

The Evolution of HTTP

The Consolidation of Two Simple Protocols for Widespread Web Automation

Scenera Research LLC

Cary, North Carolina

www.sceneraresearch.com

Thanks to its simplicity and reliability, HTTP has grown to be the standard communications protocol for the Web. It allows users without great knowledge of network protocols to send and receive text-based requests and responses. Because HTTP is so well-established worldwide, we found that one of the best ways to automate Web browsing on such a large scale would be to extend HTTP to allow asynchronous, loosely-coupled operation. This extension would allow additional new Web capabilities to be consolidated into HTTP. Furthermore, rather than adding new commands to HTTP that could cause migration issues, we have the extended HTTP as far as allowed by the HTTP 1.1 specifications so that they remain fully HTTP 1.1 compliant.

The HTTP extensions we propose integrate an asynchronous communications model into HTTP with a principal-based publish/subscribe model, which provides loosely-coupled communications with additional benefits. Publish/subscribe protocols currently coexist with HTTP as separately defined protocols and are primarily used to provide presence services for a variety of messaging applications. Extending HTTP to support asynchronous messaging that includes publish/subscribe message flows in a manner compliant with current standards will permit a relatively smooth transition from the current Web to a more active, automated Web.

Introduction

The Internet handles trillions of pieces of data of varying sizes and types every day, 24 hours a day, seven days a week, and requires rigorous rules to keep everything moving smoothly and reliably. This follows a set of rules specified as protocols that are analogous to rules that regulate vehicles on roads and highways, such as traffic signs, signals, road markings, and laws. Like cars traveling at rush hour, at times the movement of data on the Internet can be so busy and crowded that it slows down and becomes erratic, so all it takes is for one person to break the rules to cause a serious crash.

The rules and requirements that oversee the operation of automobiles are created, put to law, and enforced by experts assigned by governing bodies, and drivers learn those rules through education and practice. The establishment of rules for operating the Internet, including the Web, are also created and standardized by experts working together through official groups, and network programmers learn them through education and practice. The various organizations that regulate the global Web have been

successful in their missions, as we see every time we send or read an e-mail, download a file, make an online purchase or sale, or talk on a Voice-Over-Internet (VOIP) telephone.

Protocols

The most important regulations that manage the operations that keep the Web running smoothly are contained in protocol specifications and the hardware and software that adhere to them. Protocols define the software and hardware conventions that regulate how data moves through the Web. They are created and maintained by networking and standardization experts and are defined by standards organizations, such as the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C). Requests for Comments (RFCs) are IETF documents that have been the backbone of Internet standards for forty years and are available for global public viewing on the Web (1).

The Internet has levels of protocols, known as “layers.” An application layer operates at the top layer of the Internet, and it primarily includes the client-server protocols used by Web browsers and Web servers, as well as other familiar Internet applications. They are the closest to users and include the most widely known protocols, such as Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Simple Mail Transfer Protocol (SMTP). The lowest protocol layers operate closer to the physical media, both wired and wireless, and they are hidden from applications and users by the application layer protocols.

Two of the principle goals of arranging protocols in layers are to identify and distinguish their operational responsibilities and to define how the protocols within and between the layers interact with one another. This layered approach makes it much easier for people without networking expertise to program and use the Web at the uppermost application layer while leaving access to the Internet’s lower software and hardware infrastructure and operational layers to those who have the necessary training and expertise.

The layered protocol design is called the TCP/IP Model and was named for the collection of protocols that enable today’s Internet, termed the TCP/IP Suite. This model was developed in the 1970’s by the U.S. Defense Advanced Research Projects Agency (DARPA) for the development of ARPANET, which was the precursor to the system that evolved into the Internet. The protocols at each layer of this model embody the rules needed to manage communication between network nodes. The exact definition of each protocol can be found in its corresponding RFC documents.

The protocol layers above the physical layer are listed and described in Table 1. While there is considerable variation in their functionality, all implementations must meet the basic requirements of protocol specifications.

