



CoAP: An Application Protocol for Billions of Tiny Internet Nodes

Carsten Bormann • *Universität Bremen*

Angelo P. Castellani • *University of Padova*

Zach Shelby • *Sensinode*

The Constrained Application Protocol (CoAP) is a transfer protocol for constrained nodes and networks, such as those that will form the Internet of Things. Much like its older and heavier cousin HTTP, CoAP uses the REST architectural style. Based on UDP and unencumbered by historical baggage, however, CoAP aims to achieve its modest goals with considerably less complexity.

More and more devices are becoming connected. Automation systems, mobile personal gadgets, building-automation devices, cellular terminals, the smart grid, and so on all benefit from interacting with other objects close to them or halfway around the globe. Although many limited, special-purpose networks have been (and will be) built for this, ubiquitous IP technology is now enabling a leap from sheer quantity to a new quality, often called the Internet of Things. Over the next decade, this could grow to trillions of embedded devices and will greatly increase the Internet's size and scope.

The IETF has already undertaken much standardization work to make the packets flow. The IPv6 over Low-Power Wireless Area Networks (6LoWPAN)¹ standards (RFCs 4944 and 6282) now enable IPv6 even on very constrained networks – including the popular IEEE 802.15.4 wireless standard, ISM (industrial, scientific, medical) band telemetry radios, and low-rate power-line communications (PLC), all while using very simple embedded microcontrollers. IETF Routing Over Low-power and Lossy networks (ROLL) standardization provides a routing solution optimized for these constrained networks. In the ZigBee Smart Energy Profile 2.0, the ZigBee

Alliance has demonstrated how to combine these components to build a complete, mature IPv6 stack for integrating embedded devices and systems into the Internet.

Networking alone, however, doesn't make the Internet useful. Applications today depend on the Web architecture, using HTTP to access information and perform updates. HTTP is based on Representational State Transfer (REST), an architectural style that makes information available as resources identified by URIs: applications communicate by exchanging representations of these resources using a transfer protocol such as HTTP. This powerful and extensible paradigm is quickly spreading to Internet of Things applications, letting developers of Web-based applications continue using their existing skills. The IETF Constrained RESTful Environments (CoRE) working group aims to make the REST paradigm available for devices and networks that might be too constrained to use the typical approaches around the HTTP protocol. The main product of the CoRE WG is the Constrained Application Protocol (CoAP).²

The CoRE World

What's so special about the environment the CoRE work targets? The large number of devices

envisaged will never emerge if they're too expensive, especially if they consume too much power. (A trillion devices using a watt each would consume half the entire globally available electricity.) Worse, many devices will need to live for years off primary batteries, limiting them to an average consumption on the order of microwatts. They can only achieve such thriftiness by sleeping most of the time, making some of them responsive to incoming packets only for short intervals.

These power limitations also lead to constraints on available networking. Because stringing wires is prohibitively expensive in many applications, most devices will connect wirelessly, often using crowded ISM spectrum. At a transmission power of roughly a milliwatt, many packet losses will occur, and the net data rate achievable could make old modems look good. Also, wireless standards such as IEEE 802.15.4 need fragmentation (which exacerbates losses) to transport more than a few dozen bytes of payload, requiring strict frugality with regard to packet size. Constrained networks aren't a wireless Ethernet!

Constraints on nodes themselves include not only power limitations but also a desire to limit manufacturing costs. There is no single type of constrained, Internet-connected device – rather, the trend is toward a wider variety of such devices than in the current Internet – but commercially available chips do group into certain capability clusters. We can examine these device classes to determine the resulting design constraints.

Heroic attempts to get some Internet functionality into impossibly limited devices get considerable press, but these devices won't be full-fledged citizens of the Internet of Things. These *class-0* devices most likely will form a symbiotic relationship with larger devices to take part in global conversations.

We can discern another cluster of chips with about 10 Kbytes of RAM and roughly 100 Kbytes of code space (Flash or ROM); these are the *class-1* devices. Interestingly, this class hasn't changed much in the past decade. Moore's law tends to be less effective in the embedded space than in personal computing devices: chip makers are more likely to invest gains from increases in transistor count and density into reducing cost and power requirements than into continual increases in computing power.

Class-1 devices can't easily talk to other Internet nodes using HTTP, Transport Layer Security (TLS) and related security protocols, and XML-based data representations. However, they have enough power to participate in meaningful conversations

constrained networks limited to packets of maybe 60 to 80 bytes of payload, will need application protocols that fit this environment.

