COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

02/09 VOL.52 NO.02

Inspiring Women in Computing

Beyond Web 2.0

Point/Counterpoint on Net Neutrality

Improving Performance on the Internet

Compiler Research: The Next 50 Years

Making Sense of Sensors

Association for Computing Machinery

FAST '09

7th USENIX Conference on File and Storage Technologies

FEBRUARY 24–27, 2009 | SAN FRANCISCO, CA

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

FAST '09 brings together storage system researchers and practitioners to explore new directions in the design, implementation, evaluation, and deployment of storage systems. The 2009 program includes:

Training Program

- **Updated! Clustered and Parallel Storage System Technologies**
  Brent Welch and Marc Unangst, *Panasas*
- **New! Security and Usability: What Do We Know?**
  Simson Garfinkel, *Naval Postgraduate School*
- **Updated! Storage Class Memory, Technology, and Uses**
  Richard Freitas, Winfried Wilcke, Bülent Kurdi, and Geoffrey Burr, *IBM Almaden Research Center*
- **New! Web-Scale Data Management**
  Christopher Olston and Benjamin Reed, *Yahoo! Research*

Technical Sessions

Refereed Paper sessions on:

- Augmenting File System Functionality
- Meta-data and Optimization
- Distributed Storage
- Data Integrity
- and more

Work-in-Progress Reports and a Poster Session

Don't miss this opportunity to meet with premier storage system researchers and practitioners for three and one-half days of ground-breaking file and storage information and training.

Register by February 9, 2009, and save! **http://www.usenix.org/fast09/aq**

COMMUNICATIONS OF THE ACM

Departments

Last Byte

News

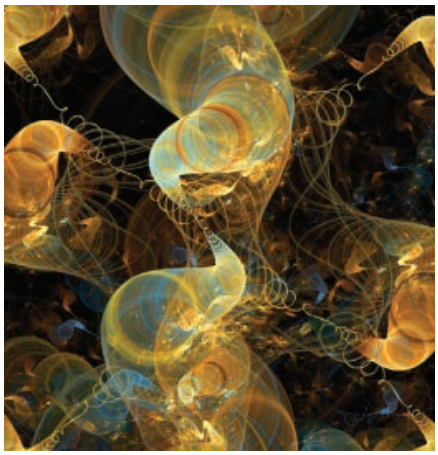
Viewpoints

PHOTOGRAPH OF JACK THIELEPAPE

Association for Computing Machinery
Advancing Computing as a Science & Profession

02/09
VOL. 52 NO. 2

Practice



About the Cover: While women's representation in computing has improved, we still have a long way to go to inspire, recruit, and retain females for a life-long career in the field. Engaging girls to the wonders of computing is just the beginning.

COVER PHOTOGRAPH BY MICHAEL GOLDMAN/GETTY IMAGES. FRACTAL BY SCOTT DRAVES

Contributed Articles

Review Articles

Research Highlights

Virtual Extension

As with all magazines, page limitations often prevent the publication of articles that might otherwise be included in the print edition. To ensure timely publication, ACM created *Communications*' Virtual Extension (VE).

VE articles undergo the same rigorous review process as those in the print edition and are accepted for publication on their merit. These articles are now available to ACM members in the Digital Library.

Association for Computing Machinery
Advancing Computing as a Science & Profession

COMMUNICATIONS OF THE ACM

A monthly publication of ACM Media

Communications of the ACM is the leading monthly print and online magazine for the computing and information technology fields. *Communications* is recognized as the most trusted and knowledgeable source of industry information for today's computing professional. *Communications* brings its readership in-depth coverage of emerging areas of computer science, new trends in information technology, and practical applications. Industry leaders use *Communications* as a platform to present and debate various technology implications, public policies, engineering challenges, and market trends. The prestige and unmatched reputation that *Communications of the ACM* enjoys today is built upon a 50-year commitment to high-quality editorial content and a steadfast dedication to advancing the arts, sciences, and applications of information technology.

ACM, the world's largest educational and scientific computing society, delivers resources that advance computing as a science and profession. ACM provides the computing field's premier Digital Library and serves its members and the computing profession with leading-edge publications, conferences, and career resources.

BPA Audit Pending

Subscriptions
Annual subscription cost is included in the society member dues of $99.00 (for students, cost is included in $42.00 dues); the nonmember annual subscription rate is $100.00.

ACM Media Advertising Policy
Communications of the ACM and other ACM Media publications accept advertising in both print and electronic formats. All advertising in ACM Media publications is at the discretion of ACM and is intended to provide financial support for the various activities and services for ACM members. Current Advertising Rates can be found by visiting http://www.acm-media.org or by contacting ACM Media Sales at (212) 626-0654.

Single Copies
Single copies of *Communications of the ACM* are available for purchase. Please contact acmhelp@acm.org.

COMMUNICATIONS OF THE ACM

(ISSN 0001-0782) is published monthly by ACM Media, 2 Penn Plaza, Suite 701, New York, NY 10121-0701. Periodicals postage paid at New York, NY 10001, and other mailing offices.

POSTMASTER

Please send address changes to
Communications of the ACM
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA

Association for Computing Machinery

Printed in the U.S.A.

DOI:10.1145/1461928.1461929

Eugene H. Spafford

USACM's Policy Role

As ACM members we understand that computing technologies enable and support much of modern society. Furthermore,

we envision computing advances effecting positive transformations in government, business, and society. Those changes may conflict with traditions and policies developed in other times, however. Laws and regulations significantly impact our efforts in computing, and policymakers often do not understand the underlying technology. ACM members have a professional duty to ensure that the public comprehends and benefits from advances in computing. Thus, ACM clearly must have a role in educating policymakers and shaping policy.

The ACM U.S. Public Policy Committee (USACM) is chartered to address public policy issues in the U.S. in a nonpartisan, proactive manner. In conjunction with the ACM's Washington office, USACM members track legislative and regulatory issues at the federal level (and sometimes at the state level). Membership represents a cross section of the ACM, including representatives from the SIG Board and several ACM Committees. USACM regularly produces briefs, educational materials, and Congressional testimony on key computing issues. Members also regularly engage key personnel in government agencies and advocacy organizations. Our activities are directed to ensure that policymakers understand both the capabilities and limitations of computing.

USACM's priorities for providing information will continue to be important under the new administration:

Privacy. Government is in an extraordinary position to compile information about people and organizations for legitimate reasons: law enforcement, tax collection, national security, census collection, among others. The private sector is also accumulating ever-increasing amounts of personal information. Unfortunately, "leaks" and information misuse imperil privacy and enable crime, often because of insufficient attention is devoted to privacy during system design and operation.

Currently, there are two looming privacy concerns: implementation of the REAL-ID Act, a de facto national ID program using state driver's licenses; and using databases with sensitive, personal information to verify employment eligibility (E-Verify). USACM will continue to advise legislators how to mitigate the concerns with these initiatives.

Reliability and Security. Appropriate computing technology can render government activities more effective and economical. Unfortunately, technology can suffer failures—whether accidental or malicious—with impacts that are not always understood by policymakers. Given proposed reforms to health care, financial systems, and cyber security improvements, USACM will undoubtedly need to provide continued suggestions about safeguards and protections.

Accuracy. Many people do not understand computing's limitations. In response, USACM has provided guidance on issues such as biometrics and data-matching error rates. We continue to advise policymakers about how to use computing technologies so as to avoid adverse consequences.

Voting. Using computers in election systems without adequate protection against fraud and error is a long-standing concern for USACM. Several voting bills have stalled in Congress in recent years, but the issue continues to be active at both state and federal levels. We will continue to pursue appropriate reforms and safeguards.

Intellectual Property (IP). ACM is a major publisher and the ACM trademark is an important asset. Many ACM members produce intellectual property. USACM respects the legal framework that allows property owners and creators to have some control over how their IP is used. However, we have concerns about regulations (for example, the Digital Millennium Copyright Act) and technologies that impact scholarship, fair use, reverse analysis for accessibility and security, and other reasonable uses of IP. USACM continues to champion an equitable IP regime.

Accessibility. USACM advocates for computing access that is fair and inclusive for everyone, including people with disabilities. Our recent public statement on this topic involved contributions by SIGACCESS, SIGCHI, and SIGWEB. USACM is promoting this position for both existing and future systems.

In addition to these projects, USACM also works closely with the Computing Research Association on diversity and science funding policy. K–12 education policy is addressed by our peer, the ACM Education Policy Committee.

These issues all relate to computing technology in important ways, but none can be solved with technology alone. Instead, workable solutions require people who understand the technology, and who invest the effort to understand and participate in the policy environment. USACM has addressed this challenge for 15 years, and the set of issues to address keeps growing. That's good, because it reflects the growing importance of computing and the ACM!

For more information about USACM, visit http://www.acm.org/usacm/.

Eugene H. Spafford is a professor of CS at Purdue University and executive director of CERIAS. He is also chair of ACM's U.S. Public Policy Committee.

ACM, Uniting the World's Computing Professionals, Researchers, Educators, and Students

Dear Colleague,

At a time when computing is at the center of the growing demand for technology jobs worldwide, ACM is continuing its work on initiatives to help computing professionals stay competitive in the global community. ACM's increasing involvement in initiatives aimed at ensuring the health of the computing discipline and profession serve to help ACM reach its full potential as a global and diverse society which continues to serve new and unique opportunities for its members.

As part of ACM's overall mission to advance computing as a science and a profession, our invaluable member benefits are designed to help you achieve success by providing you with the resources you need to advance your career and stay at the forefront of the latest technologies.

MEMBER BENEFITS INCLUDE:

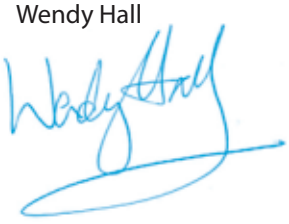
- Access to ACM's **Career & Job Center** offering a host of exclusive career-enhancing benefits
- **Free e-mentoring services** provided by MentorNet®
- **Full access to over 3,000 online courses** from SkillSoft®
- **Full access to 600 online books** from Safari® Books Online, featuring leading publishers, including O'Reilly (Professional Members only)
- **Full access to 500 online books** from Books24x7®
- A subscription to ACM's flagship monthly magazine, ***Communications of the ACM***
- Full member access to the new ***ACM Queue*** website featuring blogs, online discussions and debates, plus video and audio content
- The option to subscribe to the full **ACM Digital Library**
- The **Guide to Computing Literature**, with over one million searchable bibliographic citations
- The option to connect with the **best thinkers in computing** by joining **34 Special Interest Groups** or **hundreds of local chapters**
- **ACM's 40+ journals and magazines** at special member-only rates
- ***TechNews***, ACM's tri-weekly email digest delivering stories on the latest IT news
- ***CareerNews***, ACM's bi-monthly email digest providing career-related topics
- ***MemberNet***, ACM's e-newsletter, covering ACM people and activities
- **Email forwarding service & filtering service**, providing members with a free acm.org email address and **Postini** spam filtering
- And much, much more

ACM's worldwide network of over 92,000 members range from students to seasoned professionals and includes many of the leaders in the field. ACM members get access to this network and the advantages that come from their expertise to keep you at the forefront of the technology world.

Please take a moment to consider the value of an ACM membership for your career and your future in the dynamic computing profession.

Sincerely,

Wendy Hall



President
Association for Computing Machinery

Association for Computing Machinery

Advancing Computing as a Science & Profession

Association for Computing Machinery

Advancing Computing as a Science & Profession

membership application & *digital library* order form

Priority Code: ACACM28

You can join ACM in several easy ways:

Online
http://www.acm.org/join

Phone
+1-800-342-6626 (US & Canada)
+1-212-626-0500 (Global)

Fax
+1-212-944-1318

Or, complete this application and return with payment via postal mail

Special rates for residents of developing countries:
http://www.acm.org/membership/L2-3/

Special rates for members of sister societies:
http://www.acm.org/membership/dues.html

Please print clearly

Name

Address

City | State/Province | Postal code/Zip

Country | E-mail address

Area code & Daytime phone | Fax | Member number, if applicable

Purposes of ACM

ACM is dedicated to:
1) advancing the art, science, engineering, and application of information technology
2) fostering the open interchange of information to serve both professionals and the public
3) promoting the highest professional and ethics standards

I agree with the Purposes of ACM:

Signature

ACM Code of Ethics:
http://www.acm.org/serving/ethics.html

choose one membership option:

PROFESSIONAL MEMBERSHIP:

- ❑ ACM Professional Membership: $99 USD
- ❑ ACM Professional Membership plus the ACM Digital Library: $198 USD ($99 dues + $99 DL)
- ❑ ACM Digital Library: $99 USD (must be an ACM member)

STUDENT MEMBERSHIP:

- ❑ ACM Student Membership: $19 USD
- ❑ ACM Student Membership plus the ACM Digital Library: $42 USD
- ❑ ACM Student Membership PLUS Print *CACM* Magazine: $42 USD
- ❑ ACM Student Membership w/Digital Library PLUS Print *CACM* Magazine: $62 USD

All new ACM members will receive an ACM membership card.
For more information, please visit us at www.acm.org

Professional membership dues include $40 toward a subscription to *Communications of the ACM*. Member dues, subscriptions, and optional contributions are tax-deductible under certain circumstances. Please consult with your tax advisor.

RETURN COMPLETED APPLICATION TO:

Association for Computing Machinery, Inc.
General Post Office
P.O. Box 30777
New York, NY 10087-0777

Questions? E-mail us at acmhelp@acm.org
Or call +1-800-342-6626 to speak to a live representative

Satisfaction Guaranteed!

payment:

Payment must accompany application. If paying by check or money order, make payable to ACM, Inc. in US dollars or foreign currency at current exchange rate.

❑ Visa/MasterCard ❑ American Express ❑ Check/money order

❑ Professional Member Dues ($99 or $198) $ _____

❑ ACM Digital Library ($99) $ _____

❑ Student Member Dues ($19, $42, or $62) $ _____

Total Amount Due $ _____

Card # Expiration date

Signature

DOI:10.1145/1461928.1461931

Seven Principles for Secure E-Voting

E-VOTING CAN BE as secure and confidential as paper-based voting, as discussed in the "Point/Counterpoint" "The U.S. Should Ban Paperless Electronic Voting Machines" by David L. Dill and Daniel Castro (Oct. 2008). However, to work properly, such systems must first incorporate seven design principles:

Proven security. All protocols and techniques must be mathematically proven secure. One-time-pad-based methods qualify, while popular cryptographic methods (such as AES, DES, RSA, and SHA) do not; historically, every cipher not proven secure has been broken;

Trustworthy design responsibility. Government security agencies (such as the U.S. National Security Agency and the German Bundesamt für Sicherheit in der Informationstechnik) should be responsible for creating secure voting systems, though this work must be inspected and audited by experts selected and approved by all major parties taking part in elections. Private companies have shown they are unable to secure critical systems, including government ciphers and nuclear launch codes, so should not be entrusted to secure elections;

Published source code. The source code of all election software must be published and made publicly accessible;

Vote verification. All voters must be able to verify their votes as part of a complete nationwide tally of votes, as well as of individual voting-district-based tallies;

Voter accessibility. A full list of voters must be available to all citizens, allowing them to verify its accuracy; date and place of birth might be necessary parts of voter records to assure detection of duplicate entries;

Ensure anonymization. Techniques like onion routing must be used to ensure anonymization; and

Expert oversight. The government's responsibility in the election system (including defense against denial-of-service attacks) must be handled by a team of experts selected and approved by all major parties taking part in elections.

Simple, mathematically secure methods for e-voting are conceivable, and voters' confidence can be increased by allowing them to verify their votes in the nationwide tally, as well as review the full list of voters. As with any cryptographic method, the system must still rely on a chain of mutual trust, which will always be necessary. The chain of trust inherent in practical cryptography cannot be ignored in e-voting. Moreover, boot-from-CD voting software like Linux Live CDs, one-time pads, and onion routing would support more direct democracy. The economics of e-voting allow for much cheaper voting, thereby allowing more elections on a larger number of specific policy decisions.

Frank Gerlach,
Baden-Württemberg, Germany

Dill Responds:

Rather than critiquing Gerlach's complex yet vague proposal, I return to the question addressed in the debate. Suppose, for the sake of argument, that an elaborated version of that scheme allowed its operator to change votes without detection. The current and proposed standards and certification processes would be completely ineffective at protecting voters from such a system. A certification system that would be able to assure the security of paperless voting systems will not exist for many years, maybe never. On the other hand, it is possible to write testable requirements for secure voter-verified paper-ballot systems. That's one reason they should be mandated.

David L. Dill, Stanford, CA

Castro Responds:

Many of Gerlach's suggestions can (and often are) used in today's e-voting systems and elections. Security, accountability, usability, and cost will all continue to be important factors in evaluating these systems. However, I disagree with the premise that in essence "nationalizing" the voting-machine industry is a good solution. The quality of an engineer's final product is not dependent on whether or not the engineer works for private industry or for government. Competition can ensure that innovation continues and better voting system standards protect the electorate from unnecessary risk.

Daniel Castro, Washington, D.C.

Send IT Employees to Teach

Reading "Crossroads for Canadian CS Enrollment" by Jacob Slonim et al. (Oct. 2008), I thought of something not discussed directly in the article: Why not have industry employees contribute directly to the education process? Many of the causes the authors attributed to the decline of CS enrollment can be, at least partially, addressed by increasing the participation of IT employees in high school and university education. A general proposal would involve companies in the technology community initiating employee-teaching programs that give employees the option of serving as high school teachers or as visiting university professors for some period of time, say, two to five years. They could use their practical knowledge and industry ties to:

Share industry trends and create curricula for high schools and university CS departments that more closely align with the demands of industry;

Teach university courses on industrial topics or new technologies not traditionally addressed by CS departments;

Provide the kind of computing knowledge high school faculty often lack;

Serve as a visible representative of computing in the educational setting, as well as a role model for students; and

Create a direct line of communication between industrial and educational organizations.

This program should coincide with other industrial initiatives (such as donating equipment to schools and giv-

ing universities access to online training materials).

Declining interest in computing is disheartening and must be addressed for the future health of both the Canadian and the U.S. technology industries. For industry, money and effort spent today on education should be seen as an investment in its own future.

Bill Bushey, New Paltz, NY

Trickle Algorithm Corrections

The article "The Emergence of a Networking Primitive in Wireless Sensor Networks" on the Trickle algorithm I coauthored in July 2008 had two errors:

Sun SPOT sleeps. The article's description of the Sun SPOT platform said SPOT sleeps by writing its RAM contents to flash while requiring significant time and energy to do so. The SPOT has an external RAM bank to which it saves its internal processor state when it sleeps and, by itself, does not incur a significant cost. However, SPOT wakeup requires tens of milliseconds to stabilize timing circuits and restore processor state. The alternative, used in most simple sensor-node designs, is to require just a few kilobytes of RAM and microcontroller wakeup times of tens of microseconds. This fast wakeup time allows nodes to quickly check for network traffic while spending less energy warming up to perform the checks; and

Srcr mesh routing protocol. The citation for the Srcr mesh routing protocol designed by John Bicket incorrectly cited Douglas S.J. De Couto's MobiCom 2003 paper "A High-Throughput Path Metric for Multi-Hop Wireless Routing" when it should have cited John Bicket's MobiCom 2005 paper "Architecture and Evaluation of an Unplanned 802.11b Mesh Network." The citation for De Couto's paper also incorrectly listed the author's name as Couto, D.D. rather than as De Couto, D.

Please accept my apology for these errors. I thank Randy Smith, as well as Douglas De Couto, for pointing them out.

Philip Levis, Stanford, CA

Communications welcomes your opinion. To submit a Letter to the Editor, please limit your comments to 500 words or less and send to letters@cacm.acm.org.

DOI:10.1145/1461928.1461932

David Roman

The Dot-Org Difference

One of the first things you'll notice about the new *Communications* Web site (**cacm.acm.org**) is that it has different content than the monthly magazine. A distinctive example is the blogs. The site hosts two kinds. **Syndicated blogs**, about a dozen of them, provide an array of insight, opinion, and information from innovative technology movers and thinkers. All were recommended by ACM members and most are written by ACM members. They reflect the international scope of the computing world.

In addition, the new *Communications* site is proud to present its own **expert blog** manned by a Who's Who of leading computer science researchers and practitioners. These include MIT's Scott Aaronson, Georgia Tech's Mark Guzdial, ACM Fellow James J. "Jim" Horning, IBM Research's Tessa Lau, Google's Peter Norvig, Microsoft Research's Daniel A. Reed, MIT's Michael Stonebraker, CMU's Jeannette M. Wing, and other illustrious names. Their collective observations and insights are sure to spark great interest and debate.

The expert and syndicated blogs do several things for *Communications*' online readers: they provide valuable content; generate discussion; and encourage communication.

ACM Member News

OBAMA URGED TO INCLUDE COMPUTER SCIENCE IN K–12 MATH AND SCIENCE EDUCATION

ACM has issued a set of recommendations to the Obama administration, supporting its goal of making mathematics and science education a national priority at the K–12 level and urging it to include computer science as an integral part of the nation's education system. "Computing education benefits all students, not just those interested in pursuing computer science or information technology careers," notes

Bobby Schnabel, chair of ACM's Education Policy Committee. "To meet the nation's educational and professional needs in the face of insufficient numbers of undergraduates majoring in computer science, we need to work harder to increase interest at the K–12 level, and to expand the pipeline supplying the necessary work force for an information-based economy." Among ACM's recommendations are to consider "computer science as one of the core courses students need to develop critical 21st century skills as part of any STEM education initiative," strengthen efforts to introduce students to computer science in middle school as middle school curriculum is very influential in determining children's future interests, and "expand efforts to increase the number of females and underrepresented minorities" in STEM education.

SIGCSE AWARD WINNERS

At SIGCSE 2009, Eugene Spafford, executive director of CERIAS at Purdue University, received the 2009 Abacus Award from Upsilon Pi Epsilon for his sustained commitment to students in computing. Also, Michael J. Clancy, a senior lecturer at the University of California at Berkeley, won the Lifetime Service to Computer Science Education Community award and Elliot B. Koffman, a professor of computer and information sciences at Temple University, won the Outstanding Contribution to Computer Science award.

Science | DOI:10.1145/1461928.1461933 Kirk L. Kroeker

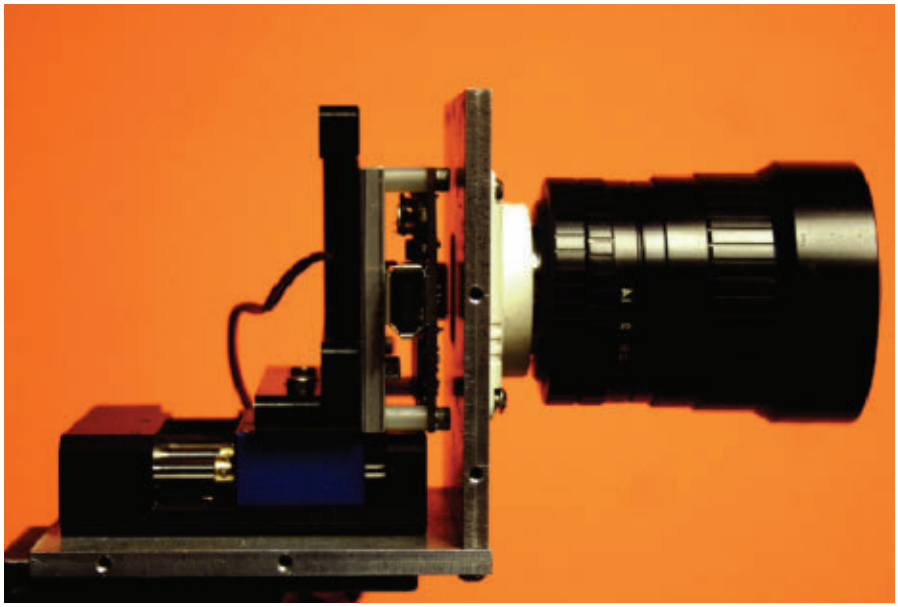
Photography's Bright Future

Researchers working in computational photography are using computer vision, computer graphics, and applied optics to bring a vast array of new capabilities to digital cameras.

While the first digital cameras were bulky, possessed few features, and captured images that were of dubious quality, digital cameras today pack an enormous array of features and technologies into small designs. And the quality of pictures captured on these cameras has improved dramatically, even to the point where most professional photographers have abandoned film and shoot exclusively with digital equipment. Given that digital photography has established itself as superior to analog film in many aspects, it might seem safe to assume that the next breakthroughs in this area will be along the lines of more megapixels or smaller handheld designs. However, researchers working in the emerging area of computational photography—a movement that draws on computer vision, computer graphics, and applied optics—say the next major breakthroughs in digital photography will be in how images are captured and processed.

Indeed, the technology powering digital photography is rapidly improving and is certainly facilitating the ability to capture images at increasingly high resolutions on ever smaller hardware. But most digital cameras today still operate much like traditional film cameras, offering a similar set of features and options. Researchers working in computational photography are pushing for new technologies and designs that will give digital cameras abilities that analog cameras do not have, such as the ability to capture multiple views in one image or change focal settings even after a shot is taken.

"There is tremendous enthusiasm for computational photography," says



A camera mechanism designed to enhance depth of field without compromising light quality. A sensor mounted on a movable platform and controlled by a microactuator can capture an image that is equally blurred everywhere but can be deblurred to produce an image with an unusually large depth of field.

PHOTOGRAPH BY SHREE NAYAR

Shree Nayar, a professor of computer science at Columbia University. "The game becomes interesting when you think about optics and computation within the same framework, designing optics for the computations and designing computations to support a certain type of optics." That line of thinking, Nayar says, has been evolving in both fields, optics on one side and computer vision and graphics on the other.

Multiple Images

One of the most visually striking examples of computational photography is high dynamic range (HDR) imaging, a technique that involves using photo-editing software to stitch together multiple images taken at different exposures. HDR images—many of which have a vibrant, surreal quality—are almost certain to elicit some degree of surprise in those who aren't yet familiar with the technique. But the traditional process of creating a single HDR image is painstaking and cannot be accomplished with moving scenes because of the requirement to take multiple images at different exposures. In addition to being inconvenient, traditional HDR techniques require expertise beyond the ability or interest of casual photographers unfamiliar with photo-editing software. However, those working in computational photography have been looking for innovative ways not only to eliminate the time it takes to create an HDR image, but also to sidestep the learning curve associated with the technique.

With computational photography, people can change a camera's focal settings after a photo is taken.

"It turns out that you can do HDR with a single picture," says Nayar. "Instead of all pixels having equal sensitivity on your detector, imagine that neighboring pixels have different sunshades on them—one is completely open, one is a little bit dark, one even darker, and so on." With this technique, early variations of which have begun to appear in digital cameras, such as recent models in the Fujifilm FinePix line, the multiple exposures required of an HDR image would be a seamless operation initiated by the user with a single button press.

Another research area in computational photography is depth of field. In traditional photography, if you want a large depth of field—where everything in a scene is in focus—the only way to do so is to make the camera's aperture very small, which prevents the camera from gathering light and causes images to look grainy. Conversely, if you want a good picture in terms of brightness and color, then you must open the camera's aperture, which results in a reduced depth of field. Nayar, whose work involves developing vision sensors and creating algorithms for scene interpretation, has been able to extend depth of field without compromising light by moving an image sensor along a camera's optical axis. "Essentially what you are doing is that while the image is being captured, you are sweeping the focus plane through the scene," he explains. "And what you end up with is, of course, a picture that is blurred, but is equally blurred everywhere." Applying a deconvolution algorithm to the blurred image can recover a picture that Nayar says doesn't compromise the quality of the image.

One of the major issues that those working in computational photography face is testing their developments on real-world cameras. With few exceptions, the majority of researchers working in this area generally don't take apart cameras or try to make their own, which means most research teams are limited to what they can do with existing cameras and a sequence of images. "It would be nicer if they could program the camera," says Marc Levoy, a professor of computer science and electrical engineering at Stanford University. Levoy, whose research involves light-field sensing and applications of computer graphics in microscopy and biology, says that even those researchers who take apart cameras to build their own typically do not program them in real time to do on-the-spot changes for different autofocus algorithms or different metering algorithms, for example.

"No researchers have really addressed those kind of things because they don't have a camera they can play with," he says. "The goal of our open-source camera project is to open a new angle in computational photography by providing researchers and students with a camera they can program." Of course, cameras do have computer software in them now, but the vast majority of the software is not available to researchers. "It's a highly competitive, IP-protected, insular industry," says Levoy. "And we'd like to open it up."

But in developing a programmable platform for researchers in computational photography, Levoy faces several

A photo containing 62 frames, taken through a group of trees, of Chicago's Newberry Library.

PHOTOGRAPH BY MIKE WAROT

major challenges. He says the biggest problem is trying to compile code written in an easy-to-use, high-level language down to the hardware of a camera. He likens the challenge to the early days of shading languages and graphics chips. Shaders used to be fixed functions that programmers could manipulate in only certain limited ways. However, graphics chipmakers changed their hardware to accommodate new programming languages. As a result, says Levoy, the shader languages got better in a virtuous cycle that resulted in several languages that can now be used to control the hardware of a graphics chip at a very fine scale.

"We'd like to do the same thing with cameras," Levoy says. "We'd like to allow a researcher to program a camera in a high-level language, like C++ or Python. I think we should have something in a year or two."

In addition to developing an open source platform for computational photography, Levoy is working on applying some of his research in this area to microscopy. One of his projects embeds a microlens array in a microscope with the goal of being able to refocus photographs after they are taken. Because of the presence of multiple microlenses, the technology allows for slightly shifting the viewpoint to see around the sides of objects—even after capturing an image. With a traditional microscope, you can of course refocus on an object over time by moving the microscope's stage up and down. But if you are trying to capture an object that is moving or a microscopic event that is happening very quickly, being able to take only a single shot before the scene changes is a serious drawback. In addition to Levoy's microlens array allowing images to be refocused after they are captured, the technology also offers the ability to render objects in three dimensions. "Because I can take a single snapshot and refocus up and down," he says, "I can get three-dimensional information from a single instant in time."

These and other developments in computational photography are leading to a vast array of new options for researchers, industry, and photography enthusiasts. But as with any advanced technologies that have interfaces designed to be used by humans, one of the major challenges for computational photography is usability. While computer chips can do the heavy lifting for many of these new developments, the perception that users must work with multiple images or limitless settings to generate a good photo might be a difficult barrier to overcome. Levoy points to the Casio EX-F1 camera as a positive step toward solving this usability problem. He says the EX-F1, which he calls the first computational camera, is a game changer. "With this camera, you can take a picture in a dark room and it will take a burst of photos, then align and merge them together to produce a single photograph that is not as noisy as it would be if you took a photograph without flash in a dark room," he says. "There is relatively little extra load on the person."

Marc Levoy's open-source camera project might enable researchers to program a camera in a high-level language, like C++ or Python.

Levoy predicts that cameras will follow the path of mobile phones, which, for some people, have obviated the need for a computer. "There are going to be a lot of people with digital cameras who don't have a computer and never will," he says. "Addressing that community is going to be interesting." He also predicts that high-end cameras will have amazing flexibility and point-and-shoot cameras will take much better pictures than they do now. Nayar is of a similar opinion. "One would ultimately try to develop a camera where you can press a button, take a picture, and do any number of things with it," he says, "almost like you're giving the optics new life." C

Based in Los Angeles, **Kirk L. Kroeker** is a freelance editor and writer specializing in science and technology.

Cognitive Computing

IBM's Brain-Like Computer

IBM has received a $4.9 million grant from DARPA to lead an ambitious, cross-disciplinary research project to create a new computing platform: electronic circuits that operate like a brain.

"The mind has an amazing ability to integrate ambiguous information across the senses, and it can effortlessly create the categories of time, space, object, and interrelationship from the sensory data," Dharmendra Modha, the IBM researcher who is leading the project, told BBC News. "The key idea of cognitive computing is to engineer mind-like intelligent machines by reverse engineering the structure, dynamics, function, and behavior of the brain."

The cognitive computing initiative will include neuroscientists, computer and materials scientists, and psychologists from IBM and five U.S. universities. Modha believes the time is right for the project as neuroscientists have learned a great deal about the inner workings of neurons and their connecting synapses, resulting in "wiring diagrams" for the brains of simple animals. Also, supercomputing is able to simulate brains up to the complexity level of small mammals; last year a Modha-led IBM team used its Blue Gene supercomputer to simulate the 55 million neurons and half-a-trillion synapses in a mouse's brain. "But the real challenge is then to manifest what will be learned from future simulations into real electronic devices—nanotechnology," according to Modha.

Along with IBM Almaden Research Center and IBM T. J. Watson Research Center, Stanford University, University of Wisconsin-Madison, Cornell University, Columbia University Medical Center, and University of California-Merced are participating in the project.

Modha acknowledges that the project is very ambitious in terms of its scope and goals. "We are going not just for a home run," he said, "but for a home run with the bases loaded."

Technology | DOI:10.1145/1461928.1461934 Alex Wright

Making Sense of Sensors

Researchers are recognizing the potential of position sensors to help them overcome the limitations of traditional user interfaces.

When Johnny Chung Lee started hacking his Nintendo Wiimote to explore whether the infrared sensors could detect simple finger movements, he scarcely expected the project to catapult him to YouTube microstardom. Yet within a few months, his four-minute demonstration video had garnered nearly two million views, earning him a loyal fan base of fellow DIY hackers, and helping him secure a plum job at Microsoft and a flattering profile in *The New York Times*. The Wiimote demo did more than boost Lee's job prospects, however. It also helped spark a surge of public interest in the possibilities of position sensors and gestural interfaces.

While position sensors have been around for years, dating at least as far back as the University of Illinois at Urbana-Champaign's early "dataglove" prototype in the 1970s, they have largely failed to penetrate the consumer mainstream despite periodic waves of hype. That may be starting to change, however, as evidenced by the popularity of CNN's presidential election night coverage featuring ubiquitous—some would say gratuitous—images of newscasters poking, pulling, and prodding interactive graphics on a giant "Magic Wall" (which was memorably parodied in a *Saturday Night Live* skit with Fred Armisen twiddling a map of the United States while declaring, "Check out Michigan—I can make it bounce!").

The CNN Magic Wall originated with Jeff Han's groundbreaking work in multitouch interaction (which also caused a YouTube splash when the first demos appeared online in 2006). In the age of the iPhone, however, multitouch screens have quickly become so commonplace as to seem almost banal. But researchers are now starting to explore more provocative visions for sensor-based applications that stretch far beyond the relatively simple manipulation of images on a two-dimensional screen. (For more about multitouch devices, see Ted Selker's article "Touching the Future" in the Dec. 2008 issue of *Communications*.)

Johnny Chung Lee discussing creative uses of the Wiimote at Creativity World Forum 2008.

At Philips Research Labs, researcher Mark Mertens recently filed a patent for a "throwable display" that updates itself based on the unit's position and trajectory. Envisioned primarily as a gaming device, the system employs a combination of an accelerometer and a triangulation system based on GHz radiation that measures flight time and location in relation to a set of fixed beacons and/or human actors. For example, the display could show a humanoid image sticking out its tongue to a player from a distance; then, when the display gets closer to the player, it might change its expression to a please-don't-hit-me smile. Once the sensors can deliver accurate data about the unit's location, orientation, and speed, says Mertens, "the rest is mathematics."

"You shouldn't just look at [displays], but interact with them physically: throw them, kick them, spin them," says Mark Mertens.

In this case, sensors are only part of the story. To stretch beyond traditional two-dimensional interactions, Mertens wants to explore new possibilities in display technology, incorporating Philips' innovative pillow-shaped display. Mertens thinks the combination

PHOTOGRAPH BY PIETER BAERT

of sensors and new, more flexible displays opens up all kinds of opportunities for innovation. "You shouldn't just look at [displays]," he says, "but interact with them physically: throw them, kick them, spin them."

In a sensor-equipped world, almost anything can become an interface. At Microsoft, researcher Paul Dietz is working on a new sensor technology called SurfaceWare that allows liquid containers to detect their contents and send automatic signals in response to changing conditions. In the technology's simplest application, a near-empty glass could automatically signal for a refill. Beyond realizing efficiency gains for harried bartenders, the technology holds out all kinds of possibilities for "smart" liquid containers.

While working on a predecessor project called iGlassware, Dietz used passive RFID tags married to capacitance sensors to determine the contents of a container, but that solution didn't work well with thick liquids that tend to coat a container's interior. So Dietz developed an optical prism embedded in the glass container that reflects light when in the air but not when covered with fluid, enabling the container to detect a wider range of fluids. Building on this work, Dietz is developing a new class of surface interaction widgets called dynamic tags. "Basically, these are transducers that can sense physical quantities in the environment and present the measurement in an optical form that can be read," says Dietz.

Before coming to Microsoft, Dietz worked on a series of sensor-based technologies for Mitsubishi and Disney, where he created a set of location-aware water installations, including a fountain that withdraws its stream when an observer tries to touch it, a musical harp with strings made of water, and a liquid touch screen known as the TouchPond.

Foldable Interfaces

While the market for sensor-based pillows, drinking glasses and water fountains remains to be proven, other researchers are working on sensor-based interfaces with more practical applications. Before his brush with YouTube fame, Lee developed a foldable interface designed to mimic one of the world's most familiar analog interfaces: the newspaper. Using an innovative approach to tracking objects with projected light, the display dynamically updates its contents based on its location and orientation. Like the Wiimote demo, this prototype relies on infrared sensors to determine the object's position. After experimenting with a number of alternative sensor technologies, Lee settled on infrared sensors like the ones found in the Wiimote.

Johnny Chung Lee's inspiration for the foldable display came from *Renaissance*, an animated cyberpunk/science-fiction detective film.

An alternative approach might have involved using a camera to track the display, by giving the display unit unusual optical characteristics like an orange light flashing at regular intervals. But Lee believes that camera-based displays are inherently limiting. He prefers infrared sensors because they are cheaper and more reliable.

"One of the problems of camera tracking is that if there is more than one point, the camera can't tell them apart. But a light sensor can tell them apart. With cameras, the lights are broadcasting outward, but with a projector, the light 'knows' where it came from," Lee explains. "There's relatively little infrared interference in the world. Very little else is blinking at 50 kHz."

The foldable display looks like something right out of *Minority Report*, the 2002 Steven Spielberg film that has served as a touchpoint for many interface designers. However, Lee says his inspiration actually came from *Renaissance*, an animated cyberpunk/science-fiction detective film by Christian Volckman. Lee admits to finding inspiration throughout the pop culture world, often basing his projects on ideas found in Japanese anime and other films. "Usually I already have a list of things I'd like to do," Lee says, "but films sometimes give me the inspiration to actually do it." In a world where most people have become accustomed to the traditional display-keyboard-mouse interface, looking to films and other reference points outside the world of computer science may help developers find fertile new ground for thinking about alternative interfaces.

Some developers find inspiration in science-fiction and fantasy films. For others, the inspiration comes from hard-won experience. Dietz traces his original inspiration back thirty years to his childhood, when he and his brothers hacked a Heathkit H-8 computer kit to control a toy train set by using LEDs as light receivers and small magnetic reed switches wired into the computer. As Dietz recalls, "The train would automatically start up and go from station to station, switching the switches, displaying a live map, and even playing appropriate background music. Not bad for 1978!"

Whatever the inspiration, researchers are gradually recognizing the potential of position sensors to help them overcome the limitations of traditional user interfaces. "We are leading increasingly sedentary lives, locked with our eyes onto the very small world behind the screen of a traditional display," says Mertens, "and that is still how many computer games force you to behave."

Mertens believes the future of interaction design involves bringing the computer out from the other side of the glass, and building bridges into the physical realm. "Instead of playing behind the screen in a virtual computer world, you have to take the display into the real world," he says.

When sensors start to do more than just transmit sensory data to a traditional two-dimensional computer screen, the way we interact with computers will fundamentally shift, as physical objects become "smarter" about themselves and the world around them. When that starts to happen—when computers start taking shape in three dimensions—sensors may just start making sense. C

Alex Wright is a writer and information architect who lives and works in New York City.

Society | DOI:10.1145/1461928.1461935 Samuel Greengard

The First Internet President

Barak Obama's presidential campaign utilized the Internet and information technology unlike any previous political campaign. How politicians and the public interact will never be the same.

When Barack Obama stepped on stage in Chicago's Grant Park to deliver his victory speech last November 4, it represented a defining moment in American history. Although the news media and the public couldn't help but recognize the historic fact that Obama had become the nation's first black president, it was no less significant that the 47-year-old community activist and politician had also become America's first Internet president.

Throughout a two-year campaign, Obama's political team—including campaign manager David Plouffe and senior adviser David Axelrod—tapped into information technology to redefine the election process and interact with people in new and different ways. The campaign team mined email addresses and used them to build a database of more than 13 million people, they turned to social networking sites such as Facebook to amass followers and disperse information, and they posted videos on the campaign Web site BarackObama.com as well as on YouTube.

It was a winning strategy. However, the ripple effects are likely to extend far beyond future elections and into the White House and government itself. Political observers say that politics has reached a critical threshold and there's no turning back. "The ability to connect via the Internet to groups, segments, and individuals changes everything. It flattens the process and creates a bottom-up approach to participation," says Joe Trippi, who pioneered the use of the Internet in Howard Dean's 2004 presidential bid and has worked on the presidential campaigns of Edward Kennedy, Walter Mondale, Gary Hart, Dick Gephardt, and John Edwards.

"This was a watershed election," adds Mitch Kapor, co-founder of the Electronic Frontier Foundation and now a principal at Kapor Enterprises. "It has set a tone for the country. There's a growing recognition that information technology is here to stay. It has moved into the mainstream of American politics."

Barak Obama's Web site enabled his presidential campaign to communicate directly with supporters, launch canvassing and get-out-the-vote campaigns, and raise millions of dollars.

Amassing Media

Throughout history, political candidates have searched for every advantage in the quest to get elected. Town hall meetings, radio addresses, and television appearances have all served as valuable tools to capture hearts, minds, and votes. However, as these tools have evolved, one thing has become perfectly clear: traditional mass media—despite its long reach and powerful influence—has unmistakable limitations. It cannot be targeted to specific segments and demographic groups as it offers a one-size-fits-all message.

Of course, since the mid-1990s, candidates have constructed Web sites and used them to promote their agenda. But in the Web 1.0 world, these sites served as little more than e-brochures, allowing candidates to post news, information, and positions on various issues. They made it easier to disperse information, but did nothing to target groups of voters more effectively. Then in 2004, Howard Dean began soliciting contributions via the Web—though the focus was still squarely on what Trippi describes as "big donor money and broadcast media." To be sure, the Inter-

IMAGE BY RICK KLAU

net offered enormous upside, but campaigns still spun a tight orbit around old world politics.

Fast forward to 2008 and the emergence of Web 2.0. "This was the first mass-participation election in the nation's history," observes Micah L. Sifry, co-founder and executive editor of the Personal Democracy Forum, a news site that examines how technology is changing politics. "Today's technologies are becoming as commonplace and mainstream as the telephone was to past generations," he explains. "It is fundamentally changing how candidates and the public interact." Blogs, wikis, social networking sites, and more sophisticated analytics tools have made it possible to mobilize support and money in ways that were unimaginable only a few years ago.

Sifry believes that Web 2.0 tools have already empowered citizens and armed them with a powerful force for affecting change in campaigns and government. More than 200 video-sharing sites exist, and independent groups on Facebook and MySpace now back or attack candidates without the support of the campaigns themselves. The largest independent group in the 2008 election cycle was a Facebook group, Stop Hillary Clinton: (One Million Strong AGAINST Hillary Clinton), and many smaller groups sprung up, including those focused on independent fundraisers and house parties.

Turning opportunity into action required more than just sophisticated servers and advanced databases, however. From the beginning, the Obama team understood the power of the technology and how it could be harnessed for political gain, says Deniece Peterson, principal analyst at INPUT, a Reston, VA, government-focused market research and consulting firm. "They knew how to use the technology to operate at a more grassroots level and engage in a more individual and personal outreach," she notes.

The campaign used Facebook to communicate online (it had 1.7 million supporters at the site), and tapped into microblogging site Twitter to deliver a constant stream of news as well as volunteer opportunities. For example, on election day, the Obama campaign used Twitter to post toll-free numbers and texting strings for finding polling locations, connecting to volunteer opportunities, and making contributions.

With a private database of millions of people, President Obama's staff could conduct polls, solicit ideas and opinions, and hold online town hall meetings.

Other social networking sites also entered the picture, including My-Space, YouTube, Flickr, Digg, Eventful, LinkedIn, BlackPlanet, FaithBase, Eons, GLEE, MiGente, My Batanga, AsianAve and DNC PartyBuilder. Meanwhile, BarackObama.com amassed more than two million registered users. In the end, the 13-million-person database represented about 10% of the total number of voters in the presidential election. Moreover, "these are people who also have a lot of impact offline," Trippi points out.

In the end, Obama received donations from more than four million persons—nearly triple the number that George W. Bush tallied in 2004. In fact, the final financial tally exceeded $750 million, with the average donation being less than $100. A steady stream of email solicitations made contributing as easy as buying a book on Amazon.com. "This approach is putting PACs [political action committees] and lobbies on notice that we've entered a new and far more democratic era," Trippi observes. "It's no longer only about the fat cats running the campaign and contributing to it. It's all about the average person."

Beyond the Bully Pulpit

It should come as no surprise that databases, social networking tools, and IT systems are likely to play a central role in the Obama administration. While new media has enormous power to help a candidate get elected, it also wields influence as a tool for operating a more efficient and transparent government—and advancing a political agenda. Obama's chief strategist, David Axel-

Computer Science

Award Winners

Industry veteran and Convey Computer cofounder Steven Wallach was awarded the Seymour Cray Computer Science and Engineering Award at SC08 in Austin, TX, for his "contribution to high-performance computing through design of innovative vector and parallel computing systems, notably the Convex mini-supercomputer series, a distinguished industrial career and acts of public service." In a statement, Wallach said, "This is one of our industry's greatest honors and I am deeply honored. At Convex Computer Corp., and now at Convey we are showing that you can have a highly productive software environment coupled with high performance computing. I have always believed that the machine that is simplest to program will ultimately win."

Also honored at SC08 was University of Illinois computer science professor William Gropp, who won the Sidney Fernbach Award, which honors innovative uses of high-performance computing in problem solving. Gropp played a major role in creating MPI, the standard interprocessor communication interface for large-scale parallel computers. Gropp is also co-author of MPICH, one of the most influential MPI implementations to date, and co-wrote two books on MPI.

Microsoft presented its first Jim Gray eScience Award to Carole Goble, a professor of computer science at the University of Manchester in the United Kingdom.

As director of the U.K.'s myGrid project, Goble helped create Taverna, open source software that enables scientists to quickly analyze complex data sets with a regular computer. With Taverna, extensive database research that used to take several days can now be completed in several hours.

Tony Hey, a vice president of Microsoft External Research, said Goble was chosen because her research helped scientists conduct data-intensive science. "She's a data person," said Hey, "and I think that would have pleased Jim."

rod, made it clear that the Internet and technology will play a prominent role in government when he recently stated, "We'll continue to have a conversation with [the public]."

In fact, by converting the existing campaign database into a private database, the president would have a powerful link to his constituency. Some, like Trippi, believe that President Obama could eventually cultivate 20 to 30 million people and make the database a central tool for communicating with the public. His staff could conduct polls, solicit ideas and opinions, and hold online town hall meetings.

With this iPhone application, Obama supporters could organize and prioritize their contacts by key battleground states.

The upshot? Instead of addressing the public on the radio for only a few minutes every Saturday, a two-way discourse could ensue. At press conferences, "questions could come directly from citizens," Trippi says. What's more, online groups could discuss key issues and policies and provide immediate feedback. "The president will have the ability to bypass Congress and appeal directly to citizens in a way that has never before been possible," Sifry explains. At the same time, however, citizens will have the ability to make their voices heard and petition for change.

Transparency will likely emerge as a key issue as well. Already, the Obama administration is exploring the possibility of putting lobbyist and campaign filings online, creating public/private communities to help streamline environmental reviews, and providing an opportunity for citizens to comment online about pending legislation. The administration's Change.gov site has already invited the American public to offer suggestions and comments on healthcare reform. More than 3,000 people had reportedly contributed ideas during the first month the site was live.

Make no mistake, the full power of the technology is becoming apparent. By creating persistent identities for online users—a single real identity that a person uses online—it is possible to take government to a new level. "You begin building the foundation for a robust community where there is participation; you begin to tap into the power of social networking," Sifry explains. "Once you have a system like this in place and someone enters a comment, it's possible to notify that individual the next time a healthcare bill or education bill comes up. It's also possible to poll the person and ask for additional information and ideas. It creates a whole new world of interaction."

Yet, it's clear that a new era is unfolding. As campaigns and government become more familiar with digital tools and the technology advances, the face of politics will continue to change. Kapor believes that sophisticated analytics will help guide decisions and provide tools for more advanced clickstream and data analysis. At the same time, new tools will emerge. "Most social networking tools, including Facebook, Twitter, and YouTube, either didn't exist or weren't a factor four years ago," he says. "It's impossible to imagine what will be available by the next election cycle."

The way Sifry sees it, there's no turning back. "We're going to see the Internet and information technology play a far more prominent role in politics and government moving forward," he concludes. "Political parties, advocacy groups, and others are looking at the technology and understanding that they have to use it to make an impact. It is now a major force in politics." C

Samuel Greengard is an author and freelance writer based in West Linn, OR.

Neuroscience

Extracting Brain Images

Researchers at ATR Computational Neuroscience Laboratories in Kyoto, Japan, have developed a brain analysis technology that can reconstruct images inside a person's brain and display them on a computer monitor. Future development of the technology could lead to the ability to read people's dreams while they sleep, according to the researchers, whose study was published in the U.S. journal *Neuron*.

Led by chief researcher Yukiyasu Kamitani, a team reconstructed images seen by study participants by analyzing changes in their cerebral blood flow. The researchers used a functional magnetic resonance imaging (fMRI) machine to map the blood flow changes that occurred in the cerebral visual cortex as the study participants viewed different images. The participants were shown some 400 black-and-white images for 12 seconds each, and the fMRI machine monitored the changes in their brain activity, while a computer program analyzed the data and learned how to correlate the changes in brain activity with the various images.

Next, the participants viewed a different set of images, such as the individual letters in a word, and the system reconstructed and displayed what the participants were viewing based solely on their brain activity.

"These results are a breakthrough in terms of understanding brain activity," said Kang Cheng, a researcher at the RIKEN Brain Science Institute in Saitama, Japan. "In as little as 10 years, advances in this field of research may make it possible to read a person's thoughts with some degree of accuracy."

The researchers believe that their technology could have applications in the fields of art and design and might lead to new treatments for psychiatric disorders by enabling doctors to directly view a patient's mind.

PHOTOGRAPH BY ACHIM HEPP AND VINCE BROWN

News | DOI:10.1145/1461928.1461936 Kirk L. Kroeker

SIGGRAPH Debuts in Asia

ACM's premier computer graphics conference hosts its first-ever graphics event in Asia, with a more global focus.

EACH YEAR, COMPUTING enthusiasts from around the world are drawn to SIGGRAPH, the annual ACM conference that has built a reputation as a must-attend event for the latest breakthroughs in computer graphics and related technologies. In a move that conference organizers say was motivated by a desire to foster a more global focus at the event and to recognize Asia's growing importance in graphics and digital media, SIGGRAPH Asia took place in Singapore, marking the first time that SIGGRAPH has appeared outside North America.

The inaugural SIGGRAPH Asia conference was held December 10–13, 2008 at the Suntec Singapore International Convention & Exhibition Centre, with programs mirroring the format of previous SIGGRAPH events. "It was a real SIGGRAPH-level conference in quality and look-and-feel," says G. Scott Owen, president of ACM SIGGRAPH and professor emeritus at Georgia State University. "I was very pleased with SIGGRAPH Asia, and everyone I talked to was also pleased."