Protocol Layering: The TCP/IP Model			
Layer		General Function	Example Protocols
4	Application	Application specific. Examples include Web browsing, e-mail, file transfer, etc.	HTTP, FTP, SMTP, DNS, SIP, Telnet, SOAP, XMPP
3	Transport	Application-to-Application datagrams and connections.	TCP, UDP, RSVP
2	Internet	Node-to-Node communication across network boundaries.	IP, ICMP, IGMP, IPSec
1	Link	Node-to-Node communication across a single physical media link.	ARP, NDP, OSPF, Ethernet

Table 1. The TCP/IP Model over a physical layer, which is designed to allow computers to communicate across networks. Its name originates from two of its most important protocols: Transmission Control Protocol and Internet Protocol.

A simple example of a protocol that most of us come in contact with is the one that dictates that when an e-mail is sent, there must be a response indicating whether or not the message is received intact. If no response is received, the protocol assumes that something in the message or its transmission failed and it tells the server to send it again. After a specified number of failed attempts, the protocol tells the server to stop trying to send the e-mail and to send a failed transmission response to the sender instead. Finally, the sender receives an e-mail from the server saying that the e-mail could not be sent.

Communication Protocols

Communications protocols determine how data is formatted and transmitted across networks. All types of communication devices use protocols for the successful relay of information, such as radios, telephones, and devices connected to the Web.

Important attributes that network protocols define are whether or not transmissions are synchronous or asynchronous and the type of coupling that the data senders and data receivers have.

Hypertext Transfer Protocol

Today's most common Web browsers use the Hypertext Transport Protocol (HTTP) to exchange messages over the Internet with Web servers. HTTP is a request-response protocol that defines how messages are formatted and transmitted and what actions Web servers and Web browsers should take in response to various commands included in received messages.

The success of the Web is largely a result of the simplicity and flexibility of HTTP, which has allowed the Web to expand, adapt, and evolve to the current Web system since the early 1990's. HTTP has a small set of commands, and only two are required for the request-response communication it needs to run nearly everything we see on the Web: GET and POST.

HTTP's basic function is to send a request from one network entity (a Web browser) to another network entity (a Web server), and then wait for a response. For instance, to open a Web page a user enters or

selects a Web address (URL) in a Web browser and an HTTP command is sent to the Web server telling it to fetch and reply with the page so it can be displayed in the browser.

In HTTP, a reply can only be sent in response to a request; if no request is made, no reply is sent. This is termed “synchronous communication.” Although HTTP is a very efficient protocol, its synchronous design is just one model of communication that requires a requester to know the location of desired data and to ask for it. Because of this, finding information can require extensive use of search engines and success can depend on one’s search skills. Rather than requiring users to search for, locate, then request information, the Web of the future will provide and send information as it is created in real time.

Publish/Subscribe Protocol

Another protocol model that is prevalent on the Web is publish/subscribe. Publish/subscribe allows one network entity, called a subscriber, to subscribe to information provided by another network entity, called a publisher. The publisher posts, or publishes, information to a publish/subscribe service, which then transmits the posted information to its subscribers so that the information can be viewed simultaneously by any number of them in real time. This is “asynchronous communication” and it allows the Web to provide real-time data. Current protocols that support presence services (described below), such as Session Initiation Protocol for Instant Messaging and Presence Leveraging (SIP-SIMPLE) and Extensible Messaging and Presence Protocol (XMPP-IM), support a publish/subscribe protocol model.

Like HTTP, publish/subscribe protocols can be simple and flexible, and the commands of HTTP and publish/subscribe are very similar. Publish/subscribe protocols typically have a small set of commands: PUBLISH, SUBSCRIBE, and NOTIFY. The SUBSCRIBE command can be viewed as a type of GET command, and PUBLISH and NOTIFY both can be viewed as types of POST or PUT commands. Seen this way, a publish/subscribe protocol can be simple and adaptable, like HTTP. And because publish/subscribe and HTTP can be viewed as variants of one another, it is plausible to consider supporting both of them in one protocol so they work together to provide synchronous and asynchronous communication on the Web. Consolidating them in HTTP would be a most efficient way to support them together.

Asynchronous, Loosely-Coupled Web Communication

As mentioned earlier, HTTP is a synchronous protocol because it requires that a reply be sent in response to a request. Requests and responses generally have a one-to-one correspondence, and requesters must know the identity of the responder in order to send a request, which is a tightly-coupled relationship.

