The Internet Protocol Journal

June 2016 Volume 19, Number 2

A Quarterly Technical Publication for Internet and Intranet Professionals

In This Issue

You can download IPJ back issues and find subscription information at:
www.protocoljournal.org

ISSN 1944-1134

From The Editor

A major design feature of the *Internet Protocol* (IP) is its ability to run over a variety of underlying network technologies. If you look through the *Request For Comments* (RFC) document series, you will find numerous specifications of the form "IP over xxx," where "xxx" is anything from Ethernet to X.25, Frame Relay, Bluetooth, WiFi, and even "Avian Carriers" (pigeons), the latter being one of the more famous April Fools RFCs. Because each of these technologies has different capabilities in terms of how much data can be carried in a "packet" or datagram, IP employs the concept of *fragmentation* and *reassembly* in cases where the originating datagram is larger than what the underlying network medium can support. In our first article, Geoff Huston explains fragmentation and reassembly for both IPv4 and IPv6. Special thanks go to Mansour Ganji of Vodafone New Zealand for suggesting this topic.

We've covered various aspects of the *Internet of Things* (IoT) in previous editions of this journal. This time, Akbar Rahman and Chonggang Wang discuss ongoing work within the *Internet Engineering Task Force* (IETF) and elsewhere to develop Resource Discovery mechanisms for IoT devices.

The long-awaited proposal to transition the *Internet Assigned Numbers Authority* (IANA) Stewardship Functions from the U.S. Government to a new entity was finally submitted in early March of this year. Vint Cerf explains the history and background of this process. At the end of his article you will find pointers to further information about this important Internet milestone.

As always, we welcome your feedback, suggestions, book reviews, articles, and sponsorship support. You can contact us by e-mail to **ipj@protocoljournal.org** and visit our website for subscription information, back issues, author guidelines, sponsor information, and much more.

—Ole J. Jacobsen, Editor and Publisher
ole@protocoljournal.org

Fragmentation

by Geoff Huston, APNIC

One of the more difficult design exercises in packet-switched network architectures is that of the design of packet fragmentation. In this article I will examine *Internet Protocol* (IP) *packet fragmentation* in detail and look at the design choices made by IP Version 4, and then compare that with the design choices made by IP Version 6.

Packet-switched networks dispensed with a constant time base, in turn allowing individual packets to be sized according to the needs of the application as well as the needs of the network. Smaller packets have a higher ratio of packet header to payload, and are consequently less efficient in data carriage. On the other hand, within a packet-switching system the smaller packet can be dispatched faster, reducing the level of *head-of-line blocking* in the internal queues within a packet switch and potentially reducing network-imposed jitter as a result. Larger packets allow larger data payloads, in turn allowing greater carriage efficiency. Larger payloads per packet also allows a higher internal switch capacity when measured in terms of data throughput. But larger packets take longer to be dispatched, potentially causing increased jitter.

Various network designs have adopted various parameters for packet size. Ethernet, standardized in the mid-1970s, adopted a variable packet size, with supported packet sizes of between 64 and 1,500 octets. *Fiber Distributed Data Interface* (FDDI), a fibre ring local network, used a packet size of up to 4,478 octets. Frame Relay used a variable packet size of between 46 and 4,470 octets. The choice of variable-sized packets allows applications to refine their behaviour. Jitter and delay-sensitive applications, such as digitised voice, may prefer to use a stream of smaller packets to attempt to minimise jitter, while reliable bulk data transfer may choose a larger packet size to increase carriage efficiency. The nature of the medium may also have a bearing on this choice. If there is a high *Bit Error Rate* (BER) probability, then reducing the packet size minimises the impact of sporadic errors within the data stream, possibly increasing throughput.

IPv4 and Packet Fragmentation

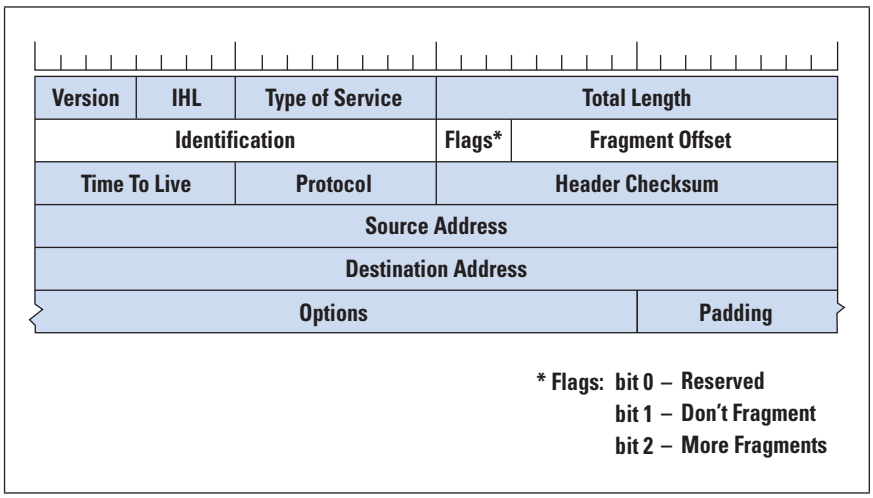
In designing a network protocol that is intended to operate over a wide variety of substrate carriage networks, the designers of IP could not rely on a single packet size for all transmissions. Instead the IP designers of the day provided a packet-length field in the IP Version 4 header[1]. This field was a 16-bit octet count, allowing for an IP packet to be anywhere from the minimum size of 20 octets (corresponding to an IP header without any payload) to a maximum of 65,535 octets. So IP itself supports a variable size packet format. But which packet size should an implementation use?

The tempting answer is to use the maximum size permitted by the network interface of the local device, with the caveat that an application may nominate the explicit use of smaller-sized packets. But there is a complication here. The Internet was designed as an "internetwork" network system, allowing an IP packet to undertake an end-to-end journey from source to destination across numerous different networks. For example, consider a host connected to a FDDI network, which is connected to an Ethernet network. The FDDI-connected host may elect to send a 4,478-octet packet, which will fit into a FDDI network, but the packet switch that attempts to pass the packet into the Ethernet network will be unable to do so because it is too large.

The solution adopted by IPv4 was *forward fragmentation*. The basic approach is that any IP router that is unable to forward an IP packet into the next network because the packet is too large for this network may split the packet into a set of smaller IP fragments, and forward each of these fragments. The fragments continue along the network path as autonomous packets, and the addressed destination host is responsible for reassembling these fragments back into the original IP packet.

The behaviour is managed by a 32-bit field in the IPv4 header, which is subdivided into three sub-fields (Figure 1).

Figure 1: IPv4 Packet Header Fragmentation Fields



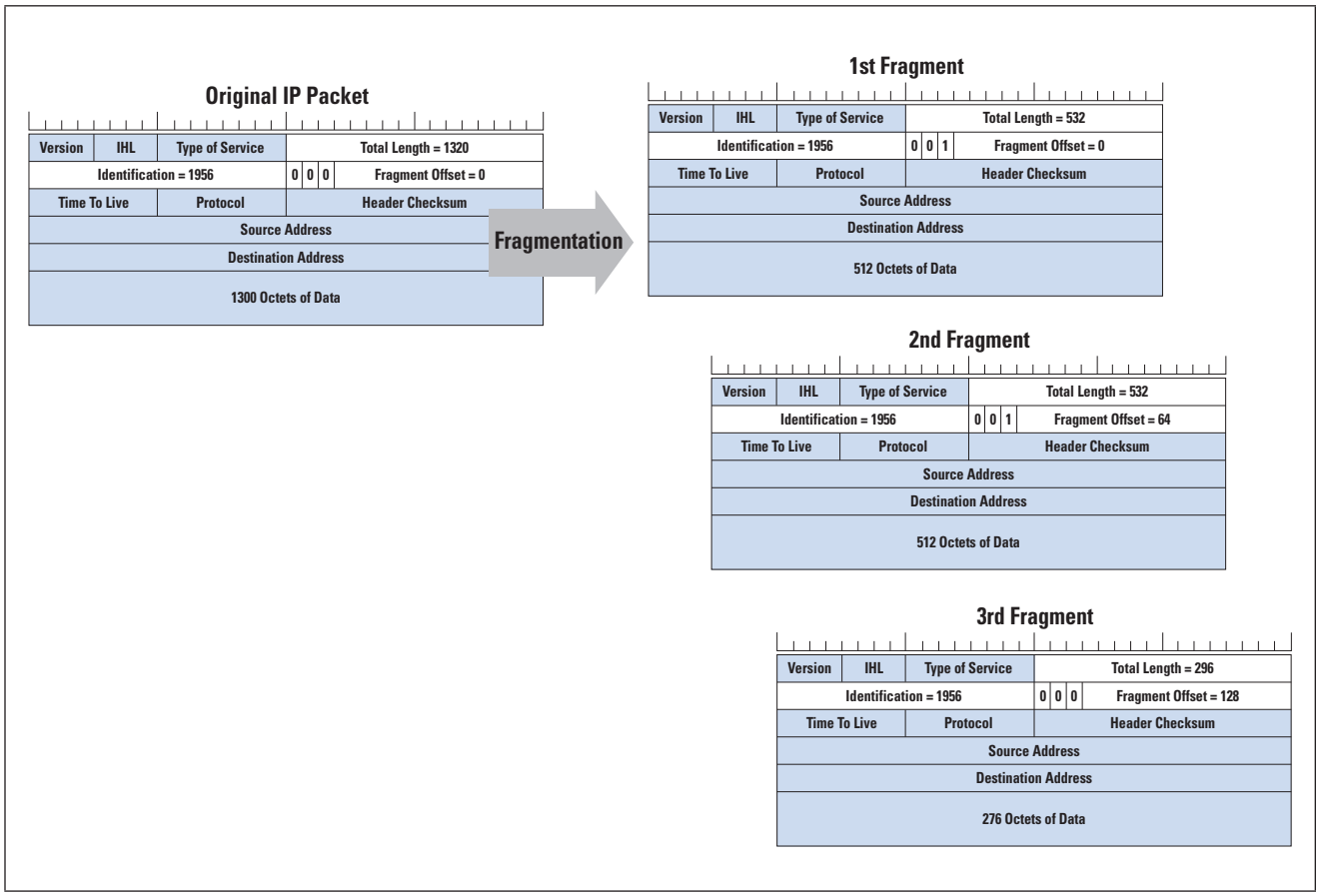
The first sub-field is a 16-bit packet identifier which allows fragments that share a common packet-identifier value to be identified as fragments of the same original packet.