REST

HTTP is the most popular application protocol on the Internet; it supports the architecture we refer to as “the Web.” What does the Web bring to constrained networks and devices?

First and foremost, the Web is a *loosely coupled* application-layer architecture. *Resources* are key to Web architecture: server-controlled abstractions an application process makes available, identified via URIs. Clients access these server-controlled resources in a synchronous request-response fashion using

HTTP has undergone more than a decade of organic growth, leading to considerable implementation baggage that overwhelms small devices.

beyond a simple symbiotic relationship to a single gateway node, so giving them the power of the Internet is worthwhile.

Another cluster of embedded devices sports around 50 Kbytes of RAM and maybe 250 Kbytes of code space. These *class-2* devices can indeed speak the exact same protocols used among desktops, laptops, and rack-mount servers. However, even these devices can benefit from constrained protocols – they'd use less power and fewer network resources, would leave more functionality available to applications, and could also more easily communicate with class-1 devices in their environment.

In short, an Internet of Things that wants to make good use of inexpensive class-1 devices, and of

methods such as GET, PUT, POST, and DELETE (see Figure 1).

The server owns the original state of a resource, and the access to its representation allows for caching, proxying, and redirecting requests and responses, enabling seamless interoperation through proxies. Web resources often contain links to other resources, which creates a distributed Web between Internet end points, resulting in a highly scalable and flexible architecture. These core Web concepts are commonly described as REST (<http://java.sun.com/developer/technicalArticles/WebServices/restful/>).³

CoAP

Loosely speaking, the Web consists of three technologies: HTML, HTTP/REST,

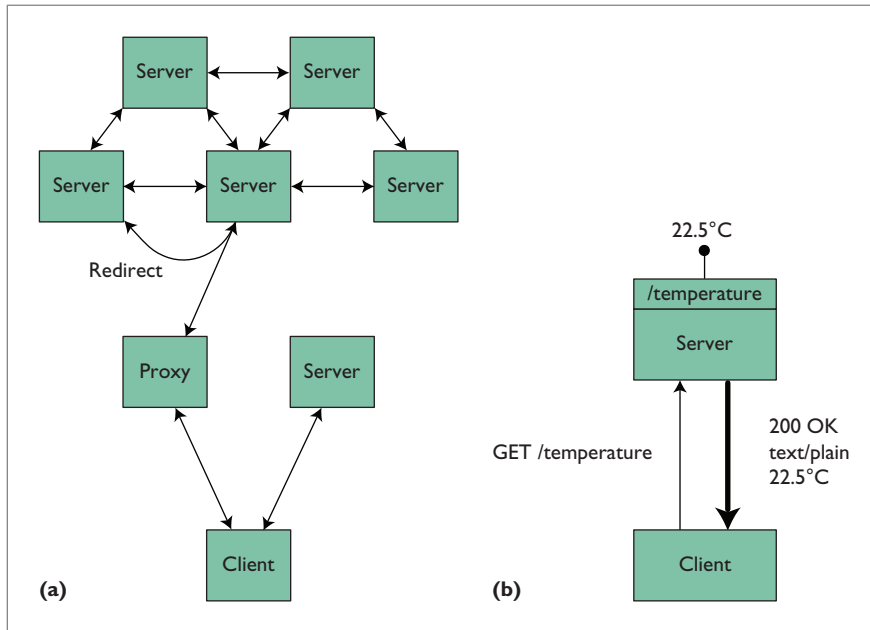


Figure 1. The Web architecture. (a) Clients access servers directly and via proxies; (b) a GET request elicits a 200 OK response.

and URIs. Only the latter two are useful where machines talk to machines. Special data formats to replace HTML for these applications are being defined,⁴ often based on XML and its compact binary representation, EXI, or on the JavaScript Object Notation (JSON, RFC 4627).

HTTP itself is a powerful and well-tried protocol, but it's relatively expensive both in implementation code space (a problem for class-1 devices) and network resource usage. Part of the problem is that HTTP has undergone more than a decade of organic growth, leading to considerable implementation baggage that overwhelms small devices.

However, HTTP is designed to interoperate through proxies; what we really need in constrained environments is REST, not necessarily all HTTP's bells and whistles. CoAP is a fresh approach to a Web application transfer protocol that tries to get by with very limited resources. CoAP isn't just "compressed HTTP" – although it provides the same basic set of services, it does so with a very frugal design (see Figure 2).