Asynchronous protocols are designed so there is no need for a one-to-one correspondence between the requests and responses exchanged between network entities that communicate with each other. In fact, a message can be sent for which no response is required. In the publish/subscribe model a publisher (sender) of information does not need to know the identity of a receiver nor does the publisher require a response in most cases. Even so, a request-response protocol can be defined to operate over a publish/subscribe model. Such a request-response protocol can be “loosely-coupled,” meaning that a requester/publisher does not need to know which, if any, receiver of the published information will

respond. The asynchronous nature of the publish/subscribe model will facilitate Web automation, and a loosely-coupled request-reply model should make for more reliable and robust network applications.

Because publish/subscribe protocols are asynchronous and allow receivers and responders to be loosely-coupled, when their function is added to HTTP the resultant protocol would be uniquely qualified to support a more proactive and responsive Web.

Presence

Presence is a type of service that is supported by a principal-based publish/subscribe protocol model. Its services are the building blocks of an automated network that convey the capability, availability, and willingness of a user to communicate with others. Basically, a presence service provides metadata that is comparable to a person saying, “I’m here” or “I’m available.” While that may not seem impressive, today it is widely used to establish communication sessions, and much of its potential remains unrealized.

Presence is a relatively nascent service that is currently expanding in many directions and will continue to expand with future development. The most well-known and widely-used technology used with presence today is instant messaging, such as AOL Instant Messenger, MS Office Communicator, and Yahoo! Messenger. In these applications, presence services allow users to know whether or not other users, known as buddies, contacts, or friends, are connected to a user’s instant messenger. Presence services also convey the statuses of users, such as “available,” “busy,” “on the phone,” “out to lunch,” etc., and often provide information on how their users can be contacted, such as a telephone number or an e-mail address.

Presence services gather and deliver presence information. Presence information indicates a user’s status or availability, typically for communication, and can also provide contact information that indicates where or how to contact the user. Users are referred to as principals, and published information is always published for an identified principal. Hence, publish/subscribe protocols that support presence services are called principal-based publish/subscribe protocols.

Presence services and principal-based publish/subscribe services support two types of clients, “presentities” and “watchers,” that represent principals. Presentities publish information that identifies with a publishing user or principal. Presence service presentities publish presence information to a presence service for distribution. For example, the user of an instant messenger is a principal represented by a presentity, and that user’s status in the service is the presence information. Presence services distribute published information to watchers and receive presence information from presence services on behalf of watching users.

Publish/subscribe information, such as presence information, is stored as tuples. Tuples are simply structured data, and the word “tuple” is derived from the English terms for multiples, such as double, triple, quadruple, and quintuple. A presence tuple is an “n-tuple,” or infinite multiple, and its name conveys its key feature: a tuple can store data objects of any size and any type. Each tuple represents an identifiable principal such as a human user or a device. In short, principal-based publish/subscribe

services can support tuples that hold information of any size and any type. This feature gives publish/subscribe services tremendous flexibility.

Publish/Subscribe Protocols

As the popularity of presence services grows, the number of presence applications using different protocols also increases. Many presence applications use proprietary architectures and protocols to implement presence service components, while others use open, non-proprietary architectures and protocols that are based on standards.

The protocols most often used with presence are those that meet the requirements and architecture described in Request for Comments documents RFC 2778 and RFC2779 (2, 3). Those RFCs describe a principal-base publish/subscribe protocol model for providing presence services. Current protocols compliant with those RFCs have varying levels of interoperability. For instance, one of the most open and flexible presence protocols is XMPP-IM. However, although it is flexible, it is relatively complex when compared to HTTP and it is incompatible with many firewalls, which are ubiquitous on the Web. The protocol SIP-SIMPLE is like HTTP, but it adds commands that are not supported by the HTTP 1.1 specification. Thus, current browsers and Web servers do not recognize SIP-SIMPLE commands.