The second sub-field is a 3-bit vector of flags. The first bit is unused. The second is the *Don't Fragment* flag. If this flag is set the packet cannot be fragmented, and must be discarded when it cannot be forwarded. The third bit is the *More Fragments* field, and is set for all fragments except the final fragment.

The third sub-field is the fragmentation offset value that is the offset of this fragment from the start of the IP payload of the original packet, measured in *octawords* (64-bit units).

For example, a router attempting to pass a 1320-octet IP packet into a network whose maximum packet size is 532 octets would need to split the IP packet into three parts. The first packet would have a fragmentation offset of 0 and the *More Fragments* bit set. The total length would be 532 octets, and the IP payload would be 512 octets, making a total of 532 octets for the packet. The second packet would have a fragmentation offset value of 64, the *More Fragments* bit set, total length of 532, and an IP payload of 512 octets, making a total of 532 octets for the packet. The third packet would have a fragmentation offset value of 128, the *More Fragments* bit clear, total length of 296, and an IP payload of 276 octets, making a total of 296 octets for the packet (Figure 2).

Figure 2: Example of IPv4 Packet Fragmentation



The advantage of this approach is that as long as it is permissible to fragment the IP packet, all packet flows are "forward,"meaning that the sending host is unaware that packet fragmentation is occurring, and all the IP fragment packets continue to head towards the original destination, where they are reassembled.

Another advantage is that while the router performing the fragmentation has to expend resources to generate the packet fragments, the ensuing routers on the path to the destination have no additional processing overhead, assuming that they do not need to further fragment these IP fragments. Fragments can be delivered in any order, so the fragments may be passed along parallel paths to the destination.

To complete the IPv4 story we must describe the IPv4 behaviour when the *Don't Fragment* bit is set. The router that is attempting to fragment such a packet is forced to discard it. Under these circumstances the router is expected to generate an *Internet Control Message Protocol* (ICMP) "Unreachable" error (type 3, code 4), and in later versions of the IP specification it was expected to add the *Maximum Transmission Unit* (MTU) of the next-hop network into the ICMP packet. The original sender would react to receiving such an ICMP message by changing its local maximum packet size associated with that particular destination address, and thus it would "learn" a viable packet size for the path between the source and destination.

Evaluating IPv4 Fragmentation

A case has been made that the IP approach to fragmentation contributed to its success. This design allowed transport protocols to operate without consideration of the exact nature of the underlying transmission networks, and avoid additional protocol overhead in negotiating an optimal packet size for each transaction. Large *User Datagram Protocol* (UDP) packets could be transmitted and fragmented in real time as required without requiring any form of end-to-end network path packet size discovery. This approach allowed IP to be used on a wide variety of substrate networks without requiring extensive tailoring.

But it wasn't all good news.

Cracks in the IP fragmentation story were described in a 1987 paper by Kent and Mogul, "Fragmentation Considered Harmful."[2]

TCP has always attempted to avoid IP fragmentation. The initial opening handshake of *Transmission Control Protocol* (TCP) exchanges the local and remote *Maximum Segment Size* (MSS), and the sender will not send a TCP segment larger than that notified by the remote end at the start of the TCP session. The reason that TCP attempted to avoid fragmentation was that fragmentation was inefficient under conditions of packet loss in a TCP environment. Lost fragments can be repaired only by resending the entire packet, including resending all those fragments that were successfully transmitted in the first place. TCP can perform a data repair more efficiently if it limits its packet size to one that does not entail packet fragmentation.

This form of fragmentation also posed vulnerabilities for hosts. For example, an attacker could send a stream of fragments with a close to maximally sized fragment offset value, and random packet identifier values.

If the receiving host believed that the fragments represented genuine incoming packets, then a credulous implementation might generate a reassembly buffer for each received fragment that may represent a memory buffer starvation attack. It is also possible, either through malicious attack or by poor network operation, that fragments may overlap or overrun, and the task of reassembly requires care and attention in implementation of fragment reassembly.

Lost fragments represent a slightly more involved problem than lost packets. The receiver has a packet reassembly timer upon the receipt of the first fragment, and will continue to hold this reassembly state for the reassembly time. The reassembly timer is a factor in the maximal count of packets in flight, because the packet identifier cannot be recycled within a period defined by the sender-received path delay plus the reassembly timer of the receiver. For higher-delay high-capacity network paths, this limit of 65,535 packets in flight can be a potential performance bottleneck[3].

Fragmentation also consumes router processing time, forcing the processing of oversized packets from a highly optimised fast path into a processor queue.

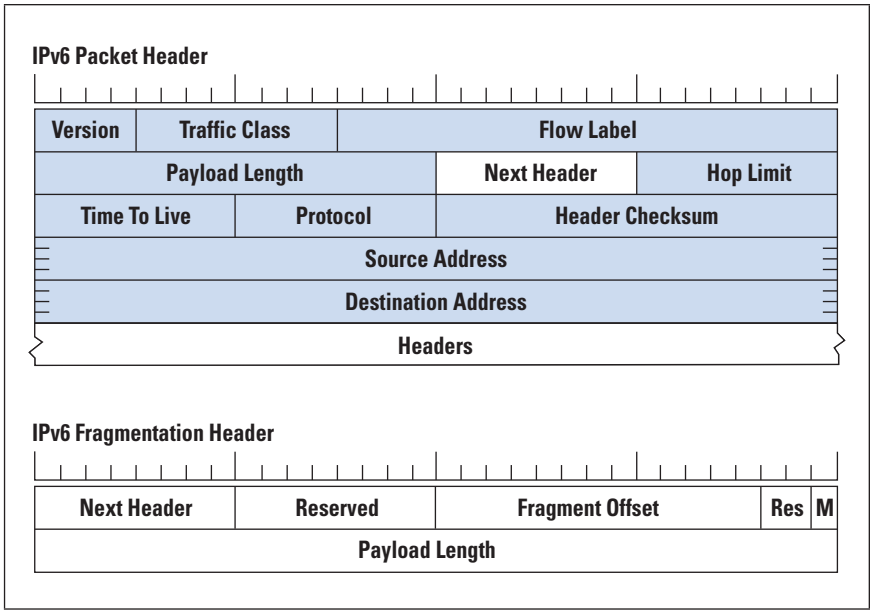
And then there is the "middleware problem." Filters and firewalls perform their function by applying a set of policy rules to the packet stream. But these rules typically require the presence of the transport layer header. How can a firewall handle a fragment? One option is to pass all trailing fragments through without inspection, but this process exposes the internal systems to potential attack[4]. Another option is to have the firewall rebuild the original packet, apply the filter rules, and then refragment the packet and forward it on if the packet is accepted by the filter rules. However, this process now exposes the firewall to various forms of memory starvation attack. *Network Address Translators* (NATs)[5] that use the transport-level port addresses as part of the NAT binding table have a similar problem with trailing fragments. The conservative approach is for the NAT to reassemble the IP packet at the NAT, apply the NAT transform, and then pass the packet onward, fragmenting as required.

IPv6 and Fragmentation

When it came time to think about the design of what was to become IPv6, the *forward fragmentation* approach was considered to be a liability, and while it was not possible to completely discard IP packet fragmentation in IPv6, there was a strong desire to redefine its behaviour.

The essential change between IPv4 and IPv6 is that in IPv6 the *Don't Fragment* bit is always on, and because it's always on, it's not explicitly contained in the IPv6 packet header (Figure 3). There is only one fragmentation flag in the Fragmentation Header, the *More Fragments* bit, and the other two bits are reserved. The other change was that the packet-identifier size was doubled in IPv6, using a 32-bit packet identifier field.

Figure 3: IPv6 Packet Header and Fragmentation Header



An IPv6 router cannot fragment an IPv6 packet, so if the packet is too large for the next hop the router is required to generate an ICMPv6 type 2 packet, addressed to the source of the packet with a *Packet Too Big* (PTB) code, and also providing the MTU size of the next hop. While an IPv6 router cannot perform packet fragmentation, the IPv6 sender may fragment an IPv6 packet at the source.

Evaluating IPv6 Packet Fragmentation

The hope was that these IPv6 changes would fix the problems seen with IPv4 and fragmentation.

Our experience appears to point to a different conclusion.