A central element of CoAP's reduced complexity is that, instead of TCP, it uses UDP and defines a very simple "message layer" for retransmitting lost packets. Within UDP packets, CoAP uses a four-byte binary header, followed by a sequence of options (each with a one-byte header, extensible to two bytes for longer option values). This compact but easily parsable encoding enables a total header size of 10 to 20 bytes for a typical request. Differential encoding of option types provides the future extensibility needed without burdening simple implementations.

On top of CoAP's message layer, the CoAP base specification defines the familiar four request methods, GET, PUT, POST, and DELETE. Similarly, response codes are patterned after the HTTP response codes (as in the familiar "404 not found"), but encoded in a single byte ("4.04" standing for $4 \times 32 + 04$) (see Figure 3 on p. 66).

Interworking with HTTP

CoAP would already be useful if we could use it only for communicating between CoAP end points, but it reaches its full potential by

interworking with HTTP. The REST architectural style enables this through proxies or, more generally, intermediaries that behave like a server to a client and play a client toward another server. (REST terminology reserves the term "proxy" for intermediaries specifically configured on a client. It also has a "gateway" that acts as if it were the origin server; these are often called "reverse proxies" in the Web because they can be much less intrusive than the general concept of a gateway.)

We can generally build intermediaries that speak CoAP on one side and HTTP on the other without encoding specific application knowledge. This lets us deploy new applications without having to upgrade all the intermediaries involved – a requirement that's typically the bane of architectures heavily relying on gateways in the general sense.

In many cases, an intermediary can perform the translation between CoAP and HTTP without posing further requirements either on the client or server. Where equivalent methods, response codes, and options are present in both protocols, the mapping between CoAP and HTTP is straightforward, and even completely stateless intermediaries can handle the self-describing REST-based messages by applying a static mapping.

Both CoAP and HTTP identify resources using URIs. Existing HTTP end points might be unaware of CoAP's URI schemes – say, `coap://` URIs. A reverse-proxy-style intermediary can make a set of CoAP resources available at what look like regular `http://` or `https://` URIs, enabling older Web clients to access CoAP servers transparently (see Figure 2). Similarly, an *interception proxy* (RFC 3040) deployed in a network location suitable for traffic interception that automatically redirects client requests to itself might provide such a service.