Because so many presence protocols are proprietary and many of the non-proprietary ones have limited interoperability or Web compatibility, users cannot always use generic clients, such as Web browsers, to take advantage of the presence services or other asynchronous services. Instead, users must install application-specific clients, as is the common case with instant messaging.

An asynchronous communications protocol that is interoperable with networked applications and compatible with Web browsers is needed to automate the Web. A principal-based publish/subscribe protocol can meet those requirements.

Publish/Subscribe Potential

Many instant messengers that currently use presence services to determine a user's status and address use principal-based publish/subscribe protocols to provide their services. However, the underlying publish/subscribe protocols are not being used to support services other than presence. For example, publish/subscribe commands can deliver instant messages, support a loosely-coupled remote procedure call, and also support auctions, workflow systems, and many other applications. Services and applications built on publish/subscribe protocols and services are designed around the data required by the service or application. This data is structured in a tuple, and the flexibility of tuples would permit more diverse types of information than they have yet been asked to handle by presence services (4).

In short, both presence services and principal-based publish/subscribe services have unrealized potential that would benefit the development of an automated Web.

HTTP-Publish/Subscribe-Presence Consolidation

As mentioned earlier, HTTP became the backbone of the Web due to its simplicity, interoperability, and flexibility, making it relatively easy to consolidate new Web operations as needed to expand the Web., As exemplified in presence services, principal-based publish/subscribe protocols are also simple and

flexible, and we believe that consolidating a publish/subscribe protocol into HTTP will provide asynchronous messaging and loosely-couple communication to the Web, enabling a next phase in its evolution toward becoming more proactive, automated, and data-centered.

There are many ways to format the communication between subscribers and publishers with the consolidated HTTP-publish/subscribe protocol. An example scenario includes these actions (5):

- First, an HTTP request message is sent that allows the user to request a subscription to a tuple identified by a uniform resource identifier (URI) in the request. The HTTP message includes subscription data in an HTTP content header identifying the message as a subscription request.
- Next, an HTTP request is sent to allow an identified principal to publish information to update a tuple identified by a URI. The tuple represents the Web resource or publishing principal that is subscribed to by sender of the first HTTP request message, as described above.
- Last, in response to the published information, an HTTP message, such as response to the subscription request, is sent notifying the subscriber of the newly published information. Notifications are sent to the subscriber whenever the tuple representing the subscribed to resource is updated.

Requests for subscriptions and tuple updates typically are processed by a publish/subscribe service operating on a server. Once the service receives and processes subscription requests and receives published information, it notifies appropriate subscribers. A subscribing principal can operate in or be represented by any type of network device, including PCs, mobile phones, PDAs, digital cameras, and sensors. The general flow of information is illustrated in Figure 1.

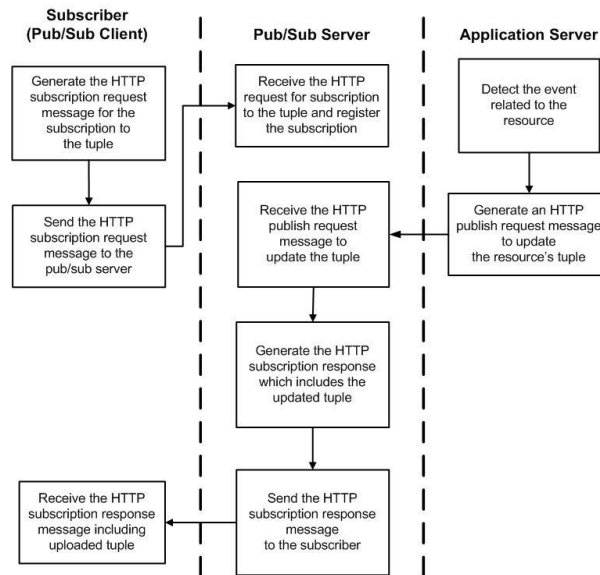


Figure 1. The general flow of information through an application server using the consolidated HTTP-Publish/Subscribe Protocol. It includes tuple updating and subscription processing.