The first problem is that there is widespread ICMP packet filtering in today's Internet. For IPv4 this approach was basically a reasonable defense tactic, and if you were willing to have a packet fragmented you cleared the *Don't Fragment* bit before sending the packet so that you didn't rely on receiving an ICMP message to indicate a path sender MTU problem. But in IPv6 the equivalent *Don't Fragment* bit function is jammed in the "on" position, and fragmentation can be performed only if the original sender receives the ICMPv6 PTB message and then resends the packet fragmented into a size that meets the specified MTU size. But when ICMPv6 PTB messages are filtered, the large packet is silently discarded within the network without any discernible trace. Attempts by the sender to time out and resend the large IPv6 packet will meet with the same fate, so this situation can lead to a wedged state.

This scenario has been seen in the context of the HTTP protocol, where the path MTU is smaller than the MTU of the host systems at either end. The TCP handshake completes because none of the opening packets is large. The opening HTTP GET packet also makes it through because this packet is normally not a large one.

However, the first response may be a large packet. If it is silently discarded because of the combination of fragmentation required and ICMPv6 filtering, then neither the client nor the server can repair the situation. The connection hangs.

The second problem is that the ICMPv6 PTB message is sent backwards to the source from the interior of a network path. Oddly enough, the IPv6 ICMP PTB message is perhaps the one critical instance in the entire IP architecture in which the IP source address is interpreted by anything other than the intended destination. The problems here include path asymmetry, in that the source address may be unreachable from the point of the generation of the ICMP packet. There is also the case of tunneling IP-in-IP. Because IPv6 fragmentation can be performed only at the source, should the ICMP message be sent to the tunnel ingress point or to the original source? Using the tunnel ingress assumes that the tunnel egress performs packet reassembly, potentially burdening the tunnel egress. This situation is further confounded in the cross protocol case of IPv6-in-IPv4 and IPv4-in-IPv6.[6]

The third problem is the combination of IPv6 packet fragmentation and UDP. UDP is an unreliable datagram delivery service, so a sender of a UDP packet is not expected to cache the packet and be prepared to resend it. A UDP packet-delivery error can occur only at the level of the application, not at the IP or UDP protocol level. So what should a host do upon receipt of an ICMP PTB message if resending the IP packet is not an option? Given that the sender does not cache sent UDP packets, the packet header in the ICMPv6 message is unhelpful. Because the original packet was UDP, the sender does not necessarily have a connection state, so it is not clear how this information should be retained and how and when it should be used. How can a receiver even tell if an ICMPv6 PTB packet is genuine? If the sender adds an entry into its local IPv6 forwarding table, it is exposing itself to a potential resource starvation problem. A high volume flow of synthetic PTB messages has the potential to bloat the local IPv6 forwarding table. If the sender ignores the PTB message, the application is left to attempt to recover the transaction.

If it makes little sense in the context of an attempt to fragment a UDP packet, it makes less sense to fragment a TCP packet. In the context of a TCP session, a received ICMPv6 PTB message can be interpreted as a redefinition of the remote end MSS value, and the outgoing TCP segments can be reframed to conform to this MSS.

Wither Fragmentation?

The basic problem here is that the network was supposed to operate at the IP level and be completely unaware of transport, implying that IP-level fragmentation was meant to work in a manner that does not involve transport protocol interaction.

So much of today's network (firewalls, filters, etc.) is transport-aware and the trailing fragments have no transport context, meaning that transport-aware network middleware needs to reassemble the packet, and this process could represent a problem and a *Denial of Service* (DoS) vulnerability in its own right.

So is fragmentation worth it at all?

I'd still say that it's more useful to have it than not. But the IPv4 model of *forward fragmentation* in real time has proved to be more robust than the IPv6 model because the IPv4 model requires only that traffic flows in one direction and is an IP-level function. It has its problems, and no doubt the papers that warned that IP fragmentation was "harmful" were sincere in taking that view[2]. But it is possible to make it worse, and the IPv6 model requiring a *backward* ICMPv6 message from the interior of the network was in retrospect a decision that did just that!

So what should we do now?

It is probably not a realistic option to try to alter the way that IPv6 manages fragmentation. There was an effort in 2013 in one of the IETF's IPv6 Working Groups to deprecate the IPv6 Fragment Header[7]. That's possibly an overreaction to the problem of packet fragmentation and IPv6, but there is no doubt that the upper-level protocols simply should not assume that IPv6 fragmentation operates in the same manner as IPv4, or even operates in a reliable manner at all!

The implication is that transport protocol implementations, and even applications, should try to manage their behaviour on the assumption that ICMP message filtering is sufficiently prevalent that it is prudent to assume that all ICMP messages are dropped. The result is a default assumption that large IPv6 packets that require fragmentation are silently dropped.

How can we work around this problem and operate a network that uses variable-sized packets but cannot directly signal when a packet is too large? RFC 4281[8] describes a *Path MTU Discovery* process that operates without relying on ICMP messages, and IPv6 TCP implementations should rely on this mechanism to establish and maintain a viable MTU size that can support packet delivery. In this way TCP can manage the path MTU and the application layer need not add explicit functions to manage persistent silent drop of large segments.

Path MTU Discovery

Path MTU discovery was specified in RFC 1191[9]. The approach was to send packets with the *Don't Fragment* bit set. When a router on the path is unable to forward the packet because it is too large for the next hop, the *Don't Fragment* field directs the router to discard the packet and send a *Destination Unreachable* ICMP message with a code of "Fragmentation Required and DF set" (type 3, code 4).

RFC 1191 advocated the inclusion of the MTU of the next-hop network in the next field of the ICMP message.

A host receiving this form of ICMP message should store the new MTU in the local forwarding table, with an associated time to allow the entry to time out. Also the host should identify all active TCP sessions that are connected to the same destination address as given in the IP packet header fragment of the ICMP message, and notify the TCP session of the revised path MTU value.

RFC 1981[10] defined much the same behaviour for IPv6, relying on the MTU information conveyed in the ICMPv6 PTB message in exactly the same manner as its IPv4 counterpart.

The problem of filtered ICMP messages is a difficult one, and attention has turned to path MTU Discovery ideas that do not rely on an ICMP message to operate correctly. RFC 4821 describes a mechanism that refines the RFC 1191 ICMP-based process by adding an alternate process that is based on detection and reporting of packet loss as an inference of path MTU problems when there is no ICMP feedback. This process uses a probe procedure that attempts to establish a working MTU size through probing the path with various sized packets to establish the upper-bound MTU. The trade-off here is the number of round-trip intervals taken to perform the probes and the accuracy of the path MTU estimate.

Because these probes take time, the entire exercise tends to be of value only in long-held TCP and TCP-like flows. For shorter sessions the pragmatic advice is to clamp the local MTU to a conservative value (1,280 is a good first choice for IPv6, and RFC 4821 also suggests 1,024 for IPv4) and try to avoid the entire issue of fragmentation in the first place.

UDP is a different story. The lightweight UDP protocol shim does not admit much in the way of additional functions, and one possible approach is to insist that UDP-based applications limit themselves to the local MTU size, or to be even more conservative, limit themselves to the 1,280-octet IPv6 minimum unfragmented packet size.

The major issue with such advice for UDP lies in the *Domain Name System* (DNS). Efforts to improve the security of the DNS with *Domain Name System Security Extensions* (DNSSEC) have added additional data into DNS responses. In addition, if you want to maintain the lightweight efficiency of the DNS, then it's not possible to keep DNSSEC responses under 1500 octets all the time, let alone under 1,280 octets. One option here is to insist that larger DNS responses use TCP, but this option imposes some considerable cost overhead on the operation of the DNS. What the DNS has chosen to do appears to represent a reasonable compromise.

The first part of the approach is that the management of the packet MTU is passed into the application layer. The application conventionally operates with a maximum UDP payload size that assumes that UDP fragmentation is working, and a DNS query normally offers an *Extension Mechanisms for DNS* (EDNS) buffer size of 4,096 octets. The responder uses this information to assemble its UDP response of up to 4,096 octets in length, a process that conventionally causes the source to perform UDP packet fragmentation for large responses. This fragmented response may not reach the querier for a variety of reasons, in which case the EDNS buffer size is dropped back to a more conservative value that is not expected to trigger UDP fragmentation, but may not be able to contain the complete response. The intended result is that if the network cannot complete a UDP transaction that entails a fragmented UDP response, the transaction is repeated using a smaller maximum UDP packet size, and the truncated response explicitly signals to the client to retry the query using TCP[12]. This process is protocol-agnostic, in that it operates as intended in the case of IPv4 forward fragmentation, where trailing fragments are filtered out by middleware, and in the case of IPv6, where there is no forward fragmentation, and it operates whether or not the responder receives any ICMP PTB messages.

Conclusion

What we have learned through all this discussion is that packet fragmentation is extremely challenging, and is sensibly avoided if at all possible.

Rather than trying to bury packet fragmentation to an IP-level function performed invisibly at the lower levels of the protocol stack, a robust approach to packet fragmentation requires a more careful approach that lifts the management of Path MTU into the end-to-end transport protocol and even into the application.

IPv6 UDP-based applications that want a lightweight operation should look at keeping their UDP packets under the IPv6 1,280-octet unfragmented packet limit. And if that's not possible, then the application itself needs to explicitly manage Path MTU, and not rely on the lower levels of the protocol stack to manage it.

IPv6 TCP implementations should never assume that IPv6 PTB messages are reliably delivered. High-volume flows should use RFC 4821 Path MTU Discovery and management procedures to ensure that the TCP session can avoid Path MTU blackholing. For short flows, MSS clamping still represents the most viable approach.