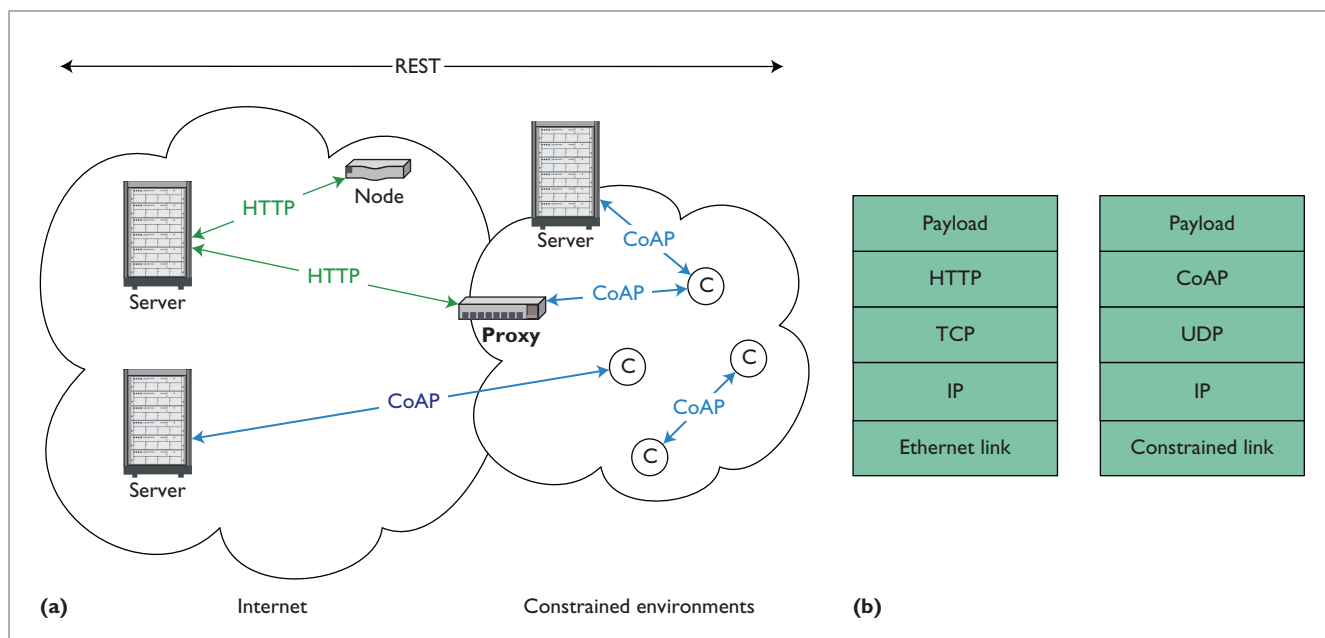


Figure 2. Implementing the Web architecture with HTTP and the Constrained Application Protocol (CoAP). (a) HTTP and CoAP work together across constrained and traditional Internet environments; (b) the CoAP protocol stack is similar to, but less complex than, the HTTP protocol stack.

By mapping a single HTTP request to a multicast CoAP request and then aggregating multiple responses back into a single HTTP response body, future types of intermediaries might even support more complex communication patterns across HTTP and CoAP, such as group communication.

Block

Basic CoAP messages work well for the small payloads we expect from temperature sensors, light switches, and similar building-automation devices. Occasionally, however, applications will need to transfer larger payloads – for instance, for firmware updates. With HTTP, TCP does the grunt work of slicing large payloads up into multiple packets and ensuring that they all arrive and are handled in the right order. Although UDP supports larger payloads through IP fragmentation, it's limited to 64 KiB and, more importantly, doesn't really work well for constrained applications and networks.

Instead of relying on IP fragmentation, CoAP simply adds a pair of “Block” options, transferring multiple

blocks of information from a resource representation in multiple request-response pairs.⁵ The block options enable a server to be truly stateless in the most likely cases: the server can handle each block transfer separately, with no need for a connection setup or other server-side memory of previous block transfers.

Observe

In HTTP, transactions are always client-initiated, and the client must perform GET operations again and again (polling) if it wants to stay up to date about a resource's status. This pull model becomes expensive in an environment with limited power, limited network resources, and nodes that sleep most of the time. Web developers have come up with some more or less savory workarounds for HTTP (RFC 6202), but, as a new protocol, CoAP can do better.

CoAP uses an asynchronous approach to support pushing information from servers to clients: observation.⁶ In a GET request, a client can indicate its interest in further updates from a resource by

specifying the “Observe” option. If the server accepts this option, the client becomes an observer of this resource and receives an asynchronous notification message each time it changes. Each such notification message is identical in structure to the response to the initial GET request.

Instead of trying to create another complex publish-subscribe architecture, CoAP effectively provides a minimal enhancement to the REST model, just adding the well-known observer design pattern.⁷

Discovery

In the machine-to-machine (M2M) environments that will be typical of CoAP applications, devices must be able to discover each other and their resources. Resource discovery is common on the Web, and is called Web discovery in the HTTP community. One form of Web discovery occurs when humans access a server's default resource (such as index.html), which often includes links to other Web resources available on that or related servers.

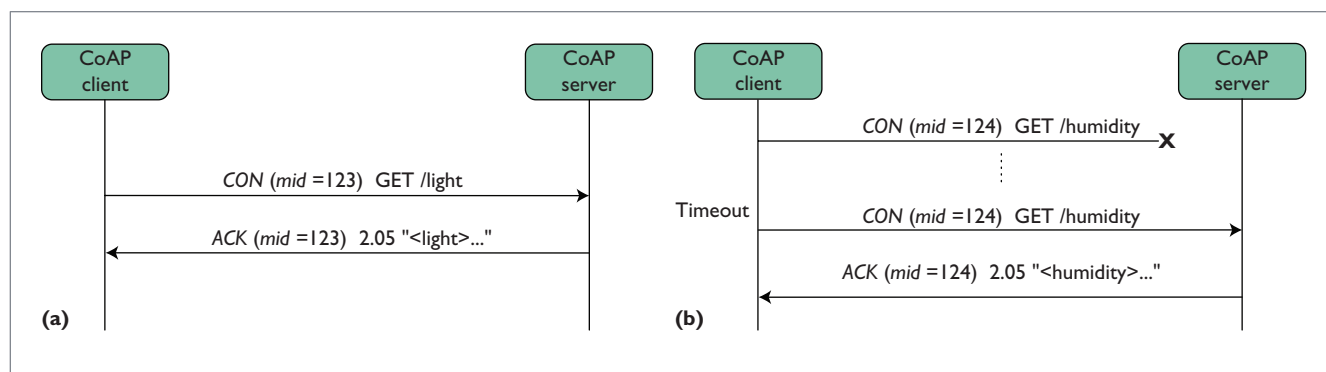


Figure 3. Constrained Application Protocol (CoAP) request–response examples. (a) A confirmable GET request elicits a 2.05 response piggy-backed in an ACK; (b) packet loss is fixed by retransmission, reusing the message id (mid).