The conference, which drew 3,389 attendees, served as a showcase for digital artwork and interactive platforms. It also offered workshops, technical presentations, and an exhibition that highlighted the newest products and services from 81 companies.

Yong Tsui Lee, conference chair of SIGGRAPH Asia 2008 and a professor at Nanyang Technological University, says it wasn't difficult to create interest in the show among those who were already familiar with SIGGRAPH. "Its reputation as the world's largest and highest-quality event in computer graphics and interactive techniques helped open doors," he says. "Its name helped draw a good audience." Still, Lee says, it was difficult to built interest in the show among people who didn't know SIGGRAPH well.

PHOTOGRAPH COURTESY OF SIGGRAPH ASIA 2008

"So, a lot of hard work was needed."

By most accounts, the hard work paid off. SIGGRAPH Asia featured notable speakers such as Cornell University's Don Greenberg, Pixar's Rob Cook, and Lucasfilm Animation Singapore's Lee Stringer and Matt Aldrich. At the Computer Animation Festival, attendees were able to view 68 films selected for screening from 685 submissions, and the Emerging Technologies program featured projects designed to push the boundaries of computer graphics. In all, SIGGRAPH Asia's Technical Papers program received 320 submissions, from which 59 were selected for publication, making for an acceptance rate slightly lower than SIGGRAPH's 10-year average.

Every SIG conference depends on the help of many volunteers, and SIGGRAPH Asia garnered the support of 135 volunteers from 17 countries. In terms of the number of attendees, Lee says it would have been nice to have more, but "looking at how some of our sessions filled up, I suppose 3,400 is about what we could handle comfortably this time."

SIGGRAPH Asia 2009 is slated for Japan, with preparations for the show already under way. "My committee and I are discussing ways to continue the SIGGRAPH Asia 2008 program, but we are considering some changes," says Masa Inakage, conference chair of SIGGRAPH Asia 2009 and dean of Keio University's Graduate School of Media Design. In particular, Inakage says he plans to expand the show's Emerging Technologies program and tailor some course content to reflect Japanese and other Asian cultures. "We are hoping to attract a much larger number of attendees," he says, "locally and internationally." C

Based in Los Angeles, **Kirk L. Kroeker** is a freelance editor and writer specializing in science and technology.

DOI:10.1145/1461928.1461937 Ashish Arora, Matej Drev, and Chris Forman

Economic and Business Dimensions

The Extent of Globalization of Software Innovation

Will the software development laboratories follow the production mills?

Has innovative activity in software become global? The question attracts interest because software has become a global business. For example, exports of business services and computer and information services grew at an average annual rate of 27% in India (1995–2003) and at a rate of 46% in Ireland (1995–2004), with similar rapid growth in Brazil, China, and Israel.[3] Yet, the question remains open. While software production takes place in many countries, innovative software is not created everywhere.

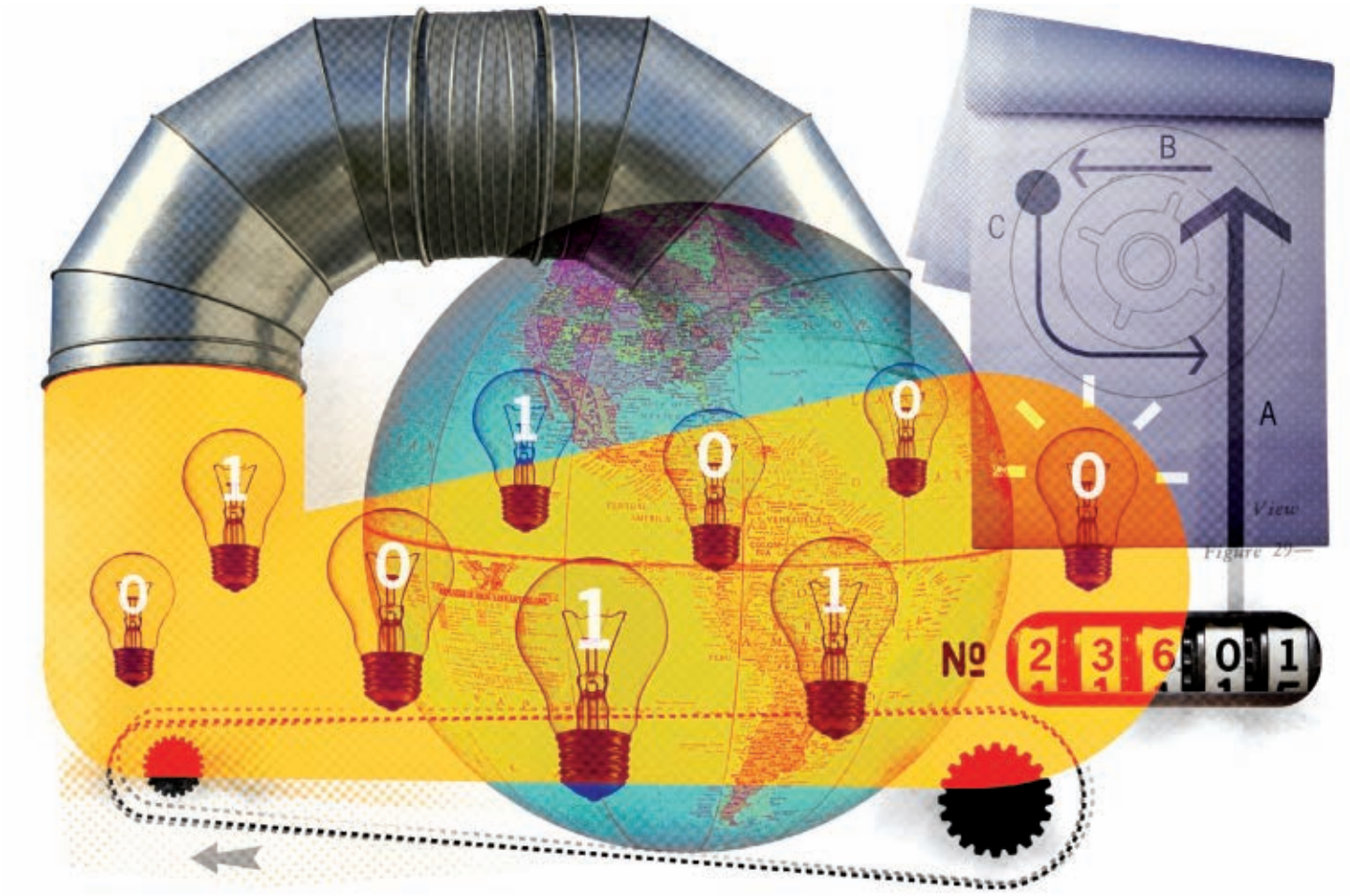
Moreover, there is increasing evidence of growing software development operations in countries such as India. For example, the Microsoft India Development Center (located in Hyderabad) has grown from two products and 20 employees in 1998 to 70 products and over 1,500 employees today. SAP Labs India is now that company's second-largest Research and Development and Global Services and Support center, with 25% of its employees engaged in research and new product development. IBM's India Software Labs similarly have large operations in Bangalore, Gurgaon, Pune, Hyderabad, and Mumbai. Yet, despite these anecdotes the question is not easy to answer at a general level. In many cases innovation cannot be defined. For that matter, sometimes software cannot be defined.

In this column we summarize and extend an investigation into software innovation.[2] We look at software developed to be sold as a standalone product, software developed to be sold as a service (such as software available for use on salesforce.com), and software developed by a user firm. We exclude software for semiconductor chips and also software written for application-specific computers such as some CAD software.

One measure of innovation appears in patent data. To be sure, not all inventions are patented and not all patents are important. However, information in the text of the patents and their technical classes can help identify patents related to software. Moreover, patents provide a consistent source of data about innovative activity and can provide some useful insights.

In 2007, 12,692 U.S. software patents were issued to inventors in the U.S.—a larger number of patents than all other areas of the world combined (6,397). That is striking because growth in software patenting between 1988 and 2007 in the U.S. was comparable to that of the rest of the world: patenting by U.S. inventors grew at an average annual rate of 32.6%, compared to 34% in the rest of the world. Software innovation in the so-called underdog countries has been growing, but not faster than in the U.S. (see the figure here).

We also interviewed the Indian software research and development labs of several large multinational firms. We found some evidence of increasing autonomy among software development centers. In particular, we found that most organizations started by giving their Indian software labs assigned development or testing work. Yet, it rarely stopped with those assignments.



Increasingly, centers have been given control of key components of software applications and in some cases have been responsible for design and development of entire products.

This was not so for all software. Indian software development centers were most likely to have design control in software projects for which there was a large local market. For example, we found significant autonomy for software projects that involved software development and testing tools and for mobile applications. The Indian market for both of these products is large.

A key determinant of the location of development activities in software is the location of the user. This is particularly true with business software, which is often bundled with a set of business rules and assumptions about business processes.

Growth in emerging economies translates into more lead users, and, thus, more local innovation. Already we have seen this in areas such as software tools, mobile technologies, and security software, but there is evidence in other areas as well. For example, i-flex, which began as an offshoot of Citibank developing products for banks in India, has leveraged its success with Indian banks to banks in other emerging markets, and in banks in developed economies as well. In other words, as the distribution of users shifts, so does the locus of software innovation.

In many cases innovation cannot be defined. For that matter, sometimes software cannot be defined.

This should come as no surprise. Historically, some of the most significant examples of business software—such as IBM's SABRE airline reservation and SAP's ERP software—have arisen through close collaboration between software firms and their users. Since knowledge of business needs is frequently not well codified, such collaboration will often be most effective when performed by firms that are in close proximity to one another.

We see the same pattern today. Conditions for development of innovative new software products (and software firms) are propitious when they occur in proximity to potential lead users. A good example is Israel's longstanding strength in security software, which arose in part due to the advanced needs of the Israeli defense forces.

Our patent data also illustrated the importance of proximity to users. Most software innovation occurring outside the U.S. is found in U.S. multinational firms. One can see this by looking at the country of assignee for patents in-

ILLUSTRATION BY STUART BRADFORD

vented outside the U.S. Among the underdog countries (Brazil, China, India, Israel, and Ireland) most commonly cited as having robust software industries, the fraction of patents assigned to U.S. firms has generally been increasing over time, ranging from 15.4% in 1990 to a high of 64.6% in 2002. This suggests that multinationals—U.S. software firms—can serve as a partial, though highly imperfect, conduit for the needs of lead users.

However, while U.S. software firms appear to be a potential conduit for user needs, they appear to be moving their innovative activity offshore much more slowly than they are moving some programming and maintenance activity. The percentage of U.S.-assigned patents that were invented in the U.S. fell from 94.3% in 1996 to 91.4% in 2007. This decrease in the share of U.S.-assigned patents invented in the U.S. is due in large part to the increase in offshore activity in the underdogs: the percent of U.S.-assigned patents invented in the underdogs rose from 1% in 1996 to 2.6% in 2005.

It will be a while before countries such as India become significant sources of software innovation. There simply are not enough highly talented computer scientists and software engineers in these countries. Moreover, the state of the science and technology infrastructure is weak. The firms we interviewed told us that supplies of fresh engineering graduates were plentiful in India, but it was much more difficult to find developers with project management experience and more difficult still to find those with critical business knowledge.

Conditions for development of innovative new software products (and software firms) are propitious when they occur in proximity to potential lead users.

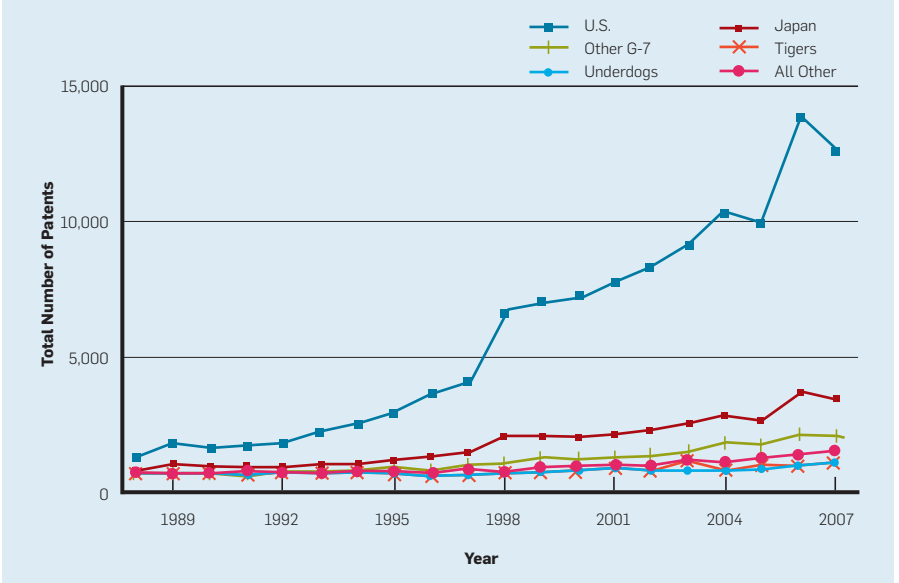
As of 2004 the number of engineering graduates in India and the U.S. were approximately equal, and the number of Indian engineering graduates is growing much faster than that in the U.S.—from 42,000 in 1992 to 128,000 in 2003.[1] Higher education growth in China has been similarly rapid. Yet, do not be misled by these numbers. Despite these improvements in educational infrastructure, R&D as a percent of GDP is only 1.44% in China compared to 2.68% in the U.S. (both 2004 data) and 0.85% in India (2000 data).

What will the future bring? First, it does appear software product development and testing activities have become increasingly global. Firms continue to experiment with new methods of managing global software development, and such methods will likely increase the share of software work that can be modularized and produced away from the point of product design and architecture.

Second, these trends raise a big question. Entry- and mid-level programming jobs have frequently provided U.S. IT workers with the skills needed to perform more complicated activities such as product design and strategy. In other words, training by U.S. firms has traditionally bestowed an uncompensated benefit to entry-level workers by providing them with certain types of general training and skills, which are very valuable to the workers later in the careers. Indeed, our interviews suggest that lack of these skills has been a significant barrier to innovation in countries outside the U.S.

Many of these entry-level jobs are now going overseas. A declining demand today for entry-level programming jobs in the U.S. and increasing demand elsewhere could make it more difficult for U.S. workers—and relatively easier for those from other countries—to perform complex software design activity in the future. The implications of this would be a more globally dispersed pattern of innovation in software than we see today. C

U.S. software patents invented in the U.S. and other countries.



References

1. Arora, A. and Bagde, S. The Indian software industry: The human capital story. Working Paper, Heinz School of Public Policy and Management, Carnegie Mellon University, 2006.
2. Arora, A., Forman, C., and Yoon, J.W. Software. In J.T. Macher and D.C. Mowery, Eds., *Innovation in Global Industries: U.S. Firms Competing in a New World.* National Academies Press, Washington, D.C., 2008, 53–99.
3. Organization for Economic Cooperation and Development. *Information Technology Outlook 2006.* OECD Publications, Paris, 2006.

Ashish Arora (ashish@andrew.cmu.edu) is H.J. Heinz Professor in the Heinz School of Public Policy and Management at Carnegie Mellon University, Pittsburgh, PA.

Matej Drev (mdrev@andrew.cmu.edu) is a Ph.D. student in the Heinz School of Public Policy and Management at Carnegie Mellon University, Pittsburgh, PA.

Chris Forman (chris.forman@mgt.gatech.edu) is the Robert and Stevie Schmidt assistant professor of IT management in the College of Management at Georgia Institute of Technology, Atlanta, GA.

The authors gratefully acknowledge funding from the Software Industry Center at Carnegie Mellon University. Chris Forman acknowledges funding from an Industry Studies Fellowship from the Alfred P. Sloan Foundation.

DOI:10.1145/1461928.1461938 George H.L. Fletcher and James J. Lu

Education

Human Computing Skills: Rethinking the K–12 Experience

Establishing the fundamentals of computational thinking is essential to improving computer science education.

Since the dot-com downturn, the conundrum we face in computer science is how such a useful discipline can have such difficulties attracting students, despite continuing growth of the IT industry. We attribute the blame for student disinterest to career instability, but similar and even stronger arguments have long existed for other disciplines with little impact on enrollment. Recent data from the National Center for Education Statistics[a] indicates the computer and information sciences conferred fewer degrees than either the visual and performing arts or the social sciences and history—certainly not majors that are typically associated with iron-clad career guarantees. Not surprisingly, CS enrollment also lags far behind that of other practically perceived disciplines such as education or business.

Of course, the problem is deeper than just the issue of career trends. Through the years, despite our best efforts to articulate that CS is more than "just programming," the misconception that the two are equivalent remains. This equation continues to project a narrow and misleading image of our discipline[3]—and directly impacts the character and number of students we attract.

In an effort to broaden awareness of and participation in the computing sciences, Jeannette Wing's recent call for teaching computational thinking (CT)[6] as a skill on par with reading, writing, and arithmetic (the so-called three Rs) places the core of CS, we believe correctly, in the category of basic knowledge. Just as proficiency in basic language arts helps us to effectively communicate and proficiency in basic math helps us to successfully quantitate, proficiency in computational thinking helps us systematically and efficiently process information and tasks.

But while teaching everyone to think computationally is a noble goal, there are pedagogical challenges. It is not enough to simply repackage CS1, or CS0, and teach it at an earlier stage, as many K–12 programs already do. Perhaps the most confounding issue is the role of programming, and whether we can separate it from teaching basic computer science. How much programming, if any, should be required for CT proficiency?

We believe that to successfully broaden awareness of the depth, breadth, and beauty of computer science, and thereby increase participation in our discipline, efforts must be made to lay the foundations of CT long before students experience their first programming language. In order to pursue alternatives to the traditional

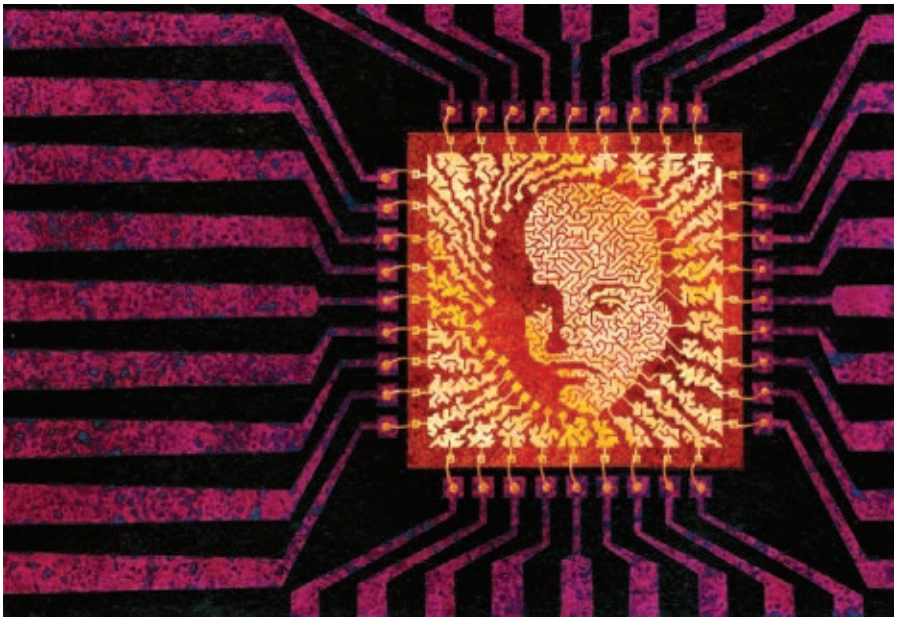


ILLUSTRATION BY MICHAEL SHUMATE

a See http://nces.ed.gov/programs/digest/d06/figures/fig_15.asp.

"programming-first" approach, then, it is necessary to consider more carefully the implications of aligning CT with the three Rs.

Programming: Describing Computational Processes

While being educated implies proficiency in basic language and quantitative skills, it does not imply knowledge of or the ability to carry out scholarly English and mathematics. Indeed, for those students interested in pursuing higher-level English and mathematics, there exist milestone courses to help make the critical intellectual leaps necessary to shift from the development of useful skills to the academic study of these subjects. Analogously, we believe the same dichotomy exists between CT, as a skill, and computer science as an academic subject. Our thesis is this: *Programming is to CS what proof construction is to mathematics and what literary analysis is to English*.

The shift to the study of CS as an academic subject cannot, of course, be achieved without intense immersion in crafting programs. In this respect, the traditional programming-first curriculum is appropriate. Knowledge of programming should not, however, be necessary to proclaim literacy in basic computer science. Just as math students come to proofs after 12 or more years of experience with basic math, and English students come to literary analysis after an even longer period of reading and writing, programming should begin for *all* students only after they have had substantial practice acting and thinking as computational agents.

We need to start teaching computational thinking early and often. But what does this entail in the absence of programming?

Missing in K–12 curricula are the mathematics equivalents of algebra and calculus, and the English equivalents of literature and composition. Missing also is the spectrum of service courses college mathematics and English departments offer to non-majors that enhance their reading, writing, and quantitative skills. Currently, the setting in which students are introduced to CT is also where they first learn programming. This pedagogical approach is akin to teaching basic arithmetic alongside proof construction, and elementary reading and writing with linguistics and discourse analysis. It can be done, but perhaps not optimally. Writing descriptions in unfamiliar formal languages cannot be easy when one does not yet have a solid grasp of the concepts and processes the descriptions are designed to capture. A corollary to our thesis, then, is the following: *Substantial preparation in computational thinking is required before students enroll in programming courses*.

The redesign and implementation of K–12 curricula to provide adequate exposure to and practice in CT should, of course, be coupled with ongoing efforts to rethink the ways in which we transition students into programming and higher-level CS.[5]

Learning to Understand Computational Processes

We need to start teaching computational thinking early and often. But what does this entail in the absence of programming? The emphasis should be on the development of human computing skills,[b] not particular programming language or pseudocode manifestations of algorithms. Gaining familiarity with algorithmic notions such as basic flow of control is important. Also central is the development of skills for abstracting and representing information, and for evaluating properties of processes.

As a glue for connecting these concepts into an identifiable discipline, a language—a computational thinking language (CTL)—that captures core CT concepts must permeate the pedagogy. Again, this is not a programming language, but rather vocabularies and symbols that can be used to annotate and describe computation, abstraction, and information, and provide notation around which semantic understanding of computational processes can be hung (see, for example [2]).

In grade school mathematics, we already speak of representing word problems, and applying algebraic rules to derive simpler forms. But at a more advanced level, we can augment the vocabulary with the search space of possible algebraic simplifications, induced by the initial state, the final state, and the set of applicable operations. We may explore the process of finding the derivation through a blind or a heuristic search.

Of course, concepts that are presented in association with the CTL must be grade-appropriate. At Grade 3, when students first encounter multistep calculations and small combinatorial problems, the phrase "search space" may simply mean a set of prespecified choices in a multiple-choice exercise, and a heuristic can be explained as a pruning strategy for eliminating those choices that are inconsistent with smaller or related problems to which students know the answer. A nice example is when, as a way of developing reading comprehension skills, students are given the task of putting a set of simple sentences into chronological order. The heuristic for pruning wrong answers is simply to first order a subset of the given sentences and eliminating inconsistent choices. Later in middle school, when permutation is first introduced, the search space concept may be revisited and broadened to include those sentence orderings that result from applying the permute operation to some initial sentence ordering.

As students encounter more complex calculations in middle school, richer CT notation should be introduced as part of the CTL. In particular, basic tuple notation for state representation, together with a simple rewrite system to annotate state changes can assist in clarifying the computational aspects of many problem solving algorithms. As an example, consider the standard notion of the greatest common divisor (gcd) of two integers *a* and

b For example, the CS Unplugged program; http://csunplugged.org.

b, a useful tool that students encounter during the study of fractions. Euclid's classic algorithm, applied to computing gcd(56,21), for example, proceeds as follows: (56,21)⇒(35,21)⇒(14,21)⇒(14,7)⇒(7,7). Here, "⇒" is an abstraction of the function λ *a,b.if a<b,(a,b-a)*; else *(a-b,b)*. We can compare the efficiency of Euclid's algorithm to an exhaustive linear search. This comparison further provides a chance to discuss representation and decision. For linear search, a single number captures the complete state since each subsequent state is a unary function of the current state. For Euclid's algorithm, a pair representation is necessary as each subsequent state is conditioned on both of the current values.

Discussion

Again, note that computers are not explicitly part of this discussion. During the introduction and development of basic CT, students are the computing agents that perform the process execution. Software tools can be used to assist in this, just as they are used in teaching basic language and math skills. But, to reemphasize, the focus here is on developing the computing skills of the students, as computers.[c]

Through practice and repeated encounters, students will become accustomed to thinking and communicating in the CTL. For students that follow up with more advanced CS courses, starting with programming, the challenge will no longer be in learning to think computationally, but in learning the nuances of new languages, how to formally describe computations in these languages, and in subsequent courses on how such descriptions are executed on a von Neumann machine. For students that do not pursue CS further, their background in computational thinking will be of substantial benefit in their professional careers and in everyday life.

On the issue of enrollment, we suspect that most students major in mathematics, English, and the humanities not for the abundance of career opportunities in these fields, but more for intellectual interests, born out of gradual and sustained exposure. We conjecture that students with similar development in CT will also be more likely to choose CS based on intellectual motivations, and, furthermore, that they will be better prepared for programming and the major curriculum.[1] Exposure to basic computer science will raise awareness in students of what CS might be as a field of inquiry, leading to a broader participation of a wider variety of students in our discipline.

Exposure to basic computer science will raise awareness in students of what CS might be as a field of inquiry, leading to a broader participation of a wider variety of students in our discipline.

To truly integrate computational thinking into current K–12 curricula undoubtedly presents significant challenges. It will necessarily be a gradual and evolutionary process, and requires coordination among many constituents of the wider education community. We consider definitive efforts toward achieving broader CT literacy as some of the most exciting, challenging, and necessary next steps in the maturation of the computer science discipline. C

c For a fascinating account of the history of human computing, see [4].

References

1. Carter, L. Why students with an apparent aptitude for computer science don't choose to major in computer science. In *Proceedings of SIGCSE 2006* (Houston, TX), 27–31.
2. Cohen, A. and Haberman, B. Computer science: A language of technology. *SIGCSE inroads 39*, 4 (2007), 65–69.
3. Denning, P.J. and McGettrick, A. Recentering computer science. *Commun. ACM 48*, 11 (Nov. 2005), 15–19.
4. Grier, D.A. *When Computers Were Human*. Princeton University Press, Princeton, NJ, 2005.
5. Guzdial, M. Paving the way for computational thinking. *Commun. ACM 51*, 8 (Aug. 2008), 25–27.
6. Wing, J.M. Computational thinking. *Commun. ACM 49*, 3 (Mar. 2006), 33–35.

George H.L. Fletcher (fletcher@vancouver.wsu.edu) is an assistant professor of computer science in the School of Engineering and Computer Science at Washington State University in Vancouver.

Calendar of Events

February 14–19
ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Raleigh, NC, Sponsored: SIGPLAN, Contact: Daniel A Reed, Email: Daniel.Reed@microsoft.com

February 20–21
IWIC '09: International Workshop on Intercultural Collaboration 2009, CA, USA, Contact: Pamela Hinds, Email: phinds@leland.stanford.edu

February 27–March 1
I3D '09: Symposium on Interactive 3D Graphics and Games, Boston, MA, Contact: Manuel Menezes Oliviera Neto, Phone: 55-51-3332-0927, Email: oliveira@inf.ufgrs.br

March 2–6
2nd International Conference on Simulation Tools and Techniques, Rome, Italy, Contact: Oliver Dalle, Phone: 33-492-387-937, Email: Olivier.dalle@sophia.inria.fr

March 3–7
The 40th ACM Technical Symposium on Computer Science Education, Chattanooga, TN, Sponsored: SIGCSE, Contact: Sue E Fitzgerald, Email: sue.fitzgerald@usermail.com

March 9–13
HRI '09: International Conference on Human Robot Interaction, La Jolla, CA, Sponsored: SIGART, Contact: Matthias Scheutz, Email: mscheutz@indiana.edu

March 11–13
ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Washington DC, Sponsored: SIGOPS, SIGPLAN, Contact: Antony L Hosking, Email: hosking@cs.purdue.edu

DOI:10.1145/1461928.1461940

Kristina Irion

Privacy and Security
International Communications Surveillance

Aren't we all foreigners when our Internet traffic transits through other countries and is subject to regional intelligence-gathering policies?

When you send email, do you know through which countries your communication will be routed? In a world where countries use the Internet to gather intelligence from communications traffic that transits local facilities, this question has become increasingly important for Internet users—individuals and businesses alike. Such interception is an established investigative tool of intelligence services and law enforcement agencies all over the world provided for by domestic laws. For governments concerned with national-security threats, the exploitation of all available sources for intelligence gathering seems obvious. This surveillance is constantly being expanded—to the detriment of communications privacy. Countries are increasingly adopting legislation that provides for preventive surveillance and the massive collection of communications data without either adequate procedural limitations or strict oversight of the activities. In the worst case governments stretch—or even ignore—existing rules in order to facilitate intelligence gathering no matter what.

The fact that Internet traffic is supranational in character offers a promising new avenue for intelligence gathering by targeting international communications originating from, terminating in, or simply passing through a given country and subjecting it to the local standards of legal interception. Unlike the public switched telephony network (PSTN), which delivered only the destined traffic to the international gateways, Internet traffic is not confined to the territory of a state and is more likely to cross borders while in transit. Not long ago the overwhelming majority of data flowed through the U.S., where the world's top Internet backbone providers' switching equipment was located (the situation has since changed somewhat). The U.S. was therefore in the position of being able to exercise control over most of the world's information transmitted via the Internet.

In 2005 the U.S. National Security Agency's warrantless wiretapping, a program authorized by the Bush administration, was disclosed. Afterward the government pursued legislation to expand surveillance powers. The short-lived 2007 Protect America Act and its immediate successor, the Foreign Intelligence Surveillance Act (FISA) Amendments Act of 2008, permit warrantless interception of international communications during transit through the U.S. and the targeting of non-U.S. persons reasonably believed to be located outside the U.S. Under the latter U.S. act, the highest level of protection is afforded to purely domestic communications, interception of which would require a warrant whereas international communications (with at least one foreign end-

ILLUSTRATION BY CELIA JOHNSON

point) are more exposed to surveillance activities. If this logic were to be adopted worldwide, a citizen would have privacy of communications only in the nation in which they had citizenship—and then only if the communications remained fully within the nation's borders, a situation not always guaranteed when using the decentralized architecture of the Internet. From the perspective of all other countries the same Internet communications would be treated as foreign communications and are thus susceptible to surveillance when on transit through their territories. This privacy threat is not just abstract, but is a realistic assessment in a communications environment powered by Internet technologies.

International adoption of the Internet has diminished the strategic predominance of the U.S. over the Internet's core infrastructure, while the percentage of international transit traffic carried via U.S. routes continues to decrease.[3] Many regions of the world are catching up, setting up new intra-regional hubs for Internet exchanges that carry international Internet traffic.[3] Europe has been mostly self-sustaining for some time now and is also attracting the majority of traffic from neighboring regions like Africa and the Middle East.

At the same time, the European protection of the confidentiality of communications as a revered fundamental value in the national constitutions has seen some severe setbacks. It remains to be seen whether electronic communications are any better protected against sweeping legal interception. The European Union (EU) Directive mandating data retention laws in the EU Member States largely compromised the expectation of privacy when communicating electronically. Providers of telephony, email services, and Internet access are required to log communications metadata—the information on who is calling whom when and for how long—and retain these records for up to two years. Such preemptive storing of such data for every user based within the territory of the EU is "just in case." As a consequence, the substance of this fundamental right is condensed to communications content, while the circumstances of individual communications via electronic means are subject to data retention. The EU Data Retention rule, however, only applies to metadata of Internet email and telephony that originates and/or terminates within the EU. Communications content and mere transit of international traffic through European Internet exchanges are not affected by this rule.

Neighboring countries are no less outraged about Sweden's approach to international surveillance, which affects their national communications sectors.

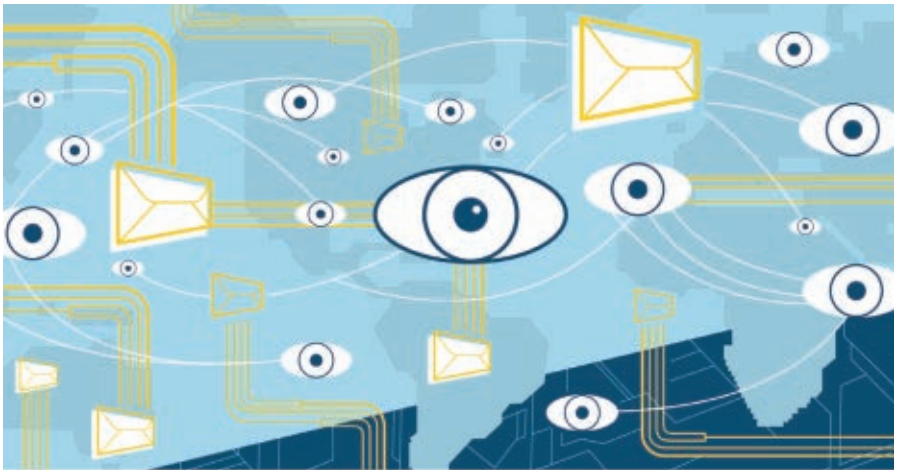
However, a number of European countries do maintain their own domestic surveillance programs that also target international communications, with Sweden recently catching up—and now overtaking—other nations. In June 2008, the Swedish parliament passed the New Signal Surveillance Act or FRA law, as it had been dubbed, which grants, in the name of national security, Sweden's National Defense Radio Establishment (Försvarets Radioanstalt—FRA) the power to access the complete Internet and telephone communications in and out of Sweden. The bill, which took effect at the beginning of this year, obliges all operators of Internet exchange points in Swedish territory to channel traffic though FRA's facilities. Sweden's move toward international communications surveillance went far beyond existing legal standards in other European countries. In his personal blog, Peter Fleischer, Google's Global Privacy Counsel, compared the Swedish government initiative to those of governments from China to Saudi Arabia and the U.S. eavesdropping program.[1]

The passage of the Swedish law involved some odd circumstances. The Swedish center-right alliance in power succeeded with a very narrow majority of 143 votes to 138 in favor of the law after the introduction of some last-minute changes to address oversight of FRA's surveillance activities. Even after this, the first version of the FRA law might have clashed with the privacy protection granted under the European Convention of Human Rights (ECHR). The law's language is vague and its provisions might exceed what is necessary in a democratic society. In order to address some of these obvious weaknesses, the parliament asked the government to propose further amendments in a number of areas by autumn 2008, well before the original law would have taken effect.[5] It seems quite unusual though that a law was passed along with a mandate to fix the flaws even before the law takes effect.

From a political perspective, the surveillance initiative has been a disaster for the Swedish government: its sweeping effect upset political allies, voters, businesses, and neighboring countries alike. Beginning with the legislative process, opposition to the law only grew once the law was adopted. By now awareness about the issue is extremely high and the government has lost significant credibility with the public. The Swedish daily newspaper *Expressen* offered a protest email form on its Web site. The newspaper reported six million protest email messages had been generated, a significant number in a nation with a population of nine million (of course, the responses may not all have been from different individuals or from Sweden).

Sweden's reputation as a leading location for international ICT services was considerably damaged. In a public statement, the CEOs of eight major Nordic telecom companies have warned the country that their companies will relocate their activities away from Sweden in order to protect the interests of their international customers and to abide by the legal requirements elsewhere.[4] The Swedish-Finnish telecom operator TeliaSonera has already moved email and Web servers from Sweden. Google and other companies are publicly contemplating withdrawing from Swedish territory as well, a negative trend that would be self-perpetuating.

Neighboring countries are no less outraged about Sweden's approach to international surveillance, which affects their national communications



sectors. Since Sweden serves as a local hub for transit traffic from and to Norway, Finland, and Russia, the wider cross-border impact is evident. For example, Russia would be exposed to the surveillance scheme so long as a significant share of its domestic Internet traffic is routed via Sweden. Apart from the diplomatic distortions these countries have a palpable incentive to bypass Sweden in the near future and connect to other available Internet exchange services or set up their own facilities.

The Swedish example illustrates how not to expand surveillance. It also shows that new surveillance legislation can have repercussions far beyond strict privacy concerns. The subsequent efforts Sweden put into amending the first FRA law were only trying to adjust the sweeping effects it created. The new draft law presented by the government at the end of September last year is significantly more tailored and contains additional safeguards; it introduces a special court designated to authorize signal surveillance requests from the government and the armed forces to FRA.[2]

Because of the possibility that many Big Brothers may be watching, as individuals we lose out when we communicate via the Internet.

European constitutional tradition protects everyone's communications in a country's territory, and thus the same principles and safeguards apply to the legal interception of domestic and international communications. It does not matter where the originator or the recipient of a particular communication are located in order to benefit from the protection guaranteed inside a country. Maintaining the same thresholds for all communications entering a jurisdiction would also help to overcome the expansion of surveillance to foreign individuals and businesses that typically do not have much influence in a legislative process otherwise. For example, it is likely the Swedish people are most concerned about their own rights to privacy in communications and might not be as interested in the rights of foreigners, whose transit communications would be intercepted. Still, their engagement promotes as well the case of foreigners that otherwise do not carry significant political weight.

International agreements that guarantee the right to communications privacy should play a more significant role to uphold the level of protection against unfettered surveillance powers. One example is Article 8 of the European Convention of Human Rights, which provides the right to respect for one's private life and correspondence. More importantly, the possibility to turn to the ECHR for an appraisal of domestic surveillance laws helped to clarify the requirements that legal interception rules have to meet in order to comply with the fundamental right. Its judgments concerning the strategic monitoring schemes, as opposed to monitoring of individual communications, in Germany and the U.K. show that the key to permissible legal interception lies in the precise description of the authority and procedures that would empower a proportionate level of surveillance.[a]

In my opinion, there is no other international mechanism that would compare to the supranational oversight of domestic surveillance laws of the ECHR. Unsurprisingly, many country's national interests point toward intelligence gathering, where internationally enforceable standards for legal interception would just be burdensome. The prospect to access international traffic that is passing through their territories' Internet infrastructure is perceived as an addition to conventional methods of interception.

Because of the possibility that many Big Brothers may be watching, as individuals we lose out when we communicate via the Internet. But the privacy of the individual is only one part of this complex issue; business is another. When international ICT companies choose a regional location for doing business, countries that maintain a proportionate and safeguarded policy regarding legal interception could find themselves with a competitive advantage over neighbors that do not have such policies. And that would ultimately be good for privacy, business, and the security of the nation choosing to adequately protect communications privacy. C

a European Court of Human Rights (ECHR), case of *Weber and Saravia v. Germany*, no. 54934/00, decision of June 29, 2006; case of *Liberty and Others v. the United Kingdom*, no. 58243/00, decision of July 1, 2008.

References

1. Fleischer, P. Sweden and government surveillance (May 31, 2007); http://peterfleischer.blogspot.com/2007/05/sweden-and-government-surveillance.html.
2. Government unites around new FRA law. *The Local* (Sept. 26, 2008); http://www.thelocal.se/14576/20080926/.
3. Markoff, J. Internet traffic begins to bypass the U.S. *The New York Times* (Aug. 30, 2008).
4. Surveillance sweep—A new surveillance law causes a rumpus in Sweden. *The Economist* (July 22, 2008); http://www.economist.com/agenda/displaystory.cfm?story_id=11778941.
5. Sveriges Riksdag. New Signal Surveillance Act (FöU15) (June 18, 2008); http://www.riksdagen.se/templates/R_PageExtended____16402.aspx.

Kristina Irion (irionk@ceu.hu) is an assistant professor in the Department of Public Policy at Central European University in Budapest, Hungary.

ILLUSTRATION BY CELIA JOHNSON

DOI:10.1145/1461928.1461939

Peter G. Neumann

Inside Risks

U.S. Election After-Math

Recounting problems still associated with election integrity, transparency, and accountability.

From the perspective of computer-based election integrity, it was fortunate that the U.S. presidential electoral vote plurality was definitive. However, numerous problems remain to be rectified, including some experienced in close state and local races.

Elections represent a complicated system with end-to-end requirements for security, integrity, and privacy, but with many weak links throughout. For example, a nationwide CNN survey for the 2008 election tracked problems in registration (26%), election systems (15%), and polling place accessibility and delays (14%). Several specific problems are illustrative.

Registration. In Cleveland and Columbus, OH, many voters who had previously voted at their current addresses were missing from the polling lists; some were on the Ohio statewide database, but not on the local poll registry, and some of those had even received postcard notification of their registration and voting precinct. At least 35,000 voters had to submit provisional ballots. (Sec. of State Jennifer Brunner rejected the use of a list of 200,000 voters whose names did not identically match federal records.) Several other states attempted to disenfranchise voters based on nonmatches with databases whose accuracy is known to be seriously flawed.

Machines. In Kenton County, KY, a judge ordered 108 Hart voting machines shut down because of persistent problems with straight-party votes not being properly recorded for the national races. As in 2002 and 2004, voters reported touch-screen votes flipped from the touched candidate's name to another. Calibration and touching are apparently quite sensitive.

In Maryland, Andrew Harris, a longtime advocate in the state senate for avoiding paperless touch-screen and other direct-recording voting machines (DREs) ran for the Representative position in Congressional District 1. He trailed by a few votes over the 0.5% margin that would have necessitated a mandatory recount. Of course, recounts in paperless DREs are relatively meaningless if the results are already incorrect.

In California, each precinct typically had only one DRE, for blind and other disabled voters who preferred not to vote on paper. In Santa Clara County, 57 of those DREs were reported to be inoperable. (Secretary of State Debra Bowen was a pioneer in commissioning a 2007 study on the risks inherent in existing computer systems.[1])

There were also various reports in other states of paperless DREs that were inoperable including some in which more than half of the machines could not be initialized. In Maryland and Virginia, there were reports of voters having to wait up to five hours.

Every Vote Should Count. Close Senate races in Minnesota and Alaska required ongoing auditing and recounting, particularly as more uncounted votes were discovered. Numerous potential undervotes also required manual reconsideration for voter intent. Anomalies are evidently commonplace, but must be *resolvable*—and, in close elections must be *resolved*.

Deceptive Practices. The George Mason University email system was hacked, resulting in the sending of misleading messages regarding student voting. Numerous misleading phone calls, Web sites, and email messages have been reported, including those that suggested Democrats were instructed to vote on Wednesday instead of Tuesday to minimize poll congestion.[2]

Conclusion

The needs for transparency, oversight, and meaningful audit trails in the voting process are still paramount. Problems are very diverse. Despite efforts to add voter-verified paper trails to paperless direct-recording voting machines, some states still used unauditable systems for which meaningful recounts are essentially impossible. The electronic systems are evidently also difficult to manage and operate.

Systematic disenfranchisement continues. Although there seems to have been very little voter fraud, achieving accuracy and fairness in the registration process is essential. To vary an old adage, It's not just the votes that count, it's the votes that don't count.

An extensive amount of work remains to be done to provide greater integrity throughout the election process. C

References

1. Bishop, M. and Wagner, D. Risks of e-voting. *Commun. ACM 50*, 11 (Nov. 2007); http://www.sos.ca.gov/elections/elections_vsr.htm.
2. E-Deceptive Campaign Practices, EPIC and the Century Found. (Oct. 20, 2008), http://votingintegrity.org/pdf/edeceptive_report.pdf; and Deceptive Practices 2.0: Legal and Policy Responses Common Cause, The Lawyers Committee for Civil Rights under Law, and the Century Found. (Oct. 20, 2008), http://www.tcf.org/list.asp?type=TP&topic=6.

Peter G. Neumann moderates the ACM Risks Forum (www.risks.org).

The Best Place to Find the Perfect Job... Is Just a Click Away!

No need to get lost on commercial job boards.
The ACM Career & Job Center is tailored specifically for you.

JOBSEEKERS

- Manage your job search
- Access hundreds of corporate job postings
- Post an anonymous resume
- Advanced Job Alert system

EMPLOYERS

- Quickly post job openings
- Manage your online recruiting efforts
- Advanced resume searching capabilities
- Reach targeted & qualified candidates

NEVER LET A JOB OPPORTUNITY PASS YOU BY!

START YOUR JOB SEARCH TODAY!

http://www.acm.org/careercenter

Association for Computing Machinery

Advancing Computing as a Science & Profession

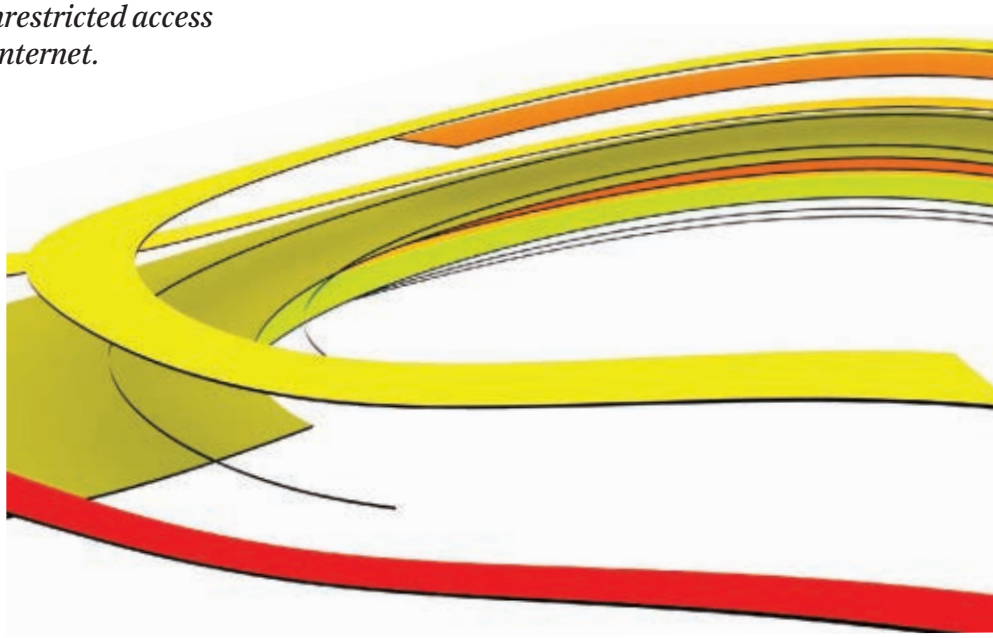
POWERED BY JOBTARGET

DOI:10.1145/1461928.1461942

Barbara van Schewick David Farber

Point/Counterpoint
Network Neutrality Nuances

A discussion of divergent paths to unrestricted access of content and applications via the Internet.



Point: Barbara van Schewick

IMAGINE SWITCHING ON your computer to try an exciting new Internet application you heard about. It does not work. You call customer support, but they cannot help you. If you are like most users, you give up. Maybe the application was not so great after all. If you are technically sophisticated, you may run some tests, only to find out your ISP is blocking the application. Welcome to the future without network neutrality rules.

Most of us take the ability to use the applications and content of our choice for granted. To us, "Internet access" means access to everything the Internet has to offer, not access to a selection of Internet applications and content approved by our ISP. This assumption was justified in the past, when the Internet was application-blind, making it impossible for ISPs such as AT&T, EarthLink, or Comcast to interfere with the applications and content running over their network.[a] Today's world is different: ISPs have access to sophisticated technology that enables them to block applications or content they do not like, or degrade their performance by slowing the delivery of the corresponding data packets.

Whether and how the law should react to this changed situation is the subject of the network neutrality debate. Network neutrality proponents argue that ISPs have incentives to use this new technology, and that the existing laws in many countries do not sufficiently constrain the ISPs' ability to do so. Proponents contend that users, not network providers should continue to decide how they want to use the Internet if the Internet is to realize its full potential and that the law should forbid ISPs to block applications and content or to discriminate against them. While the debate does not end here (in particular, whether a nondiscrimination rule should ban Quality of Service or restrict ISPs' ability to charge unaffiliated application or content providers for the right to offer their products to the ISPs' customers is controversial even among network neutrality proponents), a rule against blocking and discrimination is at the core of all network neutrality proposals.[b]

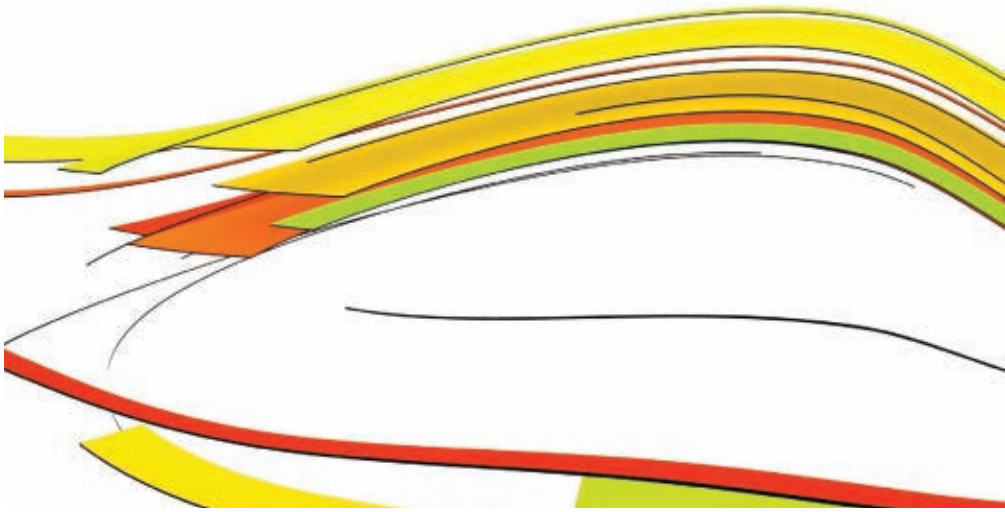
But does a network provider really have an incentive to discriminate against applications? Research shows that while a network provider does not generally have an incentive to exclude applications or content,[c] there are cases

a The application-blindness of the Internet was a consequence of its design, which was based on the broad version of the end-to-end arguments.

b In some proposals, such a rule takes the form of users' rights to use the (lawful) applications and (legal) content of their choice. There usually is an exception that allows network providers to block malicious applications and content, such as those involved in denial-of-service attacks.

c More applications and content make the Internet more attractive, so network providers generally have an incentive to foster, not exclude additional applications.

ILLUSTRATION BY LEANDER HERZOG



in which it does have an incentive to do so—to increase its own profit, to manage bandwidth on its network, or to exclude unwanted content. Consistent with these theoretical predictions—and in spite of heightened public attention from the ongoing controversy about the need for network neutrality regulation, examples of discriminatory conduct have started to appear in practice. As Lawrence Lessig has put it, "if 'network neutrality' was 'a solution in search of a problem' in 2002, and 2006, the network owners have been very kind to network neutrality advocates by now providing plenty of examples of the problem to which network neutrality rules would be a solution."[2]

For example, network providers may want to exclude applications that threaten their traditional sources of income. In 2005, Madison River, a rural phone company in North Carolina, blocked the Internet telephony application Vonage, which threatened its revenue from traditional phone services. In 2007, Comcast, the second-largest provider of Internet services in the U.S., shut down peer-to-peer file sharing connections, degrading the performance of applications such as Vuze that legally deliver television content to end users based on a peer-to-peer protocol and threaten Comcast's traditional cable-based content delivery services. ISPs such as AT&T or Verizon, which offer co-branded services with Yahoo may have an incentive to increase their joint advertising revenue with Yahoo by slowing down Web sites or portals that compete with Yahoo. Network providers need not necessarily be able to monopolize the market for a specific application to make discrimination profitable; the increased revenue from selling more copies at the market price may be incentive enough.

Does a network provider really have an incentive to discriminate against applications?

Network providers may also be motivated to interfere with applications to manage bandwidth on their network. Because of the prevailing flat-rate pricing structure, network providers have an incentive to block or degrade applications that consume more bandwidth or consume it in unexpected ways. After all, if the use of the network increases, the network provider's costs increase as well, but due to flat-rate pricing, its revenue stays the same. For the network provider, blocking or degrading selected applications is a quick fix that requires less investment than upgrading the network or devising a nondiscriminatory solution. Comcast's blocking of BitTorrent and other peer-to-peer file-sharing applications is an example of this type of behavior.