I'm not sure that we should go as far as deprecating IP fragmentation in IPv6. The situation is not that dire. But we should treat Path MTU with a lot more respect, and include explicit consideration of the trade-offs between lightweight design and robust behaviour in today's network.

References

[1] Jon Postel, "Internet Protocol," RFC 791, September 1981.

[2] Kent, C. and J. Mogul, "Fragmentation Considered Harmful," Proc. SIGCOMM '87 Workshop on Frontiers in Computer Communications Technology, August 1987.

[3] Matt Mathis, Ben Chandler, and John W. Heffner, "IPv4 Reassembly Errors at High Data Rates," RFC 4963, July 2007.

[4] G. Ziemba, D. Reed, and P. Traina, "Security Considerations for IP Fragment Filtering," RFC 1858, October 1995.

[5] Geoff Huston, "Anatomy: A Look Inside Network Address Translators," *The Internet Protocol Journal*, Volume 7, No. 3, September 2004.

[6] Pekka Savola, "MTU and Fragmentation Issues with In-the-Network Tunneling," RFC 4459, April 2006.

[7] Bonica, R. W. Kumari, R. Bush, and H. Pfeifer, "IPv6 Fragment Header Deprecated," Internet Draft, work in progress, **draft-bonica-6man-frag-deprecate**, July 2013.

[8] Matt Mathis and John W. Heffner, "Packetization Layer Path MTU Discovery," RFC 4821, March 2007.

[9] Jeffrey C. Mogul and Stephen E. Deering, "Path MTU Discovery," RFC 1191, November 1990.

[10] Stephen E. Deering, Jack McCann, and Jeffrey Mogul, "Path MTU Discovery for IP Version 6," RFC 1981, August 1996.

[11] Mukesh Gupta, Stephen E. Deering, and Alex Conta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 4443, March 2006.

[12] Paul Vixie, Joao Damas, and Michael Graff, "Extension Mechanisms for DNS (EDNS(0))," RFC 6891, April 2013.

[13] Godred Fairhurst and Lars Eggert, "Unicast UDP Usage Guidelines for Application Designers," RFC 5405, November 2008.

GEOFF HUSTON, B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region. He has been closely involved with the development of the Internet for many years, particularly within Australia, where he was responsible for building the Internet within the Australian academic and research sector in the early 1990s. He is author of numerous Internet-related books, and was a member of the Internet Architecture Board from 1999 until 2005. He served on the Board of Trustees of the Internet Society from 1992 until 2001. At various times Geoff has worked as an Internet researcher, an ISP systems architect, and a network operator. E-mail: **gih@apnic**

Resource Discovery in the Internet of Things

by Akbar Rahman and Chonggang Wang,
InterDigital Communications, Inc.

The *World Wide Web* (WWW or Web) is a global collection of connected documents and other resources that reside on the Internet. The introduction of the *Internet of Things* (IoT) is expected to dramatically increase the size of the Web in the near future and thus necessitates a fundamental change to the existing mechanisms of discovering resources. In IoT, the vision is that a significant number of new types of devices (or "things") such as fridges, car sensors, traffic lights, and so on will be dynamically connected to the Web for communication and control. These IoT devices will have radically different characteristics from existing Web servers and users. This article looks at a key protocol development occurring in the *Internet Engineering Task Force* (IETF) for allowing IoT devices to discover resources via a new logical node called a *Resource Directory* (RD).

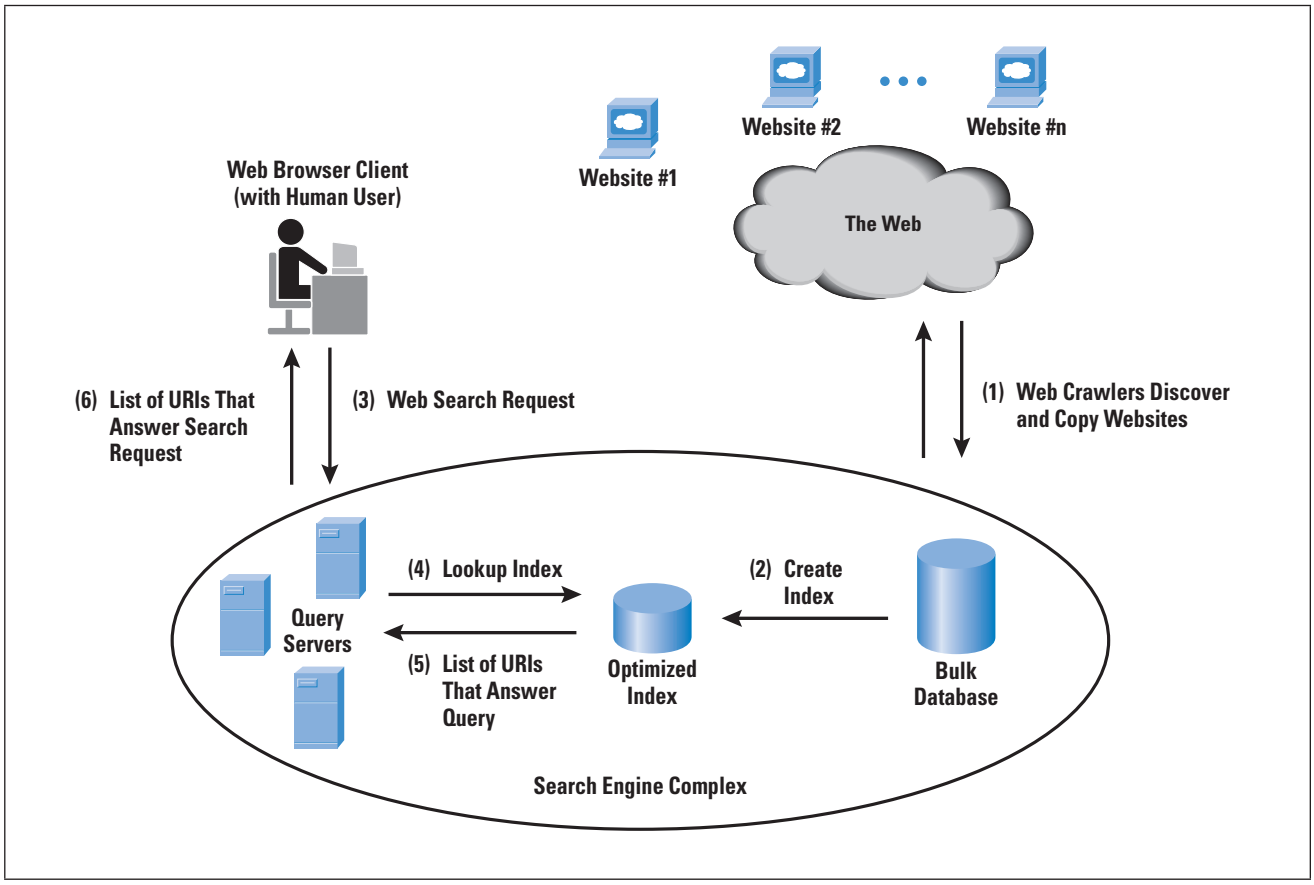
Resource Discovery in the Traditional Web

The basic unit of addressing on the Web is the *Uniform Resource Identifier* (URI), which identifies a resource[1]. The resource may, for example, be a restaurant-review website page for a human user to read. Or in a more abstract form, the resource may be a software process to be triggered by a *Business-to-Business* (B2B) Web application as part of an automated stock market trading system. The key challenge in all cases is how users can quickly find the correct URI for the resource that they are interested in out of all possible URIs in the entire Web space. This process is referred to as *resource discovery.*

The most well-known and powerful resource discovery mechanism in the current Web is the one employed by Web search engines such as *Baidu, Bing, Google, Yahoo,* etc. Specifically, search engines use the mechanism of Web *crawlers* (also called spiders, ants, or robots) to periodically browse the Web to create a dynamic index of the resources of most publicly available websites. A website is defined as a server that hosts resources users can access with the *Hypertext Transfer Protocol* (HTTP)[2]. Human users can then send a search request, via a Web browser client, to look up the specific resources that they are interested in.

Figure 1 shows the overall resource discovery process based on Web crawlers. Figure 1 is given in the context of a search engine, but academic researchers, market research companies, and others follow very similar processes. However, unlike a search engine, these other entities typically do not send crawlers to cover the entire Web to discover all possible resources. Instead, they send crawlers to cover parts of the Web to discover the specific type of resource that they are interested in.

Figure 1: Overview of Traditional Resource Discovery Process by Web Search Engines



For example, a market research company may send its Web crawlers to discover all the resources related to a specific type of product in a given geographic area as part of a pricing comparison study.

In Figure 1, Web crawlers start crawling out from the search engine server to an initially provisioned seed list of URIs. This seed list typically consists of very popular websites with a lot of URIs to other sites (that is, *hyperlinks*). From these initial websites, the Web crawlers then crawl outward to all connected hyperlinks. At each new website that it discovers, the Web crawler creates a copy of the website, which it sends back to the search engine[3]. The search engine records all the received information in a bulk database and later processes it to create an optimized index for fast lookups. Then when a given search request comes from a Web browser client looking for some specific resource, the search engine can go quickly through its index using its own proprietary algorithm to find one or more matches.

Finally, the search engine will return to the client a list of URIs and selected application content pertaining to the resources that match the client's search parameters. This information is then displayed on the user's Web browser interface. Human users will then select ("click") the URI(s) that they want to visit.