Machines can also perform Web discovery if standardized interfaces and resource descriptions are available. New approaches from the IETF include the well-known resource path `/well-known/scheme` (RFC 5785) and the HTTP link header (RFC 5988). Several related techniques are common today. In CoRE, we're dealing with autonomous devices and embedded systems; thus, the importance of uniform, interoperable resource discovery is much greater than on the current Web. To ensure interoperability between CoAP end points, the protocol includes a technique for discovering and advertising resource descriptions. Because these descriptions are machine-interpreted, we're also standardizing the description format itself. To achieve resource discovery, CoAP servers are encouraged to provide a resource description available via the well-known URI `/well-known/core` for resource discovery. Clients then access this description with a GET request on that URI. The same description could be advertised, or even posted to a description directory. The description format is based on the HTTP link header format as an Internet media type carried in the payload, which is simple and easy to parse.⁸

Security

For applications that require some level of security, HTTP is usually

employed in combination with TLS (formerly Secure Sockets Layer, or SSL). This protects the message content's confidentiality and integrity. Server authentication often employs a public-key infrastructure (PKI) based on certification authorities (CAs). This approach works quite well in practice, but suffered some well-publicized attacks in 2011.⁹

Similarly, we can use CoAP on top of Datagram Transport Layer Security (DTLS).¹⁰ We expect CoAP deployments to use a wider variety of key-management options available for TLS than most HTTP applications do today; the IETF TLS working group is developing some additional particularly lightweight combinations.¹¹

Standardization Activities and Adoption

CoRE technology has already become widespread in both open source communities and industry applications, with implementations of CoAP and related specifications available in several programming languages along with Firefox and Wireshark support. Many informal tests established interoperability between dozens of implementations; the European Telecommunications Standards Institute (ETSI) will conduct a formal interop event on 24–25 March 2012, colocated with IETF 83 in Paris.

Several other standards activities are using IETF CoRE standards as

part of more complete M2M systems. The ETSI M2M Technical Committee has specified a service-layer architecture for M2M that includes bindings for both HTTP and CoAP. The ZigBee IP Smart Energy 2.0 specification includes support for CoAP for constrained battery-powered devices.

As a tiny but well-designed and quite functional stand-in for HTTP, CoAP is slated to become a ubiquitous application protocol for the future Internet of Things – or, really, the “Internet of Innumerable Embedded Systems.” □

References

1. Z. Shelby and C. Bormann, *6LoWPAN: The Wireless Embedded Internet*, Wiley, 2009.
2. Z. Shelby et al., “Constrained Application Protocol (CoAP),” IETF Internet draft, work in progress, Oct. 2011.
3. R.T. Fielding, *Architectural Styles and the Design of Network-Based Software Architectures*, PhD thesis, Univ. of California, Irvine, 2000.
4. C. Jennings, Z. Shelby, and J. Arkko, “Media Types for Sensor Markup Language (SENML),” IETF Internet draft, work in progress, Oct. 2011.
5. C. Bormann and Z. Shelby, “Blockwise Transfers in CoAP,” IETF Internet draft, work in progress, July 2011.
6. K. Hartke and Z. Shelby, “Observing Resources in CoAP,” IETF Internet draft, work in progress, Oct. 2011.

7. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
8. Z. Shelby, "CoRE Link Format," IETF Internet draft, work in progress, Nov. 2011.
9. N. Leavitt, "Internet Security Under Attack: The Undermining of Digital Certificates," *Computer*, Dec. 2011, pp. 17–20.
10. E. Rescorla and N. Modadugu, "Datagram Transport Layer Security version 1.2," IETF Internet draft, work in progress, July 2011.
11. P. Wouters et al., "TLS Out-of-Band Public Key Validation," IETF Internet draft, work in progress, Nov. 2011.