Finally, network providers may have an incentive to block unwanted content that threatens the company's interests or does not comply with the network provider's chosen content policy. In 2005, Telus, Canada's second largest ISP, blocked access to a Web site that was run by a member of the Telecommunications Workers Union. At the time, Telus and the union were engaged in a contentious labor dispute, and the Web site allowed union members to discuss strategies during the strike. In 2007, Verizon Wireless rejected a request by NARAL Pro-Choice America, an abortion rights group, to let them send text messages over Verizon Wireless' network using a five-digit short code. In the same year, AT&T deleted words from a Webcast of a Pearl Jam concert in which the singer criticized George W. Bush. Both providers argued that the rejected or deleted content violated their content policies.[d] While the latter two examples are not direct examples of ISPs restricting content on their networks (Verizon Wireless restricted a service on its wireless mobile network, not the wireless Internet, while AT&T acted in its role as a content provider, not as ISP), it is easy to imagine virtually identical incidents in which an ISP enacts a content policy and restricts content on its network accordingly.

If ISPs have an incentive to block selected applications or content or discriminate against them, why should we care? Preventing discrimination is necessary if the Internet is to realize its full economic, social, and political potential. Discrimination restricts users' ability to choose the application and content they want to use. This ability to choose is fundamental if the Internet is to create maximum value, for us as individuals and for society. The Internet is a general-purpose technology. It does not create value through its existence

d They later changed their view after the incidents had been widely reported.

Network providers may be motivated to interfere with applications to manage bandwidth on their network.

ILLUSTRATION BY LEANDER HERZOG

alone. It creates value by enabling us to do things we could not do otherwise, or to do things more efficiently. Applications are the tools that enable us to realize this value. Through the applications and content it offers, the Internet has enabled us to become more productive in our professional and private lives, to interact with relatives, friends, and complete strangers, to get to know them, communicate, or work with them, focusing on anything we like, to educate ourselves using a wide variety of sources, and to participate in social, cultural, and democratic discourse. In the process, it has spurred economic growth, improved democratic discourse, and created a decentralized environment for social and cultural interaction in which anyone can participate.

ISPs ability to discriminate changes this. In a world without network neutrality rules, ISPs determine which applications and content can become successful, distorting competition in the markets for applications and content. As we have seen, who network providers decide to support and who they decide to exclude may be motivated by a number of factors that are not necessarily aligned with users' preferences, leading to applications that users would not have chosen and forcing users to engage in an Internet usage that does not create the value it could. If I am working on an open source project that uses BitTorrent to distribute its source code, and the network provider chooses to single out BitTorrent to manage bandwidth on its network, I am unable to use the application that best meets my needs and use the Internet in the way that is most valuable to me. If I am interested in content that my network provider has chosen to restrict, my ability to educate myself, contribute to a discussion on this subject, and make informed decisions is impeded. Instead, ISPs gain the power to shape public discourse based on their own interests and idiosyncratic content policies. In addition, the risk of being cut off from access to users at any time and at the sole discretion of the network provider reduces independent innovators' incentive to innovate and their ability to secure funding. Throughout the history of the Internet, successful applications such as email, the Web, search engines, or social networks have been developed by independents, not network providers. By threatening the supply of all those exciting new applications that have not even been thought of yet, discrimination by network providers reduces the Internet's ability to create even more value in the future.

Network providers may have an incentive to block unwanted content that threatens the company's interests or does not comply with the network provider's chosen content policy.

But do we really need regulation? That competition will solve any problems, should they exist, is a common argument against network neutrality. If AT&T blocks an application that its customers want to use, the arguments goes, customers will switch to another provider that lets them use the application.

This argument comes in many flavors: Some, like many European regulators, use it to argue that the problems that network neutrality is designed to address are caused by the concentrated market structure in the U.S., but are not relevant to European countries that, due to open access regulation, have more competition among ISPs than the U.S.[e] Others, particularly in the U.S., argue that governments should focus on increasing competition among ISPs instead of enacting network neutrality rules.

These arguments neglect a number of factors that make competition less effective in disciplining discriminatory conduct than one might expect. First, if all network providers block the same application, there is no provider to switch to. For example, in many countries all mobile network providers block Internet telephony applications to protect their revenue from mobile voice services, leaving customers who would like to use Internet telephony over their wireless Internet connection with no network provider to turn to.

Second, customers do not have an incentive to switch if they do not realize the network provider interferes with their preferred application. If network providers secretly slow down packets or use methods that are difficult to detect, their customers may attribute an application's or Web site's bad performance to bad design, and happily switch to the network provider's supposedly superior offering.

Third, finding another ISP and making the switch requires significant time, effort, and money, reducing customers' willingness to switch. All this suggests that while increasing competition is good for other reasons, it is not a substitute for a robust network neutrality regime.

Finally, some argue that allowing network providers to discriminate against applications and content is necessary to foster broadband deployment. This argument concedes that network providers have an incentive to discriminate to increase their profits. By removing the ability to discriminate, network neutrality rules reduce network providers' profits.

Fewer profits may mean less money to deploy broadband networks. I am not convinced that network neutrality rules would reduce ne twork providers' profit enough to push deployment incentives beyond the socially efficient level,[f] or that network providers would really use the additional profits to deploy more networks instead of using the money to please their shareholders.

e In the U.S., most residents have a choice between at most two providers, the local telephony company and the local cable modem provider (residents in 34% of ZIP codes in the U.S. have only one or zero cable modem or ADSL provider who serves at least one subscriber living within the ZIP code). These providers are not required to (and generally do not) let independent ISPs offer Internet services over their infrastructure. By contrast, Europeans often have the choice between cable and DSL services, and can choose among a number of ISPs offering their services over the DSL network.

f After all, network providers would still be able to offer their own applications or content, but they would not be able to give them an advantage over competing products.

Still, there is a potential trade-off here that legislators need to resolve. Allowing discrimination reduces user choice and application-level innovation. It distorts competition in applications and content, harms economic growth and constrains democratic discourse. Sacrificing the vital innovative and competitive forces that drive the Internet's value to get more broadband networks seems too high a price; as Tim Wu has put it, it is like selling the painting to get a better frame.[6] While it is impossible to protect application-level innovation and user choice once network providers are allowed to discriminate, there are ways to solve the problem of broadband deployment that do not similarly harm application-level innovation and user choice (for example, if insufficient profits really are the problem, subsidizing network deployment may be one).

Changes in technology have given network providers an unprecedented ability to control applications and content on their network. In the absence of network neutrality rules, our ability to use the lawful Internet applications and content of our choice is not guaranteed. The Internet's value for users and society is at stake. Network neutrality rules will help us protect it. C

References

1. Frischmann, B.M. and van Schewick, B. Network neutrality and the economics of an information superhighway: A reply to Professor Yoo. *Jurimetrics Journal 47* (Summer 2007), 383–428.
2. Lessig, L. *Testimony before the United States Senate, Committee on Commerce, Science, and Transportation, at its Hearing on: The Future of the Internet.* 2nd Session 110th U.S. Congress, 2008.
3. van Schewick, B. Towards an economic framework for network neutrality regulation. *Journal on Telecommunications and High Technology Law 5*, 2 (2007), 329–391.
4. van Schewick, B. *Written Testimony before the Federal Communications Commission at its Second En Banc Hearing on Broadband Management Practices.* 2008; http://www.fcc.gov/broadband_network_management/hearing-ca041708.html.
5. van Schewick, B. *Architecture and Innovation: The Role of the End-to-End Arguments in the Original Internet.* MIT Press, Cambridge, MA, Forthcoming 2009.
6. Wu, T. Why you should care about network neutrality. *Slate Magazine* (May 1, 2006).

Barbara van Schewick (schewick@stanford.edu) is the co-director of Stanford Law School's Center for Internet and Society, an assistant professor of Law at Stanford Law School, and an assistant professor Electrical Engineering (by courtesy) at Stanford's Department of Electrical Engineering in Stanford, CA.

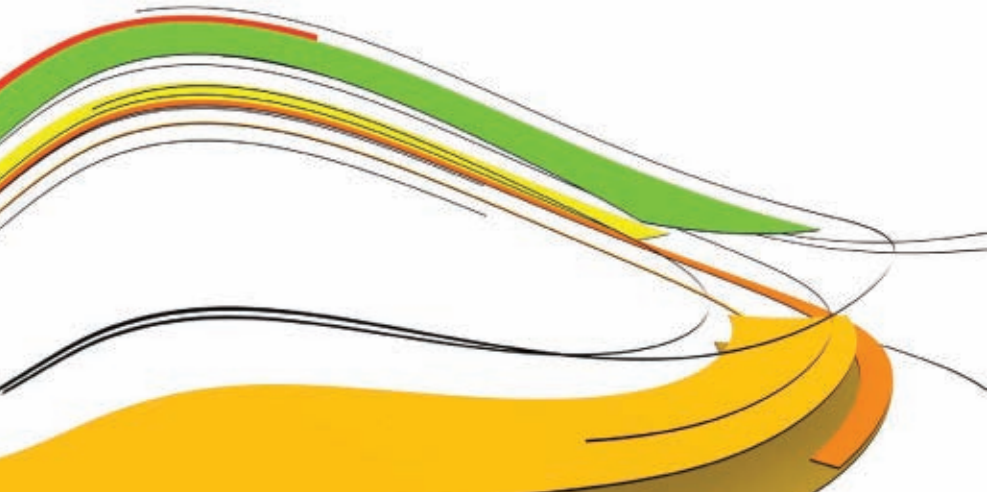
This column builds and partly draws on my academic work on network neutrality; see [1,3–5]. Thanks to Joseph Grundfest, Lawrence Lessig, Larry Kramer, and Mark Lemley for comments on an earlier version of this material.

Counterpoint: David Farber

LET'S SAY THAT I am completely in favor of network neutrality. But what would such a strong position actually mean? The definition of "network neutrality reshapes itself like our lungs. It expands, drawing in causes ranging from freedom of speech to open access. Then it contracts, exhaling a lot of hot air, and starts all over again. I would like very much to sharpen the focus to those essential issues that will form the basis of a future expansion of broadband Internet services as well as the widespread deployment of such capabilities.

There is one constant about the Internet: it has continued to evolve and change, often in ways none of us—even those of us directly involved in its development—would have predicted. This simultaneously makes the Internet so valuable and so vulnerable. Its growth and expansion into all corners of society have made it a major part of global life—but this expansion and the value of the Internet also frequently leads to efforts by some to try and predict and control its direction.

We've had cycles in the past where the Internet faced challenges due to rapid growth or the development of new forms of malware or online attacks. For example, in the dial-up era of the 1980s, the growth of list servers and FTP file downloads caused great concern about congestion. In the 1990s, there was a similar fear that the Internet would "crash" due to the rise of the Web. At various times, fears of new forms of viruses and botnets have arisen as well. In every case, the cooperative efforts of network providers, applications developers, and volunteers with a great amount of expertise have helped us make changes in protocols or add capacity that have helped us get through.



The evolution of the Internet is thus driven equally by competition and cooperation, and by and large we continue to find ways, as messy and informal as they often are, to address problems as they arise.

I am concerned that we may succumb to fears about possible dangers to the Internet's future and react with proposals to legislate or regulate its operations. Many of these ideas are designed around presumptions as to how the Internet will evolve. We have seen the Internet become a truly mass-market phenomenon on a global basis. Broadband networks have been and are being deployed that are moving us toward higher levels of speed and capability. Some now suggest there is the potential for abuses that might harm consumers as these networks grow. They argue that the companies deploying wire-line broadband networks might use their position as network owners to favor the applications and services they provide and/or harm competing servic-

ILLUSTRATION BY LEANDER HERZOG

es and applications offered by search engine companies, online content providers, and Internet portals.

The fundamental fact of the U.S. broadband capabilities is that we have a landline duopoly consisting of the cable companies and the "telephone" facilities—cable data and DSL. Each member of the duopoly is expanding their physical plant—both are evolving to fiber to either the home or close to the home and both are targeting a complete set of subscriber services from phone to video to data. In the long run there is no obvious winner in terms of technology and maybe even in terms of services offered. In the short run the competition between these duopolies encourages the modernization of their physical plants and the enhancement of their services.

All this sounds great for all. So where is the problem? The issue of Net neutrality first arose in the public's view in a remark by one of the carriers—SBC (now the "new ATT") —that services that use the facilities provided to reach their customers should pay a fee to the carriers for the use of their facilities. This trial balloon raised a firestorm with Internet-based companies such as Google who essentially argued that their customers were paying their Internet access fees in order to gain access to Google's services and that Google was paying the terminating carrier directly for high-speed access. They and others demanded that the U.S. Congress pass laws requiring essentially open access to the data networks on an equal-service-to-all basis. They asked the FCC to exercise their regulatory powers to require this in the absence of any specific Congressional actions.

There is one constant about the Internet: it has continued to evolve and change, often in ways none of us would have predicted.

I am concerned that we may succumb to fears about possible dangers to the Internet's future and react with proposals to legislate or regulate its operations.

In both cases the cure might be worse that the disease. The ability of the U.S. Congress to pass effective legislation in the telecommunications area is open to question. The Telecommunications Act of 1996 was an attempt to provide unbundled access to the last-mile wire loop in order to allow and encourage the rise of alternative data carriers. Experience has shown that this portion of the bill was worse than ineffective. Those companies who believed the law would be effective lost a lot of money as the incumbents resisted and were forced to the courts. The incumbents argued that the existence of these unbundling obligations undermined the incentives for the incumbents or anyone else to make the heavy investment required for building new next-generation broadband networks. Counting on the FCC has yielded mixed results. Like Congress, its decisions are subject to influence by political considerations and special interest goups as administrations come and go. The FCC has a minimum amount of technical knowledge about the Internet and thus even when it acts, often misses the mark and can end up in lengthy court actions that an innovative new company cannot survive.

Recently there have been a number of activities that have been attacked, sometimes validly, as being against the public good and against the FCC guidelines. The result of this was an FCC action telling Comcast to stop a particular form of network action being used by them for network management, There were public hearings prior to the action during which time there was a mix of technical input and public discussion, all informal—that is, not sworn testimony as required in formal hearings. I will not argue the validity of the criticism, except to say technique used seemed not to actually work but I would comment that the procedure, such as was used, most likely is not an effective way of gathering critical technical information and is all too easily turned into a political show. It is also possibly illegal but the courts will eventually determine that when someone sues.

One of the major dangers that face the future of Internet business is whether those who control our access to the Net will implement procedures under the guise of managing the Net that will discourage competition to those services they offer—such as video over the Internet competing with the cable delivery of the cable operator.

In an Op-Ed article in the *Washington Post* in January 2007, Michael Katz, Christopher Yoo, Gerald Faulhaber, and I argued: "Public policy should intervene where anticompetitive actions can reliably be identified and the cure will not be worse than the disease. Policymakers must tread carefully, however, because it can be difficult, if not impossible, to determine in advance whether a particular practice promotes or harms competition. Current antitrust law generally solves this problem by taking a case-by-case approach under which private parties or public agencies can challenge business practices and the courts require proof of harm to competition before declaring a practice illegal. This is a sound approach that has served our economy well."[a]

Today, innovation and enhancements can occur at all levels of the Internet. Network providers, applications providers, portals, search engines, and content providers can all innovate in various ways and make needed improvements that can benefit the Internet's evolution. We should encourage this innovation, while preserving the other core strengths of the Internet: its cooperative spirit and openness to en-

a Farber, D., Katz, M. Hold off on Net neutrality. *Washington Post* (Jan. 19, 2007), A19; http://www.washingtonpost.com/wp-dyn/content/article/2007/01/18/AR2007011801508.html.

try by new players with new ideas and innovations.

This requires, I believe, a new commitment to transparency, openness, and sharing of information as much as possible. Network capacity should be managed in a way that brings users the benefits of differentiated services but at the same time network providers must be very transparent about:

- What consumers can expect with regard to how their connection will work and what services it normally should be able to run; and
- Their traffic management practices and how those practices are likely to affect consumers' connections and the applications they are running.

I also think network providers should work together with those of us who informally keep involved in the Internet's workings to voluntarily develop better information about the Internet's overall health including capacity constraints and bottlenecks, the impact of a variety of applications on network capacity, and congestion problems. I think we can do this without violating proprietary information restrictions.

I believe applications providers should also be transparent about how their offerings affect customers and their network connections. They should ensure customers know how the use of their applications might affect the speeds they have and the speeds of the connections of those in their neighborhood.

Finally, all participants in the broadband value chain—from the content portals and search engines to the applications providers to the network providers—should also embrace key principles designed to ensure consumers have control over and full use of their broadband connections to:

- Access any content on the Internet;
- Run any application they choose; and
- Attach any devices to their broadband connection that do not harm the network.

Government agencies should continue to actively monitor what is going on with the Internet. If allegations emerge regarding actions that are alleged to be harmful and anticompetitive, companies and consumers should be able to petition to government and have the incident or practice investigated. Most importantly, all of us who care about the Internet and how it works—from those in the media, to academics, to bloggers, to industry players—must remain vigilant and ready to expose, discuss, and publicly upbraid what we feel are examples of "bad actors."

No one would have predicted even five or six years ago many of the advances and services we see today on the Internet. Few even knew what a search engine was, for example, or had used Instant Messaging or viewed a video online. All of this happened in large part because the Internet has not been subject to the slow, cumbersome regulatory processes of government. Inserting government into questions around network management and the evolution of the Internet's underlying technologies and applications will simply erode the cooperative spirit that has driven its evolution, substituting instead filings, charges, and countercharges. I shudder to see this happen. C

David Farber (dave@farber.net) is Distinguished Career Professor of Computer Science and Public Policy at the School of Computer Science, Heinz School, and Department of Engineering and Public Policy at Carnegie Mellon University, Pittsburgh, PA.

Rebuttal: Barbara van Schewick

David Farber and I both want to preserve users' ability to use the applications and content of their choice and the Internet's openness to innovation. We differ in how to get there.

Farber appeals to network providers "to embrace key principles" designed to protect users' ability to use the Internet as they want and to disclose any limitations, including those resulting from the network providers' traffic management practices. To a limited degree, this appeal would be backed up by the force of law: The regulatory regime he envisages would prohibit only "anticompetitive" practices (in the sense the term is used in antitrust law). Consumers and companies could petition the government to investigate (and presumably ban) specific allegedly anticompetitive conduct after the fact.

I don't think these measures will be sufficient to protect users' ability to use the Internet as they want and enable the Internet to realize its economic, social, and democratic potential. Appeals to shared values may have worked in the past, when most networks were operated by academic institutions. Today, networks are run by companies. Their goal is to create value for their shareholders, not to do what's in the public interest. To the extent commercial network providers do have an incentive to block or slow down applications or content, appeals won't be able to stop them.

I agree with Farber that network providers should disclose any limitations on users' ability to use the Internet. As disclosure may expose competitive weaknesses compared to rival providers, network providers may need regulatory pressure to engage in it. While disclosure will support competition by helping consumers make more informed choices, it will not be sufficient to prevent discrimination: Disclosure removes only one of the obstacles (incomplete information) highlighted in my statement that prevent competition in the broadband services market from being effective in disciplining providers.

If appeals and disclosure alone are not sufficient to restrict network providers' incentives to block or slow down applications, the scope of the regulatory regime determines whether network providers can act on their incentives. In this respect, Farber's regime would only capture a subset of the cases in which network providers have an incentive to exclude applications.

First, discrimination designed to exclude unwanted content or manage bandwidth on a network may often lack an anticompetitive motivation. In the examples of content-based discrimination described in my statement, none of the content providers whose content was blocked was com-

peting with the network provider. Similarly, a network provider may have an incentive to exclude or slow down selected bandwidth-intensive applications to manage bandwidth on its network, even if the network provider does not offer a competing application itself. At the same time, the resulting harm—users' inability to participate in social, cultural or democratic discourse related to the blocked content, their inability to use the Internet in the way that is most valuable to them, or application developers' difficulty to obtain funding for an application—is caused by the blocking as such, not by the motivations that were driving it.

Second, even blocking that hurts a competitor is not necessarily prohibited by Farber's proposed regime. In U.S. antitrust law, which Farber's regime is designed to mirror, the term "anticompetitive" has a much narrower meaning than nonlawyers would expect.[a] For example, if a network provider excludes an application such as BitTorrent from access to the provider's Internet service customers, this only constitutes "anticompetitive" conduct under U.S. antitrust law if it creates a "dangerous probability of success" that the network provider will monopolize the nationwide market for BitTorrent-like applications. That the network provider's customers cannot use BitTorrent, or that BitTorrent is excluded from a part of the nationwide market, is irrelevant in the context of antitrust law, but not in the context of the network neutrality debate that focuses on different types of harm.

Prohibiting only "anticompetitive" conduct will not prevent all relevant discrimination. To protect user choice and the Internet's ability to realize its potential, we need rules that prohibit blocking and discrimination of applications and content regardless of the underlying motivation and independent of the network provider's market share. C

a In particular, as Farber explains in his excerpt from an Op-Ed with economists Michael Katz and Gerald Faulhaber and legal scholar Chris Yoo, it requires a proof of "harm to competition," not just to a competitor.

Rebuttal: David Farber

I THINK ALL of us would agree with a basic premise underlying Barbara van Schewick's comments—that the ability of consumers and business to access the content and applications of their choice, without interference, is vital to the continued evolution of the Internet and the innovation, social progress, and economic advancements it promises.

But van Schewick paints with too broad a brush. She asserts "a rule against blocking and discrimination is at the core of all Net neutrality proposals." If only that were the case—ask six different Net neutrality proponents and you'll get six different definitions.

Van Schewick suggests there are incentives to discriminate or interfere with traffic. Providers are not free to operate in the market as they wish with no government role or policies. Since 2003, the FCC has had a set of principles in place it uses to oversee the broadband market. It uses them to assess developments in the market and where necessary, to engage in enforcement activities. The FCC used these principles in the Comcast case. Whether you agree or disagree with the FCC's findings and conclusions, the reality is the principles have acted as a framework not only for the FCC, but also for industry, consumers, and advocates. While these principles are not regulations, they are powerful in the sense that they set expectations and they have the merit of being flexible and adaptive and in that sense much more in sync with the Internet's core underpinnings.

In addition to the FCC's principles, the consumers are protected by many "eyes" watching the Internet and how it is working, including the FCC's enforcement role, the consumer protection and antitrust oversight of the FTC, and competition among providers. Because the Internet's protocols are open and because there are literally thousands of networks, millions of Web sites, and more than a billions of users online, there are lots of folks watching what is going on at all levels of the Internet. Companies doing dumb things won't get away with it too long, despite her comments to contrary. Consumers do have to be aware of what is going on in order to help ensure that companies are not taking actions that may harm them. That is why while regulations and new laws are potentially harmful in my view, there are some actions that should be taken.

There does need to be far more transparency on the part of companies regarding how their broadband services work, what types of network management activities they engage in and how those activities might affect consumers. Content and applications providers too need to be far more transparent about how their applications affect the Internet and consumers themselves.

Moreover, more transparency at the higher levels of the Internet—particularly the backbone—would help academics and Internet experts to better understand how well the Internet is working, what applications may be causing the most problems, and where network congestion problems are occurring or likely to occur.

Finally, Internet experts and academics need to avoid policy polemics and engage in more rigorous analysis, assessments, and fact-based reporting on issues like congestion. While there is not as much data out there as we would like, we can do more to develop rigorously balanced analysis that can help policymakers understand emerging issues around broadband networks and applications. C

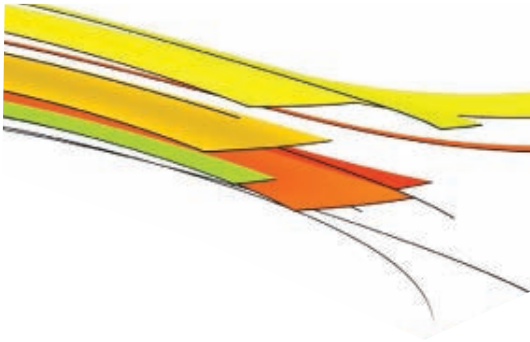


ILLUSTRATION BY LEANDER HERZOG

DOI:10.1145/1461928.1461943

While still primarily a research project, transactional memory shows promise for making parallel programming easier.

BY ULRICH DREPPER

Parallel Programming with Transactional Memory

With the speed of individual cores no longer increasing at the rate we came to love over the past decades, programmers have to look for other ways to increase the speed of our ever-more-complicated applications. The functionality provided by the CPU manufacturers is an increased number of execution units, or CPU cores.

To use these extra cores, programs must be parallelized. Multiple paths of execution have to work together to complete the tasks the program has to perform, and as much of that work as possible has to happen concurrently. Only then is it possible to speed up the program (that is, reduce the total runtime). Amdahl's Law expresses this as:

$$\frac{1}{(1-P) + P/S}$$

Here P is the fraction of the program that can be parallelized, and S is the number of execution units.

Synchronization Problems

This is the theory. Making it a reality is another issue. Simply writing a normal program by itself is a problem, as can be seen in the relentless stream of bug fixes available for programs. Trying to split a program into multiple pieces that can be executed in parallel adds a whole dimension of additional problems:

- Unless the program consists of multiple independent pieces from the onset and should have been written as separate programs in the first place, the individual pieces have to collaborate. This usually takes on the form of sharing data in memory or on secondary storage.
- Write access to shared data cannot happen in an uncontrolled fashion. Allowing a program to see an inconsistent, and hence unexpected, state must be avoided at all times. This is a problem if the state is represented by the content of multiple memory locations. Processors are not able to modify an arbitrary number (in most cases not even two) of independent memory locations atomically.
- To deal with multiple memory locations, "traditional" parallel programming has had to resort to synchronization. With the help of mutex (mutual exclusion) directives, a program can ensure that it is alone in executing an operation protected by the mutex object. If all read or write accesses to the protected state are performed while holding the mutex lock, it is guaranteed that the program will never see an inconsistent state. Today's programming environments (for example, POSIX) allow for an arbitrary number of mutexes to coexist, and there are special types of mutexes that allow for multiple readers to gain access concurrently. The latter is allowed since read accesses do not

change the state. These mechanisms allow for reasonable scalability if used correctly.

▸ Locking mutexes opens a whole new can of worms, though. Using a single program-wide mutex would in most cases dramatically hurt program performance by decreasing the portion of the program that can run in parallel (P in the formula). Using more mutexes increases not only P but also the overhead associated with locking and unlocking the mutexes. This is especially problematic if, as it should be, the critical regions are only lightly contended. Dealing with multiple mutexes also means there is the potential for deadlocks. Deadlocks happen if overlapping mutexes are locked by multiple threads in a different order. This is a mistake that happens only too easily. Often the use of mutexes is hidden in library functions and not immediately visible, complicating the whole issue.

The Programmer's Dilemma

The programmer is caught between two problems:

▸ Increasing the part of the program that can be executed in parallel (P).

▸ Increasing the complexity of the program code and therefore the potential for problems.

An incorrectly functioning program can run as fast as you can make it run, but it will still be useless. Therefore, the parallelization must go only so far as not to introduce problems of the second kind. How much parallelism this is depends on the experience and knowledge of the programmer. Over the years many projects have been developed that try to automatically catch problems related to locking. None is succeeding in solving the problem for programs of sizes as they appear in the real world. Static analysis is costly and complex. Dynamic analysis has to depend on heuristics and on the quality of the test cases.

For complex projects it is not possible to convert the whole project at once to allow for more parallelism. Instead, programmers iteratively add ever more fine-grained locking. This can be a long process, and if the testing of the intermediate steps isn't thorough enough, problems that are not caused by the most recently added set of changes might pop up. Also, as experience has shown, it is sometimes very difficult to get rid of the big locks. For an example, look at the BKL (big kernel lock) discussions on the Linux kernel mailing list. The BKL was introduced when Linux first gained SMP (symmetric multiprocessing) support in the mid-1990s, and we still haven't gotten rid of it in 2008.

People have come to the conclusion that locking is the wrong approach to solving the consistency issue. This is especially true for programmers who are not intimately familiar with all the problems of parallel programming (which means almost everybody).

Looking Elsewhere

The problem of consistency is nothing new in the world of computers. In fact, it has been central to the entire solution in one particular area: databases. A database, consisting of many tables with associated indexes, has to be updated atomically for the reason already stated: consistency of the data. It must not happen that one part of the update is performed while the rest is not. It also must not happen that two updates are interleaved so that in the end only parts of each modification are visible.

The solution in the database world is transactions. Database programmers explicitly declare which database operations belong to a transaction. The operations performed in the transaction can be done in an arbitrary order and do not actually take effect until the transaction is committed. If there are conflicts in the transaction (that is, there are concurrently other operations modifying the same data sets), the transaction is rolled back and has to be restarted.

The concept of the transaction is something that falls out of most programming tasks quite naturally. If all changes that are made as part of a transaction are made available atomically all at once, the order in which the changes are added to the transaction does not matter. The lack of a requirement to perform the operations in a particular order helps tremendously. All that is needed is to remember to modify the data sets always as part of a transaction and not in a quick-and-dirty, direct way.

ILLUSTRATION BY OLI LARUELLE

Transactional Memory

The concept of transactions can be transferred to memory operations performed in programs as well. One could of course regard the in-memory data a program keeps as tables corresponding to those in databases, and then just implement the same functionality. This is rather limiting, though, since it forces programmers to dramatically alter the way they are writing programs, and systems programming cannot live with such restrictions.

Fortunately, this is not needed. The concept of transactional memory (TM) has been defined without this restriction. Maurice Herlihy and J. Eliot B. Moss in their 1993 paper[2] describe a hardware implementation that can be implemented on top of existing cache coherency protocols reasonably easily.[1]

The description in the paper is generic. First, there is no need to require that transactional memory is implemented in hardware, exclusively or even in part. For the purpose mentioned in the paper's title (lock-free data structures), hardware support is likely going to be essential. But this is not true in general, as we will see shortly. Second, the description must be transferred to today's available hardware. This includes implementation details such as the possible reuse of the cache coherency protocol and the granularity of the transactions, which most likely will not be a single word but instead a cache line.

Hardware support for TM will itself be mostly interesting for the implementation of lock-free data structures. To implement, for example, the insert of a new element into a double-linked list without locking, four pointers have to be updated atomically. These pointers are found in three list elements, which mean it is not possible to implement this using simple atomic operations. Hardware TM (HTM) provides a means to implement atomic operations operating on more than one memory word. To provide more general support for transactional memory beyond atomic data structures, software support is needed. For example, any hardware implementation will have a limit on the size of a transaction. These limits might be too low for nontrivial programs or they might differ among implementations. Software can and must complete the HTM support to extend the reach of the TM implementation meant to be used for general programming.

This has been taken a step further. Because today's hardware is mostly lacking in HTM support, software TM (STM) is what most research projects are using today. With STM-based solutions it is possible to provide interfaces to TM functionality, which later could be implemented in *hybrid TM* implementations, using hardware if possible. This allows programmers to write programs using the simplifications TM provides even without HTM support in the hardware.

Show Me The Problem

To convince the reader that TM is worth all the trouble, let's look at a little example. This is not meant to reflect realistic code but instead illustrates problems that can happen in real code:

```
long counter1;
long counter2;
time_t timestamp1;
time_t timestamp2;

void f1_1(long *r, time_t *t) {
  *t = timestamp1;
  *r = counter1++;
}

void f2_2(long *r, time_t *t) {
  *t = timestamp2;
  *r = counter2++;
}

void w1_2(long *r, time_t *t) {
  *r = counter1++;
  if (*r & 1)
    *t = timestamp2;
}

void w2_1(long *r, time_t *t) {
  *r = counter2++;
  if (*r & 1)
    *t = timestamp1;
}
```

Assume this code has to be made thread safe. This means that multiple threads can concurrently execute any of the functions and that doing so must not produce any invalid result. The latter is defined here as return counter and timestamp values that don't belong together.

It is certainly possible to define one single mutex lock and require that this mutex is taken in each of the four functions. It is easy to verify that this would generate the expected results, but the performance is potentially far from optimal.

Assume that most of the time only the functions `f1_1` and `f2_2` are used. In this case there would never be any conflict between callers of these functions: callers of `f1_1` and `f2_2` could peacefully coexist. This means that using one single lock unnecessarily slows down the code.

So, then, use two locks. But how to define them? The semantics would have to be in the one case "when counter1 and timestamp1 are used" and "when counter2 and timestamp2 are used," respectively. This might work for `f1_1` and `f2_2`, but it won't work for the other two functions. Here the pairs counter1/timestamp2 and counter2/timestamp1 are used together. So we have to go yet another level down and assign a separate lock to each of the variables.

Assuming we would do this, we could easily be tempted to write something like this (only two functions are mentioned here; the other two are mirror images):

```
void f1_1(long *r, time_t *t) {
  lock(l_timestamp1);
  lock(l_counter1);

  *t = timestamp1;
  *r = counter1++;
}

void w1_2(long *r, time_t *t) {
  lock(l_counter1);

  *r = counter1++;
  if (*r & 1) {
    lock(l_timestamp1);
    *t = timestamp2;
    unlock(l_timestamp1);
  }

  unlock(l_counter1);
}
```

The code for `w1_2` in this example is wrong. We cannot delay getting the `l_timestamp1` lock because it might produce inconsistent results. Even though it might be slower, we always have to get the lock:

```
void w1_2(long *r, time_t *t) {
  lock(l_counter1);
  lock(l_timestamp1);

  *r = counter1++;
  if (*r & 1) {
    *t = timestamp2;

  unlock(l_timestamp1);
  unlock(l_counter1);
}
```

It's a simple change, but the result is also wrong. Now we try to lock the required locks in `w1_2` in a different order from `f1_1`. This potentially will lead to deadlocks. In this simple example it is easy to see that this is the case, but with just slightly more complicated code it is a very common occurrence.

What this example shows is: it is easy to get into a situation where many separate mutex locks are needed to allow for enough parallelism; and using all the mutex locks correctly is quite complicated by itself.

As can be expected from the theme of this article, TM will be able to help us in this and many other situations.

Rewritten Using TM

The previous example could be rewritten using TM as follows. In this example we are using nonstandard extensions to C that in one form or another might appear in a TM-enabled compiler. The extensions are easy to explain.

```
void f1_1(long *r, time_t *t) {
  tm_atomic {
    *t = timestamp1;
    *r = counter1++;
  }
}

void f2_2(long *r, time_t *t) {
  tm_atomic {
    *t = timestamp2;
    *r = counter2++;
  }
}

void w1_2(long *r, time_t *t) {
  tm_atomic {
    *r = counter1++;
    if (*r & 1)
      *t = timestamp2;
  }
}
```

```
void w2_1(long *r, time_t *t) {
  tm_atomic {
    *r = counter2++;
    if (*r & 1)
      *t = timestamp1;
  }
}
```

All we have done in this case is enclose the operations with a block header by `tm_atomic`. The `tm_atomic` keyword indicates that all the instructions in the following block are part of a transaction. For each of the memory accesses, the compiler could generate code as listed here. Calling functions is a challenge since the called functions also have to be transaction-aware. Therefore, it is potentially necessary to provide two versions of the compiled function: one with and one without support for transactions. In case any of the transitively called functions uses a `tm_atomic` block by itself, nesting has to be handled. The following is one way of doing this:

1. Check whether the same memory location is part of another.
2. If yes, abort the current transaction.
3. If no, record that the current transaction referenced the memory location so that step 2 in other transactions can find it.
4. Depending on whether it is a read or write access, either load the value of the memory location if the variable has not yet been modified or load it from the local storage in case it was already modified; or write it into a local storage for the variable.

Step 3 can fall away if the transaction previously accessed the same memory location. For step 2 there are alternatives. Instead of aborting immediately, the transaction can be performed to the end and then the changes undone. This is called the lazy abort/lazy commit method, as opposed to the eager/eager method found in typical database transactions (described previously).

What is needed now is a definition of the work that is done when the end of the `tm_atomic` block is reached (that is, the transaction is committed). This work can be described as follows:

1. If the current transaction has been aborted, reset all internal state, delay for some short period, then retry, executing the whole block.
2. Store all the values of the memory

ILLUSTRATION BY OLI LARUELLE

locations modified in the transaction for which the new values are placed in local storage.

3. Reset the information about the memory locations being part of a transaction.

The description is simple enough; the real problem is implementing everything efficiently. Before we discuss this, let's take a brief look at whether all this is correct and fulfills all the requirements.

Correctness and Fidelity

Assuming a correct implementation (of course), we are able to determine whether a memory location is currently used as part of another implementation. It does not matter whether this means read or write access. Therefore, it is easy to see that we are not ever producing inconsistent results. Only if all the memory accesses inside the `tm_atomic` block succeed and the transaction is not aborted will the transaction be committed. This means, however, that as far as memory access is concerned, the thread was completely alone. We have reduced the code back to the initial code without locks, which obviously was correct.

The only remaining question about correctness is: will the threads using this TM technology really terminate if they are constantly aborting each other? Showing this is certainly theoretically possible, but in this article it should be sufficient to point at a similar problem. In IP-based networking (unlike token-ring networks) all the connected machines could start sending out data at the same time. If more than one machine sends data, a conflict arises. This conflict is automatically detected and the sending attempt is restarted after a short waiting period. IP defines an exponential backup algorithm that the network stacks have to implement. Given that we live in a world dominated by IP-based networks, this approach must work fine. The results can be directly transferred over to the problem of TM.

One other question remains. Earlier we rejected the solution of using a single lock because it would prevent the concurrent execution of `f1_1` and `f2_2`. How does it look here? As can easily be seen, the set of memory locations used for the two functions is disjunct. This means that the sets of memory locations in the transactions in `f1_1` and `f2_2` is disjunct as well, and therefore the checks for concurrent memory uses in `f1_1` will never cause an abort because of the execution of `f2_2` and vice versa. Thus, it is indeed trivially possible to solve the issue using TM.

Add to this the concise way of describing transactions, and it should be obvious why TM is so attractive.

The problem of consistency is nothing new in the world of computers. In fact, it has been central to the entire solution in one particular area: databases.

Where is TM Today?

Before everybody gets too excited about the prospects of TM, we should remember that it is still very much a topic of research. First implementations are becoming available, but we still have much to learn. The VELOX project (http://www.velox-project.eu/), for example, has as its goal a comprehensive analysis of all the places in an operating system where TM technology can be used. This extends from lock-free data structures in the operating system kernel to high-level uses in the application server. The analysis includes TM with and without hardware support.

The project will also research the most useful semantics of the TM primitives that should be added to higher-level programming languages. In the previous example it was a simple `tm_atomic` keyword. This does not necessarily have to be the correct form; nor do the semantics described need to be the most optimal.

A number of self-contained STM implementations are available today. One possible choice for people to get experience with is TinySTM (http://tinystm.org). It provides all the primitives needed for TM while being portable, small, and depending on only a few services, which are available on the host system.

Based on TinySTM and similar implementations, we will soon see language extensions such as `tm_atomic` appear in compilers. Several proprietary compilers have support, and the first patches for the GNU compilers are also available (http://www.hipeac.net/node/2419). With these changes it will be possible to collect experience with the use of TM in real-world situations to find solutions to the remaining issues—and there are plenty of issues left. Here are just a few:

Recording transactions. In the explanation above we assumed that the exact location of each memory location used in the transaction is recorded. This might be inefficient, though, especially with HTM support. Recording information for every memory location would mean having an overhead of several words for each memory location used. As with CPU caches, which theoretically could also cache individual words, this often constitutes too high of a price. Instead, CPU caches today handle cache lines of 64 bytes at once. This would mean for a TM implementation that step 2 in our description would not have to record an additional dependency in case the memory location is in a block that is already recorded.

But this introduces problems as well. Assume that in the final example code all four variables are in the same block. This means that our assumption of `f1_1` and `f2_2` being independently executable is wrong. This type of block sharing leads to high abort rates. It is related to the problem of false sharing, which in this case also happens and therefore should be corrected anyway.

These false aborts, as we might want to call them, are not only a performance issue, though. Unfortunate placement of variables actually might lead to problems never making any progress at all because they are constantly inadvertently aborting each other. This can happen because of several different transactions going on concurrently that happen to touch the cache memory blocks but different addresses. If blocking is used, this is a problem that must be solved.

Handling aborts. Another detail described earlier is the way aborts are handled. What has been described is the so-called lazy abort/lazy commit method (lazy/lazy for short). Transactions continue to work even if they are already aborted and the results of the transaction are written into the real memory location only when the entire transaction succeeded.

This is not the only possibility, though. Another one is the exact opposite: the eager/eager method. In this case transactions will be recognized as aborted as early as possible and restarted if necessary. The effect of store instructions will also immediately take effect. In this case the old value of the memory location has to be stored in memory local to the transaction so that, in case the transaction has to be aborted, the previous content can be restored.

There are plenty of other ways to handle the details. It might turn out that no one way is sufficient. Much will depend on the abort rate for the individual transaction. It could very well be that compilers and TM runtimes will implement multiple different ways at the same time and flip between them for individual transactions if this seems to offer an advantage.

Semantics. The semantics of the `tm_atomic` block (or whatever it will be in the end) have to be specified. It is necessary to integrate TM into the rest of the language semantics. For example, TM must be integrated with exception handling for C++. Other issues are the handling of nested TM regions and the treatment of local variables (they need not be part of the transaction but still have to be reset on abort).

Performance. Performance is also a major issue. Plenty of optimizations can and should be performed by the compiler, and all this needs research. There are also practical problems. If the same program code is used in contested and uncontested situations (for example, in a single-threaded program), the overhead introduced through TM is too high. It therefore might be necessary to generate two versions of each function: one with TM support and the other without. The TM runtime then has to make sure that the version without TM support is used as frequently as possible. Failure to the one side means loss of performance, failure to the other side means the program will not run correctly.

Conclusion

TM promises to make parallel programming much easier. The concept of transaction is already present in many programs (from business programs to dynamic Web applications) and it proved to be reasonably easy to grasp for programmers. We can see first implementations coming out now, but all are far from ready for prime time. Much research remains to be done. C

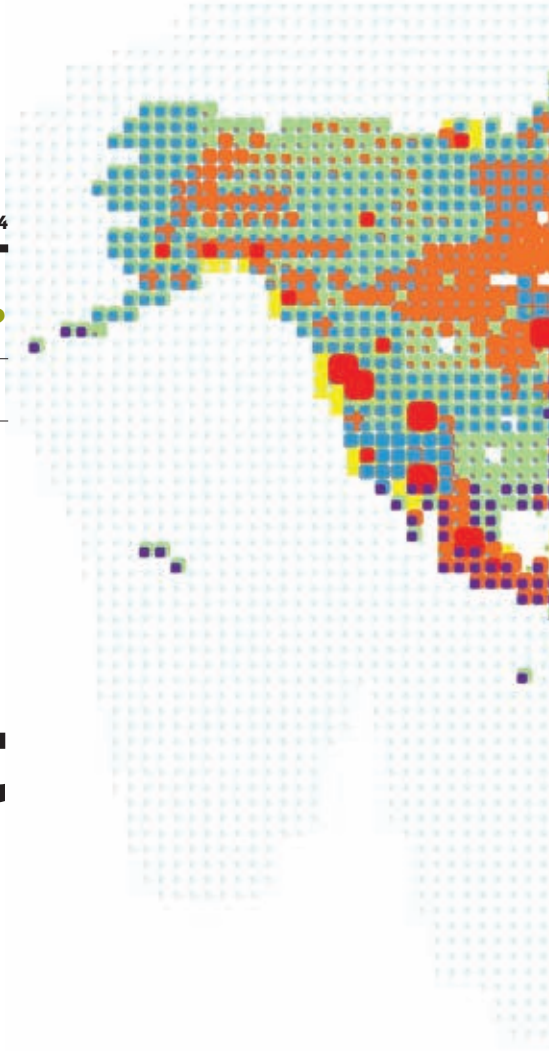
References

1. Drepper, U. What every programmer should know about memory (2007); http://people.redhat.com/drepper/cpumemory.pdf.
2. Herlihy, M., Moss, J.E.B. Transactional memory: Architectural support for lock-free data structures. In *Proceedings of 20th International Symposium on Computer Architecture* (1993); http://citeseer.ist.psu.edu/herlihy93transactional.html.

Ulrich Drepper (drepper@redhat.com) is a consulting engineer at Red Hat, where he has worked for the past 12 years. He is interested in all kinds of low-level programming and has been involved with Linux for almost 15 years.

A previous version of this appeared in the September 2008 issue of *ACM Queue*.

ILLUSTRATION BY OLI LARUELLE



DOI:10.1145/1461928.1461944

Given the Internet's bottlenecks, how can we build fast, scalable, content-delivery systems?

BY TOM LEIGHTON

Improving Performance on the Internet

WHEN IT COMES to achieving performance, reliability, and scalability for commercial-grade Web applications, where is the biggest bottleneck? In many cases today, we see that the limiting bottleneck is the *middle mile*, or the time data spends traveling back and forth across the Internet, between origin server and end user.

This wasn't always the case. A decade ago, the *last mile* was a likely culprit, with users constrained to sluggish dial-up modem access speeds. But recent high levels of global broadband penetration—more than 300 million subscribers worldwide—have not only made the last-mile bottleneck history, they have also increased pressure on the rest of the Internet infrastructure to keep pace.[5]

Today, the *first mile*—that is, origin infrastructure—tends to get most of the attention when it comes to designing Web applications. This is the portion of the problem that falls most within an application architect's control. Achieving good first-mile performance and reliability is now a fairly well-understood and tractable problem. From the *end user*'s point of view, however, a robust first mile is necessary, but not sufficient, for achieving strong application performance and reliability.

This is where the middle mile comes in. Difficult to tame and often ignored, the Internet's nebulous middle mile injects latency bottlenecks, throughput constraints, and reliability problems into the Web application performance equation. Indeed, the term *middle mile* is itself a misnomer in that it refers to a heterogeneous infrastructure that is owned by many competing entities and typically spans hundreds or thousands of miles.

This article highlights the most serious challenges the middle mile presents today and offers a look at the approaches to overcoming these challenges and improving performance on the Internet.

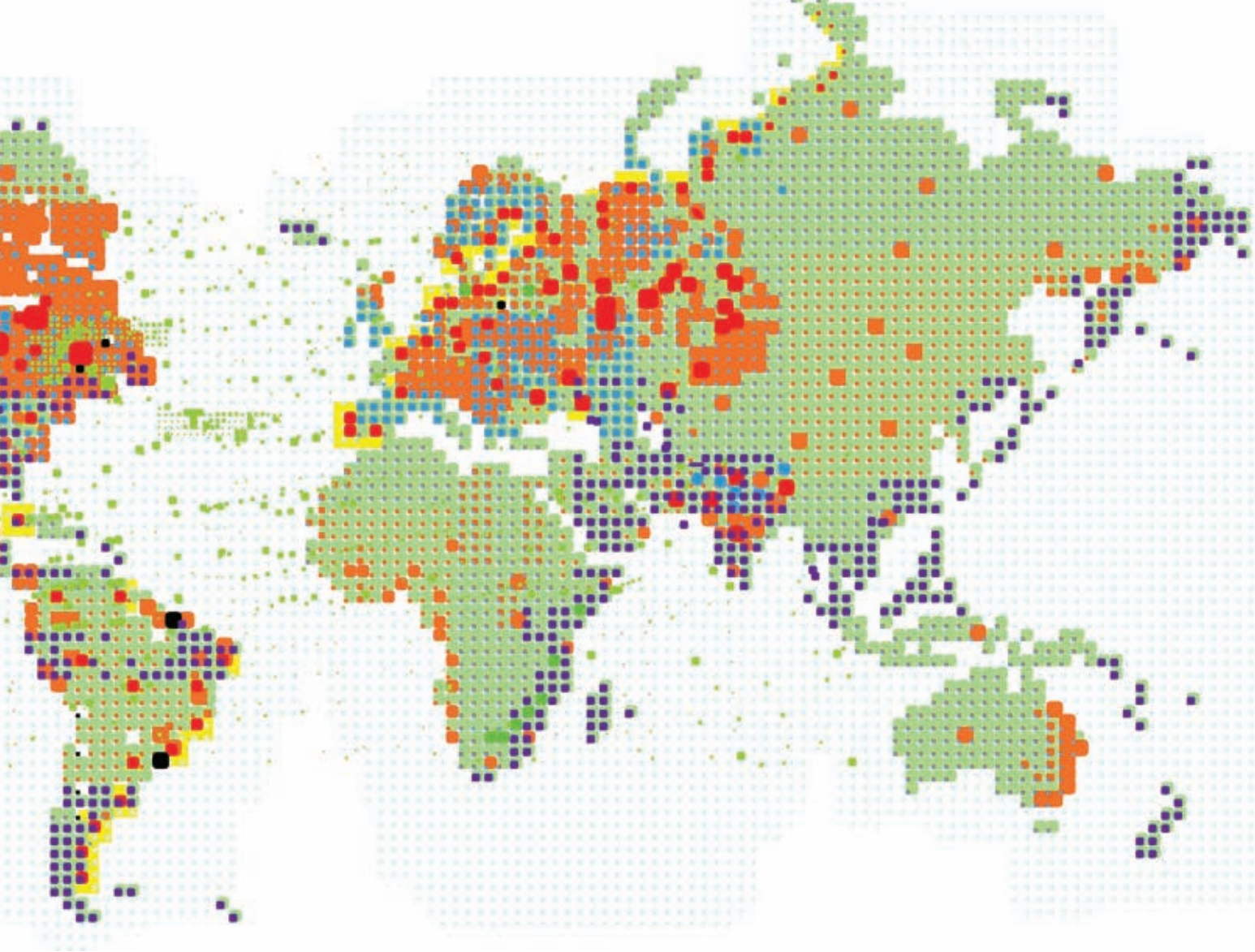


ILLUSTRATION BY STUDIO TONNE

Stuck in the Middle

While we often refer to the Internet as a single entity, it is actually composed of 13,000 different competing networks, each providing access to some small subset of end users. Internet capacity has evolved over the years, shaped by market economics. Money flows into the networks from the first and last miles, as companies pay for hosting and end users pay for access. First- and last-mile capacity has grown 20- and 50-fold, respectively, over the past five to 10 years.

On the other hand, the Internet's middle mile—made up of the peering and transit points where networks trade traffic—is literally a no man's land. Here, economically, there is very little incentive to build out capacity. If anything, networks want to minimize traffic coming into their networks that they don't get paid for. As a result, peering points are often overburdened, causing packet loss and service degradation.

The fragile economic model of peering can have even more serious consequences. In March 2008, for example, two major network providers, Cogent and Telia, de-peered over a business dispute. For more than a week, customers from Cogent lost access to Telia and the networks connected to it, and vice versa, meaning that Cogent and Telia end users could not reach certain Web sites at all.

Other reliability issues plague the middle mile as well. Internet outages have causes as varied as transoceanic cable cuts, power outages, and DDoS (distributed denial-of-service) attacks. In February 2008, for example, communications were severely disrupted in Southeast Asia and the Middle East when a series of undersea cables were cut. According to TeleGeography, the cuts reduced bandwidth connectivity between Europe and the Middle East by 75%.[8]

Internet protocols such as BGP (Border Gateway Protocol, the Internet's primary internetwork routing algorithm) are just as susceptible as the physical network infrastructure. For example, in February 2008, when Pakistan tried to block access to YouTube from within the country by broadcasting a more specific BGP route, it accidentally caused a near-global YouTube blackout, underscoring the vulnerability of BGP to human error (as well as foul play).[2]

The prevalence of these Internet reliability and peering-point problems means that the longer data must travel through the middle mile, the more it is subject to congestion, packet loss, and poor performance. These middle-mile problems are further exacerbated by current trends—most notably the increase in last-mile capacity and demand. Broadband adoption continues to rise, in terms of both penetration and speed, as ISPs invest in last-mile infrastructure. AT&T just spent approximately $6.5 billion to roll out its U-verse service, while Verizon is

spending $23 billion to wire 18 million homes with FiOS (Fiber-optic Service) by 2010.[6,7] Comcast also recently announced it plans to offer speeds of up to 100Mbps within a year.[3]

Demand drives this last-mile boom: Pew Internet's 2008 report shows that one-third of U.S. broadband users have chosen to pay more for faster connections.[4] Akamai Technologies' data, shown in Figure 1, reveals that 59% of its global users have broadband connections (with speeds greater than 2 Mbps), and 19% of global users have "high broadband" connections greater than 5Mbps—fast enough to support DVD-quality content.[2] The high-broadband numbers represent a 19% increase in just three months.