Following are some key observations about resource discovery in the traditional Web:

- In terms of network configuration, the search engine functionality, or whichever entity dispatches the Web crawler, is typically located on a set of centralized servers and related databases with high-speed and large-bandwidth Internet connectivity. The resources that the Web crawlers discover may be widely distributed across the entire Internet. The Web browser-based clients that interface with the human users and send the search (lookup) requests are typically located at the edge of the network.
- Search engines primarily use a *pull model* to get resource information. In this approach, the receiving node (that is, search engine) goes out and explicitly requests information (via Web crawlers) from the sending node (content websites). However, a small number of URIs such as the initial seed list of URIs (for example, very popular websites) may be obtained without using the pull model, but these URIs are always a small fraction of the URIs in a search engine index.
- The list of resources the search engine returns for a given Web search (lookup) request may vary from a few URIs to potentially hundreds or even thousands of URIs. The order that these URIs are presented to the human user via the search engine Web interface is called the *ranking* of the resources. This ranking is critical because when a large number of URIs are returned to users for a given search request, users will typically select ("click") only the top few ranked URIs.
- The ranking of resources is ultimately an algorithmic decision internal to the search engine. However, it can be affected by external input such as *Search Engine Optimization* (SEO) techniques that website developers use to try to get search engines to rank their specific URIs higher than other URIs with similar application content. For example, a simple SEO technique is to have website content clearly tagged (titles, section headings, etc.) and correlated to the website metadata. This metadata is an important input for the search index engine. A more sophisticated SEO technique is to have hyperlinks to a given website from as many other websites as possible because search engines consider this factor a measure of content popularity. There are many other SEO techniques[4].

The Resource Discovery Problem in IoT

As mentioned previously, a key characteristic of current Web discovery technology is the use of Web crawlers to fan out and discover resources across the Internet. The implicit assumption in this approach is that Web servers are always active and available for Web crawlers that arrive in an unscheduled manner to discover easily. However, this assumption conflicts with the expected nature of many IoT devices that may have only intermittent connectivity to the Web.

The primary reason for this intermittent connectivity is that many IoT devices have a limited power supply (for instance, battery or solar power). To conserve their power they may "wake up" or become active only when required to perform a specific function. For example, a fire-detection sensor acting as a mini Web server may wake up and connect to the Web only to send a warning message to a remote controller when it senses a certain amount of smoke in its vicinity. At most other times, the fire-detection sensor is "asleep" (that is, in a low power state and not active) and unreachable via the Web. A secondary reason for intermittent connectivity is that many IoT devices are connected to the Web by low-power and lossy wireless networks. These wireless networks are more susceptible to interference and temporary loss of connectivity than traditional wired or cellular networks[5].

Another key difference between IoT devices and other Web infrastructure is that most IoT devices may be deployed in semi-closed networks. For example, the IoT devices such as a lighting or heating control system in a home may have Internet connectivity only through a fire-walled home gateway. So the IoT devices and their associated resources may be accessible by the home owner through a smart phone control application with the proper security credentials from anywhere in the Internet. However, Web crawlers dispatched by a search engine will not discover the home IoT devices because they will not be able to traverse the fire-walled home gateway.

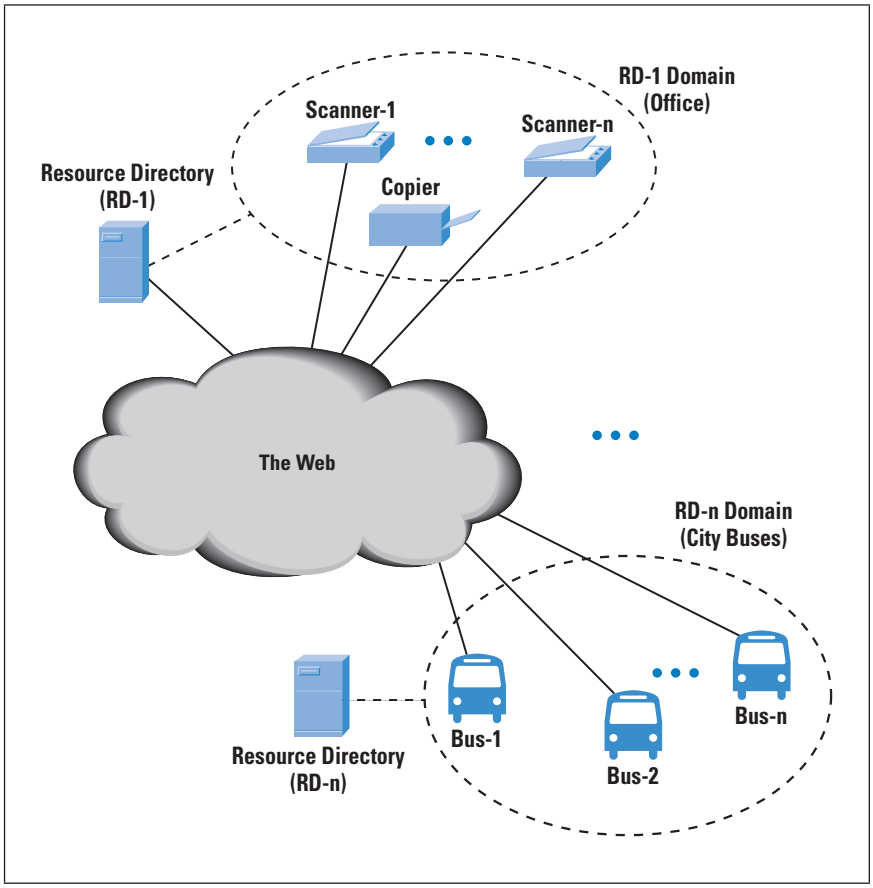
Therefore, the current pull model of Web discovery cannot be applied directly to the expected deployments of IoT networks. In other words, current Web crawler technology is unable to reliably discover a significant percentage of IoT devices that may be asleep or unconnected for significant periods of time, or may be located in semi-closed networks. The result is that traditional Web discovery techniques will not produce accurate discovery results for IoT scenarios.

Resource Directories to Solve the IoT Discovery Problem

The solution currently being standardized in the IETF to address the IoT resource discovery problem is based on a new logical network node called the *Resource Directory* (RD)[6, 7]. The RD idea was originally conceived and validated in the *European Union* (EU)-funded *SENSEI* research program before coming to the IETF for standardization[8]. The RD is defined in [6] to be applicable to a given *domain* and not the entire Web. The domain is a logical grouping of IoT devices that are related to an RD. An RD may support multiple domains. The details of defining the extent of a given domain boundary, however, are left to implementation and are not specified. Typically, the RD domains specified in IETF use cases are building-wide, campus-wide, or city-wide. The domain concept maps well into the expected deployment model of IoT devices in semi-closed networks.

In the simplest case, there would be a one-to-one mapping between each semi-closed network and a domain. The RD approach thus provides a distributed resource discovery mechanism for IoT scenarios. Figure 2 shows some typical RD domains.

Figure 2: Typical Resource Directory Domains



The resource registration step is done in a push fashion by IoT devices acting as mini Web servers pushing their resource information into the RD resource database. Figure 3 shows the architecture of a given RD. All the IoT devices acting as Web servers will first register their resources (URIs) via a registration interface.

Discovery can then be performed on the registered resources by an IoT client using the lookup interface. Mutual authentication, encryption, and access control are required for both the registration and lookup interfaces to ensure security and privacy of the entire resource discovery process.

A given device may use both the registration interface (as a Web server) and the lookup interface (as a client). The client may be located anywhere in the Web, but must have some knowledge regarding which specific RD to direct the resource discovery request to. For example, a newly installed home light controller may perform a lookup on its own home RD to find all the lights installed in the house.

Or, a national smart-grid controller may perform a lookup on a known RD in a remote city to find all the electric transformers located in that city.

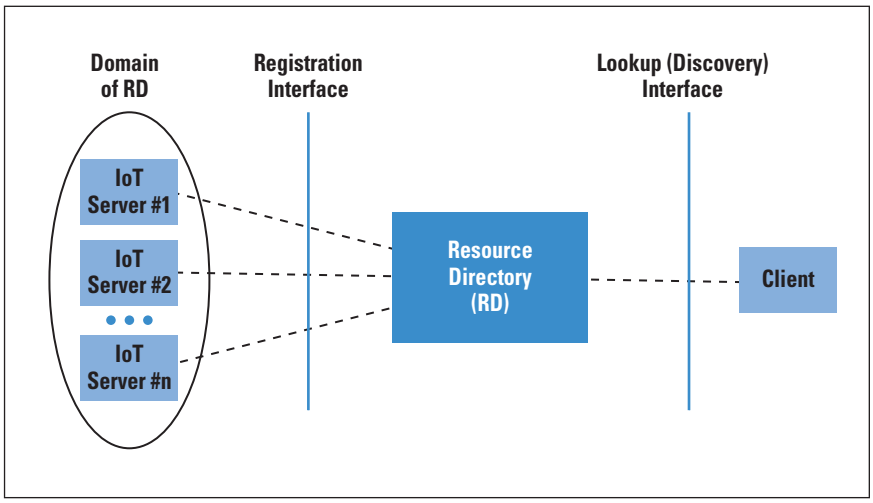


Figure 3: IoT Resource Directory Architecture (adapted from [6])

IoT devices communicate with the RD using a *Representational State Transfer* (REST)-based protocol similar to HTTP but optimized for IoT. This protocol is referred to as the *Constrained Application Protocol* (CoAP)[9]. The resource information pushed by the IoT servers into the RD uses CoAP messages with a specific payload format termed the *Link Format*[7]. Only the URI, hyperlinks, and some metadata are sent from the IoT device to the RD. Application content is not sent to the RD. Table 1 shows a comparison of the main resource discovery features of a traditional Web search engine and an IoT RD.