Carsten Bormann is Honorarprofessor for Internet technology at the Universität Bremen and a board member of its Center for Computing and Communications Technology (TZI). He's a protocol designer by heart, a standardization geek by necessity, and coauthor of *6LoWPAN: The Wireless Embedded Internet* (Wiley, 2009). Contact him at cabo@tzi.org.

Angelo P. Castellani is a PhD student at the University of Padova. His research interests include the Internet of Things and wireless sensor networks. Castellani has an ME in telecommunication engineering from the University of Rome, Sapienza. Contact him at castellani@dei.unipd.it.

Zach Shelby is cofounder and chief nerd at Sensinode, where he leads Internet of Things research and standardization activities. He's actively involved with IETF standardization and EU research projects, and is coauthor of *6LoWPAN: The Wireless Embedded Internet* (Wiley, 2009). Contact him at zach@sensinode.com.

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

IEEE computer society

PURPOSE: The IEEE Computer Society is the world's largest association of computing professionals and is the leading provider of technical information in the field.

MEMBERSHIP: Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

COMPUTER SOCIETY WEBSITE: www.computer.org

Next Board Meeting: 11–15 June, Seattle, Wash., USA

EXECUTIVE COMMITTEE

President: John W. Walz*

President-Elect: David Alan Grier;* **Past President:** Sorel Reisman;* **VP, Standards**

Activities: Charlene (Chuck) Walrad;† **Secretary:** Andre Ivanov (2nd VP);* **VP,**

Educational Activities: Elizabeth L. Burd;* **VP, Member & Geographic Activities:**

Sattupathuv Sankaran;† **VP, Publications:** Tom M. Conte (1st VP);* **VP, Professional**

Activities: Paul K. Joannou;* **VP, Technical & Conference Activities:** Paul R. Croll;†

Treasurer: James W. Moore, CSDP;* **2011–2012 IEEE Division VIII Director:** Susan K.

(Kathy) Land, CSDP;† **2012–2013 IEEE Division V Director:** James W. Moore, CSDP;†

2012 IEEE Division Director VIII Director-Elect: Roger U. Fujii†

*voting member of the Board of Governors

†nonvoting member of the Board of Governors

BOARD OF GOVERNORS

Term Expiring 2012: Elizabeth L. Burd, Thomas M. Conte, Frank E. Ferrante, Jean-Luc Gaudiot, Paul K. Joannou, Luis Kun, James W. Moore, William (Bill) Pitts

Term Expiring 2013: Pierre Bourque, Dennis J. Frailey, Atsuhiko Goto, André Ivanov, Dejan S. Milojicic, Paolo Montuschi, Jane Chu Prey, Charlene (Chuck) Walrad

EXECUTIVE STAFF

Executive Director: Angela R. Burgess; **Associate Executive Director, Director,**

Governance: Anne Marie Kelly; **Director, Finance & Accounting:** John Miller;

Director, Information Technology & Services: Ray Kahn; **Director, Membership**

Development: Violet S. Doan; **Director, Products & Services:** Evan Butterfield;

Director, Sales & Marketing: Chris Jensen

COMPUTER SOCIETY OFFICES

Washington, D.C.: 2001 L St., Ste. 700, Washington, D.C. 20036-4928

Phone: +1 202 371 0101 • **Fax:** +1 202 728 9614

Email: hq.ofc@computer.org

Los Alamitos: 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314 • **Phone:** +1 714 821 8380 • **Email:** help@computer.org

Membership & Publication Orders

Phone: +1 800 272 6657 • **Fax:** +1 714 821 4641 • **Email:** help@computer.org

Asia/Pacific: Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-0062, Japan • **Phone:** +81 3 3408 3118 • **Fax:** +81 3 3408 3553 • **Email:** tokyo.ofc@computer.org

IEEE OFFICERS

President: Moshe Kam; **President-Elect:** Gordon W. Day; **Past President:** Pedro A.

Ray; **Secretary:** Roger D. Pollard; **Treasurer:** Harold L. Flescher; **President, Standards**

Association Board of Governors: Steven M. Mills; **VP, Educational Activities:** Tariq

S. Durrani; **VP, Membership & Geographic Activities:** Howard E. Michel; **VP,**

Publication Services & Products: David A. Hodges; **VP, Technical Activities:** Donna

L. Hudson; **IEEE Division V Director:** James W. Moore, CSDP; **IEEE Division VIII**

Director: Susan K. (Kathy) Land, CSDP; **President, IEEE-USA:** Ronald G. Jensen

revised 24 Jan. 2012