A Question of Scale

Along with the greater demand and availability of broadband comes a rise in user expectations for faster sites, richer media, and highly interactive applications. The increased traffic loads and performance requirements in turn put greater pressure on the Internet's internal infrastructure—the middle mile. In fact, the fast-rising popularity of video has sparked debate about whether the Internet can scale to meet the demand.

Consider, for example, delivering a TV-quality stream (2Mbps) to an audience of 50 million viewers, approximately the audience size of a popular TV show. The scenario produces aggregate bandwidth requirements of 100Tbps. This is a reasonable vision for the near term—the next two to five years—but it is orders of magnitude larger than the biggest online events today, leading to skepticism about the Internet's ability to handle such demand. Moreover, these numbers are just for a single TV-quality show. If hundreds of millions of end users were to download Blu-ray-quality movies regularly over the Internet, the resulting traffic load would go up by an additional one or two orders of magnitude.

Another interesting side effect of the growth in video and rich media file sizes is that the distance between server and end user becomes critical to end-user performance. This is the result of a somewhat counterintuitive phenomenon that we call the Fat File Paradox: given that data packets can traverse networks at close to the speed of light, why does it takes so long for a "fat file" to cross the country, even if the network is not congested?

It turns out that because of the way the underlying network protocols work, latency and throughput are directly coupled. TCP, for example, allows only small amounts of data to be sent at a time (that is, the TCP window) before having to pause and wait for acknowledgments from the receiving end. This means that throughput is effectively throttled by network round-trip time (latency), which can become the bottleneck for file download speeds and video viewing quality.

Packet loss further complicates the problem, since these protocols back off and send even less data before waiting for acknowledgment if packet loss is detected. Longer distances increase the chance of congestion and packet loss to the further detriment of throughput.

Figure 2 illustrates the effect of distance (between server and end user) on throughput and download times. Five or 10 years ago, dial-up modem speeds would have been the bottleneck on these files, but as we look at the Internet today and into the future, middle-mile distance becomes the bottleneck.

Figure 1: Broadband penetration by country.

Broadband

Ranking	Country	% > 2Mbps
—	Global	59%
1	South Korea	90%
2	Belgium	90%
3	Japan	87%
4	Hong Kong	87%
5	Switzerland	85%
6	Slovakia	83%
7	Norway	82%
8	Denmark	79%
9	Netherlands	77%
10	Sweden	75%
...		
20.	United States	71%

Fast Broadband

Ranking	Country	% > 5Mbps
—	Global	19%
1	South Korea	64%
2	Japan	52%
3	Hong Kong	37%
4	Sweden	32%
5	Belgium	26%
6	United States	26%
7	Romania	22%
8	Netherlands	22%
9	Canada	18%
10	Denmark	18%

Source: Akamai's State of the Internet Report, 02 2008

Four Approaches to Content Delivery

Given these bottlenecks and scalability challenges, how does one achieve the levels of performance and reliability required for effective delivery of content and applications over the Internet? There are four main approaches to distributing content servers in a content-delivery architecture: centralized hosting, "big data center" CDNs (content-delivery networks), highly distributed CDNs, and peer-to-peer networks.

Centralized Hosting. Traditionally architected Web sites use one or a small number of collocation sites to host content. Commercial-scale sites generally have at least two geographically dispersed mirror locations to provide additional performance (by being closer to different groups of end users), reliability (by providing redundancy), and scalability (through greater capacity).

This approach is a good start, and for small sites catering to a localized audience it may be enough. The performance and reliability fall short of expectations for commercial-grade sites and applications, however, as the end-user experience is at the mercy of the unreliable Internet and its middle-mile bottlenecks.

There are other challenges as well: site mirroring is complex and costly, as is managing capacity. Traffic levels fluctuate tremendously, so the need to provision for peak traffic levels means that expensive infrastructure will sit underutilized most of the time. In addition, accurately predicting traffic demand is extremely difficult, and a centralized hosting model does not provide the flexibility to handle unexpected surges.

"Big Data Center" CDNs. Content-delivery networks offer improved scalability by offloading the delivery of cacheable content from the origin server onto a larger, shared network. One common CDN approach can be described as "big data center" architecture—caching and delivering customer content from perhaps a couple dozen high-capacity data centers connected to major backbones.

Although this approach offers some performance benefit and economies of scale over centralized hosting, the potential improvements are limited because the CDN's servers are still far away from most users and still deliver content from the wrong side of the middle-mile bottlenecks.

It may seem counterintuitive that having a presence in a couple dozen major backbones isn't enough to achieve commercial-grade performance. In fact, even the largest of those networks controls very little end-user access traffic. For example, the top 30 networks *combined* deliver only 50% of end-user traffic, and it drops off quickly from there, with a very long tail distribution over the Internet's 13,000 networks. Even with connectivity to all the biggest backbones, data must travel through the morass of the middle mile to reach most of the Internet's 1.4 billion users.

A quick back-of-the-envelope calculation shows that this type of architecture hits a wall in terms of scalability as we move toward a video world. Consider a generous forward projection on such an architecture—say, 50 high-capacity data centers, each with 30 outbound connections, 10Gbps each. This gives an upper bound of 15Tbps total capacity for this type of network, far short of the 100Tbps needed to support video in the near term.

Highly Distributed CDNs. Another approach to content delivery is to leverage a very highly distributed network—one with servers in thousands of networks, rather than dozens. On the surface, this architecture may appear quite similar to the "big data center" CDN. In reality, however, it is a fundamentally different approach to content-server placement, with a difference of two orders of magnitude in the degree of distribution.

By putting servers within end-user ISPs, for example, a highly distributed CDN delivers content from the right side of the middle-mile bottlenecks, eliminating peering, connectivity, routing, and distance problems, and reducing the number of Internet components depended on for success.

Moreover, this architecture scales. It can achieve a capacity of 100Tbps, for example, with deployments of 20 servers, each capable of delivering 1Gbps in 5,000 edge locations.

On the other hand, deploying a highly distributed CDN is costly and time consuming, and comes with its own set of challenges. Fundamentally, the network must be designed to scale efficiently from a deployment and management perspective. This necessitates development of a number of technologies, including:

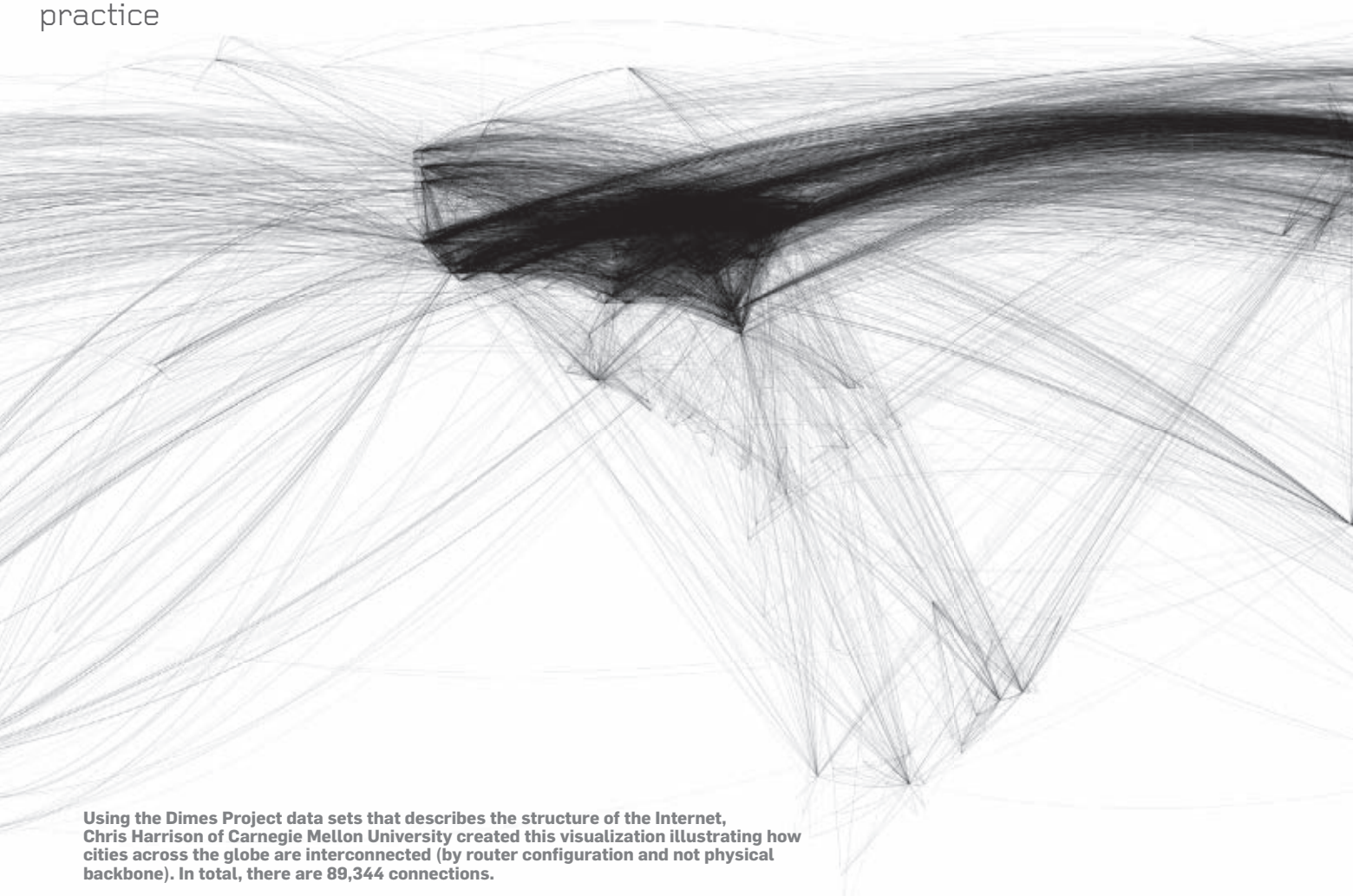
- Sophisticated global-scheduling, mapping, and load-balancing algorithms
- Distributed control protocols and reliable automated monitoring and alerting systems
- Intelligent and automated failover and recovery methods
- Colossal-scale data aggregation and distribution technologies (designed to handle different trade-offs between timeliness and accuracy or completeness)
- Robust global software-deployment mechanisms
- Distributed content freshness, integrity, and management systems
- Sophisticated cache-management protocols to ensure high cache-hit ratios

These are nontrivial challenges, and we present some of our approaches later on in this article.

Peer-to-Peer Networks. Because a highly distributed architecture is critical to achieving scalability and perfor-

Figure 2: Effect of distance on throughput and download times.

Distance from Server to User	Network Latency	Typical Packet Loss	Throughput (quality)	4GB DVD Download Time
Local: <100 mi.	1.6ms	0.6%	44Mbs (HDTV)	12 min.
Regional: 500–1,000 mi.	16ms	0.7%	4Mbs (not quite DVD)	2.2 hrs.
Cross-continent: ~3,000 mi.	48ms	1.0%	1Mbs (not quite TV)	8.2 hrs.
Multi-continent: ~6,000 mi.	96ms	1.4%	0.4Mbs (poor)	20 hrs.



Using the Dimes Project data sets that describes the structure of the Internet, Chris Harrison of Carnegie Mellon University created this visualization illustrating how cities across the globe are interconnected (by router configuration and not physical backbone). In total, there are 89,344 connections.

mance in video distribution, it is natural to consider a P2P (peer-to-peer) architecture. P2P can be thought of as taking the distributed architecture to its logical extreme, theoretically providing nearly infinite scalability. Moreover, P2P offers attractive economics under current network pricing structures.

In reality, however, P2P faces some serious limitations, most notably because the total download capacity of a P2P network is throttled by its total uplink capacity. Unfortunately, for consumer broadband connections, uplink speeds tend to be much lower than downlink speeds: Comcast's standard high-speed Internet package, for example, offers 6Mbps for download but only 384Kbps for upload (one-sixteenth of download throughput).

This means that in situations such as live streaming where the number of uploaders (peers sharing content) is limited by the number of downloaders (peers requesting content), average download throughput is equivalent to the average uplink throughput and thus cannot support even mediocre Web-quality streams. Similarly, P2P fails in "flash crowd" scenarios where there is a sudden, sharp increase in demand, and the number of downloaders greatly outstrips the capacity of uploaders in the network.

Somewhat better results can be achieved with a hybrid approach, leveraging P2P as an extension of a distributed delivery network. In particular, P2P can help reduce overall distribution costs in certain situations. Because the capacity of the P2P network is limited, however, the architecture of the non-P2P portion of the network still governs overall performance and scalability.

Each of these four network architectures has its trade-offs, but ultimately, for delivering rich media to a global Web audience, a highly distributed architecture provides the only robust solution for delivering commercial-grade performance, reliability, and scale.

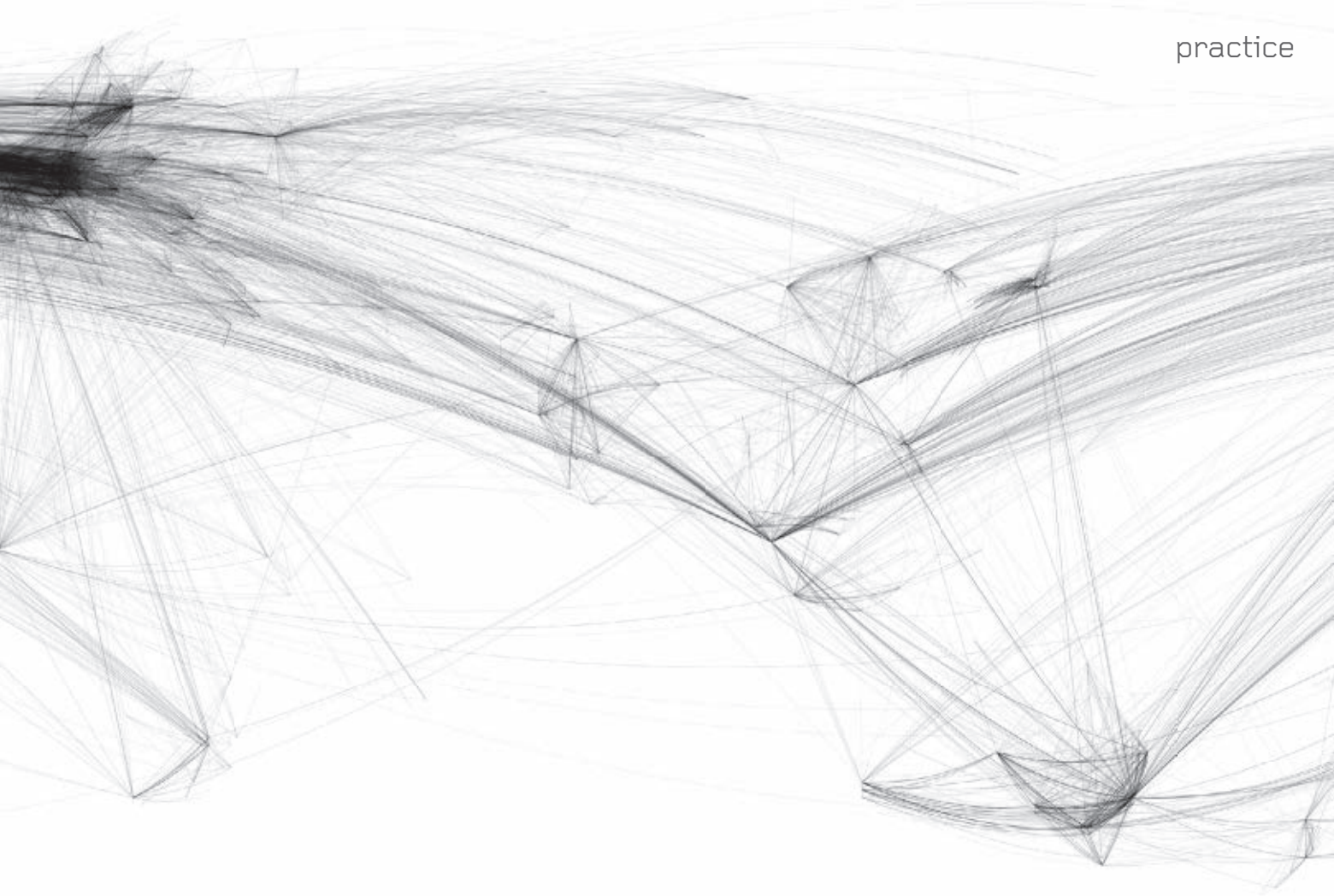
Application Acceleration

Historically, content-delivery solutions have focused on the offloading and delivery of static content, and thus far we have focused our conversation on the same. As Web sites become increasingly dynamic, personalized, and application-driven, however, the ability to accelerate *uncacheable* content becomes equally critical to delivering a strong end-user experience.

Ajax, Flash, and other RIA (rich Internet application) technologies work to enhance Web application responsiveness on the browser side, but ultimately, these types of applications all still require significant numbers of round-trips back to the origin server. This makes them highly susceptible to all the bottlenecks I've mentioned before: peering-point congestion, network latency, poor routing, and Internet outages.

Speeding up these round-trips is a complex problem, but many optimizations are made possible by using a highly distributed infrastructure.

Optimization 1: Reduce transport-layer overhead. Architected for reliability over efficiency, protocols such as TCP have substantial overhead. They require multiple round-trips (between the two communicating parties) to set



up connections, use a slow initial rate of data exchange, and recover slowly from packet loss. In contrast, a network that uses persistent connections and optimizes parameters for efficiency (given knowledge of current network conditions) can significantly improve performance by reducing the number of round-trips needed to deliver the same set of data.

Optimization 2: Find better routes. In addition to reducing the number of round-trips needed, we would also like to reduce the time needed for each round-trip—each journey across the Internet. At first blush, this does not seem possible. All Internet data must be routed by BGP and must travel over numerous autonomous networks.

BGP is simple and scalable but not very efficient or robust. By leveraging a highly distributed network—one that offers potential intermediary servers on many different networks—you can actually speed up uncacheable communications by 30% to 50% or more, by using routes that are faster and much less congested. You can also achieve much greater communications reliability by finding alternate routes when the default routes break.

Optimization 3: Prefetch embedded content. You can do a number of additional things at the application layer to improve Web application responsiveness for end users. One is to prefetch embedded content: while an edge server is delivering an HTML page to an end user, it can also parse the HTML and retrieve all embedded content *before* it is requested by the end user's browser.

The effectiveness of this optimization relies on having servers near end users, so that users perceive a level of application responsiveness akin to that of an application being delivered directly from a nearby server, even though, in fact, some of the embedded content is being fetched from the origin server across the long-haul Internet. Prefetching by forward caches, for example, does not provide this performance benefit because the prefetched content must still travel over the middle mile before reaching the end user. Also, note that unlike link prefetching (which can also be done), embedded content prefetching does not expend extra bandwidth resources and does not request extraneous objects that may not be requested by the end user.

With current trends toward highly personalized applications and user-generated content, there's been growth in either uncacheable or long-tail (that is, not likely to be in cache) embedded content. In these situations, prefetching makes a huge difference in the user-perceived responsiveness of a Web application.

Optimization 4: Assemble pages at the edge. The next three optimizations involve reducing the amount of content that needs to travel over the middle mile. One approach is to cache page fragments at edge servers and dynamically assemble them at the edge in response to end-user requests. Pages can be personalized (at the edge) based on characteristics including the end user's location, connection speed, cookie values, and so forth. Assembling the page at the edge not only offloads the origin server, but also results in much lower latency to the end user, as the middle mile is avoided.

Optimization 5: Use compression and delta encoding. Compression of HTML and other text-based components can reduce the amount of content traveling over the middle mile to one-tenth of the original size. The use of delta encoding, where a server sends only the *difference* between a cached HTML page and a dynamically generated version, can also greatly cut down on the amount of content that must travel over the long-haul Internet.

While these techniques are part of the HTTP/1.1 specification, browser support is unreliable. By using a highly distributed network that controls both endpoints of the middle mile, compression and delta encoding can be successfully employed regardless of the browser. In this case, performance is improved because very little data travels over the middle mile. The edge server then decompresses the content or applies the delta encoding and delivers the complete, correct content to the end user.

Optimization 6: Offload computations to the edge. The ability to distribute applications to edge servers provides the ultimate in application performance and scalability. Akamai's network enables distribution of J2EE applications to edge servers that create virtual application instances on demand, as needed. As with edge page assembly, edge computation enables complete origin server offloading, resulting in tremendous scalability and extremely low application latency for the end user.

While not every type of application is an ideal candidate for edge computation, large classes of popular applications—such as contests, product catalogs, store locators, surveys, product configurators, games, and the like—are well suited for edge computation.

Putting it All Together

Many of these techniques require a highly distributed network. Route optimization, as mentioned, depends on the availability of a vast overlay network that includes machines on many different networks. Other optimizations such as prefetching and page assembly are most effective if the delivering server is near the end user. Finally, many transport and application-layer optimizations require bi-nodal connections within the network (that is, you control both endpoints). To maximize the effect of this optimized connection, the endpoints should be as close as possible to the origin server and the end user.

Note also that these optimizations work in synergy. TCP overhead is in large part a result of a conservative approach that guarantees reliability in the face of unknown network conditions. Because route optimization gives us high-performance, congestion-free paths, it allows for a much more aggressive and efficient approach to transport-layer optimizations.

Highly Distributed Network Design

It was briefly mentioned earlier that building and managing a robust, highly distributed network is not trivial. At Akamai, we sought to build a system with extremely high reliability—no downtime, ever—and yet scalable enough to be managed by a relatively small operations staff, despite operating in a highly heterogeneous and unreliable environment. Here are some insights into the design methodology.

The fundamental assumption behind Akamai's design philosophy is that a significant number of component or other failures are occurring at all times in the network. Internet systems present numerous failure modes, such as machine failure, data-center failure, connectivity failure, software failure, and network failure—all occurring with greater frequency than one might think. As mentioned earlier, for example, there are many causes of large-scale network outages—including peering problems, transoceanic cable cuts, and major virus attacks.

Designing a scalable system that works under these conditions means embracing the failures as natural and expected events. The network should continue to work seamlessly despite these occurrences. We have identified some practical design principles that result from this philosophy, which we share here.[1]

Principle 1: Ensure significant redundancy in all systems to facilitate failover. Although this may seem obvious and simple in theory, it can be challenging in practice. Having a highly distributed network enables a great deal of redundancy, with multiple backup possibilities ready to take over if a component fails. To ensure robustness of all systems, however, you will likely need to work around the constraints of existing protocols and interactions with third-party software, as well as balancing trade-offs involving cost.

For example, the Akamai network relies heavily on DNS (Domain Name System), which has some built-in constraints that affect reliability. One example is DNS's restriction on the size of responses, which limits the number of IP addresses that we can return to a relatively static set of 13. The Generic Top Level Domain servers, which supply the critical answers to akamai.net queries, required more reliability, so we took several steps, including the use of IP Anycast.

We also designed our system to take into account DNS's use of TTLs (time to live) to fix resolutions for a period of time. Though the efficiency gained through TTL use is important, we need to make sure users aren't being sent to servers based on stale data. Our approach is to use a two-tier DNS—employing longer TTLs at a global level and shorter TTLs at a local level— allowing less of a trade-off between DNS efficiency and responsiveness to changing conditions. In addition, we have built in appropriate failover mechanisms at each level.

Principle 2: Use software logic to provide message reliability. This design principle speaks directly to scalability. Rather than building dedicated links between data centers, we use the public Internet to distribute data—including control messages, configurations, monitoring information, and customer content—throughout our network. We improve on the performance of existing Internet protocols—for example, by using multirouting and limited retransmissions with UDP (User Datagram Protocol) to achieve reliability without sacrificing latency. We also use software to route data through intermediary servers to ensure communications (as described in Optimization 2), even when major disruptions (such as cable cuts) occur.

Principle 3: Use distributed control for coordination. Again, this principle is important both for fault tolerance and scalability. One practical example is the use of leader election, where leadership evaluation can depend on many factors

including machine status, connectivity to other machines in the network, and monitoring capabilities. When connectivity of a local lead server degrades, for example, a new server is automatically elected to assume the role of leader.

Principle 4: Fail cleanly and restart. Based on the previous principles, the network has already been architected to handle server failures quickly and seamlessly, so we are able to take a more aggressive approach to failing problematic servers and restarting them from a last known good state. This sharply reduces the risk of operating in a potentially corrupted state. If a given machine continues to require restarting, we simply put it into a "long sleep" mode to minimize impact to the overall network.

Principle 5: Phase software releases. After passing the quality assurance (QA) process, software is released to the live network in phases. It is first deployed to a single machine. Then, after performing the appropriate checks, it is deployed to a single region, then possibly to additional subsets of the network, and finally to the entire network. The nature of the release dictates how many phases and how long each one lasts. The previous principles, particularly use of redundancy, distributed control, and aggressive restarts, make it possible to deploy software releases frequently and safely using this phased approach.

Principle 6: Notice and proactively quarantine faults. The ability to isolate faults, particularly in a recovery-oriented computing system, is perhaps one of the most challenging problems and an area of important ongoing research. Here is one example. Consider a hypothetical situation where requests for a certain piece of content with a rare set of configuration parameters trigger a latent bug. Automatically failing the servers affected is not enough, as requests for this content will then be directed to other machines, spreading the problem. To solve this problem, our caching algorithms constrain each set of content to certain servers so as to limit the spread of fatal requests. In general, no single customer's content footprint should dominate any other customer's footprint among available servers. These constraints are dynamically determined based on current levels of demand for the content, while keeping the network safe.

Practical Results and Benefits

Besides the inherent fault-tolerance benefits, a system designed around these principles offers numerous other benefits.

Faster software rollouts. Because the network absorbs machine and regional failures without impact, Akamai is able to safely but aggressively roll out new software using the phased rollout approach. As a benchmark, we have historically implemented approximately 22 software releases and 1,000 customer configuration releases per month to our worldwide network, without disrupting our always-on services.

Minimal operations overhead. A large, highly distributed, Internet-based network can be very difficult to maintain, given its sheer size, number of network partners, heterogeneous nature, and diversity of geographies, time zones, and languages. Because the Akamai network design is based on the assumption that components will fail, however, our operations team does not need to be concerned about most failures. In addition, the team can aggressively suspend machines or data centers if it sees any slightly worrisome behavior. There is no need to rush to get components back online right away, as the network absorbs the component failures without impact to overall service.

This means that at any given time, it takes only eight to 12 operations staff members, on average, to manage our network of approximately 40,000 devices (consisting of more than 35,000 servers plus switches and other networking hardware). Even at peak times, we successfully manage this global, highly distributed network with fewer than 20 staff members.

Lower costs, easier to scale. In addition to the minimal operational staff needed to manage such a large network, this design philosophy has had several implications that have led to reduced costs and improved scalability. For example, we use commodity hardware instead of more expensive, more reliable servers. We deploy in third-party data centers instead of having our own. We use the public Internet instead of having dedicated links. We deploy in greater numbers of smaller regions—many of which host our servers for free—rather than in fewer, larger, more "reliable" data centers where congestion can be greatest.

Conclusion

Even though we've seen dramatic advances in the ubiquity and usefulness of the Internet over the past decade, the real growth in bandwidth-intensive Web content, rich media, and Web- and IP-based applications is just beginning. The challenges presented by this growth are many: as businesses move more of their critical functions online, and as consumer entertainment (games, movies, sports) shifts to the Internet from other broadcast media, the stresses placed on the Internet's middle mile will become increasingly apparent and detrimental. As such, we believe the issues raised in this article and the benefits of a highly distributed approach to content delivery will only grow in importance as we collectively work to enable the Internet to scale to the requirements of the next generation of users. C

References

1. Afergan, M., Wein, J., LaMeyer, A. Experience with some principles for building an Internet-scale reliable system. In *Proceedings of the 2nd Conference on Real, Large Distributed Systems 2.* (These principles are laid out in more detail in this 2005 research paper.)
2. Akamai Report: The State of the Internet, 2nd quarter, 2008; http://www.akamai.com/stateoftheinternet/. (These and other recent Internet reliability events are discussed in Akamai's quarterly report.)
3. Anderson, N. Comcast at CES: 100 Mbps connections coming this year. ars technica (Jan. 8, 2008); http://arstechnica.com/news.ars/post/20080108-comcast-100mbps-connections-coming-this-year.html.
4. Horrigan, J.B. Home Broadband Adoption 2008. Pew Internet and American Life Project; http://www.pewinternet.org/pdfs/PIP_Broadband_2008.pdf.
5. Internet World Statistics. Broadband Internet Statistics: Top World Countries with Highest Internet Broadband Subscribers in 2007; http://www.internetworldstats.com/dsl.htm.
6. Mehta, S. Verizon's big bet on fiber optics. *Fortune* (Feb. 22, 2007); http://money.cnn.com/magazines/fortune/fortune_archive/2007/03/05/8401289/.
7. Spangler T. AT&T: U-verse TV spending to increase. *Multichannel News* (May 8, 2007); http://www.multichannel.com/article/CA6440129.html.
8. TeleGeography. Cable cuts disrupt Internet in Middle East and India. *CommsUpdate* (Jan. 31, 2008); http://www.telegeography.com/cu/article.php?article_id=21528.

Tom Leighton co-founded Akamai Technologies in August 1998. Serving as chief scientist and as a director to the board, he is Akamai's technology visionary, as well as a key member of the executive committee setting the company's direction. He is an authority on algorithms for network applications. Leighton is a Fellow of the American Academy of Arts and Sciences, the National Academy of Science, and the National Academy of Engineering.

A previous version of this article appeared in the October 2008 issue of *ACM Queue* magazine.

DOI:10.1145/1461928.1461945

2^w is a result of the exponentially growing Web building on itself to move from a Web of content to a Web of applications.

BY T.V. RAMAN

Toward 2^w, Beyond Web 2.0

FROM ITS INCEPTION as a global hypertext system, the Web has evolved into a universal platform for deploying loosely coupled distributed applications. As we move toward the next-generation Web platform, the bulk of user data and applications will reside in the network cloud. Ubiquitous access results from interaction delivered as Web pages augmented by JavaScript to create highly reactive user interfaces. This point in the evolution of the Web is often called Web 2.0. In predicting what comes after Web 2.0—what I call 2^w, a Web that encompasses all Web-addressable information—I go back to the architectural foundations of the Web, analyze the move to Web 2.0, and look forward to what might follow.

For most users of the Internet, the Web is epitomized by the browser, the program they use to log on to the Web. However, in its essence, the Web, which is both a lot more and a lot less than the browser, is built on three components:

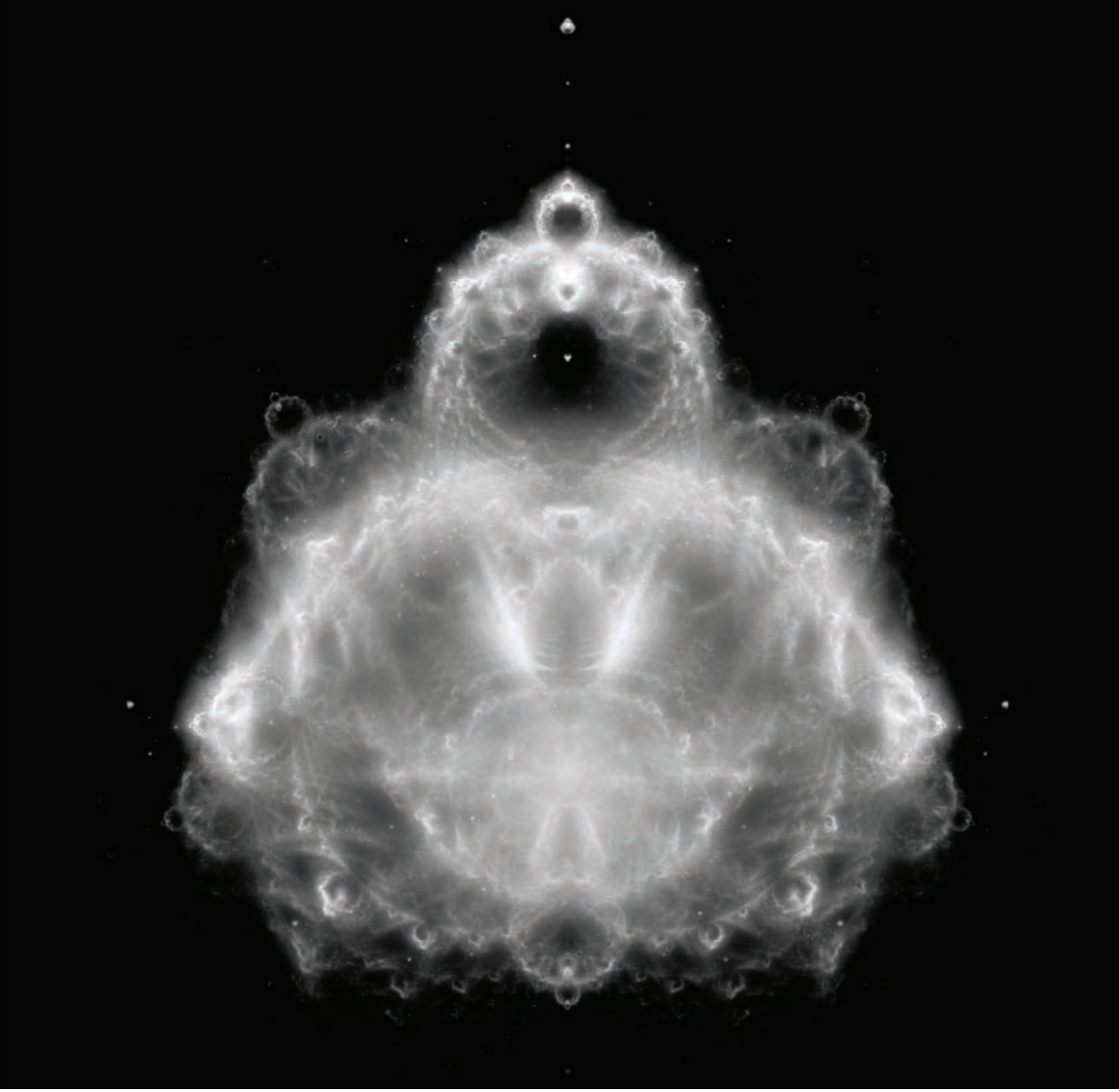
URL. A universal means for identifying and addressing content[6,7];

HTTP. A protocol for client-server communication[5]; and

HTML. A simple markup language for communicating hypertext content.[8]

Together, they constitute the global hypertext system. This decentralized architecture[35] was designed from the outset to create an environment where content producers and consumers come together without everyone having to use the same server and client. To participate in the Web revolution, one needed only to subscribe to the basic architecture of Web content delivered via HTTP and addressable via URLs. This yielded the now well-understood network effect that continues to produce exponential growth in the amount of available Web content. In the 1990s, the browser, a universal lens for viewing the Web, came to occupy center stage as the Web's primary interface. Deploying content to users on multiple platforms was suddenly a lot simpler; all one needed to enable universal access was to publish content to the Web. Note that this access was a direct consequence (by design) of the underlying Web contract, whereby Web publishers are isolated from the details of the client software used by their consumers. As Web browsers began to compete on features, this began to change in what became known as the browser wars, 1995–1999[36]; browser vendors competed by introducing custom tags into their particular flavors of HTML. This was perhaps the first of the many battles that would follow and is remembered today by most Web developers as the `blink` and `marquee` tag era marked by visual excess.

In 1997, HTML 3.2 attempted to ease the life of Web developers by documenting the existing authoring practice of the time. HTML 3.2 was in turn followed by HTML4[28] as a baseline markup language for the Web. At the same time, Cascading Style Sheets (CSS)[9] were introduced as a means of separating presentational information (style rules) from Web-page con-

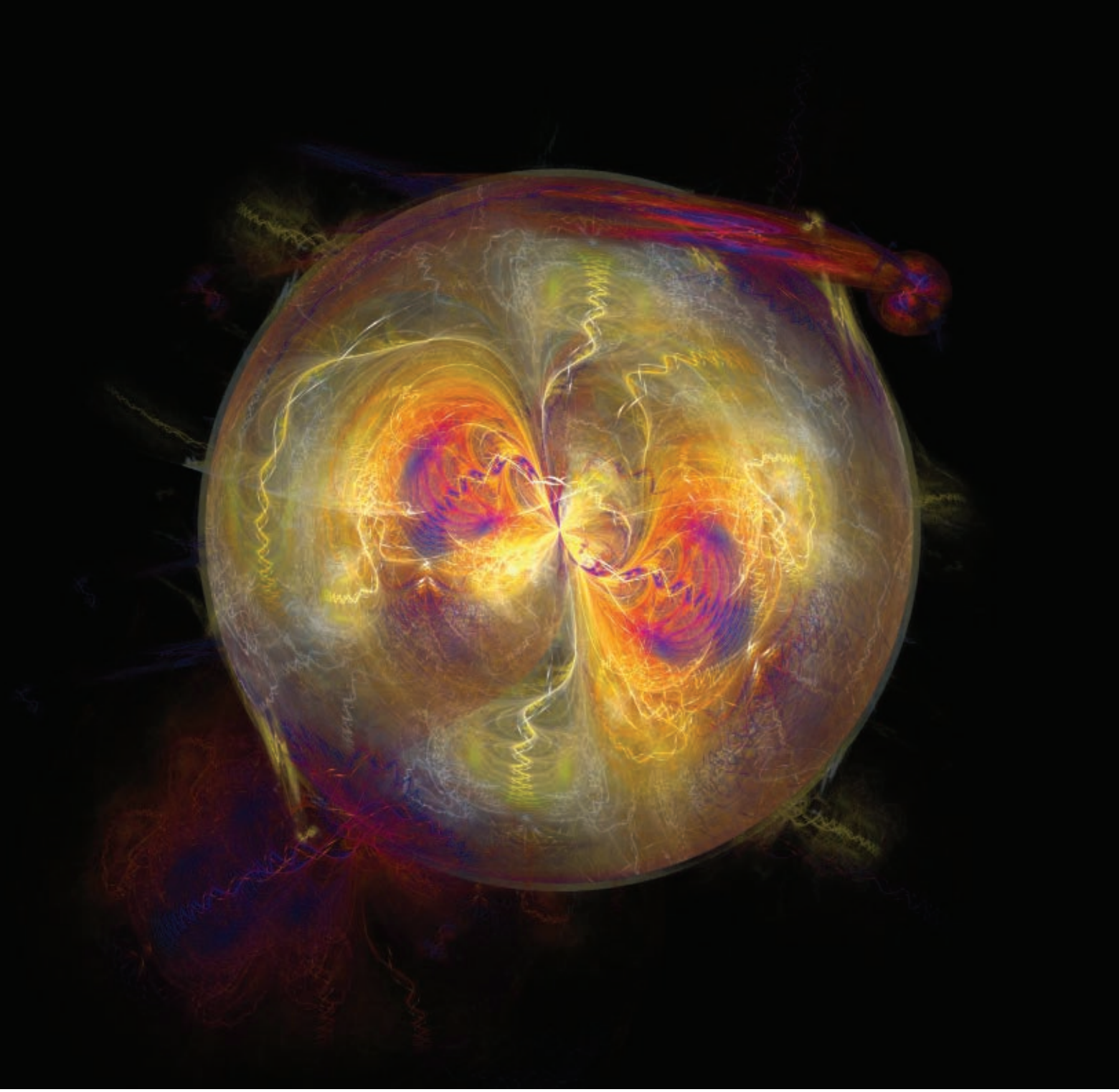


The self-similar repeating nature of fractals is a metaphor for the growth of the entire Web. This image by Jared Tarbell is a revisualization of the familiar Mandelbrot set; www.complexification.net/gallery/machines/buddabrot/.

IMAGE BY JARED TARBELL

tent. CSS enabled Web developers to flexibly style their content and was in part responsible for reducing their urge to invent new HTML tags purely to achieve a particular visual effect. But by 1998–1999, the browser wars were all but done, with Web developers coding mostly to the then-dominant browser, Microsoft's Internet Explorer 5. New features were no longer necessarily exposed via new tags in the HTML vocabulary; with CSS, a developer could easily create new presentational structures using the generic `div` and `span` tags. The behavior of constructs appearing in Web pages could be customized via JavaScript[18] and the HTML Document Object Model (DOM).[2,23] Thus, as the browser wars came to a close with the Web appearing to settle on HTML4, the Web community was already inventing a new highly interactive Web.

On the negative side, the dominance of a single browser during this period meant that all new behavior outside the scope of the HTML4 specification was implemented based on Internet Explorer; worse, that implementation in turn was a result of reverse engineering various features from the previously popular Netscape browser. This was particularly true with respect to interactive elements created through JavaScript and the HTML DOM, while incompatibilities between the CSS specifications and the predominant



Dreams 243.06260 and 243.06540 (page 58) were created by software artist Scott Draves through an evolutionary algorithm running on a worldwide cyborg mind consisting of 60,000 computers and people; ScottDraves.com.

implementation within Internet Explorer made it virtually impossible for Web developers to create content that would play consistently across multiple browsers.

Note that this period also saw significant movement away from Tim Berners-Lee's original vision of the Web. Web authors had started down the slippery slope of authoring for the dominant browser, thereby losing sight of the Web contract that had carefully arranged for Web content to be independent of the software that consumed it.

This breach might have seemed insignificant at the time, at least with respect to deploying Web content. The network effect that led to exponential growth in Web content during the 1990s meant that the Web had already taken off and that the slowdown in the network effect resulting from content coming to depend on a particular class of Web browsers did not immediately hamper growth. But the increasing interdependency between creator and consumer was not without cost; despite high hopes, the first round of the mobile Web fizzled in early 2000 partly because it was impossible to support mainstream Web content authored for a desktop browser on small devices like cellphones and PDAs. The problems that resulted from Web authors coding to a particular browser involved

IMAGE BY SCOTT DRAVES

additional hidden costs that became obvious by 2002 with the move from Web content to Web applications. By then, HTML, which began as a simple markup language, had evolved into three distinct layers:

HTML4. The markup tags and attributes used to serialize HTML documents;

CSS. The style rules used to define the presentation of a document; and

DOM. The programmatic interface to the parsed HTML document, used to manipulate HTML from within JavaScript.

The HTML4 specification went only so far as to define the tags and attributes used to serialize HTML documents. The programmatic API—the DOM—was defined within a separate specification (DOM 2) and never fully implemented by Internet Explorer. Making matters worse, CSS 2 was still under construction, and only parts of CSS 1 had been implemented in Internet Explorer.

Authoring Web content was now fraught with risks that would become apparent only over time. Authors could, with some trouble, create Web pages that appeared the same on the browsers of the time, at least with respect to visual presentation. However, when it came to styling the layout of a document via CSS or attaching interactive behavior via DOM calls, the shape of the underlying parsed representation proved far more significant than just visual appearance on a screen. As Web developers increasingly sought to add visual style and interactivity to their Web pages, they discovered incompatibilities:

Visual layout. To achieve a consistent visual layout, Web authors often had to resort to complex sets of HTML tables; and

Inconsistent DOM. Programmatic access of the HTML DOM immediately exposed the inconsistencies in the underlying representation in browsers, meaning that such programmatic calls had to be written for each browser and moved the Web further down the slippery slope toward browser-specific content.

But even as the hard-won Web looked like it would be lost to browser-specific Web content, the Web community was building on its earlier success of having a widely deployed universal browser that could be controlled (if poorly) via HTML and JavaScript. The Web had moved from mostly static content to documents with embedded pieces of interactivity. Web developers soon came to exploit the well-known fact of client-server computing: that even in a world of powerful servers, there are more compute cycles per user on a client than there are compute cycles on a server. Islands of interactivity implemented via JavaScript within HTML evolved into highly interactive user interfaces. The introduction of XML HTTP Request (XHR)[34] across the various browsers freed Web developers from having to do a complete page refresh when updating the user interface with new content. This set the stage for Asynchronous JavaScript and XML (AJax) applications.[19]

Discovering Web Applications

From late 1999 to early 2004, the line between content and applications on the Web was increasingly blurred. As examples, consider the following static (document-oriented) content and interactive (dynamic) applications:

Online news. News stories delivered in the form of articles enhanced with embedded video clips and interactive opinion polls; and

Shopping. Shopping catalogs with browsable items with interfaces, as well as real-time auction sites, enabling users to buy and sell.

This evolution from Web content to Web applications was accompanied by the progressive discovery of the Web programming model consisting of four Web components:

HTML. Markup elements and attributes for serializing Web pages;

CSS. Style rules for determining the final visual presentation;

DOM. Programmatic access to the parsed representation of the Web page; and

JavaScript. Open-ended scripting of the Web page through the DOM.

Here, the HTML, DOM, and JavaScript formed the underlying assembly language of Web applications. Though there is a clean architectural separation among content, visual-presentation, and interaction layers, note that this programming model was discovered through Darwinian evolution, not by design. A key consequence of this phenomenon is that content on the Web does not necessarily adhere to the stated separation. As a case in point, one still sees the HTML content layer sprinkled with presentational font tags, even though one would expect CSS to exclusively control the final presentation. Similarly, the content layer (HTML) is often sprinkled with script fragments embedded within the content, either in the form of inline script elements or as the value of HTML attributes (such as `href` and `onClick`).

Another key aspect of this phase of Web evolution was the creation of Web artifacts from Web components. Think of them as online information components built from Web-centric technologies—HTML, CSS, JavaScript—accessed over the network via URLs (see the figure here). A user-configurable component includes a customizable greeting, along with a photograph. Note that all of its aspects are constructed from five basic Web technologies:

Metadata. Component metadata encapsulated via XML;

Presentation. Content to be presented to the user encoded as plain HTML;

Style. The visual presentation of the HTML, controlled via CSS;

Interaction. Runtime behavior specified by attaching a JavaScript event handler (script) that computes the appropriate greeting based on the user's preferences and updates the HTML with the appropriate content; and

URLs. All resources used by the component—the photograph, the CSS style rules, the set of script functions—are fetched via URLs.

Toward Web 2.0

The first phase of the Web—Web 1.0—concluding in 2000 was characterized by bringing useful content online through the application of Web technologies to information (such as weather forecasts) in order to make them available on the Web to millions of potential users worldwide. A consequence was that a vast amount of useful content was now available—addressable via URLs and accessible over HTTP—with the requisite content being delivered via HTML, CSS, and JavaScript.