Resource Directory Protocol Considerations

As mentioned previously, CoAP is a Web transfer protocol, similar to HTTP, but optimized for IoT scenarios. CoAP provides a request/response interaction model between clients and servers. It supports key Web concepts such as URIs and Internet media types. CoAP messages are sent over *User Datagram Protocol* (UDP), and the CoAP header is encoded in a simple binary format. A CoAP request consists of a method (that is, GET, PUT, POST, and DELETE) that is applied to a resource identified by its URI, and a payload described by an Internet media type as well as other metadata.

CoAP messages may easily be interworked with HTTP in the forward or reverse directions via special cross-protocol proxies[9]. In addition, CoAP uses *Datagram Transport Layer Security* (DTLS)[10] to provide a secure session between the communicating parties.

In CoAP, every physical IoT device is assumed to have one or more resources, each identified by a URI. A resource may contain application information gathered by the IoT device (for example, temperature), or may be a method to control the device (for example, turn it ON/OFF). An example CoAP request and response pair is shown in Table 2.

Table 1: Comparison of Resource Discovery Features of Web Search Engine versus IoT Resource Directory

Characteristic	Traditional Web Search Engine (for example, Google)	IoT Resource Directory
(1) How is resource information initially received by node?	Mainly pulled from target website by Web crawlers after initial visit	Mainly pushed by target IoT devices directly to RD (usually after power-up)
(2) How is updated resource information transferred to node?	Pulled from target website by Web crawlers that revisit according to their search engine policy	Pushed by target IoT device directly to RD according to their own update policy
(3) What resource information is transferred to node?	The entire website (that is, URIs, hyperlinks, metadata, and most application content)	URIs, hyperlinks, and metadata (but no application content is transferred)
(4) What transfer protocols are supported?	HTTP	CoAP (Also some limited HTTP support exists. Further possible enhancements are discussed in [12].)
(5) What is the scope of a client discovery request for resources?	Global (that is, covers entire Internet)	Local within given RD domain (for example, city-wide)
(6) Typical end user that generates query for resources.	Human user (via a Web browser client) sends a search request	IoT device (that is, acting as both the client and end user) sends lookup request May also be used occasionally by human user (for example, via a CoAP-enabled Web browser client as part of management activities)
(7) Are resource discovery results ranked?	Yes	No (but being discussed as a future enhance-ment in [12])
(8) Are the resource discovery results machine readable?	No (but may support it in the future with further adoption of Semantic Web concept)	Yes (that is, results strictly follow Link Format[7])

Table 2: Example CoAP GET Request and Response

Request	**`GET coap://heater.net/temperature`** Note: Where Method = **`GET`** URI = **`coap://heater.net/temperature`** URI-Scheme component = **`coap://`** URI-Host component = **`heater.net`** (or alternatively may be an IP address and Port Number) URI-Path component = **`/temperature`**
Response	2.05 Content "22.3 C" Note: Where Response code = 2.05 (indicating successful processing) Payload = 22.3 Celsius (C) temperature reading

The following sections describe the key protocol steps and security characteristics related to RDs.

Finding the Resource Directory

The first step is for the IoT devices, or *End Points* (EPs) as they are called in [6], to find the appropriate RD. The most dynamic method for finding the RD is using IP multicast. Specifically, the device sends a CoAP multicast message to the CoAP IPv4 or IPv6 addresses reserved for this purpose[11]. An alternative method would be, for example, factory preprovisioning of the RD information in the device.

Assuming the IP multicast method of finding the RD, each device (EP) sends a CoAP GET request to a specific URI-Path as shown in Figure 4. Specifically, the CoAP GET request is sent by multicast to the reserved "`/.well-known/core`" URI-Path. (Note that the URI-Scheme and URI-host components are not shown for simplicity in this and subsequent figures.) All the devices in the domain will then get this request because it is sent by IP multicast[11]. However, only the RD will reply because the request URI has a query string for resource type (rt) added to the end (that is, `?rt=core.rd*`), indicating that the message is meant for the RD. The RD then responds indicating its URI-Path (that is, `/rd`) for subsequent registration or lookup requests[6].

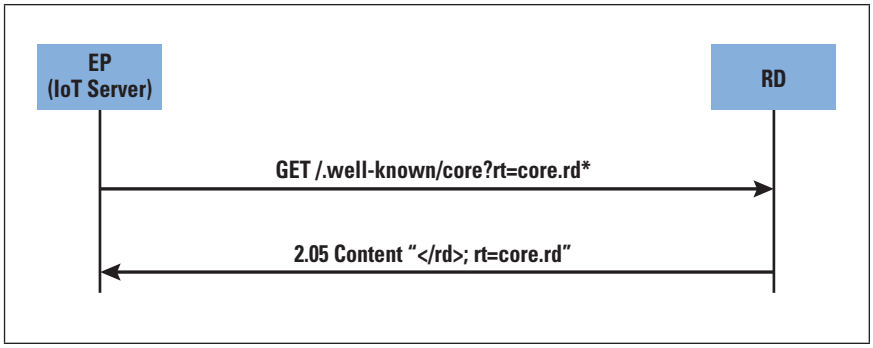
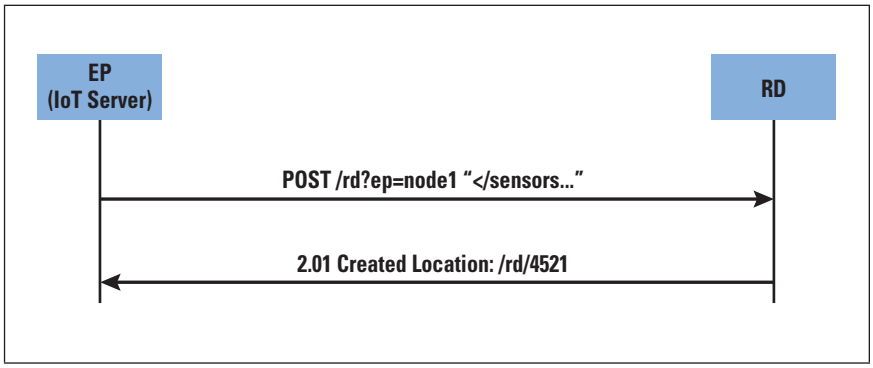


Figure 4: Finding a Resource Directory (adapted from [6])

Registering Resources

After finding the RD, each IoT device (EP) will register its own resources to the RD using the RD registration interface as shown in Figure 5. This registration is accomplished by each device sending a CoAP POST request directly to the RD with its list of URIs (that is, `/sensor...`) in the message payload, along with a query string identifying the registering device (that is, `?ep=node1`). The message payload containing the list of URIs being registered is formatted in the Link Format[7]. The RD then responds with the resulting URI-Path (that is, `/rd/4521`) that it created to store the resources of the device[6].

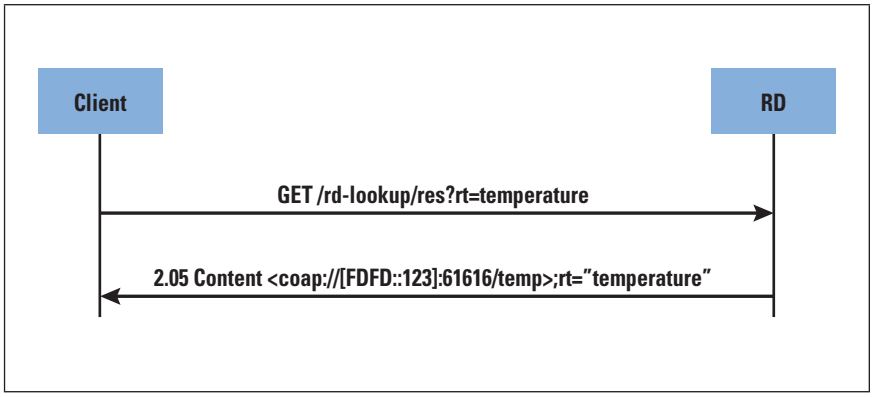
Figure 5: Registration of URIs to a Resource Directory (adapted from [6])



Resource Lookup (Discovery) by Client

The RD also supports a lookup interface for clients to make a discovery request on the RD database. The client may be located in the RD domain or may be outside of it. The client is aware of a given RD because it used the locating mechanism described previously, or it may have learned of the RD through other methods (for example, preprovisioning). Figure 6 shows a typical resource lookup request where a client is interested in finding all URIs related to "temperature." Specifically, the client will send a GET request to the RD Lookup interface indicating that it is interested in the resource type of temperature in the query string (that is, `?rt=temperature`). The RD will then respond with a message containing the list of URIs of all the devices that it has in its registration database that match this criterion[6]. The response message is formatted using the Link Format[7].

Figure 6: Resource Lookup (Discovery) Request Sent to a Resource Directory (adapted from [6])



Other types of lookup requests may also be sent. For example, the RD may be queried to find out all the URIs supported by a given IoT device. Or, the RD may be queried to find out the identities of all the IoT devices in a given domain. The great majority of lookup requests to the RD will be sent by other IoT devices, without any human in the loop, for automated command and control. However, resource lookup requests may also be sent occasionally by humans via a CoAP-enabled Web browser interface for management activities.