The next phase of this evolution—Web applications—saw the creation of useful end-user artifacts out of content already available on the Web. As an ex-

Web gadget built entirely from Web components—HTML, CSS, and JavaScript—displays a greeting and photograph both customizable by the user; the photograph is accessed via a URL.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
 <ModulePrefs title="..."/>
 <UserPref name="myname"/>
 <UserPref name="myphoto"/>
 <Content type="html"><![CDATA[
 <div id="content_div"></div>
 <style type="text/css">...</style>
 <script type="text/javascript">
 // Get userprefs
 var prefs = new gadgets.Prefs();
 function greet () {
    // Get current time
    var today = new Date();
    var time = today.getTime();
    var html = "";
    // Display appropriate greeting
    html += ...
    // Display photo if asked to
    if (prefs.getBool("photo") == true) {
    html += ...
    }
    element.innerHTML = html;
    }
    ...
    gadgets.util.registerOnLoadHandler(greet);
 </script>
 ]]>
</Content>
</Module>
```

RESTful Web APIs from major Web applications laid the software foundations for Web 2.0.

Service	Resource
Amazon	XForms Book
Google	Search Hubbell+Labrador
eBay	ASTER
Yahoo!	Browse Autos

ample, weather forecasts were available on Web 1.0, but XML HTTP Request and the ability to asynchronously refresh the content displayed in a Web page through JavaScript callbacks enabled Web sites to integrate weather forecasts into the context of user tasks (such as travel reservations). In addition to being built from Web technologies, a travel site that integrates a variety of information sources in order to deliver a complete task-oriented interface uses the same Web technologies when constructing its constituent components. Note that Web 2.0 is a result of applying Web technologies to the Web. Described differently, Web 2.0 is a consequence of Web(Web()); or writing $\mathcal{W}$ for the function Web, a more apt notation for Web 2.0 would be W^2.

Web As Platform

The notion of the Web as a new platform emerged in the late 1990s with the advent of sites providing a range of end-user services exclusively on the Web. Note that none of them had a parallel in the world of shrink-wrap software that had preceded the Web:

Portal. Yahoo! Web directory;
Shopping. Amazon online store;
Auction. eBay auction site; and
Search. Google search engine.

In addition to lacking a pre-Web equivalent, each of these services lived on the Web and, more important, exposed the services as simple URLs, an idea later known as REpresentational State Transfer, or (REST) ful, Web APIs.[16,17] All such services not only built themselves on the Web, they became an integral part of the Web in the sense that every Google search, auction item on eBay, and item for sale on Amazon were URL addressable (see the table here).

URL addressability is an essential feature of being on the Web. The URL addressability of the new services laid the foundation for Web 2.0, that is, the ability to build the next generation of solutions entirely from Web components. The mechanism of passing-in parameters via the URL defined lightweight Web APIs. Note that in contrast to all earlier software APIs, Web APIs defined in this manner led to loosely coupled systems. Web APIs like those in the table evolved informally and came to be recognized later as programming interfaces that could be used to build highly flexible distributed Web components.

That all of these services heralded publication of a new platform was reflected in the O'Reilly Hacks Series, including: *Google Hacks*[10]; *Amazon Hacks*[4]; *Yahoo! Hacks*[3]; and *eBay Hacks*.[22]

The Web had thus evolved from a Web of content to a Web of content embedded with the needed user-interaction elements. Content embedded with user interaction evolved into Web applications that could over time be composed exclusively from Web components. Being built this way and living exclusively on the Web, the new software artifacts came to form the building blocks for the next generation of the Web. Together, they define the Web as a platform with certain key characteristics:

Distributed. Web applications were distributed across the network; application logic and data resides on the network, with presentation augmented by the needed user interaction delivered to the browser;

Separable. The distributed nature of Web applications forced a cleaner separation between application logic and the user interface than in the previous generation of monolithic desktop software;

Universal. By delivering presentation and user interface to a Web browser, Web applications were more universally available than were their earlier counterparts; coding to the Web platform—or using HTML, JavaScript, and CSS[21]—enabled developers to create user interfaces that could be consistently accessed from a variety of platforms;

Zero install. With user-interface enhancements delivered via the network, users did not need to install Web applications; and

Web APIs. Web applications exposed simple URL-based APIs that evolved bottom-up that were easy to develop, document, and learn and quickly became a key enabler for Web 2.0.

Aggregations, projections, and mashups are all a direct consequence of the user's need to consume information in a form that is most suited to a given task.

User-Centric Access

As increasing amounts of information was moved onto the Web in the late 1990s, end users had a problem: To access all the information required for a given task, they needed to connect to myriad Web sites. This was true on the public Internet, as well as on corporate intranets. Moreover, the information being accessed had to be customized for the user's context (such as desktop or mobile access). The desire to deliver user-centric information access led to the binding of mobile user interfaces to Web content, another example of specialized browsing.[29,33]

A user interface designed for a large display is inappropriate for viewing on small-screen devices like cellphones and PDAs. The distributed nature of Web applications—and consequent separation of the user interface—enabled Web developers to bind specialized mobile user interfaces.

At the same time, the need to provide a single point of access to oft-used information led to portal sites that aggregated all the information onto a single Web page. In this context, the various items of information can be viewed as lightweight Web components. The environment in which these components are hosted (such as the software that generates and manages the Web page) can be viewed as a Web container. Thus, common actions (such as signing in) were refactored to be shared among the various Web applications hosted by the Web container, a piece of software managing the user's browsing context.

Web components hosted in this manner relied on the underlying Web APIs discussed earlier to retrieve and display content on behalf of the user. But as long as such aggregations were served from portal sites, users still needed to explicitly launch a Web browser in order to access their information. This turned out to be an inconvenience for frequently viewed information, motivating the move by Web developers toward Web gadgets, small pieces of Web-driven software that materialize on the user's desktop outside the Web browser. Such Web aggregation has moved over time from the server to the client where it materializes as widgets or gadgets.

Viewed this way, Web gadgets are specialized browsers. Rather than requiring the user to explicitly navigate to a Web site and drill through its various user-interface layers before arriving at the target screen, these gadgets automate away a large part of the user actions by directly embedding the final result into the user's Web environment. Finally, Web gadgets have escaped the confines of the Web browser to materialize directly on the user's desktop. Users no longer had to explicitly launch a Web browser to access the gadgets. However, the gadgets themselves continue to be built out of Web components. As an example, a typical iGoogle gadget consists of several components:

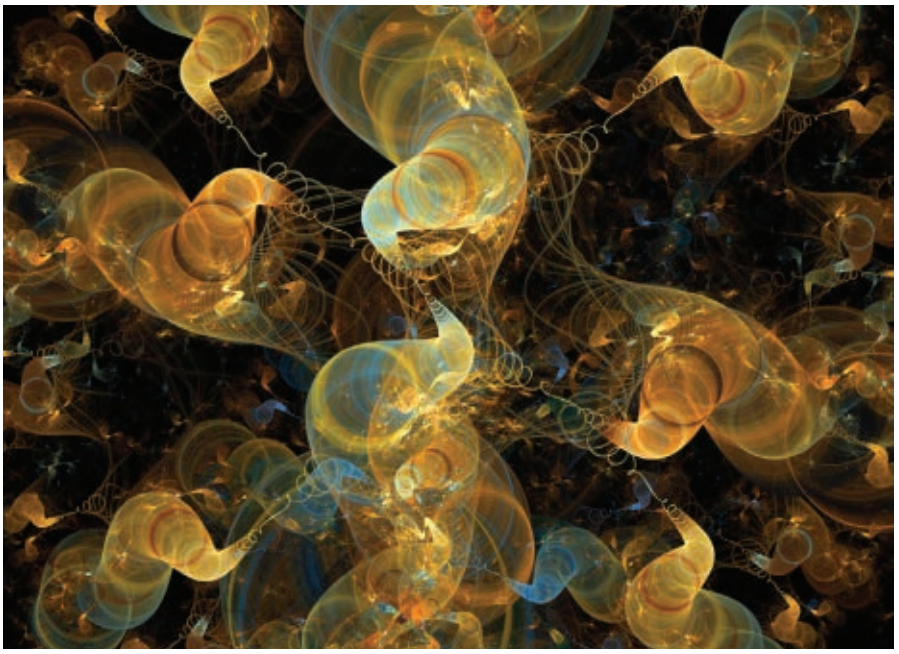
XML. A small XML file encapsulating metadata about the gadget;

HTML. The markup used to render the gadget;

CSS. Style rules to specify the final visual presentation; and

JavaScript. JavaScript functions used to retrieve and inject the relevant information into the HTML DOM before it is presented to the user.

Web gadgets relying on lightweight Web APIs, Rich Site Summaries (RSS) (letters also sometimes used to mean Really Simple Syndication), and Atom feeds[26] helped the move toward specialized browsing; retrieving information from a Web site did not always require a live human to directly interact with the user interface. Today, RSS and Atom feeds form the underpinnings of Web APIs for content retrieval. In the simplest cases, they enable



content sites to export a set of article titles and summaries. In more complex cases, such feeds are used in conjunction with newer protocols (such as the Atom Publishing Protocol[13]) layered on top of HTTP to expose rich programmatic access to Web applications. Together, these various feed-oriented APIs enable a variety of task-oriented Web tools ranging from bulk upload of data to custom information access. Note that this class of software services consists entirely of Web components.

Web gadgets thus provide specialized browsing functionality and are hosted in a variety of environments ranging from server-side containers to client-side user environments. In all cases, the hosting environment provides a number of services:

Back end. Access the Web to retrieve, filter, and format the requisite information;

Front end. Render the formatted information as HTML for realizing the final presentation and user interface;

Configuration. Provide the user interface affordances to allow users to customize the final experience by configuring the look and feel of the interface; such configuration includes adding, removing, expanding, or collapsing the gadget;

Preferences. Manage user preferences for gadgets within a container;

Single sign-on. Delegate common tasks (such as authentication) to the container, so users do not need to login to each Web application; and

Caching. Cache content to provide an optimized browsing experience.

The Web container thus provides the environment or evaluation context for Web widgets. I return to this pivotal role played by such container environments later when I address the evolving social Web.

A key aspect of all Web technologies is that the user has final control over visual presentation and user interaction. CSS emphasizes the C in Cascading by enabling users to cascade and consequently override the visual presentation chosen by the content creator. Similarly, scripting makes it possible for end users to drastically alter the interaction behavior of Web content. This flexibility was first leveraged in 1999 by Emacspeak[31]; the modules *websearch* and *url-templates* provided task-oriented Web wizards using REST APIs and XSLT Web transforms.[12] Later, similar functionality was brought to mainstream users by Greasemonkey,[27] a Firefox extension enabling them to attach arbitrary scripts to Web content. The success of Greasemonkey has been built upon by projects like Chickenfoot from MIT[25] and CoScriptor from IBM Research,[24] both providing higher-level user automation when working with Web interfaces. The ability to inject behavior into Web pages by scripting the HTML DOM was also successfully leveraged to create Google-AxsJAX,[11,30] a JavaScript library that helps developers across the Web enhance the usability of Web interfaces, with special focus on users with special needs (such as visual and hearing impairment).

Beyond Web 2.0

So here is where we stand:

- The Web, which began as a global hypertext system, has evolved into a distributed application platform delivering final-form visual presentation and user interaction;
- The separation between application logic and user interface enables late binding of the user interface,[14,15,32] promising the ability to avoid a one-size-fits-all user interface;
- More than URL-addressable content, the Web is a distributed collection of URL-addressable content and applications;
- It is now possible to create Web artifacts built entirely from Web components; and
- The underlying Web architecture ensures that when created to be URL-addressable, Web artifacts in turn become the building blocks for the next set of end-user Web solutions.

I described Web 2.0 earlier as the result of applying the Web function to itself, that is, Web 2.0 = Web^2 (). Let $\mathcal{W}$ denote the set of all URL-addressable information. Examining the properties of today's Web, we see the following additional properties with respect to $\mathcal{W}$:

Aggregation. New Web artifacts can be created by aggregating existing elements of the Web; when assigned a URL, such aggregations become elements of $\mathcal{W}$;

Projections. Information available on the Web can be filtered to suit the user's browsing context; such projections when made URL-addressable themselves become elements of $\mathcal{W}$; and

Cross-products. Discrete elements of $\mathcal{W}$ can be integrated into a single view to create Web mashups.

Note, too, that the notion of Web mashups can be generalized to cover cases where one brings together data from more than a pair of sites. Such cross-products are not limited to integrating data from multiple sources into a single view; instead, one can also integrate multiple views of the same piece of data (such as a visual representation that displays historical data both as a

IMAGE BY SCOTT DRAVES

table of numbers and as a histogram). Similarly, a multimodal view of a page, supporting both visual and spoken interaction, is also just one more type of view-to-view mashup. Bringing all this together, we can pose the question: What is the size of this Web to come? In theory, we can combine arbitrary subsets of W using the techniques I've outlined here. Each combination can in turn be deployed on the Web by making it URL-addressable and expressed mathematically as:

$$\binom{|W|}{0}+\binom{|W|}{1}+\binom{|W|}{2}+\ldots+\binom{|W|}{|W|}=2^{|W|}$$

User-Oriented Web: A Total Perspective

This number $2^{|W|}$ is extremely large and growing quickly as we build on the success of the Web; here, I denote this set 2^W. Fans of Douglas Adams's *Hitchhikers Guide To The Galaxy*[1] probably feel like they are now well entrapped within the total perspective vortex. But just as in the case of Zaphod Beeblebrox, the solution is not to focus on the totality of the Web but instead on the individual; 2^W exists for the user. As we move to a highly personalized social Web, each element of 2^W exists as it is perceived and used by a given user.

A significant portion of our social interaction increasingly happens via the Web. Note that a large portion of the impetus for the move from Web 1.0 to Web 2.0 and later to the predicted 2^W is due to user needs; aggregations, projections, and mashups are all a direct consequence of the user's need to consume information in a form that is most suited to a given task. Though the resulting set 2^W might be immense, most of these elements are relevant only when used by at least one user. Users do not use Web artifacts in a vacuum, but in a given environment or evaluation context provided by a given Web container.

Web content when combined is far more useful than its individual components. Likewise, Web applications used by collaborating users create a far richer experience than would be possible if they were used by users in isolation. Users typically converge on the use of such artifacts via popular Web containers, making the various APIs available by a given container a key distinguishing factor with respect to the types of interactions enabled within the environment. For example, OpenSocial from Google (code.google.com/apis/opensocial/), which describes itself as "many sites, one API," defines a set of APIs that can be implemented within a Web container. These APIs then expose a common set of services to gadgets being hosted within the container. Likewise, the Facebook platform provides an API for developing gadgets to be hosted in the Facebook container,[20] which can provide access to a user's contact list, enabling the various gadgets within it to provide an integrated end-user experience.

Conclusion

The Web has evolved from global hypertext system to distributed platform for end-user interaction. Users access it from a variety of devices and rely on late binding of the user interface to produce a user experience that is best suited to a given usage context. With data moving from individual devices to the Web cloud, users today have ubiquitous access to their data. The separation of the user interface from the data being presented enables them to determine how they interact with the data. With data and interaction both becoming URL-addressable, the Web is now evolving toward enabling users to come together to collaborate in ad-hoc groups that can be created and dismantled with minimal overhead. Thus, a movement that started with the creation of three simple building blocks—URL, HTTP, HTML—has evolved into the one platform that binds them all. C

References

1. Adams, D. *The Restaurant at the End of the Universe.* Ballantine Books, 1980.
2. Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Hors, A.L., Nicol, G., Robie, J., Sutor, R., Wilson, C. et al. *Document Object Model (DOM) Level 1 Specification.* W3C Recommendation, World Wide Web Consortium, 1998; www.w3.org/TR/REC-DOM-Level-1.
3. Bausch, P. *Yahoo! Hacks.* O'Reilly Media, Inc., Sebastopol, CA, 2005.
4. Bausch, P. *Amazon Hacks: 100 Industrial-Strength Tips & Tools.* O'Reilly Media, Inc., Sebastopol, CA, 2003.
5. Berners-Lee, T., Fielding, R., and Frystyk, H. *Hypertext Transfer Protocol 1.0*, 1996; www.w3.org/Protocols/Specs.html#HTTP10.
6. Berners-Lee, T. *Universal Resource Identifiers in WWW.* Internet Engineering Task Force, 1994; www.w3.org/Addressing/rfc1630.txt.
7. Berners-Lee, T., Masinter, L., and McCahill, M. *Uniform Resource Locators.* Internet Engineering Task Force Working Draft 21, 1994; www.ietf.org/rfc/rfc1738.txt.
8. Berners-Lee, T. and Connolly, D. *Hypertext Markup Language, Internet Working Draft 13*, 1993.
9. Bos, B., Lie, H.W., Lilley, C., and Jacobs, I. *Cascading Style Sheets, Level 2 CSS2 Specification.* W3C Recommendation, May 1998; www.w3.org/TR/REC-CSS2.
10. Calishain, T. and Dornfest, R. *Google Hacks: 100 Industrial-Strength Tips & Tools.* O'Reilly Media, Inc., Sebastopol, CA, 2003.
11. Chen, C.L. and Raman, T.V. AxsJAX: A talking translation bot using Google IM: Bringing Web-2.0 applications to life. In *Proceedings of the 2008 International Cross-Disciplinary Workshop on Web Accessibility*, 2008, 54–56.
12. Clark, J. et al. *XSL Transformations Version 1.0.* W3C Recommendation. World Wide Web Consortium, 1999.
13. de hOra, B. *The Atom Publishing Protocol.* 2006; bitworking.org/projects/atom/.
14. Dubinko, M. *XForms Essentials.* O'Reilly Media, Inc., Sebastopol, CA, 2003.
15. Dubinko, M., Klotz, L.L., Merrick, R., and Raman, T.V. *XForms 1.0, W3C Recommendation.* World Wide Web Consortium, Oct. 14, 2003; www.w3.org/TR/xforms.
16. Fielding, R.T. and Taylor, R.N. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology 2*, 2 (2002), 115–150.
17. Fielding, R.T. *Architectural Styles and the Design of Network-based Software Architectures, Chapter 5 Representational State Transfer.* Ph.D. Dissertation, University of California, Irvine, 2000.
18. Flanagan, D. and Novak, G.M. Java-Script: The definitive guide. *Computers in Physics 12*, 41 (1998).
19. Garrett, J.J. *Ajax: A New Approach to Web Applications.* White paper, Adaptive Path Inc., 2005.
20. Graham, W. *Facebook API Developers Guide.* firstPress, 2008.
21. Hickson, I. HTML 5 Working Draft. W3C Working Draft. World Wide Web Consortium, 2008; www.w3.org/TR/html5/.
22. Karp, D.A. *eBay Hacks.* O'Reilly Media, Inc., Sebastopol, CA, 2005.
23. Le Hors, A., Le Hegaret, P., Wood, L., Nicol, G., Robie, J., Champion, M., and Byrne, S. *Document Object Model Level 2 Core Specification.* W3C Recommendation. World Wide Web Consortium, 2000; www.w3.org/TR/DOM-Level-2-Core.
24. Leshed, G., Haber, E.M., Matthews, T., and Lau, T. CoScripter: Automating & sharing how-to knowledge in the enterprise. In *Proceedings of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems* (2008).
25. MIT CSAIL. Automation and customization of rendered Web pages. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (2005).
26. Nottingham, M. *Atom Syndication Format.* Internet RFC 4287 Internet Engineering Task Force, 2005.
27. Pilgrim, M. *Greasemonkey Hacks: Tips & Tools for Remixing the Web with Firefox.* O'Reilly Media, Inc. Sebastopol, CA, 2005.
28. Raggett, D., Le Hors, A., and Jacobs, I. *HTML 4.01 Specification.* W3C Recommendation REC-html401-19991224. World Wide Web Consortium, Dec. 1999; www.w3.org/TR/ html401.
29. Raman, T.V. Specialized browsers. Chapter 12 in *Web Accessibility*, S. Harper and Y. Yesilada, Eds. Springer, 2008; emacspeak.sf.net/raman/publications/.
30. Raman, T.V. Cloud computing and equal access for all. In *Proceedings of the 2008 International Cross-Disciplinary Workshop on Web Accessibility* (2008), 1–4.
31. Raman, T.V. Emacspeak: The complete audio desktop. In *Beautiful Code*. O'Reilly Media, Inc., Sebastopol, CA, 2007.
32. Raman, T.V. *Xforms: XML-Powered Web Forms.* Addison-Wesley Professional, 2003.
33. Raman, T.V. *Emacspeak: Toward the Speech-Enabled Semantic WWW.* 2000; emacspeak.sf.net/raman/.
34. van Kesteren, A. and Jackson, D. *The XML HTTP Request Object.* W3C Working Draft. World Wide Web Consortium, 2008; www.w3.org/TR/XMLHttpRequest/.
35. World Wide Web Consortium Technical Architecture Group. *Architecture of the World Wide Web.* W3C TAG Finding REC-webarch-20041215, 2004.
36. Yoffie, D.B. and Cusumano, M.A. Judo strategy: The competitive dynamics of Internet time. *Harvard Business Review 77*, 1 (1999), 70–81.

T.V. Raman (raman@google.com) is a research scientist at Google Research, Mountain View, CA.

DOI:10.1145/1461928.1461946

Research and education in compiler technology is more important than ever.

BY MARY HALL, DAVID PADUA, AND KESHAV PINGALI

Compiler Research: The Next 50 Years

WE PRESENT A perspective on the past contributions, current status, and future directions of compiler technology and make four main recommendations in support of a vibrant compiler field in the years to come. These recommendations were drawn from discussions among presenters and attendees at a U.S. National Science Foundation-sponsored Workshop on Future Directions for Compiler Research and Education in 2007. As 2007 was the 50th anniversary of IBM's release of the first optimizing compiler, it was a particularly appropriate year to take stock of the status of compiler technology and discuss its future over the next 50 years. Today, compilers and high-level languages are the foundation of the complex and ubiquitous software infrastructure that undergirds the global economy. The powerful and elegant technology in compilers has also been invaluable in other domains (such as hardware synthesis). It is no exaggeration to say that compilers and high-level languages are as central to the information age as semiconductor technology.

In the coming decade, 2010 to 2020, compiler research will play a critical role in addressing two of the major challenges facing the overall computer field:

Cost of programming multicore processors. While machine power will continue to grow impressively, increased parallelism, rather than clock rate, will be the driving force in computing in the foreseeable future. This ongoing shift toward parallel architectural paradigms is one of the greatest challenges for the microprocessor and software industries. In 2005, Justin Rattner, chief technology officer of Intel Corporation, said, "We are at the cusp of a transition to multicore, multithreaded architectures, and we still have not demonstrated the ease of programming the move will require..."[3]

Security and reliability of complex software systems. Software systems are increasingly complex, making the need to address defects and security attacks more urgent. The profound economic impact of program defects was discussed in a 2002 study commissioned by the U.S. Department of Commerce National Institute of Standards and Technology (NIST), concluding that program defects "are so prevalent and so detrimental that they cost the U.S. economy an estimated $59.5 billion annually, or about 0.6% of the gross domestic product." The 2005 U.S. President's Information Technology Advisory Committee (PITAC) report *Cyber Security: A Crisis of Prioritization* included secure software engineering and software assurance among its top 10 research priorities, concluding with: "Commonly used software engineering practices permit dangerous errors, such as improper handling of buffer overflows, which enable hundreds of attack programs to compromise millions of computers every year. In the future, the Nation [the U.S.] may face even more challenging problems as adversaries—both foreign and do-

mestic—become increasingly sophisticated in their ability to insert malicious code into critical software…"

Cultural Shift

To address these challenges, the compiler community must change its current research model, which emphasizes small-scale individual investigator activities on one-off infrastructures. Complete compiler infrastructures are just too complex to develop and maintain in the academic research environment. However, integration of new compiler research into established infrastructures is required to ensure the migration of research into practice. This conundrum can be solved only through a new partnership between academia and industry to produce shared open source infrastructure, representative benchmarks, and reproducible experiments. If successful, this new model will affect both commercial applications and scientific capabilities. Another 2005 PITAC report *Computational Science: Ensuring America's Competitiveness* highlighted the need for research in enabling software technologies, including programming models and their compilers, to maintain U.S. national competitiveness in computational science (see the sidebar "Agenda for the Compiler Community"). The PITAC report said, "Because the Nation's [the U.S.] research infrastructure has not kept pace with changing technologies, today's computational science ecosystem is unbalanced, with a software base that is inadequate for keeping pace with and supporting evolving hardware and application needs. By starving research in enabling software and applications, the imbalance forces researchers [in the U.S.] to build atop inadequate and crumbling foundations rather than on a modern, high-quality software base. The result is greatly diminished productivity for both researchers and computing systems."

Accomplishments

When the field of compiling began in the late 1950s, its focus was limited to the translation of high-level language programs into machine code and to the optimization of space and time requirements of programs. The field has since produced a vast body of knowledge about program analysis and transformations, automatic code generation, and runtime services. Compiler algorithms and techniques are now used to facilitate software and hardware development, improve application performance, and detect and prevent software defects and malware. The compiler field is increasingly intertwined with other disciplines, including computer architecture, programming languages, formal methods, software engineering, and computer security. Indeed, the term "compiler" has associations in the computer science community that are too narrow to reflect the current scope of the research in the area.

The most remarkable accomplishment by far of the compiler field is the widespread use of high-level languages. From banking and enterprise-management software to high-performance computing and the Web, most software today is written in high-level languages compiled either statically or dynamically. When object-oriented and data abstraction languages were first proposed back in the late 1960s and early 1970s, their potential for vastly improving programmer productivity was recognized despite serious doubts about whether they could be implemented efficiently. Static and dynamic optimizations invented by the compiler community for this purpose put these fears to rest. More recently, particularly with the introduction of Java in the mid-1990s, managed runtime systems, including garbage collection and just-in-time compilation, have improved programmer productivity by eliminating memory leaks.

Compiler algorithms for parsing, type checking and inference, dataflow analysis, loop transformations based on data-dependence analysis, register allocation based on graph coloring, and software pipelining are among the more elegant creations of computer science. They have profoundly affected the practice of computing because they are

ILLUSTRATION BY SIMON GEILFUS

incorporated into powerful yet widely used tools. The increasing sophistication of the algorithms is evident when today's algorithms are compared with those implemented in earlier compilers. Two examples illustrate these advances. Early compilers used ad hoc techniques to parse programs. Today's parsing techniques are based on formal languages and automata theory, enabling the systematic development of compiler front ends. Likewise, early work on restructuring compilers used ad hoc techniques for dependence analysis and loop transformation. Today, this aspect of compilers has been revolutionized by powerful algorithms based on integer linear programming.

Code optimizations are part of most commercial compilers, pursuing a range of objectives (such as avoiding redundant computations, allocating registers, enhancing locality, and taking advantage of instruction-level parallelism). Compiler optimization usually delivers a good level of performance, and, in some cases, the performance of compiler-generated code is close to the peak performance of the target machine. Achieving a similar result with manual tuning, especially for large codes, is extraordinarily difficult, expensive, and error prone. Self-tuning program generators for linear algebra and signal processing are particularly effective in this regard.

Tools for identifying program defects and security risks are increasingly popular and used regularly by the largest software developers. They are notably effective in identifying some of the most frequent bugs or defects (such as improper memory allocations and deallocations, race conditions, and buffer overruns). One indication of the increasing importance of program-analysis algorithms in reliability and security is the growth of the software tools industry that incorporate such algorithms.

One manifestation of the intense intellectual activity in compilers is that the main conferences in the area, including the ACM Symposium on Programming Language Design and Implementation (PLDI), ACM Symposium on Principles of Programming Languages (POPL), ACM Symposium on Principles and Practice of Parallel Programming (PPoPP), and ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA), are among the most influential, respected, and selective in computer science.[a] Another indication of the field's influence is that the phrases "programming languages" and "compilers" occur in the citations of no less than seven Turing award winners, including Peter Naur in 2005 and Fran Allen in 2006.

Compiler Challenges

The growing complexity of machines and software, introduction of multicores, and concern over security are among the more serious problems that must be addressed today. Here, we describe the role compiler technology plays in addressing them:

Program optimization. We live in the era of multicore processors; from now on, clock frequencies will rise slowly if at all, but the number of cores on processor chips is likely to double every couple of years. Therefore, by 2020, microprocessors are likely to have hundreds or even thousands of cores, heterogeneous and possibly specialized for different functionalities. Exploiting large-scale parallel hardware will be essential for improving an application's performance or its capabilities in terms of execution speed and power consumption. The challenge for compiler research is how to enable the exploitation of the power of the target machine, including its parallelism, without undue programmer effort.

David Kuck, an Intel Fellow, emphasized in a private communication the importance of compiler research for addressing the multicore challenge. He said that the challenge of optimal compilation lies in its combinatorial complexity. Languages expand as computer use reaches new application domains and new architectural features arise. Architectural complexity (uni- and multicore) grows to support performance, and compiler optimization must bridge this widening gap. Compiler fundamentals are well understood now, but where to apply what optimization has become increasingly difficult over the past few decades. Compilers today are set to operate with a fixed strategy (such as on a single function in a particular data context) but have trouble shifting gears when different

a An indication of the influence of compiler and programming language research is the citation rate rank of the field's major conferences relative to other computer science conferences and journals worldwide as reported by Citeseer (citeseer.ist.psu.edu/impact.html). Their rank on Citeseer as of December 2008 is PLDI (3rd), POPL (13th), PPoPP (14th), and OOPSLA (28th) out of a total of 1,221 computer science conferences and journals.

Collaboration, Research Challenges, Education

Agenda for the Compiler Community

The following agenda for the compiler community demands a broader collaboration between industry and academic institutions, as well as support from government funding agencies, to address the challenges discussed here.

Enablers to facilitate collaborative compiler research:

- **Open and extensible compiler infrastructure with state-of-the-art optimizations;**
- **Collections of benchmarks for evaluating advances in compilers and a strategy for keeping the benchmark collections up to date; and**
- **Methodology for measuring progress and reporting results that encourages all publications to include data in repositories to enable other researchers to reproduce the results.**

Research challenges in optimization:

- **Make parallel programming mainstream;**
- **Write compilers capable of self-improvement; and**
- **Develop performance models to support optimizations for parallel code.**

Research challenges in correctness:

- **Enable development of software as reliable as an airplane;**
- **Enable system software that is secure at all levels; and**
- **Verify the entire software stack.**

Enrich computer science education with compiler technology:

- **Expand compiler courses with examples from new problem domains (such as security); and**
- **Work with experts in other domains to incorporate compiler algorithms into their courses.**

code is encountered in a global context (such as in any whole application).

Kuck also said, "The best hope for the future is adaptive compilation that exploits equivalence classes based on 'codelet' syntax and optimization potential. This is a deep research topic, and details are only beginning to emerge. Success could lead to dramatic performance gains and compiler simplifications while embracing new language and architecture demands."

Make parallel programming mainstream. Although research in parallel programming began more than 30 years ago, parallel programming is the norm in only a few application areas (such as computational science and databases). These server-side applications deal mostly with structured data (such as arrays and relations), and computations can be done concurrently with little synchronization. In contrast, many client-side applications deal with unstructured data (such as sets and graphs) and require much-finer-grain cross-processor synchronization. Programmers have few tools for coding such applications at a high level of abstraction without explicit management of parallelism, locality, communication, load balancing, power and energy requirements, and other dimensions of optimization. Furthermore, as parallelism becomes ubiquitous, performance portability of programs across parallel platforms and processor generations will be essential for developing productive software.

Breakthroughs in compiler technology are essential for making parallel programming mainstream. They will require collaboration with other areas, including tighter integration of compilers with languages, libraries, and runtime environments, to make available the semantic information needed to optimize programs for parallel execution. The historical approach of "whole-program" analysis must be replaced with a hierarchical approach to software development and optimization in which code within a software layer is optimized without analyzing code in lower layers. This abstraction approach requires that each layer have a well-defined API with semantics that describe the information model or ontology of that layer (such as the Google map-reduce programming model). These semantics are used by the compiler to optimize software in higher layers. In the reverse direction, contextual information from higher layers can be used to specialize and optimize code at lower layers.

The most remarkable accomplishment by far of the compiler field is the widespread use of high-level languages.

Because guidance from the programmer is also necessary, interactive or semiautomatic compiler-based tools must also be developed. A noteworthy example of such an approach is a project that used race-detection software to interactively parallelize the Intel IA32 compiler.[1] Interactivity may require incorporation of compilers into integrated development environments, redesigning compiler optimizations to be less dependent on the order in which the optimizations are applied, and redesigning compiler algorithms and frameworks to make them more suitable for an interactive environment.

The advent of just-in-time compilation for languages (such as Java) blurs the distinction between compile time and runtime, opening up new opportunities for program optimization based on dynamically computed program values. As parallel client-side applications emerge, runtime dependence checking and optimization are likely to be essential for optimizing programs that manipulate dynamic data structures (such as graphs).

Develop a rigorous approach to architecture-specific optimization. Compiler front ends benefited greatly from development in the 1960s and 1970s of a systematic theory of lexical analysis and parsing based on automata theory. However, as mentioned in the private communication by David Kuck quoted earlier, there is no systematic approach for performing architecture-specific program optimization, thus hampering construction of parallelizing and optimizing compilers. Developing effective overall optimization strategies requires programmers be able to deal with a vast number of interacting transformations, nonlinear objective functions, and performance prediction, particularly if performance depends on input data. The program optimization challenge is certainly difficult but not insurmountable.

Recent research at a variety of institutions, including Carnegie Mellon University, the University of California, Berkeley, the University of Illinois, MIT, and University of Tennessee, has dem-

onstrated the potential of offline empirical search to tune the performance of application code to a specific architecture. Often called "autotuning," this approach has produced results competitive with hand tuning in such well-studied domains as linear algebra and signal processing. The basic approach is that either the application programmer expresses or the compiler derives a well-defined search space of alternative implementations for a program that is then explored systematically by compiler and runtime tools so the optimization process is able to achieve results comparable to hand tuning. The challenge is to extend the approach to parallel systems and multicore processors, as well as to integrate the technology into a coherent, easy-to-use system that applies to a large number of complex applications.

Language and compiler technology supporting autotuning will greatly facilitate the construction of libraries implementing numerical and symbolic algorithms for a variety of domains, building on examples from self-tuning linear algebra and signal-processing libraries. In addition to encapsulating efficient algorithms that can be implemented by only a handful of compiler experts and used by a vast number of nonexpert programmers, these libraries can contribute to the design of future high-level languages and abstractions by exposing new and interesting patterns for expressing computation. Furthermore, describing the characteristics of these libraries in a machine-usable format, such that compilers understand the semantic properties and performance characteristics of the library routines, will enable the implementation of interactive tools that analyze and make recommendations about the incorporation of library routines in their codes.

An even more significant challenge is for compiler researchers to extend this approach to online tuning. Evaluating the performance of different versions of a program by running them directly on the native machine is unlikely to scale to large numbers of cores or to large programs, even in an offline setting. One strategy is for compiler researchers to develop tractable machine and program abstractions to permit efficient evaluation of program alternatives, since what's needed is only the relative performance of program alternatives, rather than their absolute performance. The literature has proposed a variety of parallel computing models (such as the Parallel Random Access Machine, or PRAM, model for analyzing parallel algorithms and the LogP model for communication), but they are too abstract to serve as useful targets for performance optimization by compilers. However, through interactions with the theory and architecture communities, new models that more accurately capture today's multicore platforms should be developed. Such models could also be the foundation of a systematic theory of program parallelization and optimization for performance, power, throughput, and other criteria.

Breakthroughs in compiler technology are essential to making parallel programming mainstream.

Correctness and security. The ability to detect coding defects has always been an important mission for compilers. In facilitating programming, high-level languages both simplified the representation of computations and helped identify common program errors, including undeclared variable uses and a range of semantic errors pinpointed through increasingly sophisticated type checking. Much more can and must be done with program analysis to help avoid incorrect results and security vulnerabilities.

Regarding software security, Steve Lipner, senior director of security engineering strategy in Microsoft's Trustworthy Computing Group, said in a private communication: "With the beginning of the Trustworthy Computing initiative in 2002, Microsoft began to place heavy emphasis on improving the security of its software. Program-analysis tools have been key to the successes of these efforts, allowing our engineers to detect and remove security vulnerabilities before products are released. Today, Microsoft's engineering practices for security are formalized in the Security Development Lifecycle, or SDL, which mandates application of program-analysis tools and security-enhancing options. These tools and compiler options are the product of many years of research in program analysis and compilers, which has proven invaluable in addressing the difficult security challenges the industry faces. Microsoft's security teams eagerly look forward to the fruits of continued research in compiler technology and associated improvements in the effective-

ness of the tools that we use to make our products more secure."

Tools for program correctness and security must avoid wasting programmer time with false positives without sacrificing reliability or security and must prioritize, rank, and display the results of the analyses. Furthermore, they cannot negatively affect program performance or ease of use. As with other challenges, these issues are best addressed through a tighter integration of compilers, languages, libraries, and runtime systems. In particular, we anticipate an important role for specialized program-analysis systems that involve techniques relevant to each particular problem domain.

The foremost challenge in this area targets what traditionally is called debugging. The goal is to develop engineering techniques to detect and avoid program defects. The second challenge targets security risks, aiming to develop strategies to detect vulnerabilities to external attacks. The final challenge is to develop automatic program-verification techniques.

Enable development of software as reliable as an airplane. Improving the quality of software by reducing the number of program defects drives much research in computer science and has a profound economic influence on the overall U.S. national economy as indicated by the NIST report mentioned earlier. Compiler technology in the form of static and dynamic program analysis has proved useful in the identification of complex errors, but much remains to be done. Extending this work demands new program-analysis strategies to improve software construction, maintenance, and evolution techniques, bringing the programming process to conform to the highest engineering standards (such as those in automotive, aeronautical, and electronic engineering).

An effective strategy would likely involve analysis techniques, new language features for productivity and reliability, and new software-development paradigms. Nevertheless, at the core of these tools and strategies are advanced compiler and program-analysis techniques that guarantee consistent results while maintaining the overall quality of the code being generated, where metrics for quality might include execution time, power consumption, and code size. Beyond producing more reliable programs, the tools resulting from this research will make the profession of programming more rewarding by enabling developers and testers alike to focus on the more creative aspects of their work.

Enable system software that is secure at all levels. As with software reliability, sophisticated program-analysis and transformation techniques have been applied in recent years to the detection and prevention of software vulnerabilities (such as buffer overflows and dangling pointers) that arise from coding defects. There is some overlap between detection and prevention of software vulnerabilities and the previous challenge of software reliability in that any vulnerability can be considered a program defect. However, these challenges differ in that security strategies must account for the possibility of external attacks in the design of analyses and transformations. Thus, certain techniques (such as system call authentication and protection against SQL injection) are unique to the challenge of ensuring computer security.

Compilers play a critical role in enhancing computer security by reducing the occurrence of vulnerabilities due to coding errors and by providing programmers with tools that automate their identification and prevention. Programming in languages that enforce a strong type discipline is perhaps the most useful risk-reduction strategy. Functional language programs have more transparent semantics than imperative language programs, so language researchers have argued that the best solution to reducing software vulnerabilities is to program in functional languages, possibly extended with transactions for handling mutable state.

Enable automatic verification of the complete software stack. Formally proving that a program conforms to a given specification (program verification) is a powerful strategy for completely avoiding software defects. A manifestation is the longstanding recognition that program verification is one of the great challenges of computer science. More than 40 years ago, John McCarthy, then a professor of computer science at Stanford University, wrote: "Instead of debugging a program, one should prove that it meets its specifications, and this proof should be checked by a computer program."[2]

Although program verification is not traditionally considered a compiler challenge, we include it here due to its potential as a formal solution to the two previous challenges: the interplay between program analysis and automatic verification and growing interest in the verification of compilers.

Verifying compiler code and algorithms would be a good first step toward addressing this challenge for two reasons: First, compilers contain specifications of their own correctness, thus providing clear requirements for the verification process. And second, the verification of the code generated by compilers is a necessary aspect of the verification of software in general. Recent advances in compiler verification anticipate a future when it will be possible to rely on formal and mechanical reasoning at the source level. The ultimate goal is to prove that the compiler is extensionally correct (input-output preserving) and respects time, space, and power constraints.

Although powerful and effective verification tools would make a tremendous contribution to computing practice, the importance of program verification goes beyond its use in improving software quality. As an example of a formal reasoning system, program verification is intellectually important in and of itself. It has been argued by researchers in machine learning that program verification is an ideal subject for the development of the first advanced reasoning system. After all, programming is a subject that computer scientists who study reasoning systems really understand.

Recommendations

To address these challenges, the compiler community needs a vibrant research program with participation by industry, national labs, and universities worldwide. Advances in compiler technology will require the creativity, enthusiasm, and energy of individual researchers, but given the complexity of compiler technology and software systems, long-term projects led by compiler specialists from industry and research institutions is necessary for success. Therefore, we offer four main

Tim Burrell of Microsoft's Secure Windows Initiative describing the Phoenix compiler and automated vulnerability finding at the EUSecWest conference, May 2008, London, U.K.

recommendations:

Enable the creation of compiler research centers. There are few large compiler research groups anywhere in the world today. At universities, such groups typically consist of a senior researcher and a few students, and their projects tend to be short term, usually only as long as a Ph.D. project. Meanwhile, the few industrial groups studying advanced compiler technology tend to focus on near-term solutions, even as the compilers, the programs they translate and analyze, and the target execution environments have increased in complexity. The result is that too much of today's compiler research focuses on narrow problems, ignoring the opportunity to develop revolutionary strategies requiring long-term commitment for their development and evaluation.

In those areas of compiler research seeing diminishing returns today from incremental approaches (such as program analysis and optimization), researchers must attempt radical new solutions that are likely to be lengthy and involved. The compiler research community (university and industry) must work together to develop a few large projects or centers where long-term research projects with the support of a stable staff are carried out. Industry and funding agencies must work together to stimulate and create opportunities for the initiation of these centers. Joint industry/government funding must support the ongoing evolution and maintenance of the software.

Create significant university/industry/government partnerships for the development of infrastructure and the gathering of benchmarks while funding individual projects at universities and other research centers. Implementation and experimentation are central to the compiler area. New techniques and tools, as well as new implementations of known approaches, can be meaningfully evaluated only after they are incorporated into industrial-strength compiler infrastructures with state-of-the-art optimizations and code-generation strategies. The absence of widely accepted compiler research platforms has hindered research efforts in compiler technology, design of new programming languages, and development of optimizations for new machine architectures. The development of complete compilers requires an immense effort typically beyond the ability of a single research group. Source code is now available for the GNU compiler and for other compilers developed by industry (such as IBM's Java Jikes compiler, Intel's Open Research Compiler, and Microsoft's Phoenix framework). They may yet evolve into the desired infrastructure, but none currently meets all the needs of the community.

Experimental studies require benchmarks that are representative of the most important applications at the time of the evaluation, meaning the process of gathering representative programs is a permanent process. Numerous efforts have sought to gather benchmarks, but the collections tend to be limited and are often complicated by doubts as to how representative they truly are. A serious difficulty is that many widely used programs are proprietary. In domains where open source applications might represent proprietary software, it would suffice for the purpose of evaluating compilers to make use of open source versions that outperform their proprietary counterparts; representative input data sets must accompany any set of benchmarks. Also desirable is a description of the algorithms and data structures to enable research in new programming languages and extensions that may be better suited for expressing the computation.

Federal agencies and industry must collaborate to establish the necessary funding opportunities and incentives that will move compiler specialists at universities and industry to address the research infrastructure and benchmark challenges. Funding opportunities should also be made available for smaller academic studies and development efforts. The computer science community has achieved notable success in developing novel compiler infrastructures, including: compiler front-end development; program-analysis frameworks for research in compilers; verification tools, security applications, and software-engineering tools; and virtual-machine frameworks for both mainstream and special-purpose languages. Even if the GNU and industry-developed compilers were a useful infrastructure for research, development of new robust, easy-to-use infrastructures by the researchers who need them are critical for future advances. Not only is it important to support such research projects, the research community must recognize their academic merit.

Develop methodologies and repositories that enable the comparison of methods and reproducibility of results. The reproducibility of results is critical for comparing the strategies, machines, languages, and problem domains. Too much of the information available to-

PHOTOGRAPH BY RYO HIROSAWA

day is anecdotal. In many disciplines, reviewers and peer scientists expect papers to include sufficient information so other groups are able to reproduce the results being obtained, but papers do not adequately capture compiler experiments where numerous implementation details determine the final outcome. With the help of open source software, the Web can be used to publish the software and data used in the evaluations being reported. Major conferences and organizations (such as ACM) must provide mechanisms for publishing software, inputs, and experimental data as metadata for the publications that report these experiments. Such repositories are useful for measuring progress and could also serve as a resource to those interested in the history of technology.

Develop curriculum recommendations on compiler technology. Compiler technology is complicated, and advances in the discipline require bright researchers and practitioners. Meanwhile, computer science has grown as a discipline with numerous exciting areas of research and studies to pursue. The compiler community must convey the importance and intellectual beauty of the discipline to each generation of students. Compiler courses must clearly demonstrate to students the extraordinary importance, range of applicability, and internal elegance of what is one of the most fundamental enabling technologies of computer science.

Whereas compiler technology used to be a core course in most undergraduate programs, many institutions now offer their compiler course as an optional upper-level course for computer science and computer engineering students and often include interesting projects that deliver a capstone experience. A good upper-level compiler course combines data structures, algorithms, and tools for students as they build a large piece of software that performs an interesting and practical function. However, these courses are considered difficult by both faculty and students, and students often have other interesting choices. Thus, fewer students are exposed to the foundational ideas in compilers or to compilers as a potential area only for graduate study.

Compiler algorithms are of tremendous educational value for anyone interested in compiler implementation and machine design. Knowledge of the power and limitations of compiler algorithms is valuable to all users of compilers, debuggers, and any tool built using compiler algorithms that encapsulate many, if not most, of the important program-analysis and transformation strategies necessary for performance and correctness. Therefore, learning about compiler algorithms leads to learning about program optimization and typical programming errors in a deep and rigorous manner. For these reasons, programmers with a solid background in compilers tend to excel in their profession.

One approach to promoting knowledge of compiler algorithms involves discussion of specific compiler algorithms throughout the computer science curriculum—in automata theory, programming languages, computer architecture, algorithms, parallel programming, and software engineering. The main challenge is to define the key compiler concepts that all computer science majors must know and to suggest the content that should be included in core computer science courses.

A second complementary approach is to develop advanced courses focusing on compiler-based analyses for software engineering, scientific computing, and security. They could assume the basic concepts taught in core courses and move quickly into new material (such as virus detection based on compiler technology). The challenge is how to design courses that train students in advanced compiler techniques and apply them to areas that are interesting and relevant (such as software reliability and software engineering). They may not be called "compiler courses" but labeled in ways that reflect a particular application area (such as computer security, verification tools, program-understanding tools) or perhaps something more general like program analysis and manipulation.

Conclusion

Although the compiler field has transformed the landscape of computing, important compilation problems remain, even as new challenges (such as multicore programming) have appeared. The unsolved compiler challenges (such as how to raise the abstraction level of parallel programming, develop secure and robust software, and verify the entire software stack) are of great practical importance and rank among the most intellectually challenging problems in computer science today.

To address them, the compiler field must develop the technologies that enable more of the progress the field has experienced over the past 50 years. Computer science educators must attract some of the brightest students to the compiler field by showing them its deep intellectual foundations, highlighting the broad applicability of compiler technology to many areas of computer science. Some challenges facing the field (such as the lack of flexible and powerful compiler infrastructures) can be solved only through communitywide effort. Funding agencies and industry must be made aware of the importance and complexity of the challenges and willing to invest long-term financial and human resources toward finding solutions. C

References

1. Kirkegaard, K.J., Haghighat, M.R., Narayanaswamy, R., Shankar, B., Faiman, N, and Sehr, D.C. Methodology, tools, and techniques to parallelize large-scale applications: A case study. *Intel Technology Journal 11*, 4 (Nov. 2007).
2. McCarthy, J. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*, P. Braffort and D. Hirschberg, Eds. North-Holland Publishing Company, Amsterdam, The Netherlands, 1963, 33-70.
3. Merritt, R. Computer R&D rocks on. *EE Times* (Nov. 21, 2005); www.eetimes.com/showArticle.jhtml?articleID=174400350.

Acknowledgment

We thank Vikram Adve, Calin Cascaval, Susan Graham, Jim Larus, Wei Li, Greg Morrisett, and David Sehr for their many valuable and thoughtful suggestions. Special thanks to Laurie Hendren for organizing the discussion on education and preparing the text of the recommendation concerning curriculum recommendations on compiler technology. We gratefully acknowledge the support of the U.S. National Science Foundation under Award No. 0605116. The opinions and recommendations are those of the authors and do not necessarily reflect the views of the National Science Foundation. We also acknowledge all workshop participants whose insight and enthusiasm made this article possible, V. Adve, A. Adl-Tabatabatai, S. Amarasinghe, A. Appel, D. Callahan, C. Cascaval, K. Cooper, A. Chtchelkanova, F. Darema, J. Davidson, W. Harrod, J. Hiller, L. Hendren, D. Kuck, M. Lam, J. Larus, W. Li, K. McKinley, G. Morrisett, T. Pinkston, V. Sarkar, D. Sehr, K. Stoodley, D. Tarditi, R. Tatge, and K. Yelick.

Mary Hall (mhall@cs.utah.edu) is an associate professor in the School of Computing at the University of Utah, Salt Lake City, UT.

David Padua (padua@illinois.edu) is the Donald Biggar Willett Professor of Computer Science at the University of Illinois at Urbana-Champaign.

Keshav Pingali (pingali@cs.utexas.edu) is the W.A. "Tex" Moncrief Chair of Grid and Distributed Computing Professor in the Department of Computer Sciences at the University of Texas, Austin, and a professor in the Institute for Computational Engineering and Sciences also at the University of Texas, Austin.

DOI:10.1145/1461928.1461947

Inspiring, recruiting, and retaining women for a career in computing remains a challenge.

BY MARIA KLAWE, TELLE WHITNEY, AND CAROLINE SIMARD

Women in Computing—Take 2

"WOMEN IN COMPUTING: Where Are We Now?"—an article by Maria Klawe and Nancy Leveson in the January 1995 issue of *Communications*—addressed women's representation at the time, as undergraduate and graduate students and in the work force, in computing fields. That article, part of the issue's special section on Women and Computing, described successful activities and offered recommendations for future programs.

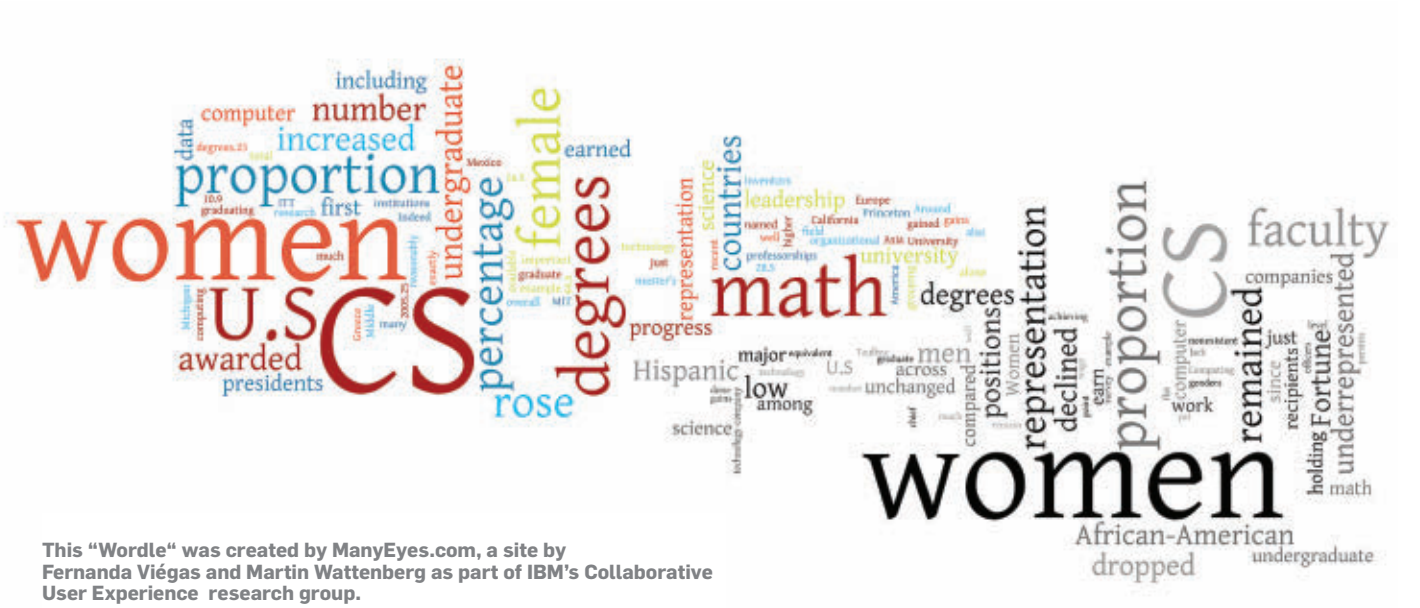
In this article, 14 years later, we assess the changes that have since occurred, including both positive and negative trends; we present strategies shown to be successful for the recruitment, retention, and advancement of women in computing; and we explore promising new initiatives for further increasing women's participation. While the 1995 article focused on the U.S. and Canada, as does the present one, we now also include data from other parts of the world.

Why should computing professionals be concerned about women and other groups underrepresented in our field? In large part, out of self-interest. Diversity often leads to enhanced abilities to perform tasks, greater creativity, and better decisions and outcomes.[17] Sadly, bias and stereotyping—often unconscious, but nevertheless pervasive—continue to affect the gender and ethnic composition of our talent pool and thus limit the possibilities of technological innovation around the world. Meanwhile, demand for computer scientists and computer engineers in the U.S. is expected to grow 37% between 2006 and 2016,[4] despite the overall economy's present travails. Clearly, society requires the contributions of women as well as men to computing.

On the Plus Side

Around the world, women have made some progress in the field of computing over the past decade. Women now play a heightened role in technology leadership, and they have gained representation at many important points in organizational hierarchies.

- The number of women earning U.S. undergraduate computer science (CS) degrees increased from 7,063 in 1995 to 11,235 in 2005.[25]
- Some countries are making gains in the numbers of women majoring in math or CS, but because data is often unavailable for computer science alone, related percentages are not exactly comparable to U.S. figures. Indeed, the percentage of U.S. female bachelor's degree recipients in math is much higher than that of CS—44.6% versus 22.2%.[25] Thus grouping math with CS may be masking lower participation in CS.
- In Asia (including only those countries for which data is available), women earned 43% of first university degrees in math and CS in 2004.[23] Women's representation in technical fields is growing in India—the percentage of female engineers graduating from ITT Bombay has grown from 1.8% in 1972 to 8% in 2005. In the Middle East, women earned 43% of first-time math and CS degrees.[23] In



This "Wordle" was created by ManyEyes.com, a site by Fernanda Viégas and Martin Wattenberg as part of IBM's Collaborative User Experience research group.

Western Europe, while the overall percentage of math and CS undergraduate degrees going to women is just 30%, some countries have been doing significantly better—Portugal was at 41% in 2004, Finland 42%, Greece 40%, and Italy 43%. In North America, Mexico also fares reasonably well, with 38% of math and CS undergraduate degrees awarded to women.[23]

▸ The total number of female CS graduate students in the U.S. grew from 9,881 in 1997 to 12,061 in 2005. The proportion of women awarded CS master's degrees rose from 26.4% in 1995 to 28.5% in 2005, and the proportion of women awarded CS doctoral degrees rose from 16.5% in 1997 to 19.8% in 2005,[25] pointing to some graduate-level progress.

▸ The proportion of newly hired women in U.S. and Canadian CS faculty increased from 18% in 1995 to 24% in 2006–2007.[6]

▸ The proportion of women in full CS professorships more than doubled between 1995 and 2007, from 5% to 10.9%.[6]

▸ The number of women in significant academic leadership positions has increased. For example, the proportion of female university presidents in the U.S. rose from 18% in 1995[31] to 23% in 2007.[1] In recent years, some high-profile research institutions—including Brown, Harvard, Michigan, MIT, Princeton, Penn, RPI, and several University of California campuses—named their first woman presidents.

▸ The percentage of U.S. information-technology patents obtained by female inventors rose from 4.4% in 1995 to 6.1% in 2005.[22]

The Bad News

The gains listed here, while encouraging, stop short of achieving equal representation and point to the fact that much work has yet to be done.

The proportion of undergraduate CS degrees received by women has declined sharply—from 37% in 1985 to 22% in 2005.[25] In research-intensive CS departments that participate in the annual Taulbee Survey conducted by the Computing Research Association (CRA), the number dropped from 19% in 2001 to 11.8% in 2006–2007.[6]

▸ Interest in CS as a major is at an all-time low both for men and women. In a 2007 teacher survey, a lack of student interest at the high-school level was cited as the number-one challenge.[5] Intention of women freshmen to major in computer science dropped from 2.8% in 1985 to 1.3% in 1995 and to 0.4% in 2006.[23, 25]

▸ Since 1995, the representation of African-American and Hispanic women among CS degree recipients has remained flat.[6] Across genders, the proportion of African-American Ph.D. recipients in the United States and Canada has remained unchanged at 1–2% since 1995, and Hispanic representation has dropped from 3% to 2%.[23]

▸ The proportion of female CS graduate students in the U.S. remained flat at 27% from 1997 to 2004 and declined to 25% in 2005.[23] Similarly in the European Union, the proportion of women earning math and CS doctorates has stood at 24%.[31]

> "My slogan is: Computing is too important to be left to men."
>
> KAREN SPARCK-JONES: PIONEER IN INFORMATION RETRIEVAL AND NATURAL LANGUAGE PROCESSING. 1935–2007

▸ Women in CS faculty positions at U.S. four-year institutions remain underrepresented, at just 15.8% of all faculty and 11% of full professors.[6] Ethnic minority women are doubly underrepresented on faculties, with Asian and African-American women holding just 3% of faculty positions. Hispanic and Native American women are virtually nonexistent among CS faculty.[25] Disparity in faculty salaries across all disciplines has remained unchanged since the 1970s—women faculty earn 81% of men's salary for equivalent

Sally Ride Science, named for the former astronaut, holds dozens of street fairs each year.

qualifications.[34]

▸ The proportion of women employed in math and CS occupations in the work force declined from 33% in 1984 to 27% in 2004.[25]

▸ The proportion of technology-industry women in top leadership positions is quite low—for example, only 5% of chief technology officers in Fortune 100 IT companies are women. Their representation on technology-company boards remains low as well, with 13% of board seats of Fortune 100 IT companies going to women, compared to 17% for Fortune 100 organizations across sectors.[21]

▸ The wage differential between men and women holding computer science degrees persists. Women with undergraduate CS degrees earn a median of $44K compared to $46K for men and $40K for underrepresented minorities.[25]

Taking Action: Issues and Exemplary Initiatives

Many initiatives are currently under way to counter such negative trends, and they have shown promise in helping to turn the tide. While it is not possible to review all such efforts in one article, we do highlight some encouraging programs at the K–12 level, in academia (undergraduate, graduate, and faculty levels), and in industry.

K–12: Appealing to Girls and Their Influencers. It is widely recognized that declining interest in technical disciplines among female students starts at a young age. Therefore early-intervention efforts are important to ensure future increases in representation.

Successful approaches at the K–12 level include:

▸ Expose girls to positive role models in the technology sphere, given that the absence of such models has proven to be a deterrent.

▸ Dispel computing-career myths and stereotypes; for example, the notion that computing is a "white male profession" discourages girls and minorities from entering the field.[2] The Image of Computing Task Force (www.imageofcomputing.com), comprised of global technology companies, professional associations, nonprofit organizations, and others, focuses its efforts on creating and disseminating positive images of computing designed to appeal to girls.

▸ Provide accurate information to key influencers of girls. Because parents and teachers with unconscious biases will subtly discourage girls from pursuing computer-related activities,[16] providing these influencers with information and resources is vital not only to igniting their daughters' and students' interest in technology at a young age but also to retaining it. Resources include the Girls Scouts' *Girls Go Tech* initiative booklet "It's Her Future." One educational program that touches on multiple audiences (girls, parents, and teachers) is Computer Mania Day—hosted by the Center for Women and Information Technology at the University of Maryland, Baltimore County—during which participants learn about pertinent issues and explore technology career for girls. This effort helps to create or strengthen positive attitudes about women's involvement in technology.[20]

▸ Provide girls with age-appropriate, hands-on technology activities; examples can be found at the *Girls Go*

Tech (www.girlsgotech.org) and Sally Ride Science (sallyridescience.com) Web sites.

▸ Enroll girls in summer computer programs in which they have immersive experiences with technology.

▸ Motivate girls and women through the potential social impact of technology.[29] Several excellent programs are built around socially relevant themes and involve teamwork, collaboration, and hands-on learning—pedagogical approaches shown to be highly effective for girls.[18] The Edge Summer Engineering Workshop for High School Girls, offered yearly by Union College, is a two-week summer residential workshop devoted to this purpose (antipasto.union.edu/edge/). Similarly, Purdue University's Engineering Projects In Community Service (EPICS) program, launched in 1995, is a learning approach in which undergraduate teams come together to apply technology solutions to an identified community problem. While EPICS does not specifically target women, it has significantly enhanced women's participation in computer science and engineering.[7] Encouraged by this result, EPICS launched a high-school summer program, based on the same model.

▸ Engage students and faculty of university CS departments to work with local middle schools and high schools while encouraging companies to implement their own outreach programs of this type.

Academia: Attracting and Retaining Students and Faculty. Over more than a decade, a host of initiatives has evolved to increase and sustain the participation of women, at all levels of academia, in computing. Many of these programs are projects of organizations specifically devoted to this purpose—for example, ACM's Committee on Women in Computing (ACM-W),[a] the Anita Borg Institute for Women and Technology (ABI), the CRA Committee on the Status of Women in Computing Research (CRA-W), MentorNet, and the National Center for Women & Information Technology (NCWIT). Readers are encouraged to visit the organizations' Web sites for more information.

a For more information on ACM's efforts to raise the profile and status of women in computing, see http://women.acm.org.

Other such programs have been initiated by funding agencies. These initiatives include the Increasing the Participation and Advancement of Women in Academic Science and Engineering Careers (ADVANCE) grants and the Broadening Participation in Computing (BPC) grants from the National Science Foundation (NSF), as well as Canada's NSERC-Industry Chairs for Women in Science and Engineering. (NSERC is the Natural Sciences and Engineering Research Council of Canada.) Still other programs, such as Google's Anita Borg scholarships and Microsoft's New Faculty Fellowships, are supported by industry.

"Today's computing is not your father's computing. Interaction design, empirical studies of user experience, project management, understanding social impacts of technology, and much more are new faces of academic computing. Check them out."

BONNIE A. NARDI: PROFESSOR, DONALD BREN SCHOOL OF INFORMATION AND COMPUTER SCIENCE, UNIVERSITY OF CALIFORNIA, IRVINE.

At the *undergraduate level*, three approaches have been the most successful:

▸ Redesign "Introduction to CS" courses to emphasize applications in areas of interest to females. Excellent examples of such redesigned courses include those at Georgia Tech and the University of British Columbia (UBC), which respectively emphasize applications in digital media and in psychology, fine arts, and biology.[27] Harvey Mudd College and Princeton each redesigned their introductory CS course to focus on science applications.[9]

▸ Require students to do assignments in their introductory CS course using "pair programming," which provides benefits (demonstrated by research conducted at UC, Santa Cruz[19]) for female and male students alike. A result of this approach is that more women are likely to complete the course, obtain a higher grade, take another CS course, and do better in the second CS course. Thus a number of institutions now impose a pair-programming requirement for their introductory CS classes.

▸ Make a computing-related course a requirement, or a highly recommended option, for all students in majors that have many females (arts and education, for example). While introducing a new general-curriculum requirement for a wide range of undergraduate majors is often politically difficult, Arizona State University has managed to do it. Otherwise, as demonstrated at UBC, a simple statement in the undergraduate handbook that "the Dean recommends that all [such] majors take at least one computer science course" can dramatically increase the number of female students taking CS courses.

Some of the successful approaches for attracting and retaining more female *computer majors* include:

▸ Create or publicize majors that combine computer science with another area. Examples include media computation at Georgia Tech; majors at Cornell cosponsored by the Faculty of Computing and Information Science and either the College of Engineering, the College of Arts and Sciences, or the College of Agriculture and Life Sciences; and informatics and business information systems at UC Irvine. At UBC, over a third of the students in double majors involving CS are female.

▸ Train instructors of introduc-

tory CS courses to encourage high-performing women to take a second course and consider majoring in computing. If the institution offers an introductory course aimed at non-CS majors, it should ensure that students who do well in it are able to become CS majors without losing credit for the introductory course.

▸ Provide and publicize opportunities for science students to enter CS majors after completing their second year. Many female students start majoring in biology or chemistry with the intention of going to medical school but realize during the second year that their choice is unlikely or undesirable.

▸ Provide bachelor's or master's programs in CS for people who already have a bachelor's degree in another field.[13]

▸ Encourage female students to participate in mentoring programs at their institution in person or by email. See, for example, Mentornet (www.mentornet.org).

▸ Encourage female students to attend computing conferences, and help to finance the excursion. For example, the Grace Hopper Celebration of Women in Computing has an outstanding track record of improving recruitment, retention, and advancement. Positive outcomes reported by attendees include inspiration, decreased feelings of isolation, renewed commitment to a computer science or technology degree, and the establishment of a professional network that aids in career advancement. ACM-W has spearheaded and supported a number of regional conferences modeled on the Hopper conference. In addition, ACM-W recently launched a program that provides scholarships to female students so that they can attend research conferences.

▸ Form an ACM-W chapter (see women.acm.org/activities.html).

▸ Engage female students in computing research during the summer after their first or second year.

Much has been written about ways to enroll more women in CS programs at the *graduate level* and retain them. Basic approaches to enhancing enrollment include the funding of visits by accepted students to the department, recruitment visits by female graduate students to their undergraduate institutions, and departmental delegations to conferences, such as Hopper, attended by many women undergraduate and graduate students majoring in computing fields. But in our view, three kinds of experiences make undergraduate females most likely to commence graduate work in CS: encouragement by a faculty member; research experience as an undergraduate; and sustained interaction with graduate students.

In the U.S., many universities and colleges offer Research Experiences for Undergraduate programs (REUs, often funded by NSF) during the summer. For over a decade, CRA-W has run its Distributed Mentor Program (DMP), which matches female undergraduates with female faculty members (usually at a different institution) for the purpose of doing research together, and the program provides funding for the effort. A research study by Harrod[12] demonstrated that students participating in DMP were significantly more likely to enter a graduate program later on. Similarly, many programs connect female undergraduates with counterparts in graduate school. Over the past few years, the Women in Computing Society program at Carnegie Mellon has sent groups of female graduate students to several academic institutions in order to talk to female undergraduates about graduate school. MentorNet provides email mentoring for undergraduates by graduate students, faculty, and computing professionals; and many departmental mentoring programs pair undergraduates with graduate students or conduct tri-mentoring programs that group an undergraduate, a graduate student, and a computing professional.

Retention initiatives fall into two groups: those conducted within the institution, usually at the departmental or school (faculty, college) level; and regional, national, or international programs that bring together women graduate students from more than one institution. Most of the within-institution initiatives are designed to build a sense of community among the students and provide mentoring, especially at critical retention points in the graduate programs. An example of the second group of initiatives is CRA-W's long history of offering graduate-student programs—beginning with academic career workshops at computing conferences and more recently at the annual Grad Cohort symposia—that bring together hundreds of women graduate students from across the U.S.

At the *faculty level*, the primary goals are to recruit more women faculty and ensure that they ultimately achieve tenure and promotion.

Significant efforts has been made over the last decade, supported by ADVANCE and other NSF grants, to establish best practices that achieve more diversity—that is, the recruitment of more women and underrepresented minorities—in science and engineer-

> "*Illegitimi non carborundum*, which is mock-Latin for 'don't let the bastards grind you down'. (See Wikipedia.) It's helped me a lot over the years!"
>
> **PROFESSOR DAME WENDY HALL: ACM PRESIDENT; SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE, UNIVERSITY OF SOUTHAMPTON.**

ing faculties.[26] Here we cite a few of these efforts' key characteristics:

▸ Provide training for faculty search committees on best practices. Unconscious bias in academic hiring and evaluating remains widespread,[33] but raising awareness of such bias helps reduce its influence. Moreover, training can help each committee build a more diverse pool of candidates, design an effective interview schedule, avoid common pitfalls when interviewing women candidates, and ensure that committee members have the answers to often-asked questions.

▸ Identify potential women candidates and build proactive relationships with them, even during years when the department is not conducting a search. For example, invite promising women graduate students and postdocs to give presentations or conduct seminars. Invite female researchers from industry to visit the department for, say, a week. Invite untenured women faculty from institutions (especially those with records of often denying tenure) to give colloquia.

▸ Be prepared to help find jobs for women candidates' spouses or partners,[28] many of whom are academics themselves, often in science or engineering disciplines. Some academic institutions in fact have programs to assist departments interested in hiring the spouse or partner of an especially talented faculty candidate.

▸ Establish parent-friendly practices in the department and institution. For example, do not schedule department meetings after 5 P.M. and encourage the institution to provide paid parental leave for faculty members with newborn children.

"Though female leaders have the same technical challenges and are expected to produce the same kind of results as male leaders, there is often a cultural context that influences their approach and a different interpretation of their performance that ups the ante."

FRANCINE BERMAN: DIRECTOR, SAN DIEGO SUPERCOMPUTER CENTER; HIGH PERFORMANCE COMPUTING ENDOWED CHAIR, JACOBS SCHOOL OF ENGINEERING, UNIVERSITY OF CALIFORNIA, SAN DIEGO.

For the last 15 years, CRA-W, through its Cohort of Associate Professors Project, has maintained programs to help young women CS faculty progress successfully; and it has also conducted workshops for older female faculty further along in their careers. Significant support/programs are also offered through the NSERC-Industry WISE chairs, ADVANCE grants, and more recently the ABI TechLeaders workshops for senior academic women.

For beginning untenured faculty, important actions by the department include:

▸ Provide a lighter teaching load for the first two years and limit the number of different courses she teaches during the first four years. Encourage each new faculty member to participate in professional-development courses offered by the institution and to invite other faculty members to observe her teaching. In addition, perform informal midterm teaching evaluations in all of her courses.

▸ At research-intensive institutions: Ensure that new faculty members receive enough startup funding to support two or more graduate students for at least two years. Provide significant help in writing grant proposals.

▸ Make certain that the new faculty member understands what is expected in order to gain tenure. Provide clear and constructive feedback annually on achievements she should focus

Composite screenshot from Storytelling Alice.

"If we want young girls to choose to learn how to program computers, we need to deeply understand the kinds of programs girls will be motivated to create and design programming environments that make those programs readily achievable."

CAITLIN KELLEHER: ASSISTANT PROFESSOR COMPUTER SCIENCE AND ENGINEERING; AS A PH.D. STUDENT WORKING WITH RANDY PAUSCH, CREATED "STORYTELLING ALICE" TO INSPIRE MIDDLE SCHOOL GIRLS TO LEARN PROGRAMMING.

A session called "Using Robots to Introduce Computer Programming to Middle Schools" at Grace Hopper Conference in Keystone, CO, on October 2, 2008.

on in the coming year.

▸ Match the new faculty member with a senior faculty member who is a compatible and effective mentor.

▸ Proactively engage the new faculty member in departmental gatherings, both formal and informal.

▸ Encourage the new faculty member to attend an Academic Career Workshop (CRA-W), Hopper conference, or similar events, and provide some of the associated funding.

▸ At research-intensive institutions, nominate deserving new faculty members for prestigious fellowships.

Key actions to take before the tenure decision:

▸ If there is a significant chance that the faculty member will not receive tenure, make sure she is aware of it as soon as possible.

▸ At research-intensive institutions: In the year before the tenure decision, encourage the faculty member to give seminars at several of the top departments in her research area and send copies of her key publications to those departments' leading researchers.

After a faculty member receives tenure, continue mentoring and provide annual feedback to ensure that she stays on track for promotion to full professor. Encourage her to assume leadership roles in the department, institution, or professional community.

Industry: Cultivating Its Most Critical Assets. Some of the successful approaches that companies may use for recruiting, retaining, and advancing more women in computing, as well as in other technical professions, include:

▸ Senior managers should be aware of the unconscious biases that often permeate industrial settings. Even when they offer competence and qualifications equivalent to those of their male counterparts, women are perceived less favorably, which leaves them at a significant disadvantage for advancement.[10] Thus company leaders should familiarize themselves with this phenomenon—for example, by attending pertinent workshops—and then revise their evaluation and promotion practices accordingly.

▸ Cast a broad net to recruit female computer scientists. Indeed, many women in technical positions in the work force earned degrees in other disciplines.[32]

▸ Address technical women's isolation, in part by developing a network to address their specific needs. Networking is paramount to career advancement, yet women in entry- to mid-level technical positions have fewer opportunities to participate in it outside their immediate department.[14]

▸ Implement a mentoring program. Indeed, make mentoring, which positively impacts career advancement and satisfaction, a basic part of the organizational culture. Sun Microsystems' SEED program, for example, is regard-

> "The best advice I've ever heard about how women should compete in the workplace was spoken by Betty Snyder Holberton, the first of my three favorite work partners: 'Look like a girl. Act like a lady. Think like a man. Work like a dog.'"
>
> **JEAN BARTIK: PROGRAMMER FOR THE GROUNDBREAKING ENIAC COMPUTER.**

PHOTOGRAPH BY KIMBERLY BLESSING

"Seek inspiration from mentors—family, friends, teachers, and/or prominent people—to create careers combining your education, talents, interests, and dreams."

MAXINE D. BROWN: ASSOCIATE DIRECTOR, ELECTRONIC VISUALIZATION LABORATORY, UNIVERSITY OF ILLINOIS AT CHICAGO; CO-AUTHORED THE 1987 NSF REPORT, VISUALIZATION IN SCIENTIFIC COMPUTING, WHICH DEFINED THE FIELD OF SCIENTIFIC VISUALIZATION.

ed as a major step in this direction.[8]

▸ Join organizations, such as the Anita Borg Institute, the Workforce Alliance of NCWIT, and Catalyst, that are actively working on solutions to problems facing women in the technical work force.

▸ Send the company's women in computing to some of the field's conferences so that they can expand their professional network. Company recruiters should attend such conferences as well.

▸ Implement "best practices"—those shown to be most successful at increasing women's representation in the technical work force (see resources provided at www.ncwit.org/resources.res.practices.php).

▸ Make an active effort to place more women in senior leadership positions. This policy is not only rewarding in its own right but it also increases the company's ability to recruit and retain other female talent.[15] Provide leadership-development opportunities through programs such as the Anita Borg Institute's TechLeaders.

▸ Correct the company's gender-related wage differential, thereby sending a strong signal to its work force that women are deemed to be critical assets.[3]

Consider the exemplary efforts of IBM. Over the past 15 years, the company has effected dramatic and systemic cultural changes, resulting in a 370% increase in its women executives and a 233% increase in ethnic minority executives (as of 2004).[30] These changes occurred in four main ways: demonstrating leadership support, engaging employees as partners, integrating diversity goals with management practices, and linking diversity goals to business goals.

When Lou Gerstner assumed control of the company in 1993, he deliberately set out to change IBM's culture by uncovering, and endeavoring to understand, differences between underrepresented population groups (including women and minorities). The first step was to establish task forces, composed of executives and employees alike, for each group. Once group needs were better understood, management implemented practices to establish and sustain diversity—in creating pools of high-potential candidates for recruitment or of outstanding employees for advancement, women and minorities had to be well represented. These changes, though company-wide, were particularly focused on IBM's technical workforce, which it considered to be one of its most critical assets.

From the beginning, IBM's diversity efforts were driven by the desire not only to do the right thing but also to broaden its customer base. Gerstner created plans to embrace group differences in order to appeal to broader sets both of employees and customers. As a result, the company extended its reach into women-owned businesses, for example, as well as into new market segments.

Government Has a Role to Play

Improving women's representation in computing must also entail more enlightened governmental institutions and policies. The following agency practices have been shown to be particularly effective:

▸ Rigorously adhere to evaluating Criterion 2. In 1997, the NSF's board approved a reformulated merit review policy that included two criteria: (1) intellectual merit and quality of research, and (2) broader impacts of the proposed activity. Criterion 2 required that proposals address areas of societal concern, including the broadening of underrepresented groups' participation in the science, technology, engineering, and mathematics (STEM) work force.[24] But although NSF continuously improves enforcement of Criterion 1 under its grants, it has yet to establish a method for ensuring observance of Criterion 2.

▸ Hold academic institutions accountable for measuring diversity among their employees through enforcement of Title IX. Four federal science agencies have made efforts

"The theoretical and practical knowledge embodied in CS is interesting as standalone study. But the real opportunity lies in equipping oneself to partner with scientists or business experts, to learn what they know and, together, to change how research or business is conducted."

ADELE GOLDBERG: FORMER RESEARCH LABORATORY MANAGER, XEROX PARC; FOUNDING CHAIRMAN AND CEO, PARCPLACE SYSTEMS; GENERAL PARTNER, PHARMA CAPITAL PARTNERS.

to ensure that grantees comply with Title IX by performing several compliance activities, such as investigating complaints and providing technical assistance, but most agencies have not conducted all the monitoring activities required.[11]

▸ Publicly advertise the agency's diversity programs and its proportion of women in leadership positions.

▸ Stay aware of new policy and legislative initiatives, even in advance of their demonstrated impacts.

▹ The House Diversity and Innovation Caucus, whose mission is to help generate policy ideas for addressing the underrepresentation of women and minorities in the STEM fields, has held a number of briefings.

▹ The America COMPETES Act of 2007 (Creating Opportunities to Meaningfully Promote Excellence in Technology, Education, and Science) seeks to strengthen research, provide technical training for 21st-century occupations, and attract the best and brightest workers.

▹ NSF's ADVANCE program, in place for about a decade, contributes to the development of a more diverse technical work force (see www.nsf.gov/funding/pgm_summ.jsp?pims_id=5383).

▹ The BPC program, while relatively new, is having significant impact on many computing educators and professionals. Its largest initiative is NCWIT, whose organizational structure includes a set of alliances—within participating communities in K–12, academia, and industry—that have been very successful. For example, the Stars Alliance is a partnership of over 20 southeastern universities that share best practices for recruiting students to computing and retaining them. The goal of the Alliance for Access to Computing Careers is to increase the field's representation of people with disabilities.

Conclusion

Every computing professional, male and female alike, can contribute to the increased participation of women in the field. At the very least, each of us should do more to encourage women with whom we daily interact. For those readers not well informed about practices and programs that help attract women to our profession and retain them, we hope this article has provided useful information and indicated actionable steps pertinent to one's particular circumstances. By way of encouragement, know that institutions that have already made decisions to implement these kinds of practices are seeing significant increases in the participation of women in computing *at all levels*. Thus we encourage our colleagues to work to effect positive change, both locally—in individual institutions—and globally. Long-term success depends on our entire community taking responsibility for making computing a broadly supportive and inclusive discipline. C

References

1. American Council on Education. Office of Women in Higher Education (2008); www.acenet.edu.
2. Badagliacco, J. Gender and race differences in computing attitudes and experiences. *Social Science Computer Review 8* (1990), 42–63.
3. Blum, T.C., Fields, D.L., and Goodman, J.S. Organization-level determinants of women in management. *Academy of Management Journal 37*, 2 (1994), 241–266.
4. Bureau of Labor Statistics. *Occupational Outlook Handbook (OOH), 2008-09 Edition*; www.bls.gov/OCO/.
5. Computer Science Teachers Association. CSTA National Secondary Computer Science Survey: Comparison of 2005 and 2007 Survey Results (2007); www.csta.acm.org/Research/sub/New_Folder/ResearchFiles/CSTASurvey05_07Comp.pdf.
6. Computing Research Association. *Taulbee Survey Report 2006–2007* (2008).
7. Coyle, E.J., Jamison, L.H., and Oakes, W.C. Integrating engineering education and community service: Themes for the future of engineering education. Journal of Engineering Education (2006), 7–11.
8. Dickinson, K. Five years of mentoring by the numbers: SEED mentoring program. Presentation at the 2006 Grace Hopper Celebration of Women and Computing; research.sun.com/SEED/hopper.presentation.oct06.pdf.
9. Dodds, Z., Libeskind-Hadas, R., Alvarado, C., and Kuenning, G. Evaluating breadth-first CS1 for scientists. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (2008); Sedgewick, R. and Wayne, K. Introduction to Programming in Java: A textbook for a first course in computer science for the next generation of scientists and engineers, 2004; www.cs.princeton.edu/introcs/home/.
10. Eagly, A. and Carau, S.J. Role congruity theory of prejudice toward female leaders. *Psychological Review 109*, 3 (2002), 573–598.
11. Government Accountability Office. *Gender Issues: Women's Participation in the Science Has Increased but Agencies Need to Do More to Ensure Compliance with Title IX.* United States Government Accountability Office Report to Congressional Requesters (Report 04-639), July 2004.
12. Harrod, M.J. Distributed mentor project: Evaluation of impact and experiences of participants. *Computing Research News* (Sept. 2000); www.cra.org/Activities/craw/reports/sep00.pdf/.
13. Humphreys, S. and Spertus, E. Leveraging an alternative source of computer scientists: Reentry programs. *Commun. ACM 34*, 2 (Feb. 2002): 53–56; Klawe, M., Cavers, I., Popowich, F., and Chen, G. A computer science post-baccalaureate diploma program that appeals to women. In *Proceedings of the Seventh International Conference on Women, Work, and Computerization* (2000).
14. Igbaria, M. and Chidambaram, L. Examination of gender effects on intention to stay among information systems employees. SIGCPR (Nashville, TN, 1995).
15. Kanter, R.M. *Men and Women of the Corporation.* Basic Books, NY, 1993.
16. Kekelis, L.S., Ancheta, R.W., and Heber, E. Hurdles in the pipeline: Girls and technology careers. *Frontiers: A Journal of Women Studies 26*, 1 (2005).
17. Mannix, E.A. and Neale, E.A. What difference makes a difference? *Psychological Science in the Public Interest 6*, 2 (2005), 31–32.
18. Margolis, J., and Fisher, A. *Unlocking the Clubhouse.* MIT Press, 2002.
19. McDowell, C., Werner, L., Bullock, H.E., and Fernald, J. Pair programming improves student retention, confidence, and program quality. *Commun. ACM 49*, 8 (Aug. 2006), 90–95.
20. Morrell, C., Cotten, S., Sparks, A., and Spurgas, A. *Computer Mania Day: An effective intervention for increasing youth's interest in technology.* Report to the Maryland Commission for Women, 2004.
21. National Center for Women & Information Technology. *NCWIT Scorecard* 2007: A Report on the Status of Women in Information Technology (2007); ncwit.org/pdf/2007_Scorecard_Web.pdf; Alliance for Board Diversity. Women and Minorities on Fortune 100 Boards (2008).
22. National Center for Women & Information Technology. *Who Invents IT? An Analysis of Women's Participation in Information Technology Patenting*, 2007.
23. National Science Board. *Science and Engineering Indicators 2008* (NSB 08-01), Arlington, VA; www.nsf.gov/statistics/seind08/.
24. National Science Foundation. *Broadening Participation in America's Science and Engineering Workforce: The 1994–2003 Decennial & 2004 Biennial Reports to Congress.* Arlington, VA, Dec. 2004.
25. National Science Foundation, Division of Science Resources Statistics. *Women, Minorities, and Persons with Disabilities in Science and Engineering* (NSF 07-315). Arlington, VA, Feb. 2007; www.nsf.gov/statistics/wmpd/.
26. *Proceedings of a Banff International Research Station Workshop.* Mentoring for Engineering Academia II. (2007); http://birs07.stanford.edu/.
27. Rich, L., Perry, H., and Guzdial, M. A CS1 course designed to address interests of women. In *Proceedings of the 35th SIGCSE Technical Symposium of Computer Science Education* (2004); www.cs.ubc.ca/~hoos/Courses/CPSC101-05/Handouts/about.pdf; Shmishek, E. Where the girls aren't. UBC Reports, 2003; www.publicaffairs.ubc.ca/ubcreports/2003/03jun05/girls.html.
28. Shiebinger, L., Henderson, A., and Gilmartin, S.K. *Dual-Career Academic Couples: What Universities Need to Know.* Clayman Institute for Gender Research, Stanford University, 2008.
29. Tillberg, H.K. and McGrath Cohoon, J. Attracting women to the CS major. *Frontiers: A Journal of Women Studies 26*, 1 (2005), 126–140.
30. Thomas, D.A. Diversity as strategy. *Harvard Business Review* (Sept. 2004).
31. Touchton, J.G., and Ingram, D. *Women Presidents in US Colleges and Universities. A 1995 Higher Education Update.* American Council on Education, Washington, DC, 1995.
32. Turner, S., Bernt, P., and Pecora, N. Why women choose information technology careers: Educational, social and familial influences. Presented at the American Educational Research Association, New Orleans, LA (ERIC Document Reproduction Service No. ED456878), Apr. 2002.
33. Valian, V. *Why So Slow? The Advancement of Women.* MIT Press, 1999.
34. West, M.S., and Curtis, J.W. *AAUP Faculty Gender Equity Indicators.* American Association of University Professors, 2006.

Maria Klawe is president of Harvey Mudd College, Claremont, CA. Prior to joining HMC, she served as dean of engineering and professor of computer science at Princeton University. She is a former president of ACM.

Telle Whitney is president and CEO of the Anita Borg Institute for Women and Technology, Palo Alto, CA.

Caroline Simard is director of research at the Anita Borg Institute for Women and Technology in Palo Alto, CA.

P. 78

Technical Perspective

Tools for Information to Flow Securely and Swift-ly

By Dan Wallach

P. 79

Building Secure Web Applications with Automatic Partitioning

By Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng

P. 88

Technical Perspective

The Complexity of Computing Nash Equilibrium

By Ehud Kalai

P. 89

The Complexity of Computing a Nash Equilibrium

By Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou

DOI:10.1145/1461928.1461948

Technical Perspective
Tools for Information to Flow Securely and Swift-ly

By Dan Wallach

BACK IN THE old days of the Web (before 1995), Web browsers were fairly simple devices. They downloaded HTML, laid out the text, and loaded a few images. There was neither JavaScript nor Java nor Flash. There was only the beginning of e-commerce sites, where all the code ran exclusively on the server. While security vulnerabilities certainly existed in both browsers and servers, the server's Web interface was simple enough that an auditor could at least look at it and reason about its security.

Today, it's a different world. With powerful client-side JavaScript and asynchronous Web requests (called "Ajax"), we now have Web "applications" that have significant portions of their state on the client side. This gets even further complicated by "mashups," where code from many Web sites might interact within a single Web browser. Building systems like this typically requires careful engineering of the whole system; the server side must be secure even if a non-conforming client is making arbitrary requests.

Meanwhile, a new generation of tools, such as the Google Web Toolkit (GWT), promise to simplify the client-server programming process by blurring the distinction between the client and server. You just write one monolithic program and draw a line through it saying "these parts go on the client and these parts go on the server." This sounds great for improving developer productivity, particularly by abstracting away the inconsistencies and peculiarities of each Web browser's JavaScript runtime system. Because the RPCs are generated automatically, possible information leaks, security holes, or a host of other issues could well present themselves, and the source code is sufficiently abstract so that it's no longer obvious how to audit such a system for correctness.

This concern motivates the following research paper, "Building Secure Web Applications with Automatic Partitioning," where the authors describe a tool they built—Swift—that provides a general-purpose programming language, an extension of Java, for building partitioned Web applications. The secret sauce in Swift is its handling of annotations, placed by the programmer, which declare security properties for objects and variables within the program. These annotations speak toward secrecy or integrity constraints on the data.

For example, say you've got a list of passwords (or hashed passwords or whatever else) on the server and you want to validate a client-supplied password. Clearly, you want to perform that comparison on the server side, such that an attacker cannot access other passwords or impersonate other users. But how do you guarantee such a thing? Swift lets us declare the list of passwords to be "sensitive." We don't want to disclose it to any user. With such annotations, the program partitioning system can figure out that the password-checking logic can only happen on the server side, satisfying the information flow constraints.

Sounds easy, right? Not really. In fact, there's an important problem. Information flow systems are really good at saying "no." You validated the password, and now you want to let the user know. Unfortunately, that very fact is sensitive information because it was derived from sensitive information. We can't release that to the user? That's a problem. Clearly, we need to carve out exceptions to the rules in order to get anything useful done. Swift allows a programmer to make these sorts of exceptions in a controlled fashion, but those will still need to be carefully audited.

Information flow technologies, whether operating statically like Swift, or operating dynamically like the "tainting" mechanism used in Perl, are clearly an important mechanism for building and maintaining secure Web applications. One only has to look at the never-ending parade of cross-site scripting, cross-site request forgery, SQL injection, and other such Web attacks, none of which rely on traditional buffer overflows, to recognize the importance of high-level automated systems built into the development tool chain to improve our assurance that systems are secure. Manual, labor-intensive code audits by security experts cannot scale to support the vast number of new Web applications being deployed each and every day.

The challenge for the research community, with sophisticated tools like Swift, is to simplify the development process, making it easier to get the security labels written properly. Ultimately, our ability to prove that a system is secure, whether Web-based or anything else, is limited by our ability to understand the security model and convince ourselves that the labels we've written and properties we've derived from those labels are consistent with our high-level security goals. Swift takes us a big step closer to achieving those goals. C

Dan Wallach (dwallach@cs.rice.edu) is an associate professor in the Department of Computer Science at Rice University, Houston, TX.

DOI:10.1145/1461928.1461949

Building Secure Web Applications with Automatic Partitioning

By Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng

Abstract

Swift is a new, principled approach to building Web applications that are *secure by construction*. Modern Web applications typically implement some functionality as client-side JavaScript code, for improved interactivity. Moving code and data to the client can create security vulnerabilities, but currently there are no good methods for deciding when it is secure to do so.

Swift automatically partitions application code while providing assurance that the resulting placement is secure and efficient. Application code is written as Java-like code annotated with information flow policies that specify the confidentiality and integrity of Web application information. The compiler uses these policies to automatically partition the program into JavaScript code running in the client browser and Java code running on the server. To improve interactive performance, code and data are placed on the client. However, security-critical code and data are always placed on the server. The compiler may also automatically replicate code across the client and server, to obtain both security and performance.

1. INTRODUCTION

Web applications are client–server applications in which a Web browser provides the user interface. They are a critical part of our infrastructure, used for banking and financial management, email, online shopping and auctions, social networking, and much more. The security of information manipulated by these systems is crucial, and yet these systems are not being implemented with adequate security assurance. Indeed, 69% of all Internet vulnerabilities are said to be related to Web applications.[20] The problem is that with current implementation methods, it is difficult to know whether an application adequately enforces the confidentiality or integrity of the information it manipulates.

Recent trends in Web application design have exacerbated the security problem. To provide a rich, responsive user interface, application functionality is moving into client-side JavaScript[6] code that executes within the Web browser. JavaScript code is able to manipulate user interface components and can store information persistently on the client side by encoding it as cookies. These Web applications are distributed applications, in which client- and server-side code exchange protocol messages represented as HTTP requests and responses. In addition, most browsers allow JavaScript code to issue its own HTTP requests, a functionality used in the Ajax development approach (Asynchronous JavaScript and XML).

With application code and data split across differently trusted tiers, the developer faces a difficult question: when is it secure to place code and data on the client? All things being equal, the developer would usually prefer to run code and store data on the client, avoiding server load and client–server communication latency. But moving information or computation to the client can easily create security vulnerabilities.

For example, suppose we want to implement a simple Web application in which the user has three chances to guess a secret number between 1 and 10, and wins if a guess is correct. Even this simple application has subtleties. There is a confidentiality requirement: the user should not learn the secret until after the guesses are complete. There are integrity requirements, too: the comparison of the guess and the secret should be computed in a trustworthy way, and the number of guesses must also be counted correctly.

The guessing application could be implemented almost entirely as client-side JavaScript code, which would make the user interface very responsive and would offload the most work from the server. But it would be insecure: a client with a modified browser could peek at the true number, take extra guesses, or simply lie about whether a guess was correct. On the other hand, suppose guesses that are not valid numbers between 1 and 10 do not count against the user. Then it is secure and indeed preferable to perform the bounds check on the client side. Currently, Web application developers lack principled ways to make decisions about where code and data can be securely placed.

The Swift system is a new way to write Web applications that are *secure by construction*. Applications are written in a higher-level programming language in which information security requirements are explicitly exposed as declarative annotations. The compiler uses these security annotations to decide where code and data in the system can be placed securely. Developing programs in this way ensures that the resulting distributed application protects the confidentiality and integrity of information. The general enforcement of information integrity also guards against common vulnerabilities such as SQL injection and cross-site scripting.

Swift applications are not only more secure, they are also easier to write: control and data do not need to be explicitly transferred between client and server through the awkward extra-linguistic mechanism of HTTP requests. Automatic placement has a performance benefit as well. Currently,

A previous version of this paper was published in *Proceedings of the 21st ACM Symposium on Operating System Principles (October 2007)*.

programmers have no help deciding how to place code and data in the system for best interactive performance. In Swift, the compiler automatically places code and data to optimize interactive performance, subject to security constraints, and automatically synthesizes secure protocols where communication is required.

Of course, others have noticed that Web applications are hard to make secure and awkward to write. Prior research has addressed security and expressiveness separately. One line of work has tried to make Web applications more secure, through analysis[10, 12, 22] or monitoring[9, 16, 23] of server-side application code. However, this work does not help application developers decide when code and data can be placed on the client. Conversely, the awkwardness of programming Web applications has motivated a second line of work toward a unified language for writing distributed Web applications.[5, 8, 19, 24] However, this work largely ignores security; the programmer controls code placement, but nothing ensures the placement is secure.

Swift differs by addressing security and expressiveness together. One novel feature illustrating this is its selective replication of computation and data onto *both* the client and server. Replication is useful for validation of form inputs, which should happen on the client so the user does not have to wait for the server to respond to invalid inputs, but must also happen on the server because client-side validation is untrusted. With Swift, programmers write validation code only once. Compared to the current practice in which developers write separate, possibly inconsistent validation code for the client and server, Swift improves both security and expressiveness.

An earlier publication[3] is a good source for further details about Swift, especially regarding the run-time system and optimization techniques.

2. ARCHITECTURE

Figure 1 depicts the architecture of Swift. The programmer writes annotated Java source code, shown at the top of the diagram. Proceeding from top to bottom, a series of program transformations converts the code into the partitioned form shown at the bottom, with Java code running on the Web server and JavaScript code running on the client Web browser.

2.1. Jif source code

The source language of the program is an extended version of the Jif 3.0 programming language.[13, 15] Jif extends the Java programming language with language-based mechanisms for information flow control and access control. Information security policies can be expressed directly within Jif programs, as *labels* on program variables. By statically checking a program, the Jif compiler ensures that these labels are consistent with flows of information in the program.

The original model of Jif security is that if a program passes compile-time static checking, and the program runs on a trustworthy platform, then the program will enforce the information security policies expressed as labels. For Swift, we assume that the Web server can be trusted, but the client machine and browser may be buggy or malicious. Therefore, Swift transforms program code so that the application runs securely, even though it runs partly on the untrusted client.

2.2. WebIL intermediate code

The first phase of program transformation converts Jif programs into code in an intermediate language we call WebIL. Like Jif, WebIL includes annotations, but the space of allowed annotations is much simpler, describing constraints on the possible locations of application code and data. For example, the annotation `S` means that the annotated code or data must be placed on the Web server. The annotation `C?S` means that it must be placed on the server, and *may* optionally be replicated on the client as well. It is convenient to program directly in WebIL, but this does not enforce secure information flow.

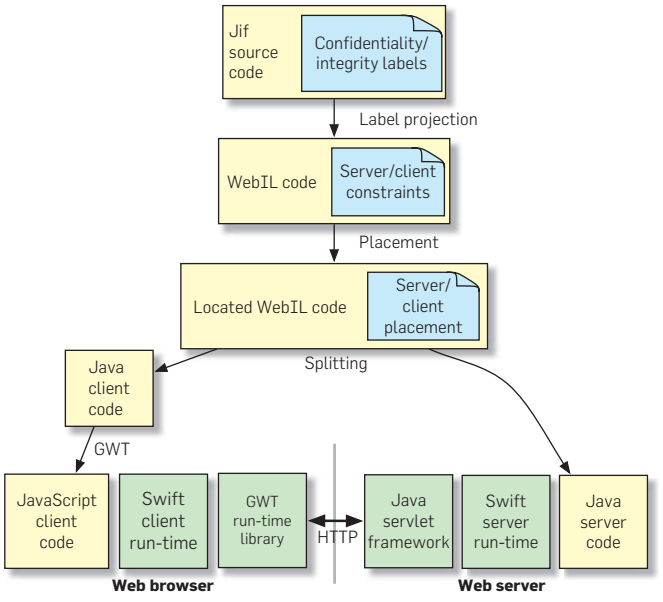
2.3. Placement and optimization

The initial WebIL annotations are merely constraints on code and data placement. The second phase of compilation decides the exact placement and replication of code and data between the client and server, in accordance with these constraints. The system uses a polynomial-time algorithm to minimize the cost of the placement, by avoiding unnecessary network messages.

2.4. Splitting code

Once code and data placements have been determined, the compiler transforms the WebIL code into two Java programs, one representing server-side computation and the other, client-side computation. This is a fine-grained transformation. Different statements within the same method may run variously on the server and the client, and similarly with different fields of the same object. Sequential statements in the program source code may become separate code fragments on the client and server that invoke each other via network messages.

Figure 1: The Swift architecture.



2.5. Client-side code

The compiler generates Java code to run on the client. This code is compiled to JavaScript code using the Google Web Toolkit (GWT).[8] The client- and server-side code uses various libraries: Swift's run-time libraries, GWT, and the Java servlet framework.

A running Swift program uses an Ajax approach to securely implement the functionality defined by the original source code. From the browser's perspective, the application is a single Web page, with most user actions, such as clicking on buttons, handled by JavaScript code. This code issues its own HTTP or HTTPS requests to the Web server, which executes Java code to compute XML responses.

2.6. Swift run-time system

In order to enforce the security requirements in the Jif source code, information flows between the client and the server must be strictly controlled. In particular, confidential information must not be sent to the client, and information received from the client cannot be trusted. The Swift compilation process generates code that satisfies these constraints. These constraints are also enforced by the Swift run-time system, which ensures that information coming from the client is never trusted.

3. WRITING SWIFT APPLICATIONS

3.1. Labels and principals

Programming with Swift starts with a program written in the Jif programming language.[13,15] A little background on Jif will therefore be helpful.

In Jif programs, information security requirements are expressed using *labels* from the decentralized label model (DLM).[14] A label is a set of policies expressed in terms of *principals*. For example, the confidentiality policy `alice→bob` says that principal `alice` owns the labeled information but permits principal `bob` to read it. Similarly, the integrity policy `alice←bob` means that `alice` permits `bob` to affect the labeled information.

Labels can be attached to types, making Jif a *security-typed* language. For example, the following declaration uses a label containing two policies separated by a semicolon:

```
int {alice → bob,alice; bob ← alice} y;
```

It means that the information in `y` is considered sensitive by `alice`, who wants to restrict its release only to `bob` and `alice`; and further that it is considered trustworthy by `bob`, who wants to permit only `alice` to affect it.

The Jif compiler uses labels to statically check that information flows within Jif code are secure. For example, consider the following code fragment (using the same variable `y`):

```
int{alice}→bob x = y
```

This code causes information to flow from `y` to `x`. For the code to be secure, the label on `x` must restrict the use of data in `x` at least as much as the label on `y` restricts the use of `y`. This is true if (1) for every confidentiality policy on `y`, there is one at least as restrictive on `x` (which is the case because `alice→bob` is at least as restrictive as `alice→bob`, `alice`) and (2) for every integrity policy on `x`, there is one at least as restrictive on `y` (which is the case because `x` has no integrity policy).

Suppose that `x` instead had the label `{alice→client}`. The information flow from `y` to `x` would still be secure if the principal `client` *acts for* the principal `bob`, meaning that anything `client` does or says can be treated as coming from `bob`. In that case, `alice→client` is at least as restrictive as `alice→bob`. Acts-for relationships increase the expressive power of labels and allow static information flow checking to work even when trust relationships change over time.

Two principals are already built into Swift programs. The principal `*` (a.k.a. `server`) represents the maximally trusted principal in the system. The principal `client` represents the other end of the current session—in ordinary, nonmalicious use, a Web browser under the control of a user. When reasoning about security, we can only assume that the client is the other end of a network connection, possibly controlled by a malicious attacker. Because the server is trusted, the principal `*` acts for `client`. The `client` may see information whose confidentiality is no greater than `*→client`, and can produce information with integrity no greater than `*←client`.

Labels defined in terms of principals are a key feature of the DLM. Because labels keep track of *whose* security is being enforced, they are useful for systems such as Web applications, where principals need to cooperate despite mutual distrust. By expressing labels in terms of principals, the DLM also enables connections between policies for information flow (labels), policies for trust and authorization (acts-for relationships), and processes for authentication. These connections are all important for Swift.

A Swift program may use and even create additional principals, for example, to represent different users of a Web application. For a user to log in as principal `bob`, server-side application code marked as trusted by `bob` must authenticate the user, establishing that the principal named by `client` acts for `bob`. Requests from that browser are then treated as coming from `bob`, and information readable by `bob` can be sent there. All this can be accomplished within the Jif programming model.

3.2. Sample application

The key features of the Swift programming model can be seen in a simple Swift Web application. Figure 2 shows fragments of the source code for the number-guessing Web application described in Section 1. Java programmers may recognize this Jif code as similar to that of an ordinary single-machine Java application with a user interface dynamically constructed out of widgets such as buttons, text inputs, and text labels. Swift widgets are similar to those in the GWT,[8] communicating via events and listeners. The key difference is that Swift controls how information flows through widgets.

The core application logic is found in the `makeGuess` method (lines 15–39). Other than security label annotations, this is essentially straight-line Java code. To implement the same functionality with technologies such as JSP[2] or GWT

Figure 2: Guess-a-Number Web application.

```
1  public class GuessANumber {
2    final label{*←*} cl = new label{*→client};
3    int{*→*; *←*} secret;
4    int{*→client; *←*} tries;
5    ...
6    private void setupUI{*→client}() {
7      guessbox = new NumberTextBox("");
8      message = new Text("");
9      button = new Button("Guess");
10     ...
11     rootpanel.addChild(cl, cl, guessbox);
12     rootpanel.addChild(cl, cl, button);
13     rootpanel.addChild(cl, cl, message);
14   }
15   void makeGuess{*→client}(Integer{*→client} num)
16     where authority(*), endorse({*←*})
17     throws NullPointerException
18   {
19     int i = 0;
20     if (num != null) i = num.intValue();
21     endorse (i, {*←client} to {*←*})
22     if (i >= 1 && i <= 10) {
23       if (tries > 0 && i == secret) {
24         declassify ({*→*} to {*→client}) {
25           tries = 0;
26           finishApp("You win!");
27         }
28       } else {
29         declassify ({*→*} to {*→client}) {
30           tries--;
31           if (tries > 0) message.setText("Try again");
32           else finishApp("Game over");
33         }
34       }
35     } else {
36       message.setText("Out of range:" + i);
37     }
38   }
39 }
40 class GuessListener
41   implements ClickListener[{*→client}, {*→client}] {
42   ...
43   public void onClick{*→client} (
44         Widget[{*→client}, {*→client}]{*→client} w)
45         : {*→client}
46   {
47     if (guessApp != null) {
48       NumberTextBox guessbox = guessApp.guessbox;
49       if (guessbox != null)
50         guessApp.makeGuess(guessbox.getNumber());
51     }
52   }
53 }
```

requires more code, in a less natural programming style with explicit control transfers between the client and server.

The code contains various labels expressing security requirements; for simplicity, only the principals `client` and $*$ are used in these labels. For example, on line 3, the variable `secret` is declared to be completely secret ($*\rightarrow*$) and completely trusted ($*\leftarrow*$); the variable `tries` on the next line is not secret ($*\rightarrow$`client`) but is just as trusted. Because Jif checks transitively how information flows within the application, the act of writing just these two label annotations constrains many other label annotations in the program. The compiler ensures that all label annotations are consistent with the information flows in the program.

The user submits a guess by clicking the button. A listener attached to the button passes the guess at line 50 to `makeGuess`. The listener reads the guess from a `NumberTextBox` widget that only allows numbers to be entered.

The `makeGuess` method receives a guess `num` from the client. The variable `num` is untrusted and not secret, as shown by its label {$*\rightarrow$`client`} on line 15. The label after the method name, also {$*\rightarrow$`client`}, is the *begin label* of the method. It controls information flows created by the fact that the method was invoked.

The code of `makeGuess` checks whether the guess is correct, and either informs the user that he has won, or else decrements the remaining allowed guesses and repeats. Because the guess is untrusted, Jif will prevent it from affecting trusted variables such as `tries`, unless it is explicitly *endorsed* by trusted code. Therefore, lines 21–37 have a *checked endorsement* that succeeds only if `num` contains an integer between 1 and 10. If the check succeeds, the number `i` is treated as a high-integrity value within the "then" clause. If the check fails, the value of `i` is not endorsed, and the "else" clause is executed. Checked endorsements are a Swift-specific extension to Jif that makes validating untrusted inputs both explicit and convenient.

Forcing the programmer to use `endorse` exposes a potential security vulnerability. In this case, the endorsement of `i` is reasonable because it is intrinsically part of the game that the client is allowed to pick any value in range.

Similarly, some information about the secret value `secret` is released when the client is notified whether the guess `i` is equal to `secret`. Therefore, the bodies of both the consequent and the alternative of the `if` test on line 23 must use an explicit `declassify` to indicate that information transmitted by the control flow of the program may be released to the client. Without the `declassify`, client-visible events—showing messages or updating the variable `tries`—would be rejected by the compiler.

The `declassify` and `endorse` operations are inherently dangerous. Jif controls the use of `declassify` and `endorse` by requiring that they occur in a code marked as trusted by the affected principals; hence the clauses `authority` ($*$) and `endorse` ({$*\leftarrow*$}) on line 16. The latter, *auto-endorse* annotation means that an invocation of `makeGuess` is treated as trusted even if it comes from the client. Jif also requires integrity to perform declassification, enforcing *robust declassification*[4]: untrusted information cannot even affect security-critical release of information.

3.3. Swift user interface framework

Swift programs interact with the user via a user interface framework that abstracts away the details of the underlying HTML and JavaScript. Swift programming has a event-driven style familiar to users of UI frameworks such as Swing.

To allow precise compile-time reasoning about information flows within the user interface framework, all framework classes are annotated with security policies that track information flow that may occur within the framework. The framework ensures that the client is permitted to view all information that the user interface displays. Conversely, all information received from the user interface is annotated as having been tainted by the client.

4. WebIL

After the Swift compiler has checked information flows in the Jif program, the program is translated to an intermediate language, WebIL. WebIL extends Java with placement annotations for both code and data. Placement annotations define constraints on where code and data may be replicated. These constraints may be due to security restrictions derived from the Jif code, or to architectural restrictions (for example, calls to a database must occur on the server, and calls to the UI must occur on the client).

Whereas Jif allows expression and enforcement of rich security policies from the DLM,[14] the WebIL language is concerned only with the placement of code and data onto two host machines, the server and the client. Thus, when translating to WebIL, the compiler projects annotations from the rich space of DLM security policies down to the much smaller space of *placement constraints*.

Using the placement constraint annotations, the compiler chooses a partitioning of the WebIL code. A partitioning is an assignment of every statement and field to a host machine or machines on which the statement will execute, or on which the field will be replicated.

WebIL can be used as a source language in its own right, allowing programmers to develop Web applications in a Java-like programming language with GUI support. This approach has benefits over traditional Web application programming, but does not enforce security as fully as Swift.

4.1. Placement annotations

Each statement and field declaration in WebIL has one of nine possible placement annotations, shown in Table 1: `C`, `S`, `Sh`, `C?Sh`, `C?S?`, `CS`, `CS?`, `C?S`, and `CSh`. Each annotation defines the possible placements for the field or statement. There are three possible placements: *client*, *server*, and *both*. The intuition is that `C` and `S` mean the statement or field must be placed on the client and server, respectively, whereas `C?` and `S?` mean it is optional. An h signifies high integrity. Figure 3 shows the result of translating Guess-a-Number into WebIL, including placement constraints.

The placement of a field declaration indicates which host or hosts the field data is replicated onto. For example, if a field has the placement *server*, that field is stored only on the server; if it has the placement *both*, it is replicated on both client and server.

Similarly, the placement of a statement indicates what machines its computation is replicated onto. For compound statements such as conditionals and loops, the placement indicates the hosts for evaluating the test expression. On line 11 of Figure 3, the comparison of the guess to the secret number is given the annotation `Sh`, meaning that it must occur only on the server. Intuitively, this is the expected placement: the secret number cannot be sent to the client, so the comparison must occur on the server. On line 3, the annotation `C?S?` indicates that there is no constraint on where to test that `num` is non-null; that test may occur on the client, server, or both.

On statements, the annotations `Sh`, `C?Sh`, and `CSh` mark high-integrity code that must be executed on the server. The Swift compiler marks statements as high-integrity if their execution may affect data that the client should not be able to influence. For example, lines 5–14 of Figure 3 are high-integrity. The client's ability to initiate execution of high-integrity statements is restricted by Swift run-time mechanisms.

4.2. Translation from Jif to WebIL

When the compiler translates from Jif to WebIL code, it replaces DLM security policies with corresponding placement constraint annotations. Based on the security policies of the Jif code, the compiler chooses annotations that ensure code and data are placed on the client only if the client cannot violate the security of the program.

Therefore, translation ensures that data may be placed on a client only if the security policies indicate that the data may be read by the principal `client`; data may originate from the client only if the security policies indicate that the data is permitted to be written by the principal `client`. Similar restrictions apply to code: code may execute on the client only if the execution of the code reveals only information

Table 1: WebIL placement constraint annotations.

Annotation	Possible placements	High integrity
C	{*client*}	N
S	{*server*}	N
Sh	{*server*}	Y
CS	{*both*}	N
CSh	{*both*}	Y
CS?	{*client, both*}	N
C?S	{*server, both*}	N
C?Sh	{*server, both*}	Y
C?S?	{*client, server, both*}	N

Figure 3: Guess-a-Number Web application in WebIL.