Resource Directory Security Characteristics

Both the RD registration and lookup interfaces are protected by multiple layers of security to ensure that only authorized parties can access the RD. Specifically, mutual authentication is first required between the RD and any device attempting to access it. This authentication is accomplished using either preshared encryption keys, raw public keys, or X.509 security certificates as the security credentials[9]. The appropriate credentials are used in the initial handshake of the DTLS session establishment to perform mutual authentication between the RD and the device or client accessing it. After the mutual authentication is completed, the cipher suite to be used for the DTLS session is negotiated. Then all subsequent messages exchanged between the RD and the device or client are securely encrypted via DTLS so that no unauthorized third party can decipher the communications[6].

In addition to the DTLS security, the RD will also perform a fine-grained access control of any device attempting to communicate with it. Access control will be performed separately on the RD registration and lookup interfaces. Access control may be performed at the domain, device, or resource level[6]. This control is especially important on the lookup interface for privacy and security reasons. For example, in a hospital setting many medical devices such as blood pressure monitoring devices may be registered to a RD, but only authorized medical staff should be able to discover a given device for privacy reasons. Or in a home setting, a visitor may be allowed to freely discover the television, but will be blocked from discovering the front door lock for security reasons.

Examples of Resource Directory Implementations

In parallel with the ongoing standardization efforts in the IETF for the RD protocol[6, 12], there are several open source and commercial instances of RDs that have successfully interoperated with various IoT devices. Some examples are briefly described in the following paragraphs.

The *Californium* open source software project is a popular CoAP framework for IoT deployments. It is written in the *Java* programming language and specifically includes support for back-end infrastructure as part of its project scope. As such, it has released software loads that implement RD functionality that can be run on general-purpose servers[13].

On the commercial front, ARM, the semiconductor and software company, has released several products for the IoT market. One of its products is a middleware offering called the "mbed Device Server." This middleware includes support of RD functionality. This middleware software can run on various server hardware platforms[14].

Another company that has done a lot of RD development work is Ericsson, the telecommunications equipment and service provider. Ericsson has done early prototyping and research[15] in the RD concept starting from the initial EU SENSEI project days[8]. The company has also participated in an open source software project for a cloud-based IoT gateway that includes RD functionality[16].

Alternative Approaches to Discovery

The RD is not the only approach to the discovery problem for IoT networks. There are other methods such as *Domain Name Service – Service Discovery* (DNS-SD), which allows lookup of a given service via DNS[17]. Another method is *Universal Plug and Play* (UPnP), which allows discovery of devices in home networks[18].

The key difference between these other discovery methods and the RD approach is that the RD is geared towards resource discovery in the context of a REST-based Web model, meaning discovery of URIs and related metadata. The other existing discovery approaches are mainly oriented to discovering IP addresses, ports, and related parameters. So they are complementary to the URI discovery methods but cannot replace them. The only other widely used URI discovery scheme is the Web crawler approach described previously, which has the shortcomings in IoT deployments as described in Table 1.

Conclusion

The existing REST-based Web architecture and protocols have been extremely successful and a driving force behind the explosive growth of the Internet during the last 20 years. Search engines like *Google* and *Bing*, which use Web crawlers to discover resources (that is, URIs) efficiently, constitute a key part of the success of the Web. However, the existing model of resource discovery is expected to undergo radical changes with the addition in the future of an increasing number of IoT devices acting as both mini Web servers and clients. The IETF is currently standardizing protocol support for the Resource Directory, which will be optimized for distributed IoT resource discovery.

It is expected that an increasing number of discovery requests in the future will be handled by RDs for scenarios involving IoT devices. In parallel, human users will continue to heavily use traditional Web search engines like Google. There is also expected to be some cross-usage because traditional Web browsers may start to support CoAP software modules (plug-ins) and hence allow human users to make direct queries to RDs. However, a limiting feature of this interaction will be the security and privacy requirements of IoT deployments. Specifically, many IoT resources such as personal health-monitoring devices will have sensitive information that is not meant for public distribution, and they may also be located in semi-closed networks. Strong security and privacy is supported by the current RD model, which requires strict mutual authentication, encryption, and access control for both registration and discovery of IoT resources.

References

[1] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter, "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986, January 2005.

[2] Roy Fielding and Julian Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," RFC 7230, June 2014.

[3] The Web Robots Pages: `http://www.robotstxt.org/`

[4] "What Is SEO, Search Engine Optimization?," `http://searchengineland.com/guide/what-is-seo`

[5] Dominique Barthel, Mischa Dohler, Thomas Watteyne, and Tim Winter, "Urban WSNs Routing Requirements in Low Power and Lossy Networks," RFC 5548, May 2009.

[6] Z. Shelby, et al., "CoRE Resource Directory," Internet Draft, work in progress, March 2016. `draft-ietf-core-resource-directory-07.txt`

[7] Zach Shelby, "Constrained RESTful Environments (CoRE) Link Format," RFC 6690, August 2012.

[8] S. Jokic, et al., "Evaluation of an XML Database Based Resource Directory Performance," `http://www.smartsantander.eu/downloads/Presentations/XML_RD_Telfor_2011_v1.0Srdjan.pdf`

[9] Zach Shelby, Carsten Bormann, and Klaus Hartke, "The Constrained Application Protocol (CoAP)," RFC 7252, June 2014.

[10] Eric Rescorla and Nagendra Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347, January 2012.

[11] Esko Dijk and Akbar Rahman, "Group Communication for the Constrained Application Protocol (CoAP)," RFC 7390, October 2014.

[12] A. Rahman, "Advanced Resource Directory Features," Internet Draft, work in progress, March 2016. `draft-rahman-core-advanced-rd-features-02.txt`

[13] Californioum (Cf) CoAP Framework: `http://www.eclipse.org/proposals/technology.californium/`

[14] ARM mbed Device Server: `https://www.mbed.com/en/development/cloud/mbed-device-server/`

[15] Ericsson Research Blog: Having a headache using legacy IoT devices?
https://www.ericsson.com/research-blog/internet-of-things/headache-using-legacy-iot-devices/

[16] Ericsson Research Blog: A Computational Engine for the Internet of Things.
https://www.ericsson.com/research-blog/internet-of-things/computational-engine-internet-things/

[17] Stuart Cheshire and Marc Krochmal, "DNS-Based Service Discovery," RFC 6763, February 2013.

[18] Wikipedia, "Universal Plug and Play":
https://en.wikipedia.org/wiki/Universal_Plug_and_Play

[19] David Lake, Ammar Rayes, and Monique Morrow, "The Internet of Things," *The Internet Protocol Journal,* Volume 15, No. 3, September 2012.

[20] William Stallings, "The Internet of Things: Network and Security Architecture," *The Internet Protocol Journal,* Volume 18, No. 4, December 2015.

AKBAR RAHMAN is a Principal Engineer at InterDigital Communications and is based in the company's office in Montreal, QC, Canada. He has been closely involved in IoT protocol development at IETF for several years. He has multiple IETF RFCs published in the areas of IoT and Internet architecture. He has a BASc degree from the University of Waterloo, Canada.
E-mail: **Akbar.Rahman@InterDigital.com**

CHONGGANG WANG is a Member of Technical Staff at InterDigital Communications and is based in the company's office in King of Prussia, PA, USA. He is Editor-in-Chief of the *IEEE IoT Journal,* and a Distinguished Lecturer for the IEEE Communications Society. He has a PhD from the Beijing University of Posts and Telecommunications.
E-mail: **Chonggang.Wang@InterDigital.com**

The IANA Transition

by Vint Cerf, Google

In this article I will explore the notable proposal sent in March 2016[0] by the *Internet Corporation for Assigned Names and Numbers* (ICANN) to the U.S. Department of Commerce, *National Telecommunication and Information Agency* (NTIA) to end the long-standing contractual relationship between ICANN and NTIA for the conduct of the *Internet Assigned Numbers Authority* functions ("IANA functions")[1, 2]. ICANN was formed in late 1998 in response to a White House "White Paper" issued by Ira Magaziner, then a senior advisor for policy to President Bill Clinton. ICANN would undertake to form a private sector entity to carry out the coordinated assignment of Internet domain names, Internet addresses, and the maintenance of parameter registries needed for the operation of the suite of protocols used in the Internet.

These functions had been managed by Jonathan Postel acting as the IANA at University of Southern California's Information Sciences Institute (and other earlier institutions where Postel had worked) under various government contracts. By 1996, the Internet was experiencing its so-called "dot boom" and the potential scale and liabilities of carrying out the IANA functions led to a serious effort to institutionalize the operation. For lack of space, I will leave out two years of community debate and fast-forward to the creation of ICANN to fulfill these functions. ICANN was conceived as a multi-stakeholder organization drawing on input from the private sector, civil society, governments of the world, and the technical community for the development of policy for the IANA functions and for the coordination of the multiple parties having a role in managing these unique identifiers and parameters.

In 1998, many organizations were involved in the evolution and operation of the Internet, its *Domain Name System* (DNS), Internet address allocation, and standards development. The *Internet Society*, founded in 1991, housed the standards-oriented *Internet Architecture Board* (IAB) and the *Internet Engineering Task Force* (IETF). There were then three *Regional Internet Registries* (RIRs)[3] for Internet address allocation—RIPE-NCC, APNIC, and ARIN—and two more to follow later (LACNIC and AFRINIC). There were nominally 13 DNS *Root Server* operators providing top-level domain name resolution. Verisign generated and distributed the official domain name root zone based on input from IANA and, under the terms of the NTIA/ICANN contract, authorization from NTIA. Many domain name registries and registrars were created to support DNS operation.