```
1  auto void makeGuess(Integer num) {
2    C?S?: int i = 0;
3    C?S?: if (num != null)
4    C?S?:   i = num.intValue();
5    C?Sh: boolean b1 = (i >= 1);
6          boolean b2;
7    C?Sh: if (b1) b2 = (i <= 10); else b2 = false;
8    C?Sh: if (b2) {
9      Sh:   boolean c1 = (tries > 0);
10           boolean c2;
11     Sh:   if (c1) c2 = (i == secret);
12     Sh:     else c2 = false;
13     Sh:   if (c2) {
14   C?Sh:     tries = 0;
15   C?S?:     finishApp("You win!");
16           } else {
17   C?Sh:     tries--;
18   C?S?:     if (tries > 0) {
19     C :       message.setText("Try again");
20             } else {
21   C?S?:       finishApp("Game over");
22             }
23           }
24         } else {
25     C :   message.setText("Out of range:"+i);
26         }
27 }
```

that the principal `client` may learn; the result of a computation on the client can be used on the server only if the labels indicate that the result can be affected by the principal `client`.

The translation to WebIL also translates Jif-specific language features. Uses of the primitive Jif type `label` are translated to uses of a class `jif.lang.Label`. Declassifications and endorsements are removed, as they have no effect on the run-time behavior of the program. However, they do affect the labels of code and expressions, and therefore affect their placement annotations.

4.3. Goals and constraints

The compiler decides the partitioning by choosing a placement for every field and statement of the WebIL program. Placements are chosen to satisfy both the placement constraints and to improve performance. Since network latency is typically the most significant component of Web application run time, fields and statements are placed in order to minimize latency arising from messages sent between the client and server. For example, it is desirable to give consecutive statements the same placement.

Replicating computation can also reduce the number of messages. Consider lines 5–8 of the Guess-a-Number application in Figure 3, which check that the user's input `i` is between 1 and 10 inclusive. To securely check that the client provides valid input, these statements must execute on the server. If the value entered by the user is not in the valid range, the server sends a message to the client to execute line 25, informing the user of the error. However, if lines 5–8 execute on *both* the client and server, no server–client message is needed, and the user interface is more responsive.

Figure 4 shows the `GuessANumber.makeGuess` method after partitioning. A placement has been chosen for each statement. The field `tries` has been replicated on both client and server, requiring all assignments to it to occur on both hosts (lines 14 and 17). Also, the compiler has replicated on both client and server the validation code to check that the user's guess is between 1 and 10 (lines 2–8). The validation code must be on the server for security, but placing it on the client allows the user to be informed of errors (on line 25) without waiting for a server response.

4.4. Partitioning algorithm

The compiler chooses placements for statements and fields in two stages. First, it constructs a weighted directed graph that approximates the control flow of the whole program. Each node in the graph is a statement, and weights on the edges are static approximations of the frequency of execution following that edge. Second, the weighted directed graph and the annotations of the statements and field declarations are used to construct an instance of an integer programming problem, which is then reduced to an instance of the maximum flow problem. This can be solved in polynomial time, directly yielding fields and statement placements.

Figure 4: Guess-a-Number after partitioning.