The original plan was for ICANN to operate under NTIA oversight for a few years and then operate as an independent organization. In fact, the contractual obligations extended from 1998 to the present.

In March 2014, however, NTIA proposed that this contractual relationship for the IANA functions should be ended and ICANN be allowed to perform the IANA functions independently. In March 2016, ICANN delivered to NTIA its consolidated proposal from all the constituent parties for the transition from the present contractual relationship to independent operation. The two-year effort leading to this comprehensive proposal was not without considerable debate among all the parties. Many ideas were surfaced, analyzed, argued over, adopted, adapted, or discarded, leading to a consolidated result. The Department of Commerce and the U.S. Congress will be evaluating the proposed new modus operandi in the weeks ahead.

Some fears have been voiced that the complex proposal poses risks that authoritarian governments within the ICANN *Governmental Advisory Committee* (GAC) or through some external means might wrest control of ICANN from its multi-stakeholder constituencies. While the proposal should be evaluated on all its merits, I am persuaded the terms and conditions of the proposed operating practices are well protected against such an outcome. A great many conditions must be satisfied before the more extraordinary powers of the *sole designator* can be exercised. The headquarters of ICANN will remain in the U.S. The many entities that cooperate with ICANN to manage core Internet identifier administration have expressed full support for the proposal.

If I have any trepidation about the proposal, it is associated with its general complexity. As the former chairman of ICANN, I am no stranger to the evolution of ICANN's structure and processes and their relative intricacy. The new proposal adds its own unique aspects to this tendency, and it remains to be seen how well the system will work. However, ICANN has shown a remarkable ability to reform and adapt when necessary, and I believe that capacity is preserved under the new proposal. There is still a good deal of work ahead to actually implement what is ultimately approved, but I am confident this community is capable of achieving a successful outcome.

[Ed.: An earlier version of this article appeared in *Communications of the ACM,* Volume 59, No. 5, May 2016.]

References

[0] "Plan to Transition Stewardship of Key Internet Functions Sent to the U.S. Government,"
https://www.icann.org/news/announcement-2016-03-10-en

[1] "NTIA Announces Intent to Transition Key Internet Domain Name Functions,"
https://www.ntia.doc.gov/press-release/2014/ntia-announces-intent-transition-key-internet-domain-name-functions

[2] "NTIA Finds IANA Stewardship Transition Proposal Meets Criteria to Complete Privatization," **https://www.ntia.doc.gov/press-release/2016/iana-stewardship-transition-proposal-meets-criteria-complete-privatization**

[3] Daniel Karrenberg, Gerard Ross, Paul Wilson, and Leslie Nobile, "Development of the Regional Internet Registry System," *The Internet Protocol Journal,* Volume 4, No. 4, December 2001.

[4] IANA Stewardship Transition Coordination Group: **http://www.ianacg.org/**

[5] NTIA IANA Functions' Stewardship Transition: **https://www.icann.org/stewardship**

VINTON G. CERF is Vice President and Chief Internet Evangelist for Google. He contributes to global policy development and continued spread of the Internet. Widely known as one of the "Fathers of the Internet," Cerf is the co-designer of the TCP/IP protocols and the architecture of the Internet. He has served in executive positions at MCI, the Corporation for National Research Initiatives and the Defense Advanced Research Projects Agency (DARPA), and on the faculty of Stanford University.

Cerf served as Chairman of the Board of the Internet Corporation for Assigned Names and Numbers (ICANN) from 2000 to 2007 and has been a Visiting Scientist at the Jet Propulsion Laboratory since 1998. He served as founding President of the Internet Society (ISOC) from 1992 to 1995. Cerf is a Fellow of the IEEE, ACM, and American Association for the Advancement of Science, the American Academy of Arts and Sciences, the International Engineering Consortium, the Computer History Museum, the British Computer Society, the Worshipful Company of Information Technologists, and the Worshipful Company of Stationers, and he is a member of the National Academy of Engineering. He currently serves as Past President of the Association for Computing Machinery and Chairman of the American Registry for Internet Numbers (ARIN), and he has completed a term as Chairman of the Visiting Committee on Advanced Technology for the U.S. National Institute of Standards and Technology. President Obama appointed him to the National Science Board in 2012.

Cerf is a recipient of numerous awards and commendations in connection with his work on the Internet, including the U.S. Presidential Medal of Freedom, U.S. National Medal of Technology, the Queen Elizabeth Prize for Engineering, the Prince of Asturias Award, the Tunisian National Medal of Science, the Japan Prize, the Charles Stark Draper Award, the ACM Turing Award, Officer of the Legion d'Honneur, and 25 honorary degrees. In December 1994, *People* magazine identified Cerf as one of that year's "25 Most Intriguing People." His personal interests include fine wine, gourmet cooking, and science fiction. Cerf and his wife, Sigrid, were married in 1966 and have two sons, David and Bennett. **vint@google.com**

RACI

The *RIPE Academic Cooperation Initiative* (RACI) connects members of the academic community with the RIPE community by inviting students and researchers to present at meetings organized by the RIPE NCC. Successful applicants receive complimentary tickets, travel and accommodation to meetings and the opportunity to present their work to some of the leading technical figures in the Internet world. Examples of relevant topics include:

- Network Measurement and Analyses
- IPv6 Deployment
- BGP Routing
- Network Security
- Internet Governance
- Peering and Interconnectivity
- The Internet of Things

For more information about RACI, including the application process and deadlines, visit: **http://ripe.net/raci**

NTIA Issues IANA Transition Proposal Report

On 9 June 2016, *The National Telecommunications and Information Administration* (NTIA) issued its assessment report on the *IANA Stewardship Transition Proposal*. (Ed.: See article on page 26). In order to be accepted, the proposal needed to be shown to have broad community support and address the following four principles:

- Support and enhance the multistakeholder model
- Maintain the security, stability, and resiliency of the Internet DNS
- Meet the needs and expectations of the global customers and partners of the IANA services
- Maintain the openness of the Internet

The NTIA further stipulated that "it would not accept a proposal that replaces its role with a government-led or intergovernmental organization solution." After thorough review the NTIA reports that it finds that "the IANA Stewardship Transition Proposal meets the criteria necessary to complete the long-promised privatization of the IANA functions."

The full report is available at:

https://www.ntia.doc.gov/report/2016/iana-stewardship-transition-proposal-assessment-report

Call for Papers

The *Internet Protocol Journal* (IPJ) is a quarterly technical publication containing tutorial articles ("What is...?") as well as implementation/operation articles ("How to..."). The journal provides articles about all aspects of Internet technology. IPJ is not intended to promote any specific products or services, but rather is intended to serve as an informational and educational resource for engineering professionals involved in the design, development, and operation of public and private internets and intranets. In addition to feature-length articles, IPJ contains technical updates, book reviews, announcements, opinion columns, and letters to the Editor. Topics include but are not limited to:

- Access and infrastructure technologies such as: Wi-Fi, Gigabit Ethernet, SONET, xDSL, cable, fiber optics, satellite, and mobile wireless.
- Transport and interconnection functions such as: switching, routing, tunneling, protocol transition, multicast, and performance.
- Network management, administration, and security issues, including: authentication, privacy, encryption, monitoring, firewalls, troubleshooting, and mapping.
- Value-added systems and services such as: Virtual Private Networks, resource location, caching, client/server systems, distributed systems, cloud computing, and quality of service.
- Application and end-user issues such as: E-mail, Web authoring, server technologies and systems, electronic commerce, and application management.
- Legal, policy, regulatory and governance topics such as: copyright, content control, content liability, settlement charges, resource allocation, and trademark disputes in the context of internetworking.

IPJ will pay a stipend of US$1000 for published, feature-length articles. For further information regarding article submissions, please contact Ole J. Jacobsen, Editor and Publisher. Ole can be reached at **ole@protocoljournal.org** or **olejacobsen@me.com**

Supporters and Sponsors

Publication of this journal is made possible by:

Supporters

Diamond Sponsors

Ruby Sponsor

Sapphire Sponsors

Emerald Sponsors

Corporate Subscriptions

Individual Sponsors

Lyman Chapin, Steve Corbató, Dave Crocker, Jay Etchings, Martin Hannigan, Hagen Hultzsch, Dennis Jennings, Jim Johnston, Merike Kaeo, Bobby Krupczak, Richard Lamb, Tracy LaQuey Parker, Bill Manning, Andrea Montefusco, Tariq Mustafa, Mike O'Connor, Tim Pozar, George Sadowsky, Scott Seifel, Helge Skrivervik, Rob Thomas, Tom Vest, Rick Wesson.

For more information about sponsorship, please contact `sponsor@protocoljournal.org`

The Internet Protocol Journal
NMS
535 Brennan Street
San Jose, CA 95131

ADDRESS SERVICE REQUESTED

The Internet Protocol Journal is published quarterly and supported by the Internet Society and other organizations and individuals around the world dedicated to the design, growth, evolution, and operation of the global Internet and private networks built on the Internet Protocol.

Email: ipj@protocoljournal.org
Web: www.protocoljournal.org

Printed in the USA on recycled paper.