```
1  auto void makeGuess(Integer num) {
2     CS : int i = 0;
3     CS : if (num != null)
4     CS :   i = num.intValue();
5     CSh: boolean b1 = (i >= 1);
6          boolean b2;
7     CSh: if (b1) b2 = (i <= 10); else b2 = false;
8     CSh: if (b2) {
9      Sh:   boolean c1 = (tries > 0);
10           boolean c2;
11     Sh:   if (c1) c2 = (i == secret);
12     Sh:     else c2 = false;
13     Sh:   if (c2) {
14    CSh:     tries = 0;
15     S :     finishApp("You win!");
16            } else {
17    CSh:     tries--;
18    CS :     if (tries > 0) {
19     C :       message.setText("Try again");
20             } else {
21     S :       finishApp("Game over");
22             }
23           }
24         } else {
25     C :   message.setText("Out of range: "+i);
26         }
27 }
```

5. EVALUATION

The Swift compiler extends the Jif compiler with about 20,000 lines of noncomment nonblank lines of Java code. Both the Swift and Jif compilers are written using the Polyglot compiler framework.[17] The Swift server and client run-time systems together comprise about 2,600 lines of Java code. The UI framework is implemented in 1,400 lines of WebIL code and an additional 560 lines of Java code that adapt the GWT UI library. We also ported the Jif run-time system from Java to WebIL, resulting in about 3,900 lines of WebIL code. The Jif run-time system provides support for run-time representations of labels and principals.

To evaluate our system, we implemented six Web applications with varying characteristics. None of these applications is large, but because they test the functionality of Swift in different ways, they suggest that Swift will work for a wide variety of Web applications. Because the applications are written in a higher-level language than is usual for Web applications, they provide much functionality (and contain many security issues) per line of code. Overall, the performance of these Web applications is comparable to what can be obtained by writing applications by hand. Therefore, we do not see any barrier to using this system on larger Web applications.

5.1. Example Web applications

Guess-a-Number: Our running example demonstrates how Swift uses replication to avoid round-trip communication between client and server. Figure 4, lines 5–8, shows that the compiler automatically replicates the range check onto the client and server, thus saving a network message from the server to the client at line 25.

Shop: This program models an important class of real-world Web applications, and is the largest Swift program written to date: 1,094 lines of Jif source. It is an online shopping application with a back-end PostgreSQL database. Users must register and log in before updating their shopping cart and making purchases.

Poll: This is an online poll that allows users to vote for one of three options and view the current winner.

Secret Keeper: This simple application allows users to store a secret on the server and retrieve the secret later by logging in. This example shows that Swift can handle complex policies with application-defined principals, and that it can automatically generate protocols for password-based authentication and authorization from high-level information security policies.

Treasure Hunt: This game has a grid of cells. Some contain bombs and others, treasure. The user chooses cells to dig in, exposing their contents, until a bomb is encountered. The game has a relatively rich user interface.

Auction: This online auction application allows users to list items for auction and to bid on items from other users. The application automatically polls the server to retrieve auction status updates and to update the display.

5.2. Code size results

Programs compiled by Swift do expand as run-time mechanisms are inserted, though avoiding this expansion was not a significant goal in the current implementation. Across the example applications, we found that expansion was roughly linear, with a server-side code expansion factor between 8 and 13. On the client side, about 800 bytes of JavaScript code were generated per line of Jif code. Much of the expansion occurs when Java code is compiled to JavaScript by GWT, so translating directly to JavaScript would probably help.

5.3. Performance results

From the user's perspective, the interactive performance of applications is primarily affected by network latency. Table 2 shows measurements of the number of network messages required to carry out the core user interface task in each application. For example, the core user interface task in Guess-a-Number is submitting a guess. The number of actual messages is compared to the optimum that could be achieved by writing a secure Web application by hand.

Messages sent from the server to the client are the most important measure of responsiveness because it is these messages that the client waits for. The table also reports the number of messages sent from the client to the server; these messages are less important because the client does not block when they are sent.

The number of server–client messages in the example applications is always optimal or nearly so. For example, in the Shop application, it is possible to update the shopping cart without any client–server communication. The optimum number of messages is not achieved for poll because the structure of Swift applications currently requires that the client hears a response to each request. For Guess-a-Number and Treasure Hunt, there are extra client–server messages triggering server-side computations that the client does not wait for, but server–client messages remain optimal.

5.4. Automatic repartitioning

One advantage of Swift is that the compiler can repartition the application when security policies change. We tested this feature with the Guess-a-Number example: if the number to guess is no longer required to be secret, the field that stores the number and the code that manipulates it can be replicated to the client for better responsiveness. Lines 9–13 of Figure 4 all become replicated on both server and client, and the message for the transition from line 13 to 14 is no longer needed. The only source-code change is to replace the label {$*\rightarrow *; *\leftarrow *$} with {$*\rightarrow$`client`; $*\leftarrow *$} on line 3 of Figure 2. Everything else follows automatically.

6. RELATED WORK

In recent years there have been a number of attempts to improve Web application security. At the same time, there has been increasing interest in unified frameworks for Web application development. As a unified programming framework that enforces end-to-end information security policies, Swift is at the confluence of these two lines of work. It is also related to prior work on automatically partitioning applications.

6.1. Information flow in Web applications

Several previous systems have used information flow control to enforce Web application security. This prior work is mostly concerned with tracking information integrity, rather than confidentiality, with the goal of preventing the client from subverting the application by providing bad information (e.g., that might be used in an SQL query). Some of these systems use static program analysis (of information flow and other program properties),[10, 12, 22] and some use dynamic taint tracking,[9, 16, 23] which usually has the weakness

Table 2: Network messages required to perform a core UI task.

Example	Task	Actual		Optimal	
		Server→Client	Client→Server	Server→Client	Client→Server
Guess-a-Number	Guessing a number	1	2	1	1
Shop	Adding an item	0	0	0	0
Poll	Casting a vote	1	1	0	1
Secret Keeper	Viewing the secret	1	1	1	1
Treasure Hunt	Exploring a cell	1	2	1	1
Auction	Bidding	1	1	1	1

that the untrusted client can influence control flow. Unlike Swift, none of this prior work addresses client-side computation or helps decide which information and computation can be securely placed on the client. Most of the prior work only controls information flows arising from a single client request, and not information flow arising across multiple client actions or across sessions.

6.2. Unified Web application development

Several recently proposed languages provide a unified programming model for implementing applications that span the multiple tiers found in Web applications. However, none of these languages helps the user automatically satisfy security requirements, nor do they support replication for improved interactive performance.

Links[5] and Hop[19] are functional languages for writing Web applications. Both allow code to be marked as client-side code, causing it to be translated to JavaScript. Links does this at the coarse granularity of individual functions, whereas Hop allows individual expressions to be partitioned. Links supports partitioning program code into SQL database queries, whereas Hop and Swift do not. Swift does not have language support for database manipulation, though a back-end database can be made accessible by wrapping it with a Jif signature. Neither Links nor Hop helps the programmer decide how to partition code securely.

Hilda[24] is a high-level declarative language for developing data-driven Web applications. It supports automatic partitioning with approximate performance optimization based on linear programming. It does not support or enforce security policies, or replicate code or data. Hilda's programming model is based on SQL and is only suitable for data-driven applications, as opposed to Swift's more general Java-based programming model.

A number of popular Web application development environments make Web application development easier by allowing a higher-level language to be embedded into HTML code. For example, JSP[2] embeds Java code, and PHP[18] and Ruby on Rails[21] embed their respective languages. None of these systems help to manage code placement, or help to decide when client–server communication is secure, or provide fully interactive user interfaces (unless JavaScript is used directly). Programming is still awkward, and reasoning about security is challenging.

The GWT[18] makes construction of client-side code easier by compiling Java to JavaScript, and gives a clean Ajax interface. GWT neither unifies programming across the client–server boundary, nor addresses security.

6.3. Automatic partitioning

That performance or security can be improved by partitioning applications across distributed computing systems is an old idea and certainly not original to Swift. Localizing security-critical functionality to trusted components has been explored in limited contexts (e.g., Balfanz[1]). Coign[11] partitions general systems automatically at the component level, though not according to information security policies.

The key feature of Swift is that it provides security by construction: the programmer specifies security requirements, and the system transforms the program to ensure that these requirements are met. Some prior work has explored this idea in other contexts.

The Jif/split system[25, 26] also uses Jif as a source language and transforms programs by placing code and data onto sets of hosts in accordance with the labels in the source code. Jif/split addresses the general problem of distributed computation in a system incorporating mutual distrust and arbitrary host trust relationships. Swift differs in exploring the challenges and opportunities of Web applications. Swift uses specialized construction techniques that exploit the trust assumptions it is based on. Supporting mutual distrust between the client and server would be an interesting extension; for example, client state could be protected in cookies. Replication is used by Jif/split only to boost integrity, whereas Swift uses replication also to improve responsiveness.

Fournet and Rezk[7] have shown that simple imperative programs annotated with confidentiality and integrity policies, as in Jif/split, can be compiled into distributed, cryptographic implementations that soundly enforce these policies.

7. CONCLUSION

We have shown that it is possible to build Web applications that enforce security by construction, resulting in greater security assurance. Further, Swift automatically takes care of some awkward tasks: partitioning application functionality across the client–server boundary, and designing protocols for exchanging information. Thus, Swift satisfies three important goals: enforcement of information security; a dynamic, responsive user interface; and a uniform, general-purpose programming model. No prior system delivers these capabilities.

What is needed for technology like Swift to be adopted widely? The two biggest impediments to adoption are first, the added annotation burden of writing security annotations, and second, the inefficiency of bulk data transfers between the server and the client.

Jif security annotations are mostly needed in method declarations, where they augment the information specified in existing type annotations. The Jif compiler is able to automatically infer most other annotations. In our experience, the annotation burden is less than the burden of managing client–server communication explicitly, even ignoring the effort that should be expended on manually reasoning about security. More sophisticated type inference algorithms might lessen the annotation burden further.

Transferring data in bulk between the client and the server can be less efficient than with a hand-coded implementation, because data transfers are tied to control transfers. The Swift implementation ensures the client and server stacks and heaps are in sync at each control transfer. It seems likely that Swift could be extended with mechanisms for delaying and securely batching updates to objects.

Because Web applications are being used for so many important purposes, better methods are needed for building them securely. Swift embodies a promising approach for this important problem.

Acknowledgments

We thank the SOSP reviewers, David Mazières, and Martín Abadi for insightful comments and useful suggestions.

This work was supported in part by the National Science Foundation under grants 0430161 and 0627649, in part by a grant from Microsoft Corporation, and in part by AF-TRUST (Air Force Team for Research in Ubiquitous Secure Technology for GIG/NCES), which receives support from the DAF Air Force Office of Scientific Research (FA9550-06-1-0244) and the NSF (0424422). C

References

1. Balfanz, D., Felten, E. Hand-held computers can be better smart cards. In *Proceedings of the 8th USENIX Security Symposium* (August 1999).
2. Bergsten, H. *JavaServer Pages*, 3rd edition. O'Reilly & Associates, 2003.
3. Chong, S. Liu, J., Myers, A.C., Qi, X., Vikram, K., Zheng, L., Zheng, X. Secure web applications via automatic partitioning. In *Proceedings of the 21st ACM Symposium on Operating System Principles* (*SOSP*) (October 2007).
4. Chong, S., Myers, A.C. Decentralized robustness. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop* (July 2006), 242–253.
5. Cooper, E., Lindley, S., Wadler, P., Yallop, J. Links: Web programming without tiers. In *Proceedings of the 5th International Symposium on Formal Methods for Components and Objects* (November 2006).
6. Flanagan, D. *JavaScript: The Definitive Guide*, 4th edition. O'Reilly, 2002.
7. Fournet, C., Rezk, T. Cryptographically sound implementations for typed information-flow security. In *IEEE Symposium on Computer Security Foundations* (June 2008), 323–335.
8. Google Web Toolkit. http://code.google.com/webtoolkit/.
9. Halfond, W., Orso, A. AMNESIA: Analysis and monitoring for neutralizing SQL-injection attacks. In *Proceedings of the International Conference on Automated Software Engineering* (*ASE'05*) (November 2005), 174–183.
10. Huang, Y.-W., Yu, F., Hang, C., Tsai, C.-H., Lee, D.-T., Kuo, S.-Y. Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th International World Wide Web Conference (WWW'04)* (May 2004), 40–52.
11. Hunt, G.C., Scott, M.L. The Coign automatic distributed partitioning system. In *OSDI '99: Proceedings of the 3rd Symposium on Operating Systems Design and Implementation* (1999), 187–200.
12. Jovanovic, N., Kruegel, C., Kirda, E. Pixy: A static analysis tool for detecting web application vulnerabilities. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2006), 258–263.
13. Myers, A.C. JFlow: Practical mostly-static information flow control. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages* (*POPL*) (January 1999), 228–241.
14. Myers, A.C., Liskov, B. Protecting privacy using the decentralized label model. *ACM Trans. Software Eng. Methodology 9*, 4 (Oct. 2000), 410–442.
15. Myers, A.C., Zheng, L., Zdancewic, S., Chong, S., Nystrom, N. Jif 3.0: Java information flow. Software release. http://www.cs.cornell.edu/jif, July 2006.
16. Nguyen-Tuong, A., Guarneri, S., Greene, D., Evans, D. Automatically hardening web applications using precise tainting. In *Proceedings of the 20th International Information Security Conference* (May 2005), 372–382.
17. Nystrom, N., Clarkson, M. R., Myers, A.C. Polyglot: An extensible compiler framework for Java. In *Proceedings of the 12th International Compiler Construction Conference* (*CC'03*) (April 2003), LNCS 2622, 138–152.
18. PHP: hypertext processor. http://www.php.net.
19. Serrano, M., Gallesio, E., Loitsch, F. HOP, a language for programming the Web 2.0. In *Proceedings of the 1st Dynamic Languages Symposium* (October 2006), 975–985.
20. Symantec Internet security threat report, volume X. Symantec Corporation, September 2006.
21. Thomas, D., Fowler, C., Hunt, A. *Programming Ruby: The Pragmatic Programmers' Guide*. The Pragmatic Programmers, 2nd edition, 2004. ISBN 0-974-51405-5.
22. Xie, Y., Aiken, A. Static detection of security vulnerabilities in scripting languages. In *Proceedings of the 15th USENIX Security Symposium* (July 2006), 179–192.
23. Xu, W., Bhatkar, S., Sekar, R. Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks. In *Proceedings of the 15th USENIX Security Symposium* (August 2006), 121–136.
24. Yang, F., Gupta, N., Gerner, N., Qi, X., Demers, A., Gehrke, J., Shanmugasundaram, J. A unified platform for data driven web applictions with automatic client–server partitioning. In *Proceedings of the 16th International World Wide Web Conference (WWW'07)* (2007), 341–350.
25. Zdancewic, S., Zheng, L., Nystrom, N., Myers, A.C. Secure program partitioning. *ACM Trans. Comput. Syst. 20*, 3 (Aug. 2002), 283–328.
26. Zheng, L., Chong, S., Myers, A.C., Zdancewic, S. Using replication and partitioning to build secure distributed systems. In *Proceedings of the IEEE Symposium on Security and Privacy* (Oakland, California, May 2003), 236–250.

Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng ({schong,liujed,andru, qixin,kvikram,zlt,xinz}@cs.cornell.edu), Department of Computer Science Cornell University.

DOI:10.1145/1461928.1461950

Technical Perspective
The Complexity of Computing Nash Equilibrium

By Ehud Kalai

Computer science and game theory go back to the same individual, John von Neumann, and both subjects deal with the mathematization of rational decision making. Yet, for many years they continued to work apart. Computer science concentrated mostly on issues of complexity; game theory mostly on issues of incentives.

But in the last dozen years we have seen a fusion of the two fields. Methodologies of each are used to solve problems in the other, and the concepts of each are incorporated to create better and more robust models in the other.

Nash equilibrium, introduced in the 1950s, is the main concept used in the analysis of strategic games. The equilibrium is simple to describe, logically appealing, powerful in making predictions, and its existence was brilliantly demonstrated by John Nash.

Despite its importance, research in the possibility of computing Nash equilibrium for games with many strategies only began in the late 1980s with studies by Gilboa and Zemel. This question was taken on full force by Papadimitriou and coauthors in a series of breakthrough papers. The research presented here is central to this literature, and (together with a follow-up paper by Chen and Deng for the case n=2) it offers a complete negative result. Computing a Nash equilibrium of an arbitrary n-person non-cooperative game with many individual strategies is at least as difficult as any problem that belongs to the class of PPAD-complete problems believed, and is considered too difficult for practical computations.

The message to users of Nash equilibrium may be devastating, as they ask: "If my computer cannot compute it, how can players in the market do it?" The negative results the authors report here even apply to algorithms where players can be centrally coordinated, and in the situations being modeled, the problem is actually even more intractable, since a solution must be found by a dispersed, uncoordinated set of players.

But as is often the case with impossibility results, this one leads to a large variety of follow-up questions dealing with the assumptions, the methodology, and the relevance.

The standard worst-case approach employed by the authors here assumes that for every proposed algorithm, one should consider the most difficult n-person game to compute, and study the computation time as the number of individual strategies becomes arbitrarily large.

Contrary to this approach, it is argued that a game designed to defeat an algorithm is not likely to be natural in applications. As a result, we have seen alternative approaches and conclusions to the question of computing a Nash equilibrium in games with many strategies.

First, motivated by economic, political, and other applications, there have been studies of computations of Nash equilibrium for restricted classes of games: games with anonymous opponents, games with graphical structures, games with continuous and/or convex payoff functions, and more. As it turns out, in many of these games researchers are able to establish positive results.

There are positive results on computing—with a high probability—an equilibrium of a many-strategies randomly generated game. This approach, however, must assume a probability distribution by which games are generated, and the results may depend on the distribution.

Finally, an old established approach is to ignore the asymptotic nature of the question, and simply find algorithms that work well in practice (similar to the simplex algorithm for linear programming working well, despite its asymptotic algorithmic inefficiency). Recent research leads to algorithms that are efficient in computing equilibria of rich classes of test games.

Ehud Kalai is the James J. O'Connor Professor of Decision and Game Sciences and a professor of Mathematics in the Kellogg School of Management at Northwestern University, Evanston, IL He is also the director for The Center for Game Theory and Economic Behavior.

DOI:10.1145/1461928.1461951

The Complexity of Computing a Nash Equilibrium

By Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou

Abstract

How long does it take until economic agents converge to an equilibrium? By studying the complexity of the problem of computing a mixed Nash equilibrium in a game, we provide evidence that there are games in which convergence to such an equilibrium takes prohibitively long. Traditionally, computational problems fall into two classes: those that have a polynomial-time algorithm and those that are NP-hard. However, the concept of NP-hardness cannot be applied to the rare problems where "every instance has a solution"—for example, in the case of games Nash's theorem asserts that every game has a mixed equilibrium (now known as the Nash equilibrium, in honor of that result). We show that finding a Nash equilibrium is complete for a class of problems called PPAD, containing several other known hard problems; all problems in PPAD share the same style of proof that every instance has a solution.

1. INTRODUCTION

In a recent CACM article, Shoham[22] reminds us of the long relationship between Game Theory and Computer Science, going back to John von Neumann at Princeton in the 1940s, and how this connection became stronger and more crucial in the past decade due to the advent of the Internet: Strategic behavior became relevant to the design of computer systems, while much economic activity now takes place on computational platforms.

Game Theory is about the strategic behavior of rational agents. It studies *games*, thought experiments modeling various situations of conflict. One commonly studied model aims to capture two players interacting in a single round. For example, the well-known school yard game of *rock–paper–scissors* can be described by the mathematical game shown in Figure 1. There are two players, one choosing a row and one choosing a column; the choices of a player are his/her *actions*. Once the two players choose, simultaneously, an action, they receive the corresponding payoffs shown in the table: The first number denotes the payoff of Row, the second that of Column. Notice that each of these pairs of numbers sum to zero in the case of Figure 1; such games are called *zero-sum games*. Three other well-known games, called *chicken, prisoner's dilemma*, and *penalty shot game*, respectively, are shown in Figure 2; the penalty shot game is zero-sum, but the other two are not. All these games have two players; Game Theory studies games with many players, but these are harder to display.[a]

The purpose of games is to help us understand economic behavior by predicting how players will act in each particular game. The predictions game theorists make about player behavior are called *equilibria*. One such prediction is the *pure Nash equilibrium*: Each player chooses an action that is a "best response" to the other player's choice—i.e., it is the highest payoff, for the player, in the line, row or column, chosen by the other player. In the game of chicken in Figure 2, a pure Nash equilibrium is when one player chooses "dare" and the other chooses "chicken." In the prisoner's dilemma, the only pure Nash equilibrium is when both players choose "defect."

Unfortunately, not all games have a pure Nash equilibrium. For example, it is easy to see that the rock–paper–scissors game in Figure 1 has none. This lack of universality is an important defect of the concept of pure Nash equilibrium as a predictor of behavior. But the rock–paper–scissors game does have a more sophisticated kind of equilibrium, called a *mixed Nash equilibrium*—and in fact one that is familiar to all who have played this game: both players pick an action uniformly at random. That is, a mixed Nash equilibrium is a probabilistic distribution on the set of actions of each player. Each of the distributions should have the property that it is a best response to the other distributions; this means that each action assigned positive probability is among the actions that are best responses, in expectation, to the distribution(s) chosen by the opponent(s).

In 1950, John Nash proved that *all games have a mixed Nash equilibrium*.[19] That is, in any game, distributions over the players' actions exist such that each is a best response to what everybody else is doing. This important—and far from obvious—universality theorem established the mixed Nash equilibrium as Game Theory's central equilibrium concept, the baseline and gold standard against which all

Figure 1: Rock–paper–scissors.

	rock	paper	scissors
rock	(0, 0)	(-1, 1)	(1, -1)
paper	(1, -1)	(0, 0)	(-1, 1)
scissors	(-1, 1)	(1, -1)	(0, 0)

[a] How about games such as chess? We can capture this and other similar games in the present framework by considering two players, Black and White, each with a huge action set containing all possible maps from positions to moves; but of course, such formalism is not very helpful for analyzing chess and similar games.

A previous version of this paper appeared in the *ACM 2006 Proceedings of the Symposium on Theory of Computing.*

Figure 2: Three other two-player games.

	chicken	dare
chicken	(0, 0)	(–5, 1)
dare	(1, –5)	(–10, –10)

In the *chicken* game above, there are two Nash equilibria, in which one player chooses "chicken," and the other player "dare." There is also a mixed equilibrium, in which each player makes a random choice that equalizes the expected payoffs to the opponent, of either of the opponent's actions.

	cooperate	defect
cooperate	(0, 0)	(–5, 1)
defect	(1, –5)	(–4, –4)

In the *prisoner's dilemma* game above, there is just one Nash equilibrium, in which both players defect. This is despite the fact that each player does better when they both cooperate.

	kick left	kick right
dive left	(1, –1)	(–1, 1)
dive right	(–1, 1)	(1, –1)

In the *penalty shot* game above, there is just one Nash equilibrium which is mixed, and in which both the goalkeeper and the penalty kicker choose left or right at random.

subsequent refinements and competing equilibrium concepts were judged.

Universality is a desirable attribute for an equilibrium concept. Of course, such a concept must also be natural and credible as a prediction of behavior by a group of agents—for example, pure Nash seems preferable to mixed Nash, in games that do have a pure Nash equilibrium. But there is a third important desideratum on equilibrium concepts, of a computational nature: *An equilibrium concept should be efficiently computable* if it is to be taken seriously as a prediction of what a group of agents will do. Because, if computing a particular kind of equilibrium is an intractable problem, of the kind that take lifetimes of the universe to solve on the world's fastest computers, it is ludicrous to expect that it can be arrived at in real life. This consideration suggests the following important question: *Is there an efficient algorithm for computing a mixed Nash equilibrium?* In this article, we report on results that indicate that the answer is *negative*—our own work [5, 7, 8, 13] obtained this for games with three or more players, and shortly afterwards, the papers[2, 3] extended this—unexpectedly—to the important case of two-player games.

Ever since Nash's paper was published in 1950, many researchers have sought algorithms for finding mixed Nash equilibria—that is, for solving the computational problem which we will call NASH in this paper. If a game is zero-sum, like the rock–paper–scissors game, then it follows from the work of John von Neumann in the 1920s that NASH can be formulated in terms of linear programming (a subject identified by George Dantzig in the 1940s); linear programs can be solved efficiently (even though we only realized this in the 1970s). But what about games that are not zero-sum? Several algorithms have been proposed over the past half century, but all of them are either of unknown complexity, or known to require, in the worst case, exponential time.

During the same decades that these concepts were being explored by game theorists, Computer Science theorists were busy developing, independently, a theory of algorithms and complexity addressing precisely the kind of problem raised in the last two paragraphs: Given a computational problem, can it be solved by an efficient algorithm? For many common computational tasks (such as finding a solution of a set of linear equations) there is a polynomial-time algorithm that solves them—this class of problems is called **P**. For other such problems, such as finding a truth assignment that satisfies a set of Boolean clauses (a problem known as SAT), or the traveling salesman problem, no such algorithm could be found after many attempts. Many of these problems can be proved **NP**-complete, meaning they cannot be solved efficiently unless **P** = **NP**—an event considered very unlikely.[11]

From the previous discussion of failed attempts to develop an efficient algorithm for NASH, one might be tempted to suppose that this problem too is **NP**-complete. But the situation is not that simple. NASH is unlike any **NP**-complete problem because, by Nash's theorem, *it is guaranteed to always have a solution*. In contrast, **NP**-complete problems like SAT draw their intractability from the possibility that a solution might not exist—this possibility is used heavily in the **NP**-completeness proof.[b] See Figure 3 for an argument (due to Nimrod Megiddo) why it is very unlikely that **NP**-completeness can characterize the complexity of NASH. (Note however that if one seeks a Nash equilibrium with additional properties—such as the one that maximizes the sum of player utilities, or one that uses a given strategy with positive probability—then the problem does become **NP**-complete.[4, 12])

Since **NP**-completeness is not an option, to understand the complexity of NASH one must essentially start all over in

Figure 3: Megiddo's proof that NASH is unlikely to be NP-complete.

Suppose we have a reduction from SAT to NASH, that is, an efficient algorithm that takes as input an instance of SAT and outputs an instance of NASH, so that any solution to the instance of NASH tells us whether or not the SAT instance has a solution. Then we could turn this into a nondeterministic algorithm for verifying that an instance of SAT has *no* solution: Just guess a solution of the NASH instance, and check that it indeed implies that the SAT instance has no solution.

The existence of such a nondeterministic algorithm for SAT (one that can verify that an unsatisfiable formula is indeed unsatisfiable) is an eventuality that is considered by complexity theorists almost as unlikely as **P** = **NP**. We conclude that NASH is very unlikely to be **NP**-complete.

[b] "But what about the TRAVELING SALESMAN PROBLEM?" one might ask. "Does not it always have a solution?" To compare fairly the TRAVELING SALESMAN PROBLEM with SAT and NASH, one has to first transform it into a search problem of the form "Given a distance matrix and a budget B, find a tour that is cheaper than B, or report that none exists". Notice that an instance of this problem may or may not have a solution. But, an efficient algorithm for this problem could be used to find an optimal tour.

the path that led us to **NP**-completeness: We must define a class of problems which contains, along with NASH, some other well-known hard problems, and then prove that NASH is complete for that class. Indeed, in this paper we describe a proof that NASH is **PPAD**-complete, where **PPAD** is a subclass of **NP** that contains several important problems that are suspected to be hard, including NASH.

1.1. Problem statement: Nash and approximate Nash equilibria

A *game in normal form* has some number k of players, and for each player p ($p \in \{1, ..., k\}$) a finite set S_p of pure *actions* or *strategies*. The set S of *pure strategy profiles* is the Cartesian product of the S_p's. Thus, a pure strategy profile represents a choice, for each player, of one of his actions. Finally, for each player p and $s \in S$ the game will specify a *payoff* or *utility* $u_s^p \geq 0$, which is the value of the outcome to player p when all the players (including p) choose the strategies in s. In a Nash equilibrium, players choose probability distributions over their S_p's, called *mixed strategies*, so that no player can deviate from his mixed strategy and improve on his expected payoff; see Figure 4 for details.

For two-player games there always exists a Nash equilibrium in which the probabilities assigned to the various strategies of the players are rational numbers—assuming the utilities are also rational. So, it is clear how to write down such a solution of a two-player game. However, as pointed out in Nash's original paper, when there are more than two players, there may be only irrational solutions. In this general situation, the problem of computing a Nash equilibrium has to deal with issues of numerical accuracy. Thus, we introduce next the concept of approximate Nash equilibrium.

If Nash equilibrium means "no incentive to deviate," then *approximate Nash equilibrium* stands for "low incentive to deviate." Specifically, if ε is a small positive quantity, we can define an ε-Nash equilibrium as a profile of mixed strategies where any player can improve his expected payoff by at most ε by switching to another strategy. Figure 4 gives a precise definition, and shows how the problem reduces to solving a set of algebraic inequalities. Our focus on approximate solutions is analogous to the simpler problem of polynomial root-finding. Suppose that we are given a polynomial f with a single variable, and we have to find a real root, a real number r satisfying $f(r) = 0$. In general, a solution to this problem (the number r) cannot be written down as a fraction, so we should really be asking for some sort of numerical approximation to r (for example, computing a rational number r such that $|f(r)| \leq \varepsilon$, for some small ε). If f happens to have odd degree, we can even say in advance that a solution must exist, in a further analogy with NASH. Of course, the analogy breaks down in that for root-finding we know of efficient algorithms that solve the problem, whereas for Nash equilibria we do not.

We are now ready to define the computational problem NASH: Given the description of a game (by explicitly giving the utility of each player corresponding to each strategy profile) and a rational number $\varepsilon > 0$, compute an ε-Nash equilibrium. This should be at least as tractable as finding an exact equilibrium, hence any hardness result for approximate equilibria carries over to exact equilibria. Note that an approximate equilibrium as defined above need not be at all close to an exact equilibrium; see Etessami and Yannakakis[9] for a complexity theory of exact Nash equilibria.

2. TOTAL SEARCH PROBLEMS

We think of **NP** as the class of search problems of the form "Given an input, find a solution (which then can be easily checked) or report that none exists." There is an asymmetry between these outcomes, in that "none exists" is *not* required to be easy to verify.

We call such a search problem *total* if the solution always exists. There are many apparently hard *total search problems* in **NP**—even though, as we argued in the introduction, they are unlikely to be **NP**-complete. Perhaps the best-known is FACTORING, the problem of taking an integer as an input and outputting its prime factors. NASH and several other problems introduced below are also total.

A useful classification of total search problems was proposed in Papadimitriou.[20] The idea is this: If a problem is total, the fact that every instance has a solution must have a mathematical proof. Unless the problem can be easily solved efficiently, in that proof there must be a "nonconstructive

Figure 4: Writing down the problem algebraically.

Recall that a game is specified by the payoffs associated with each pure strategy profile s, so that for some player p and $s \in S$, $u_s^p \geq 0$ denotes p's payoff from s. The set of pure strategy profiles of all players other than p is denoted by S_{-p}. For $j \in S_p$ and $s' \in S_{-p}$, let $u_{js'}^p$ be the payoff to p when p plays j and the other players play s'.

The problem of finding a Nash equilibrium boils down to finding a set of numbers x_j^p that satisfy the expressions below. x_j^p will be the probability that p plays j, so for these quantities to be valid probabilities we require, for each player p,

$$x_j^p \geq 0 \quad \text{and} \quad \sum_{j \in S_p} x_j^p = 1. \tag{1}$$

For a set of k mixed strategies to be a Nash equilibrium, we need that, for each p, $\Sigma_{s \in S} u_s^p x_s$ is maximized over all mixed strategies of p—where for a strategy profile $s = (s_1, ..., s_k) \in S$, we denote by x_s the product $x_{s_1}^1 \cdot x_{s_2}^2, ..., x_{s_k}^k$. (The expresssion $\Sigma_{s \in S} u_s^p x_s$ represents p's expected payoff.) That is, a Nash equilibrium is a set of mixed strategies from which no player has a unilateral incentive to deviate. It is well known[20] that the following is an equivalent condition for a set of mixed strategies to be a Nash equilibrium:

$$\sum_{s \in S_{-p}} u_{js}^p x_s > \sum_{s \in S_{-p}} u_{j's}^p x_s \Rightarrow x_{j'}^p = 0. \tag{2}$$

The summation $\Sigma_{s \in S_{-p}} u_{js}^p x_s$ in the above equation is the expected utility of player p if p plays pure strategy $j \in S_p$ and every other player q uses the mixed strategy $\{x_j^q\}_{j \in S_q}$.

We next turn to approximate notions of equilibrium. We say that a set of mixed strategies x is an *ε-approximately well supported Nash equilibrium*, or *ε-Nash equilibrium* for short, if for each p the following holds:

$$\sum_{s \in S_{-p}} u_{js}^p x_s > \sum_{s \in S_{-p}} u_{j's}^p x_s + \varepsilon \Rightarrow x_{j'}^p = 0. \tag{3}$$

Condition (3) relaxes (2) by allowing a strategy to have positive probability in the presence of another strategy whose expected payoff is better by at most ε.

step." It turns out that, for all known total search problems in the fringes of **P**, these nonconstructive steps are one of very few simple arguments:

- "If a graph has a node of odd degree, then it must have another." This is the *parity argument*, giving rise to the class **PPA**.
- "If a directed graph has an unbalanced node (a vertex with different in-degree and out-degree), then it must have another." This is the *parity argument for directed graphs*, giving rise to the class **PPAD** considered in this article. Figure 5 describes the corresponding search problems.
- "Every directed acyclic graph must have a sink." The corresponding class is called **PLS** for *polynomial local search*.
- "If a function maps n elements to $n-1$ elements, then there is a collision." This is the *pigeonhole principle*, and the corresponding class is **PPP**.

We proceed with defining more precisely the second class in the list above.

2.1. The class PPAD

There are two equivalent ways to define **NP**: First, it is the class of all search problems whose answers are verifiable in polynomial time. For example, the search problem SAT ("Given a Boolean formula in CNF, find a satisfying truth assignment, or report that none exists") is in **NP** because it is easy to check whether a truth assignment satisfies a CNF. Since we know that SAT is **NP**-complete, we can also define **NP** as the class of all problems that can be reduced into instances of SAT. By "reduce" we refer to the usual form of polynomial-time reduction from search problem A to search problem B: An efficient algorithm for transforming any instance of A to an equivalent instance of B, together with an efficient algorithm for translating any solution of the instance of B back to a solution of the original instance of A.

We define the class **PPAD** using the second strategy. In particular, **PPAD** *is the class of all search problems that can be reduced to the problem* END OF THE LINE, defined in Figure 5. Note that, since END OF THE LINE is a total problem, so are all problems in **PPAD**. Proceeding now in analogy with **NP**, we call a problem **PPAD**-*complete* if END OF THE LINE (and therefore all problems in **PPAD**) can be reduced to it.

2.2. Why should we believe that PPAD contains hard problems?

In the absence of a proof that $\mathbf{P} \neq \mathbf{NP}$ we cannot hope to be sure that **PPAD** contains hard problems. The reason is that **PPAD** lies "between **P** and **NP**" in the sense that, if $\mathbf{P} = \mathbf{NP}$, then **PPAD** itself, as a subset of **NP**, will be equal to **P**. But even if $\mathbf{P} \neq \mathbf{NP}$, it may still be the case that **PPAD**-complete problems are easy to solve. We believe that **PPAD**-complete problems are hard for the same reasons of computational and mathematical experience that convince us that **NP**-complete problems are hard (but as we mentioned, our confidence is necessarily a little weaker): **PPAD** contains many problems for which researchers have tried for decades to develop efficient algorithms; in the next section we introduce one such problem called BROUWER. However, END OF THE LINE itself is a pretty convincingly hard problem: How can one hope to devise an algorithm that telescopes exponentially long paths in every implicitly given graph?

Figure 5: END OF THE LINE: An apparently hard total search problem.

Let us say that a vertex in a directed graph is "unbalanced" if the number of its incoming edges differs from the number of its outgoing edges. Observe that, *given a directed graph and an unbalanced vertex, there must exist at least one other unbalanced vertex.* This is the parity argument on directed graphs. (**PPAD** stands for "polynomial parity argument for directed graphs.") Hence, the following is a total search problem:

Input: A directed graph G and a specified unbalanced vertex of G.
Output: Some other unbalanced vertex.

Note that, before we even begin to search G, the parity argument assures us that we are searching for something that really exists. Now, if G were presented in the form of a list of its vertices and edges, the problem could of course be solved efficiently. Suppose however that we are given a graph that is too large to be written out in full, but must be represented by a program that tells us whether an edge exists or not.

To be specific, suppose G has 2^n vertices, one for every bit string of length n (the parameter denoting the size of the problem). For simplicity, we will suppose that every vertex has at most one incoming edge and at most one outgoing edge. The edges of G will be represented by two Boolean circuits, of size polynomial in n, each with n input bits. Denote the circuits P and S (for predecessor and successor). Our convention is that there is a directed edge from vertex v to vertex w, if, given input v, S outputs w and, vice versa, given input w, P outputs v. Suppose now that some specific, identified vertex (say, the string 00…0) has an outgoing edge but no incoming edge, and is thus unbalanced. Since there is at most one incoming and one outgoing edge per node, the directed graph must be a set of paths and cycles; hence, following the path that starts at the all-zeroes node would eventually lead us to a solution. The catch is, of course, that this may take exponential time. Is there an efficient algorithm for finding another unbalanced node without actually following the path?

3. FROM NASH TO PPAD

Our main result is the following:

THEOREM 3.1. NASH *is* **PPAD**-*complete*.

In the remainder of this article we outline the main ideas of the proof; for full details see Daskalakis et al.[8] We need to prove two things: First, that NASH is in **PPAD**, that is, it can be reduced to END OF THE LINE. Second (see Section 4), that it is complete—the reverse reduction. As it turns out, both directions are established through a computational problem inspired by a fundamental result in topology, called *Brouwer's fixed point theorem*, described next.

3.1. Brouwer's fixed point theorem

Imagine a continuous function mapping a circle (together with its interior) to itself—for example, a rotation around the center. Notice that the center is fixed, it has not moved under this function. You could flip the circle—but then all points on a diagonal would stay put. Or you could do something more elaborate: Shrink the circle, translate it (so it still lies within the original larger circle), and then rotate it. A little thought reveals that there is still at least one fixed point. Or stretch and compress the circle like a sheet of rubber any way you want and stick it on the original circle; still points will be fixed, unless of course you tear the circle—the function

must be continuous. There is a topological reason why you cannot map continuously the circle on itself without leaving a point unmoved, and that's Brouwer's theorem.[16] It states that any continuous map from a *compact* (that is, closed and bounded) and *convex* (that is, without holes) subset of the Euclidean space into itself always has a *fixed point*.

Brouwer's theorem immediately suggests an interesting computational total search problem, called BROUWER: Given a continuous function from some compact and convex set to itself, find a fixed point. But of course, for a meaningful definition of BROUWER we need to first address two questions: How do we specify a continuous map from some compact and convex set to itself? And how do we deal with irrational fixed points?

First, we fix the compact and convex set to be the unit cube $[0, 1]^m$—in the case of more general domains, for example, the circular domain discussed above, we can translate it to this setting by shrinking the circle, embedding it into the unit square, and extending the function to the whole square so that no new fixed points are introduced. We then assume that the function F is given by an efficient *algorithm* Π_F which, for each point x of the cube written in binary, computes $F(x)$. We assume that F obeys a Lipschitz condition:

$$\text{for all } x_1, x_2 \in [0,1]^m : d(F(x_1), F(x_2)) \leq K \cdot d(x_1, x_2), \qquad (4)$$

where $d(\cdot,\cdot)$ is the Euclidean distance and K is the *Lipschitz constant* of F. This benign well-behavedness condition ensures that approximate fixed points can be localized by examining the value $F(x)$ when x ranges over a discretized grid over the domain. Hence, we can deal with irrational solutions in a similar maneuver as with NASH, by only seeking approximate fixed points. In fact, we have the following strong guarantee: for any ε, there is an ε-approximate fixed point—that is, a point x such that $d(F(x), x) \leq \varepsilon$—whose coordinates are integer multiples of 2^{-d}, where d depends on K, ε, and the dimension m; in the absence of a Lipschitz constant K, there would be no such guarantee and the problem of computing fixed points would become intractable. Formally, the problem BROUWER is defined as follows.

BROUWER

Input: An efficient algorithm Π_F for the evaluation of a function F: $[0, 1]^m \to [0, 1]^m$; a constant K such that F satisfies (4); and the desired accuracy ε.

Output: A point x such that $d(F(x), x) \leq \varepsilon$.

It turns out that BROUWER is in **PPAD**. (Papadimitriou[20] gives a similar result for a more restrictive class of Brouwer functions.) To prove this, we will need to construct an END OF THE LINE graph associated with a BROUWER instance. We do this by constructing a mesh of tiny triangles over the domain, where each triangle will be a vertex of the graph. Edges, between pairs of adjacent triangles, will be defined with respect to a coloring of the vertices of the mesh. Vertices get colored according to the direction in which F displaces them. We argue that if a triangle's vertices get all possible colors, then F is trying to shift these points in conflicting directions, and we must be close to an approximate fixed point. We elaborate on this in the next few paragraphs, focusing on the two-dimensional case.

Triangulation: First, we subdivide the unit square into small squares of size determined by ε and K, and then divide each little square into two right triangles (see Figure 7, ignoring for now the colors, shading, and arrows). (In the m-dimensional case, we subdivide the m-dimensional cube into m-dimensional cubelets, and we subdivide each cubelet into the m-dimensional analog of a triangle, called an *m-simplex*.)

Coloring: We color each vertex x of the triangles by one of three colors depending on the *direction* in which F maps x. In two dimensions, this can be taken to be the angle between vector $F(x) - x$ and the horizontal. Specifically, we color it red if the direction lies between 0 and –135°, blue if it ranges between 90 and 225°, and yellow otherwise, as shown in Figure 6. (If the direction is 0°, we allow either yellow or red; similarly for the other two borderline cases.) Using the above coloring convention the vertices get colored in such a way that the following property is satisfied:

(P1): None of the vertices on the lower side of the square uses red, no vertex on the left side uses blue, and no vertex on the other two sides uses yellow. Figure 7 shows a coloring of the vertices that could result from the function F; ignore the arrows and the shading of triangles.

Sperner's Lemma: It now follows from an elegant result in Combinatorics called Sperner's Lemma[20] that, in any coloring satisfying Property (P1), there will be at least one small triangle whose vertices have all three colors (verify this in Figure 7; the trichromatic triangles are shaded). Because we have chosen the triangles to be small, any vertex of a trichromatic triangle will be an approximate fixed point. Intuitively, since F satisfies the Lipschitz condition given in (4), it cannot fluctuate too fast; hence, the only way that there can be three points close to each other in distance, which are mapped in three different directions, is if they are all approximately fixed.

The Connection with PPAD: ...is the proof of Sperner's Lemma. Think of all the triangles containing at least one red and yellow vertex as the nodes of a directed graph G. There is a directed edge from a triangle T to another triangle T' if T and T' share a red–yellow edge which goes from red to yellow clockwise in T (see Figure 7). The graph G thus created consists of paths and cycles, since for every T there is at most one T' and vice versa (verify this in Figure 7). Now, we may

Figure 6: The colors assigned to the different directions of $F(x) - x$. There is a transition from red to yellow at 0°, from yellow to blue at 90°, and from blue to red at 225°.

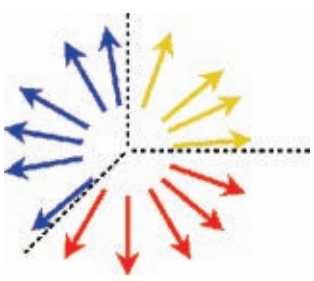


Figure 7: The subdivision of the square into smaller squares, and the coloring of the vertices of the subdivision according to the direction of $F(x) - x$. The arrows correspond to the END OF THE LINE graph on the triangles of the subdivision; the source T^* is marked by a diamond.

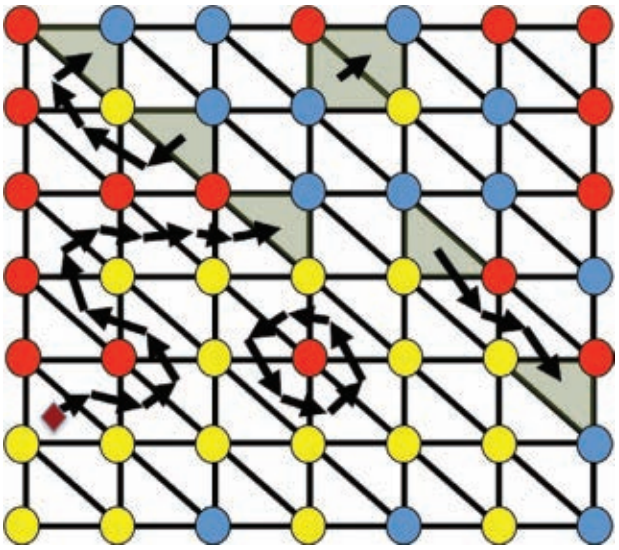
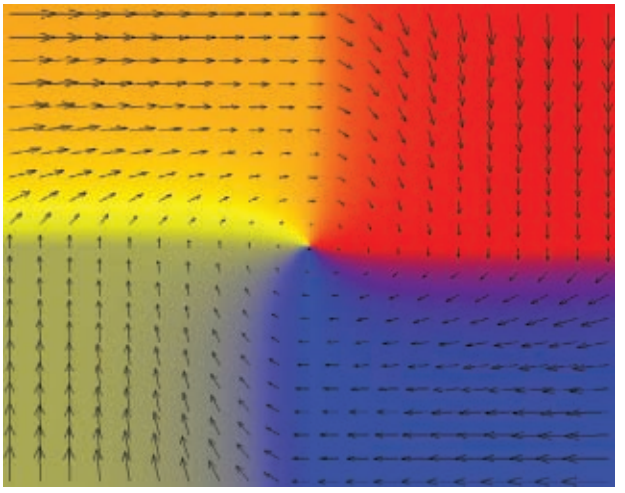


Figure 8: An illustration of Nash's function F_N for the penalty shot game. The horizontal axis corresponds to the probability by which the penalty kicker kicks right, and the vertical axis to the probability by which the goalkeeper dives left. The arrows show the direction and magnitude of $F_N(x) - x$. The unique fixed point of F_N is (1/2, 1/2) corresponding to the unique mixed Nash equilibrium of the penalty shot game. The colors respect Figure 6, but our palette here is continuous.



also assume: On the left side of the square there is only one change from yellow to red.[c] Under this assumption, let T^* be the unique triangle containing the edge where this change occurs (in Figure 7, T^* is marked by a diamond). Observe that, if T^* is not trichromatic (as is the case in Figure 7), then the path starting at T^* is guaranteed to have a sink, since it cannot intersect itself, and it cannot escape outside the square (notice that there is no red–yellow edge on the boundary that can be crossed outward). But, the only way a triangle can be a sink of this path is if the triangle is trichromatic. This establishes that there is at least one trichromatic triangle. (There may of course be other trichromatic triangles, which would correspond to additional sources and sinks in G, as in Figure 7.) G is a graph of the kind in Figure 5. To finish the reduction from BROUWER to END OF THE LINE, notice that given a triangle it is easy to compute its colors by invoking Π_F, and find its neighbors in G (or its single neighbor, if it is trichromatic).

Finally, from Nash to Brouwer: To finish our proof that NASH is in **PPAD** we need a reduction from NASH to BROUWER. Such a reduction was essentially given by Nash himself in his 1950 proof: Suppose that the players in a game have chosen some (mixed) strategies. Unless these already constitute a Nash equilibrium, some of the players will be unsatisfied, and will wish to change to some other strategies. This suggests that one can construct a "preference function" from the set of players' strategies to itself, that indicates the movement that will be made by any unsatisfied players. An example, of how such a function might look, is shown in Figure 8. A *fixed point* of such a function is a point that is mapped to itself—a Nash equilibrium. And Brouwer's fixed point theorem, explained above, guarantees that such a fixed point exists. In fact, it can be shown that an approximate fixed point corresponds to an approximate Nash equilibrium. Therefore, NASH reduces to BROUWER.

4. FROM PPAD BACK TO NASH

To show that NASH is complete for **PPAD**, we show how to convert an END OF THE LINE graph into a corresponding game, so that from an approximate Nash equilibrium of the game we can efficiently construct a corresponding end of the line. We do this in two stages. The graph is converted into a Brouwer function whose domain is the unit three-dimensional cube. The Brouwer function is then represented as a game. The resulting game has too many players (their number depends on the size of the circuits that compute the edges of the END OF THE LINE graph), and so the final step of the proof is to encode this game in terms of another game, with three players.

4.1. From paths to fixed points: The PPAD-completeness of BROUWER

We have to show how to encode a graph G, as described in Figure 5, in terms of a continuous, easy-to-compute Brouwer function F—a very different-looking mathematical object. The encoding is unfortunately rather complicated, but is the key to the **PPAD**-completeness result...

We proceed by, first, using the three-dimensional unit cube as the domain of the function F. Next, the behavior of F shall be defined in terms of its behavior on a (very fine) rectilinear mesh of "grid points" in the cube. Thus, each grid point lies at the center of a tiny "cubelet," and the behavior

[c] Suppose F gives rise to multiple yellow/red adjacencies on the left-hand side. We deal with this situation by adding an extra array of vertices to the left of the left side of the square, and color all these vertices red, except for the bottom one which we color yellow. This addition does not violate (P1) and does not create any additional trichromatic triangles since the left side of the square before the addition did not contain any blue.

of F away from the centers of the cubelets shall be gotten by interpolation with the closest grid points.

Each grid point x shall receive one of four "colors" $\{0, 1, 2, 3\}$, that represent the value of the three-dimensional displacement vector $F(x) - x$. The four possible vectors can be chosen to point away from each other such that $F(x) - x$ can only be approximately zero in the vicinity of all the four colors.

We are now ready to fit G itself into the above framework. Each of the 2^n vertices of G shall correspond with two special sites in the cube, one of which lies along the bottom left-hand edge in Figure 9 and the other one along the top left edge. (We use locations that are easy to compute from the identity of a vertex of G.) While most other grid points in the cube get color 0 from F, at all the special sites a particular configuration of the other colors appears. If G has an edge from node u to node v, then F shall also color a long sequence of points between the corresponding sites in the cube (as shown in Figure 9), so as to connect them with sequences of grid points that get colors 1, 2, and 3. The precise arrangement of these colors can be chosen to be easy to compute (using the circuits P and S that define G) and *such that all four colors are adjacent to each other (an approximate fixed point) only at sites that correspond to an "end of the line" of G*.

Having shown earlier that BROUWER is in **PPAD**, we establish the following:

THEOREM 4.1. BROUWER *is* **PPAD**-*complete*.

4.2. From BROUWER to NASH

The **PPAD**-complete class of Brouwer functions that we identified above have the property that their function F can be efficiently computed using arithmetic circuits that are built up using a small repertoire of standard operators such as addition, multiplication, and comparison. These circuits can be written down as a "data flow graph," with one of these operators at each node. In order to transform this into a game whose Nash equilibria correspond to (approximate) fixed points of the Brouwer function, we introduce players for every node on this data flow graph.

Games that Do Arithmetic: The idea is to simulate each arithmetic gate in the circuit by a game, and then compose the games to get the effect of composing the gates. The whole circuit is represented by a game with many players, each of whom "holds" a value that is computed by the circuit. We give each player two actions, "stop" and "go." To simulate, say, multiplication of two values, we can choose payoffs for three players x, y, and z such that, in any Nash equilibrium, the probability that z (representing the output of the multiplication) will "go" is equal to the product of the probabilities that x and y will "go." The resulting "multiplication gadget" (see Figure 10) has a fourth player w who mediates between x, y, and z. The directed edges show the direct dependencies among the players' payoffs. Elsewhere in the game, z may input his value into other related gadgets.

Here is how we define payoffs to induce the players to implement multiplication. Let X, Y, Z, and W denote the mixed strategies ("go" probabilities) of x, y, z, and w. We pay

Figure 9: Embedding the END OF THE LINE graph in a cube. The embedding is used to define a continuous function F, whose approximate fixed points correspond to the unbalanced nodes of the END OF THE LINE graph.

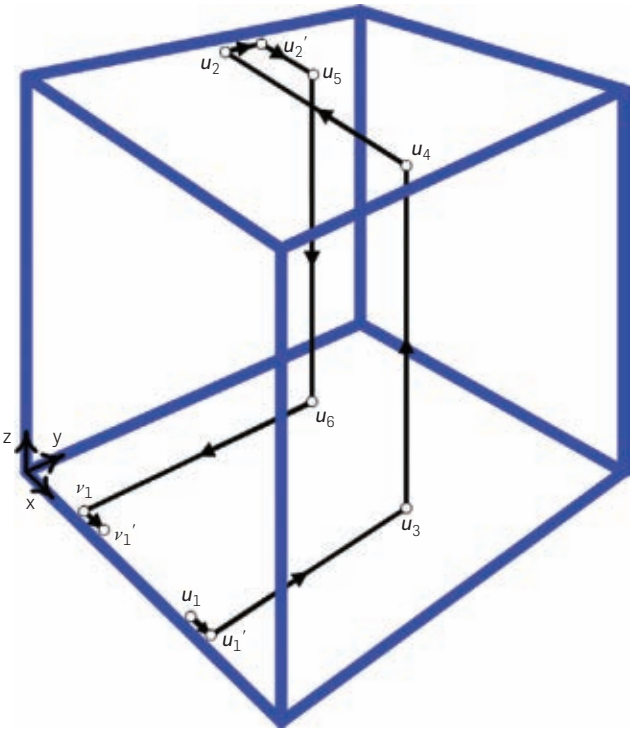
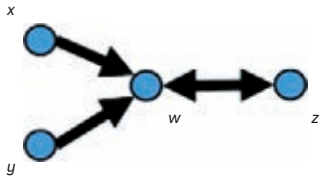


Figure 10: The players of the multiplication game. The graph shows which players affect other players' payoffs.



w the amount $\$X \cdot Y$ for choosing strategy *stop* and $\$Z$ for choosing *go*. We also pay z to play the opposite from player w. It is not hard to check that in any Nash equilibrium of the game thus defined, it must be the case that $Z = X \cdot Y$. (For example, if $Z > X \cdot Y$, then w would prefer strategy *go*, and therefore z would prefer *stop*, which would make $Z = 0$, and would violate the assumption $Z > X \cdot Y$.) Hence, the rules of the game induce the players to implement multiplication in the choice of their mixed strategies.

By choosing different sets of payoffs, we could ensure that $Z = X + Y$ or $Z = \frac{1}{2}X$. It is a little more challenging to simulate the *comparison* of two real values, which also is needed to simulate the Brouwer function. Below we discuss that issue in more detail.

Computing a Brouwer Function with Games: Suppose we have a Brouwer function F defined on the unit cube. Include three players x_1, x_2, and x_3 whose "go" probabilities represent a point x in the cube. Use additional players to compute

$F(x)$ via gadgets as described above. Eventually, we can end up with three players y_1, y_2, and y_3 whose "go" probabilities represent $F(x)$. Finally, we can give payoffs to x_1, x_2, and x_3 that ensure that in any Nash equilibrium, their probabilities agree with y_1, y_2, and y_3. Then, in any Nash equilibrium, these probabilities must be a fixed point of F.

The Brittle Comparator Problem: There's just one catch: our *comparator gadget*, whose purpose is to compare its inputs and output a binary signal according to the outcome of the comparison, is "brittle" in that if the inputs are equal then it outputs *anything*. This is inherent, because one can show that, if a nonbrittle comparator gadget existed, then we could construct a game that has no Nash equilibria, contradicting Nash's theorem. With brittle comparators, our computation of F is faulty on inputs that cause the circuit to make a comparison of equal values. We solve this problem by computing the Brouwer function at a grid of many points near the point of interest, and averaging the results, which makes the computation "robust," but introduces a small error in the computation of F. Therefore, the construction described above *approximately* works, and the three special players of the game have to play an approximate fixed point at equilibrium.

The Final Step: Three Players: The game thus constructed has many players (the number depends mainly on how complicated the program for computing the function F was), and two strategies for each player. This presents a problem: To represent such a game with n players we need $n2^n$ numbers—the utility of each player for each of the 2^n strategy choices of the n players. But our game has a special structure (called a *graphical game*, see Kearns et al.[15]): The players are vertices of a graph (essentially the data flow graph of F), and the utility of each player depends only on the actions of its neighbors.

The final step in the reduction is to simulate this game by a three-player normal form game—this establishes that NASH is **PPAD**-complete even in the case of three players. This is accomplished as follows: We color the players (nodes of the graph) by three colors, say red, blue, and yellow, so that no two players who play together, or two players who are involved in a game with the same third player, have the same color (it takes some tweaking and argument to make sure the nodes can be so colored). The idea is now to have three "lawyers," the red lawyer, the blue lawyer, and the yellow lawyer, each represent all nodes with their color, in a game involving only the lawyers. A lawyer representing m nodes has $2m$ actions, and his mixed strategy (a probability distribution over the $2m$ actions) can be used to encode the simpler stop/go strategies of the m nodes. Since no two adjacent nodes are colored the same color, the lawyers can represent their nodes without a "conflict of interest," and so a mixed Nash equilibrium of the lawyers' game will correspond to a mixed Nash equilibrium of the original graphical game.

But there is a problem: We need each of the "lawyers" to allocate equal amounts of probability to their customers; however, with the construction so far, it may be best for a lawyer to allocate more probability mass to his more "lucrative" customers. We take care of this last difficulty by having the lawyers play, on the side and for high stakes, a generalization of the rock–paper–scissors game of Figure 1, one that forces them to balance the probability mass allocated to the nodes of the graph. This completes the reduction from graphical games to three-player games, and the proof.

5. RELATED TECHNICAL CONTRIBUTIONS

Our paper[7] was preceded by a number of important papers that developed the ideas outlined here. Scarf's algorithm[21] was proposed as a general method for finding approximate fixed points, more efficiently than brute force. It essentially works by following the line in the associated END OF THE LINE graph described in Section 3.1. The Lemke–Howson algorithm[17] computes a Nash equilibrium for two-player games by following a similar END OF THE LINE path. The similarity of these algorithms and the type of parity argument used in showing that they work inspired the definition of **PPAD** in Papadimitriou.[20]

Three decades ago, Bubelis[1] considered reductions among games and showed how to transform any k-player game to a three-player game (for $k > 3$) in such a way that given any solution of the three-player game, a solution of the k-player game can be reconstructed with simple algebraic operations. While his main interest was in the algebraic properties of solutions, his reduction is computationally efficient. Our work implies this result, but our reduction is done via the use of graphical games, which are critical in establishing our **PPAD**-completeness result.

Only a few months after we announced our result, Chen and Deng[2, 3] made the following clever, and surprising, observation. The graphical games resulting from our construction are not using the multiplication operation (except for multiplication by a constant), and therefore can even be simulated by a *two-player* game, leading to an improvement of our hardness result from three- to two-player games. This result was unexpected, one reason being that the probabilities that arise in a two-player Nash equilibrium are always rational numbers, which is not the case for games with three or more players.

Our results imply that finding an ε-Nash equilibrium is **PPAD**-complete, if ε is inversely proportional to an exponential function of the game size. Chen et al.[3] extended this result to the case where ε is inversely proportional to a polynomial in the game size. This rules out a *fully polynomial-time approximation scheme* for computing approximate equilibria.

Finally, in this paper, we have focused on the complexity of computing an approximate Nash equilibrium. Etessami and Yannakakis[9] develop a very interesting complexity theory of the problem of computing the exact equilibrium (or other fixed points), a problem that is important in applications outside Game Theory, such as Program Verification.

6. CONCLUSION AND FUTURE WORK

Our hardness result for computing a Nash equilibrium raises concerns about the credibility of the mixed Nash equilibrium as a general-purpose framework for behavior

prediction. In view of these concerns, the main question that emerges is whether there exists a *polynomial-time approximation scheme* (PTAS) for computing approximate Nash equilibria. That is, is there an algorithm for ε-Nash equilibria which runs in time polynomial in the game size, if we allow arbitrary dependence of its running time on $1/\varepsilon$? Such an algorithm would go a long way towards alleviating the negative implications of our complexity result. While this question remains open, one may find hope (at least for games with a few players) in the existence of a subexponential algorithm[18] running in time $O(n^{\log n/\varepsilon^2})$, where n is the size of the game.

How about classes of concisely represented games with many players? For a class of "tree-like" graphical games, a PTAS has been given in Daskalakis and Papadimitriou,[6] but the complexity of the problem is unknown for more general low-degree graphs. Finally, another positive recent development[7] has been a PTAS for a broad and important class of games, called *anonymous*. These are games in which the players are oblivious to each other's identities; that is, each player is affected not by *who* plays each strategy, but by *how many* play each strategy. Anonymous games arise in many settings, including network congestion, markets, and social interactions, and so it is reassuring that in these games approximate Nash equilibria can be computed efficiently.

An alternative form of computational hardness, exemplified in Hart and Mansour,[14] arises where instead of identifying problems that are resistant to any efficient algorithm, one identifies problems that are resistant to specific "natural" algorithms. In Hart,[14] lower bounds are shown for "decoupled" dynamics, a model of strategic interaction in which there is no central controller to find an equilibrium. Instead, the players need to obtain one in a decentralized manner. The study and comparison of these models will continue to be an interesting research theme.

Finally, an overarching research question for the Computer Science research community investigating game-theoretic issues, already raised in Friedman and Shenker[10] but made a little more urgent by the present work, is to identify novel concepts of rationality and equilibrium, especially applicable in the context of the Internet and its computational platforms. C

References

1. Bubelis, V. On equilibria in finite games. *Int. J. Game Theory 8*, 2 (1979), 65–79.
2. Chen, X., Deng, X. Settling the complexity of 2-player Nash-equilibrium. *Proceedings of FOCS* (2006).
3. Chen, X., Deng, X., Teng, S. Computing Nash equilibria: approximation and smoothed complexity. *Proceedings of FOCS* (2006).
4. Conitzer, V., Sandholm, T. Complexity results about Nash equilibria. *Proceedings of IJCAI* (2003).
5. Daskalakis, C., Papadimitriou, C.H. Three-player games are hard. *Electronic Colloquium in Computational Complexity*, TR05-139, 2005.
6. Daskalakis, C., Papadimitriou, C. H. Discretized multinomial distributions and Nash Equilibria in anonymous games. *Proceedings of FOCS* (2008).
7. Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H. The complexity of computing a Nash Equilibrium. *Proceedings of STOC* (2006).
8. Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H. The complexity of computing a Nash Equilibrium. *SICOMP*. To appear.
9. Etessami, K., Yannakakis, M. On the complexity of Nash equilibria and other fixed points (extended abstract). *Proceedings of FOCS* (2007), 113–123.
10. Friedman, E., Shenker, S. *Learning and implementation on the Internet*. Department of Economics, Rutgers University, 1997.
11. Garey, M.R., Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
12. Gilboa, I., Zemel, E. Nash and correlated equilibria: some complexity considerations. *Games Econ. Behav*. (1989).
13. Goldberg, P.W., Papadimitriou, C.H. Reducibility among equilibrium problems. *Proceedings of STOC* (2006).
14. Hart, S. Mansour, Y. How long to equilibrium? The communication complexity of uncoupled equilibrium procedures. *Proceedings of STOC* (2007).
15. Kearns, M., Littman, M., Singh, S. Graphical models for game theory. *Proceedings of UAI* (2001).
16. Knaster, B., Kuratowski, C., mazurkiewicz, S. Ein beweis des fixpunktsatzes für n-dimensionale simplexe. *Fundamenta Mathematicae 14*, (1929), 132–137.
17. Lemke, C.E., Howson, Jr.J.T. Equilibrium points of bimatrix games. *SIAM J. Appl. Math. 12*, (1964), 413–423.
18. Lipton, R., Markakis, E., Mehta, A. Playing large games using simple strategies. *Proceedings of the ACM Conference on Electronic Commerce* (2003).
19. Nash, J. Noncooperative games. *Ann. Math. 54*, (1951), 289–295.
20. Papadimitriou, C.H. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci. 48*, 3 (1994), 498–532.
21. Scarf, H.E. The approximation of fixed points of a continuous mapping. *SIAM J. Appl. Math. 15*, (1967), 1328–1343.
22. Shoham, Y. Computer science and game theory. *Commun. ACM* 51, 8, 75–79.

Constantinos Daskalakis
(costis@cs.berkeley.edu), Computer Science Division, UC Berkeley.

Paul W. Goldberg
(P.W.Goldberg@liverpool.ac.uk), Department of Computer Science, University of Liverpool.

Christos H. Papadimitriou
(christos@cs.berkeley.edu), Computer Science Division, UC Berkeley.

The research reported here by Daskalakis and Papadimitriou was supported by NSF Grant CCF0635319.

CAREERS

Baylor University
Assistant to Full Professor of Computer Science

The Department of Computer Science seeks a productive scholar and dedicated teacher in the area of Game and Simulated Environments for a tenure-track position beginning August, 2009. The ideal candidate will hold a terminal degree in Computer Science or a closely related field, demonstrate scholarly capability in game design and development, and exhibit a passion for teaching and mentoring at the graduate and undergraduate level. For position details and application information please visit: http://cs.baylor.edu/employment

Baylor, the world's largest Baptist university, holds a Carnegie classification as a "high-research" institution. Baylor's mission is to educate men and women for worldwide leadership and service by integrating academic excellence and Christian commitment within a caring community. Baylor is actively recruiting new faculty with a strong commitment to the classroom and an equally strong commitment to discovering new knowledge as Baylor aspires to become a top tier research university while reaffirming and deepening its distinctive Christian mission as described in Baylor 2012 (http://www.baylor.edu/vision/).

Baylor is a Baptist university affiliated with the Baptist General Convention of Texas. As an AA/EEO employer, Baylor encourages minorities, women, veterans, & persons with disabilities to apply.

Baylor University
Assistant to Full Professor of Bioinformatics

The Baylor Department of Computer Science seeks a dynamic scholar to fill this position beginning August, 2009. For complete position details and application information please visit: http://cs.baylor.edu/employment.

Candidates should possess an earned doctorate in Bioinformatics, Computer Science, or a related field, and have sufficient expertise and experience to teach and perform research in the area of Bioinformatics. The successful candidate will be expected to establish, or bring, an active independently-funded research program in areas related to systems biology, genomics, or other areas related to ongoing research within the Department of Computer Science. The successful candidate will also assist in the teaching of graduate and undergraduate computer science and bioinformatics students. Your application should consist of a letter of interest, curriculum vitae, transcripts (as appropriate) and a list of three references. Salary will be commensurate with experience and qualifications.

Baylor, the world's largest Baptist university, holds a Carnegie classification as a "high-research" institution. Baylor's mission is to educate men and women for worldwide leadership and service by integrating academic excellence and Christian commitment within a caring community. Baylor is actively recruiting new faculty with a strong commitment to the classroom and an equally strong commitment to discovering new knowledge as Baylor aspires to become a top tier research university while reaffirming and deepening its distinctive Christian mission as described in Baylor 2012 (http://www.baylor.edu/vision/).

Baylor is a Baptist university affiliated with the Baptist General Convention of Texas. As an AA/EEO employer, Baylor encourages minorities, women, veterans, & persons with disabilities to apply.

Central Washington University
Computer Science Department

The Department is accepting applications for Assistant/Associate Professor. Candidates from all areas of specialization are welcomed. To apply online, please visit: https://jobs.cwu.edu CWU is an AA/EEO/Title IX Institution.

Connecticut College
Computer Science Tenure-Track Faculty Position

Assistant (preferred) or associate/full professor position in computer science to continue to develop our growing collaborations between CS and biology. Starting August 2009. See http://cs.conncoll.edu/job.html for details.

Eastern Mennonite University
Computer Science Faculty Position

Eastern Mennonite University. FT faculty position-Computer Science. Ph.D. preferred, Master's degree in CS or CIS with a related doctorate considered. Teaching intro to advanced level computer science, with a vision for continued growth in CS major. Ongoing professional experience/scholarship. Send letter of application, curriculum vitae, transcripts, and three letters of references to Dr. Marie S. Morris, Vice President, EMU, 1200 Park Road, Harrisonburg, VA 22802. http://www.emu.edu/. Email: ugdean@emu.edu. Application review begins immediately. Position will begin Fall 2009 or Fall 2010. EMU reserves the right to fill the position at any time or keep the position open. AAEO employer. We seek applicants who bring gender, ethnic, and cultural diversity.

The George Washington University
Department of Computer Science
Tenure-Track Faculty Positions

The Department of Computer Science is seeking applicants for two tenure-track positions in the broad areas of (1) systems and (2) systems security, one at the Assistant Professor level and one at the Associate or Assistant Professor level.

Basic Qualifications: All applicants must have a doctoral degree in Computer Science or a closely related field. ABD's may apply, but Ph.D. must be in hand by August 1, 2009. Associate Professor level applicants must demonstrate a strong record of externally funded research; Assistant Professor level applicants must demonstrate a potential for developing externally funded research programs. All applicants must have excellent communication skills and a strong commitment to quality teaching at both undergraduate and graduate levels as evidenced by teaching assessments, etc.

The George Washington University is a private institution that prides itself on excellent research programs, a quality undergraduate and graduate experience, and low student-teacher ratio. Located in the heart of the Nation's capital, GW affords its faculty and students unique cultural and intellectual opportunities. In the high-tech sector in particular, the Washington, DC Metropolitan area is one of the largest information technology areas in the nation, putting us in the center of activities such as security and biotechnology.

The Department of Computer Science offers an accredited Bachelor of Science program, a Bachelor of Arts program in Computer Science, and graduate degree programs at the Master's and Doctoral level. The Department has 17 full-time faculty members, numerous affiliated and adjunct faculty members, and over 425 students at all levels. The Department has active educational and research programs in security, networks, graphics, search and data mining, human computer interaction, and machine intelligence, with funding from various agencies; a center of academic excellence in security, with funding from NSF, DOD, and various other agencies and companies; and NIH-funded collaborations with the medical school in the biomedical areas. For further information please refer to http://www.cs.gwu.edu.

Review of applications will begin February 9, 2009, and will continue through the Spring 2009 semester, until the position is filled.

Application Procedure: To be considered, applicants should send curriculum vitae and a statement of teaching and research that identifies the position of interest, and should arrange for three reference letters to be sent to us. These and other relevant supporting materials should be sent to: Chair, Faculty Search Committee, Department of Computer Science / PHIL 703, The George Washington University, Washington D.C. 20052. Only complete applications will be considered. Electronic submissions are preferred, and can be sent to cssearch@gwu.edu. For more updated instructions on the application process, please visit the Department website www.cs.gwu.edu.

The George Washington University is an equal opportunity/affirmative action employer.

Gonzaga University
Assistant or Associate Professor of Computer Science

Gonzaga University seeks applicants for an Assistant or Associate Professor of Computer Science. This is a full-time, tenure track position to begin in the fall semester, 2009. Required qualifications: demonstrated expertise in data mining, database management systems, scientific visualization, or bioinformatics; Ph.D. in computer science or closely related field. The Department's facilities include a computational science lab with a 512 node cluster, and a sensor network and robotics lab. The department is housed in the newly completed Paccar Center for Applied Science.

Gonzaga, with 7000 students, is in the center of Spokane, Washington along the Spokane River. Research opportunities are available with the Pacific Northwest National Laboratories and many businesses in the area. Spokane, the health care center for the inland Northwest, has a metropolitan area population of 500,000. The city offers one of the finest four-season living environments in the Pacific Northwest, with five ski resorts, more than 60 lakes, and several national forests nearby.

Review of applications will begin 1/12/09. Applications will be accepted until the position is filled. Please send a letter, complete curriculum vita, a statement of research and teaching objectives, and the names, addresses, and telephone numbers of at least three references to: Paul De Palma, Chair, Department of Computer Science, Gonzaga University, Spokane, WA 99258-0026. Electronic submissions in pdf format are preferred and should be sent to: depalma@gonzaga.edu. Gonzaga is a Catholic, Jesuit and humanistic university interested in candidates who can contribute to its distinctive mission. The University is an AA/EEO employer and educator committed to diversity.

The Indian Institute of Technology Kharagpur, India
Associate Professor and Assistant Professor

The Indian Institute of Technology Kharagpur, India solicits applications for faculty positions in the ranks of Professor, Associate Professor and Assistant Professor in all Computer Science and Engineering related disciplines.

An earned doctorate and evidence of research outcomes are required. These positions are particularly suitable for

Recent Ph.D.s, including those with post-doctoral experience, who are able to provide intellectual leadership in developing interdisciplinary research and teaching portfolios with strong supporting disciplinary emphasis, who are drawn to an innovative academic vision and who are committed to achieving this through working and learning collegially and collectively, and

Experienced professionals in academic and industry committed to providing leadership in enhancing the research and teaching portfolios of the Institute.

For further information on the various positions, the Departments / Centres / Schools, emoluments and applications procedures, please visit http://www.iitkgp.ac.in/topfiles/faculty_top.php

North American (Alumni) Contacts:
Farrokh Mistree
farrokh.mistree@me.gatech.edu
Anjan Bose
bose@wsu.edu
Parvati Dev
parvati@parvatidev.org

Open Positions at INRIA for
Tenured and Tenure-Track Scientists

INRIA is a French public research institute in information and communication science and technology (ICST). The institute has about 160 project-teams throughout eight research centres in partnerships with universities and other research organization. INRIA focuses the activity of over 1100 researchers and faculty members, 1200 PhD students and 1000 post-docs and engineers, on research, development and industry transfer activities in the following computer science and applied mathematics areas:

- Modeling, simulation and optimization of complex dynamic systems
- Formal methods in programming secure and reliable computing systems
- Networks and ubiquitous information, computation and communication systems
- Vision and human-computer interaction modalities, virtual worlds and robotics
- Computational Engineering, Computational Sciences and Computational Medicine

In 2009, INRIA is opening over 40 new positions within his 8 research centers, at the junior

and senior levels, either tenured or tenure-track positions. These positions cover all areas of research of the institute.

Attractive conditions proposed: salaries and social benefits, programme to welcome foreign scientists, competitive and up-to-date environment close to universities, companies and research centers.

Opening competitive selection from December 2008

More information and contacts: www.inria.fr/travailler/index.en.html

National University of Singapore, School of Computing

Associate/Full Professor

Applications are invited for tenure-track positions at the Associate/Full Professor level. We are seeking outstanding candidates in the area of Computer Graphics who are looking for new opportunities to advance their careers. Recent PhD graduates with exceptional qualifications may be considered for appointment as Assistant Professor.

NUS is a highly ranked research university with low teaching loads, excellent facilities, and intensive international collaboration. The Singapore government has recently earmarked over S$500 million for research and industrial projects focused on Digital Media and related areas. Significant funding opportunities abound for strong candidates. The School of Computing consists of active and talented faculty members working in a variety of areas, and attracts the best students (both undergraduate and graduate) in the region.

NUS offers highly competitive salaries, as well as generous benefits for housing and education. Singapore offers a vibrant international environment with low-taxes.

Review of applications will be immediate and will continue until March 31, 2009. Interested candidates are requested to send the following materials to csrec@comp.nus.edu.sg:

- Curriculum Vitae
- Research Statement
- Names of at least five referees

NEC Laboratories America, Inc

Research Staff Members – Grid Storage

NEC Laboratories America is seeking researchers who are passionate about solving real world problems to join our Grid Storage Department in Princeton, NJ. The department engages in research in the areas of large scale reliable distributed systems with a focus on networked storage. We are currently looking for highly qualified individuals capable of carrying out independent research, with an interest in file and storage systems, data oriented Internet Services, or related areas.

Candidates must have:

- PhD in Computer Science (or equivalent)
- Experience in designing, building, and evaluating distributed systems and protocols
- Experience with storage systems (filesystems, object or content based storage systems, databases)
- Demonstrated knowledge of the algorithmic and practical challenges arising in handling large volumes of data
- Demonstrated knowledge of fault tolerance and availability techniques in the context of local and wide area networked systems

Knowledge of emerging technologies like SaaS platforms and knowledge of C++ are a plus.

Candidates must be proactive and assume leadership in proposing and executing innovative research projects, as well as in developing advanced prototypes leading to demonstration in an industry environment. Candidates are expected to initiate and maintain collaborations with outside academic and industrial research communities.

For consideration, forward resume and research statement to recruit@nec-labs.com and reference "Grid Storage" in the subject line.

EOE/AA/MFDV.

North Dakota State University

Assistant/Associate/Full Professor

Computer Science Department seeks to fill two tenure-track Assistant/Associate/Full Professor positions, preference for Bioinformatics and Software Engineering, starting Fall, 2009 or thereafter. NDSU offers degrees at all levels in Computer Science and Software Engineering. Research and teaching excellence is expected, normal teaching loads are three courses per year. The department has 16 Faculty, 5 Lecturers, over 170 graduate students (Master's and Ph.D) and 240 undergraduate majors.

Fargo is a clean, growing metropolitan area of 250,000 that consistently ranks near the top in national quality-of-life surveys. We have low levels of crime and pollution, excellent schools, short commutes, and proximity to the Minnesota lake country. The community has a symphony, an opera, a domed stadium, a community theater, three colleges, a research technology park and many other amenities. See http://www.cs.ndsu.nodak.edu/positions.htm for more information. NDSU is an equal opportunity institution. Job closing will be March 15, 2008 or until position is filled.

University of Arkansas, Little Rock

Department of Information Science
Assistant Professor

UALR Department of Information Science seeks tenure track Assistant Professor. Specialization: Web science, technology & development. Solid research and teaching expected. See Job #623 at ualr.edu/human_relations/positions/.

University of Central Florida

School of Electrical Engineering & Computer Science
Faculty Positions

UCF School of Electrical Engineering and Computer Science is looking for exceptional tenure-track faculty, primarily at the assistant professor level. EECS is specifically interested in hiring up to six candidates with research in the following three areas.

Software Engineering: Software processes and workflows, secure and reliable software architectures, software tools and development environments, program comprehension and visualization, software economics, engineering embedded and real-time software, ubiquitous and pervasive computing, empirical studies, and formal methods.

Space Systems: small satellites, space weather, sensors, embedded systems, imaging systems, communications, command and data management, power and propulsion.

Energy: Renewable energy research and technology, including photovoltaic applications, distributed electrical power generation and distribution, integrated power networks, renewable energy systems and storage, renewable power system control, computational methods for energy systems, and applications of power electronics in energy conversion systems.

EECS offers a competitive salary and start-up package, and UCF provides generous benefits. New faculty members have graduate student support and significantly reduced teaching loads.

Applicants should have a Ph.D. in a related area to EECS disciplines by the start of the appointment and a strong commitment to the academic process, including teaching, scholarly publications and sponsored research. Successful candidates will have a record of high-quality publications and be recognized for their potential.

EECS at UCF is the oldest Ph.D. granting CS program in the state of Florida and has a rapidly growing educational and research program with nearly $12 million in research contracts and over 500 graduate students and 2,000 undergraduates. UCF is strongly committed to continuing the buildup of strength in EECS, including a move in late 2006 to a new, state-of-the-art building: the Harris Corp. Engineering Center.

Research sponsors include NSF, NASA, DOT, ARO, ONR, PEOSTRI, RDECOM and other agencies of the DOD. Industry sponsors include Adaptec, ATI, Boeing, Canon, Electronic Arts, Harris, Honeywell, IBM, Imagesoft, Intel, Lockheed Martin, Lucent, Oracle and Sun Microsystems as well as local high-tech start-ups.

UCF has over 50,000 students and is among the nation's top-10 largest universities. Located in Orlando, EECS and UCF are at the center of the I-4 High Tech Corridor with an excellent industrial base in telecommunications, computer systems, semiconductors, defense, space, lasers, simulation, software and the world-renowned entertainment/theme park industry. Adjacent to UCF is a thriving research park that hosts many high-technology companies and the Institute for Simulation and Training. UCF also has a new medical school. Exceptional weather, easy access to the seashore, one of the largest convention centers in the nation and an international airport that is among the world's best are just a few features that make the UCF/Orlando area ideal.

To submit an application, please go to: http://www.eecs.ucf.edu/facsearch/online_app.html

UCF is an Equal Opportunity/Affirmative Action employer. Women and minorities are particularly encouraged to apply.

THE UNIVERSITY OF NORTH CAROLINA AT CHARLOTTE
CHAIRPERSON
DEPARTMENT OF COMPUTER SCIENCE

The Department invites applications for the position of Department Chair. Candidates for the position must have a Ph.D. in Computer Science or a closely related field, a record of scholarly research and publication commensurate with that of Full Professor, evidence of a commitment to excellence in teaching, and strong administrative skills. Computer Science is the largest department within the College of Computing and Informatics, and offers B.S., B.A. and M.S. programs in Computer Science, and a Computer Science track within the IT Ph.D. program. Currently, the Department has 60 Ph.D. students, 130 M.S. students, and more than 30 faculty with areas of expertise which include Visualization and Management of Data, Networking, Robotics, Knowledge Discovery, Game Design, and Computer Vision. The University is located on a 1000-acre campus in Charlotte and has over 23,000 students, with an enrollment of 35,000 expected by 2020. The Chair is expected to bring energy and enthusiasm to the continued development of the Department and its role in one of the most rapidly growing universities in the country. Applications must be made electronically at https://jobs.uncc.edu (position #1912) and must include a CV, a list of 4 references, and statements on research, teaching, and leadership/management style. Informal inquiries can be made to the Search Committee Chair, Lawrence Mays, at lemays@uncc.edu.

Review of applications will begin in immediately and continue until the position is filled. All inquires and applications will be treated as confidential. The University of North Carolina at Charlotte is an EOE/AA employer and an ADVANCE Institution. For additional information, please visit our website at http://www.cs.uncc.edu

Senior Faculty Position

Department of Computer and Information Sciences in the College of Science and Technology at Temple University

Applications are invited for the position of a senior faculty at Associate or Full Professor level in the Department of Computer and Information Sciences in the College of Science and Technology at Temple University.

Applicants are expected to have outstanding research accomplishments, a record of funded research in computer science/engineering, and a commitment to quality undergraduate and graduate programs and instruction. Applications from candidates with significant systems research are encouraged, and interdisciplinary/multidisciplinary research track records are a plus. Candidates from industry with a strong record are also encouraged. Areas of interest include, but are not limited to,

- Large-Scale Distributed Computing Systems
- Wired and Wireless Networks
- Trustworthy and Reliable Computing Systems

Applications consisting of curriculum vitae, a statement of recent achievements and visions for research and teaching, up to three representative publications, and the names and addresses of at least three references should be submitted online at **http://academicjobsonline.org.**

Review of candidates will start on February 15, 2009 and will continue until the position is filled. **For further information, check http://www.temple.edu/cis or e-mail facultysearch@cis.temple.edu.**

Temple University is an equal opportunity, equal access, affirmative action employer committed to achieving a diverse community (AA, EOE, M/F/D/V).

University of Denver
Professor and Department Chair

The Department of Computer Science (CS) at the University of Denver (DU) is seeking a dynamic and visionary individual from business or academia to lead the department during this expansion phase of the School of Engineering and Computer Science. Through its strategic planning, the faculty of our CS department have identified Software Engineering, Game Development, and Cyber Security as the key focus areas for the department. Our CS department benefits from a top quality faculty, strong partnership with industry, strong collaborations with other colleges within DU and internationally. The CS department offers degrees in both traditional and contemporary areas such as undergraduate degree in gaming, and graduate degree in Computer Science Systems Engineering. The primary focus of this new department chair will be on both educational and research programs at graduate and undergraduate levels. DU is a private university with a strong history of academic excellence, small classes, and emphasis on student engagement at all levels. DU is the oldest university in Colorado and its campus is located in the Denver metro area.

Individuals with a strong record of research, scholarship and excellence in teaching are encouraged to apply by sending their resume, statement of interest, and a list of five references to www.dujobs.org. PhD or PhD candidate in computer science or related areas and some level of leadership experience are required. The University of Denver is an AA/EOE.

The University of Michigan – Dearborn
Department of Computer and Information Science
Assistant/Associate Professors

The Department of Computer and Information Science (CIS) at the University of Michigan-Dearborn invites applications for a tenure-track faculty position in any of the following areas: computer gaming, computer and data security, digital forensics, information assurance, and multimedia. Rank and salary will be commensurate with qualifications and experience. We offer competitive salaries and start-up packages.

Qualified candidates must have, or expect to have, a Ph.D. in CS or a closely related discipline by the time of appointment and will be expected to do scholarly and sponsored research, as well as teaching at both the undergraduate and graduate levels. The CIS Department offers several BS and MS degrees, and participates in an interdisciplinary Ph.D. program in information systems engineering. The current research areas in the department include computer graphics and geometric modeling, database systems, multimedia systems and gaming, networking, real-time and secure computing, and software engineering. These areas of research are supported by several labs, including the Database and Multimedia Systems Laboratory, the Games and Multimedia Environments Laboratory, the Vehicular Networking Systems Research Laboratory, and the Virtual Engineering Laboratory.

The University of Michigan-Dearborn is located in the southeastern Michigan area and offers excellent opportunities for faculty collaboration with many industries. We are one of three campuses forming the University of Michigan system and are a comprehensive university with over 8500 students. One of university's strategic visions is to advance the future of manufacturing in a global environment.

The University of Michigan-Dearborn is dedicated to the goal of building a culturally-diverse and pluralistic faculty committed to teaching and working in a multicultural environment, and strongly encourages applications from minorities and women.

A cover letter, curriculum vitae including e-mail address, teaching statement, research statement, and three letters of reference should be sent to

Dr. William Grosky, Chair
Department of Computer and Information Science
University of Michigan-Dearborn
4901 Evergreen Road
Dearborn, MI 48128-1491
Email: wgrosky@umich.edu,
Internet: http://www.cis.umd.umich.edu
Phone: 313.583.6424, Fax: 313.593.4256

The University of Michigan-Dearborn is an equal opportunity/affirmative action employer.

University of Nebraska – Lincoln
Department of Computer Science and Engineering
Tenure-track or tenured faculty positions

We invite outstanding individuals with research interests in Embedded Systems and Computer Engineering for tenure-track or tenured faculty positions in the Department of Computer Science and Engineering. We are particularly seeking applications in the general area of embedded systems that complement existing strengths in embedded control systems, embedded software, and real-time systems. Exceptional candidates with research interest in Data Visualization and Computational Science and Engineering are also invited to apply.

Candidates are encouraged to collaborate with faculty in appropriate related areas such as robotics, biomedical engineering, computer science, electrical engineering, materials science, or mechanical engineering. The University seeks individuals with exceptional promise for, or proven record of, research achievement who will excel in teaching undergraduate and graduate courses and take a position of international leadership in defining their field of study.

Candidates will hold a PhD in Computer Engineering, Computer Science, Electrical Engineering, or a closely related discipline. Applicants will find many opportunities for research collaborations both within and outside the Computer Science and Engineering Department. To apply, go to http://employment.unl.edu and complete the Faculty/Administrative application 080943. The cover letter should include names of at least three references and statement of teaching and research. Review of applications will begin January 1, 2009, and will continue until the position has been filled. The official advertisement can be viewed at http://cse.unl.edu/search. The University of Nebraska has an active National Science Foundation ADVANCE gender equity program, and is committed to pluralistic campus community through affirmative action, equal opportunity, work-life balance, and dual careers.

University of North Carolina at Charlotte
Department of Software and Information Systems
DICyDER Center Director

The Department of Software and Information Systems at UNC Charlotte seeks to hire a tenure-track faculty member at the associate level to serve as Director of the recently established Center for Digital Identity and Cyber Defense Research (DICyDER), http://www.dicyder.uncc.edu/. DICyDER's mission is to add value to the university, community, and society through innovative educational programs, research and development in the areas of information integration, security, and privacy. The Director will be responsible for leading a strong research program by communicating research vision, planning and implementing research strategy, facilitating contract acquisition and relationship development, and providing project and group management.

The Department of Software and Information Systems is dedicated to research and education in Software Engineering and Information Technology applications, with emphasis in the areas of Information Integration & Environments and Information Security & Assurance; it offers degrees at the Bachelors, Masters, and Ph.D. levels. Current faculty members have strong research programs with substantial funding from both federal agencies and industrial partners.

Salary will be highly competitive. Applicants must have a Ph.D. in Computer Science, Information Technology, Software Engineering, or a related field, as well as a strong commitment to research and education. For further details please visit http://www.sis.uncc.edu/. Application review will start in January 2009.

Applications must be submitted online at https://jobs.uncc.edu/. To the application, attach a cover letter, curriculum vitae, a statement of teaching interests, a statement of research interests, copies of three representative scholarly publications, and a list of four references. For questions or additional information, email search-sis@uncc.edu.

Women, minorities and individuals with disabilities are encouraged to apply. UNC Charlotte is an Equal Opportunity/Affirmative Action employer.

University of North Carolina at Charlotte
Department of Software and Information Systems
Tenure-Track Faculty Positions

The Department of Software and Information Systems at UNC Charlotte invites applicants for multiple tenure-track faculty positions at both the assistant and associate levels. The Department is dedicated to research and education in Software Engineering and Information Technology applications, with emphasis in the areas of Information Integration & Environments and Information Security & Assurance; it offers de-

grees at the Bachelors, Masters, and Ph.D. levels. Current faculty members have strong research programs with substantial funding from both federal agencies and industrial partners. The department is particularly interested in faculty with research expertise in: Trusted Software Development, Software Engineering, or Modeling & Simulation. Highly qualified candidates in other areas will also be considered.

Salary will be highly competitive. Applicants must have a Ph.D. in Computer Science, Information Technology, Software Engineering, or a related field, as well as a strong commitment to research and education. For further details please visit http://www.sis.uncc.edu/. Application review will start in January 2009.

Applications must be submitted online at https://jobs.uncc.edu//. To the application, please attach a cover letter, curriculum vitae, a statement of teaching interests, a statement of research interests, copies of three representative scholarly publications, and a list of four references. For questions or additional information, please email search-sis@uncc.edu.

Women, minorities and individuals with disabilities are encouraged to apply. UNC Charlotte is an Equal Opportunity/Affirmative Action employer.

The University of Texas at Austin

Department of Computer Sciences

Tenured/Tenure-Track Faculty

The Department of Computer Sciences of the University of Texas at Austin invites applications for tenure-track positions at all levels. Excellent candidates in all areas will be seriously considered, especially in Computer Architecture. All tenured and tenure-track positions require a Ph.D. or equivalent degree in computer science or a related area at the time of employment.

Successful candidates are expected to pursue an active research program, to teach both graduate and undergraduate courses, and to supervise graduate students. The department is ranked among the top ten computer science departments in the country. It has 46 tenured and tenure-track faculty members across all areas of computer science. Many of these faculty participate in interdisciplinary programs and centers in the University, including those in Computational and Applied Mathematics, Computational Biology, and Neuroscience.

Austin, the capital of Texas, is located on the Colorado River, at the edge of the Texas Hill Country, and is famous for its live music and outdoor recreation. Austin is also a center for high-technology industry, including companies such as IBM, Dell, Freescale Semiconductor, Advanced Micro Devices, National Instruments, AT&T, Intel and Samsung. For more information please see the department web page: http://www.cs.utexas.edu/

The department prefers to receive applications online, beginning November 15, 2008. To submit yours, please visit http://services.cs.utexas.edu/recruit/faculty/

If you do not have internet access, please send a curriculum vita, home page URL, description of research interests, and selected publications, and ask three referees to send letters of reference directly to:

Faculty Search Committee, Department of Computer Sciences, The University of Texas at Austin, 1 University Station C0500, Austin, Texas 78712-0233, USA

Inquiries about your application may be directed to faculty-search@cs.utexas.edu. For full consideration of your application, please apply by January 15, 2009. Women and minority candidates are especially encouraged to apply. The University of Texas is an Equal Opportunity Employer.

The University of Texas at Tyler

Computer Science Faculty Position

The Department of Computer Science invites applications for a tenure-track faculty position. Although appointment at the assistant professor level in fall 2009 is anticipated, an exceptional candidate may be appointed at a higher rank. An earned doctorate in computer science or computer information systems, demonstrated English communication skills, and commitment to excellence in teaching, externally funded research, scholarship and service are required. All areas of specialization will be considered; preferred areas include computer security, bioinformatics, high-performance computing, gaming and simulation, artificial intelligence, information systems, and e-commerce. Information about the department, college, university and the Tyler area can be found at http://www.uttyler.edu. Send letter of application, curriculum vitae, and names and phone numbers of three references to: Faculty Search Committee, Department of Computer Science, The University of Texas at Tyler, 3900 University Boulevard, Tyler, TX 75799 or via email to cssearch@uttyler.edu. Review of applications begins in late January 2009 and will continue until the position is filled. The University of Texas at Tyler is an Equal Opportunity Employer. Women and minorities are strongly encouraged to apply. Applicants selected to fill the position must be able to present documentation of the right to work in the United States.

University of Waterloo

David R. Cheriton School of Computer Science

Faculty Position in Software Engineering

The University of Waterloo invites applications for a tenure-track or tenured faculty position in the David R. Cheriton School of Computer Science, in the area of software engineering. Candidates at all levels of experience are encouraged to apply. Preference will be given to those who focus on health informatics as an application area. Successful applicants who join the University of Waterloo are expected to develop and maintain a productive program of research, attract and develop highly qualified graduate students, provide a stimulating learning environment for undergraduate and graduate students, and contribute to the overall development of the School. A Ph.D. in Computer Science, or equivalent, is required, with evidence of excellence in teaching and research. Rank and salary will be commensurate with experience, and appointments are expected to commence during the 2009 calendar year.

With over 70 faculty members, the University of Waterloo's David R. Cheriton School of Computer Science is the largest in Canada. It enjoys an excellent reputation in pure and applied research and houses a diverse research program of international stature. Because of its recognized capabilities, the School attracts exceptionally well-qualified students at both undergraduate and graduate levels. In addition, the University has an enlightened intellectual property policy which vests rights in the inventor: this policy has encouraged the creation of many spin-off companies including iAnywhere Solutions Inc., Maplesoft Inc., Open Text Corp and Research in Motion. Please see our website for more information: http://www.cs.uwaterloo.ca/

Applications should be sent by electronic mail to cs-recruiting@cs.uwaterloo.ca

or by post to

Chair, Advisory Committee on Appointments
David R. Cheriton School of Computer Science
200 University Avenue West
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

An application should include a curriculum vitae, statements on teaching and research, and the names and contact information for at least three referees. Applicants should ask their referees to forward letters of reference to the address above. Applications will be considered as soon as possible after they are complete, and as long as positions are available.

The University of Waterloo encourages applications from all qualified individuals, including women, members of visible minorities, native peoples, and persons with disabilities. All qualified candidates are encouraged to apply; however, Canadian citizens and permanent residents will be given priority.

DOI:10.1145/1461928.1461952

Peter Winkler

Puzzled
Will My Algorithm Terminate?

Welcome to three new challenging mathematical puzzles. Solutions to the first two will be published next month; the third is as yet unsolved. In them all, I concentrate on algorithm termination, outlining some simple procedures and asking whether they always terminate or might possibly run forever.

1. Five integers with positive sum are assigned to the vertices of a pentagon. At any point you may select a negative entry (say, $-x$) and flip it to make it positive, or x, but then you must subtract x from each of the two neighboring values; thus, the sum of the five integers remains the same. For example, if the numbers are 2, 4, −3, 1, −3, you can either flip the first −3 to get 2, 1, 3, −2, −3 or the second to get −1, 4, −3, −2, 3. Now prove that no matter what numbers you start with and strategy you follow, all the numbers will eventually become non-negative, and thus the procedure terminates after finitely many steps.

2. Billiard balls numbered 1 through n but not in their correct order lie together in a trough. In a naive attempt to put them in correct order, you repeatedly pick up a ball that is not where it belongs and put it where it does belong; the balls between their old and new positions naturally slide over by one space to accommodate the new ball. For example, if five balls are in the order 1 5 3 4 2, you can pick up ball 2 and place it in the second position to yield 1 2 5 3 4. Because this knocks balls 3 and 4 out of place, it is not obvious that you have made real progress toward 1 2 3 4 5. Now, is there a starting permutation from which, if you choose your steps sufficiently badly, you will never achieve 1 2 3...n?

3. Possibly the most notorious algorithm-termination puzzle of all time—among mathematicians anyway—is the Collatz Conjecture, sometimes called the $3x+1$ problem (or the Syracuse problem, Kakutani's problem, Hasse's algorithm, or Ulam's problem). Start with any positive integer and repeat: If it is even, divide by two; if odd, multiply by 3 and add 1. For example, if you start with 22, you get 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1,... The conjecture is that no matter what number you start with, you eventually get down to the same cycle 4, 2, 1, repeating over and over. Watch out though: If you intend to give this problem to your CS 101 students, make sure you have plenty of spare computer time.

Readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (puzzled@cacm.acm.org) is Professor of Mathematics and of Computer Science and Albert Bradley Third Century Professor in the Sciences at Dartmouth College, Hanover, NH.

Emerging Software Technologies

DISNEY'S CONTEMPORARY RESORT, ORLANDO, FL
October 25-29 2009

OOPSLA

ORLANDO 2009

Scaling: Multi-core to Cloud
Mashups of Models, Data and Code
Tools for Reliability and Evolution
Enterprise Agile Management

Conference Call for Submissions

March 19, 2009
Due date for Research Program, Onward!, Practitioner report, Educators' Symposium, Essays, and proposals for Tutorials, Panels, Workshops, and DesignFest®

July 2, 2009
Due date for Posters, Demonstrations, Doctoral Symposium, Onward! Films, Student Research Competition, and Student Volunteers

PROGRAM CHAIR
Gary T. Leavens
University of Central Florida
papers@oopsla.org

CONFERENCE CHAIR
Shail Arora, Gradepoint Inc.
chair@oopsla.org

ONWARD! CHAIR
Bernd Brügge
Technische Universität München
chair@onward-conference.org

Association for Computing Machinery
Advancing Computing as a Science & Profession

OOPSLA is sponsored by ACM SIGPLAN in cooperation with SIGSOFT

www.oopsla.org/submit

CHI 2009

DIGITAL LIFE NEW WORLD

Join us in Boston, MA for the 27th Annual CHI Conference, the premier international forum for all aspects of human-computer interaction. Computing is reaching into all parts of modern life. CHI 2009 brings together people working on the design, evaluation, implementation, and study of interactive computing systems for human use. CHI serves as a forum for the exchange of ideas among computer scientists, human factors scientists, psychologists, social scientists, system designers, usability professionals, and end users.

www.chi2009.org

acm Association for Computing Machinery

SIGCHI