Web Applications

HTTP is in the application layer of the TCP/IP layering model, which publish/subscribe currently can run on either as an application protocol independent of HTTP or as a layer that operates on top of the HTTP layer. Once it is consolidated with HTTP, publish/subscribe will be folded into the application layer with HTTP. The publisher and subscriber that run on the application layer then can be loosely coupled, allowing them to communicate freely and as needed, and without the burden of mandatory responses. When needed, a direct notification connection can be established between the publisher and subscriber, thus providing direct, one-to-one, real-time communication.

Bringing publish/subscribe into the application layer with HTTP can help simplify the Web by minimizing, or even eliminating, the need for other asynchronous communication protocols. Owing to its flexibility and interoperability with Web protocols and resources, an HTTP-publish/subscribe protocol would run reliably on the Web we know today and the Web that evolves into the automated future.


Web Browsers

The HTTP-publish/subscribe protocol can help make browsing safer and extend the capabilities of existing protocols through new types of tuples and new message flow patterns. The graphical user interface (GUI) of a Web browser is configured to receive the ID of a tuple associated with a Web page or other network resource and a protocol agent that is coupled to the GUI. The protocol is configured to use the ID to request a subscription to the tuple and is configured to receive information related to the network resource and its link. A communications protocol stack coupled to the protocol agent is configured to request the subscription to the tuple and to receive real-time data using the asynchronous publish/subscribe protocol.

Figure 2 gives an example of an auction GUI in the Web browser of a PC, cell phone, or PDA. It shows the real-time data for an item that is up for auction, including the current price, the current number of bids, and remaining auction time. It also allows the user to place bids and buy and sell items on the auction site in real time. This real-time data can be received without the use of scripts as is currently required.

Take Your Chances Auctions

[MyHome](#) [Buy](#) [Sell](#) [Find](#) [Help](#)



Current bid:	\$3,475.00
Number of bids:	57
Time left:	1 hour 13 minutes
Seller ID:	Junkman
My highest bid:	\$3,200.00
My bid count:	7

Bid ...

Starting Bid:	\$3,000.00
Item Location:	Peoria, IL
Shipping:	US Only

Description: This is a natural 7.15 carat red ruby ring, 15x3.5x5.5mm, in an 18K yellow gold setting with a weight of 7.25 grams. The ring size is 7.5 and it can be adjusted as needed.

Payment: All major credit cards are accepted through PayPal unless a special payment arrangement made. Auction winners must pay for their items within seven days of the end of the auction.

Shipping: This ring is shipped in the United States only, and charges for shipping are calculated at the time the auction ends. All items will be shipped, insured, by USPS in one business day. Also look at our other auctions because we are happy to combine shipping.

Figure 2. An example of a GUI that shows a user the real-time information from an item in an auction that she is subscribed to and has bid on.

Synchronous Operations

Not all communications need to be handled by the publish/subscribe portion of the protocol or the publish/subscribe server. Because the consolidated protocol includes HTTP and operates in the same environment as other Web-based protocols, those protocols can be used the same way they are today.

One of the greatest advantages of consolidating publish/subscribe with HTTP in a manner that is compliant with the existing HTTP 1.1 standards is that when the combined protocol is used to communicate with a client that does not support the publish/subscribe content headers, the communication is processed as a traditional HTTP 1.1 message. Thus, the extensions are backwards-compatible and will not break existing browsers and Web servers. This characteristic will allow for a smooth migration to the extended HTTP.

Web Evolution

With instant messaging, telepresence, and real-time social networking developing all around us, the Web is becoming more and more automated and real-time every day. There is still considerable work to be done and it is clearly a focus of the future.

As the Web continues to evolve, more additions and expansions will be made to the protocols used to manage it, which will also add to the complexity of the network. Simple protocols have been essential to the success of the Web since its inception; however, in the direction of automation, we've already arrived at a place where not all protocols operate with all Web browsers, inhibiting normal growth. If we

promote a gradual evolution through the modification and consolidation of existing, successful protocols and by minimizing the creation of unneeded new protocols, then the components that make up the Web will be more likely to be interoperable and the network should be more reliable.

Consolidating publish/subscribe into HTTP is one such step in the right direction because it merges two simple, flexible, compatible, well-established Web protocol models into one. Stable steps like this are essential to the continued steady progress and evolution of the World Wide Web.

Acknowledgement

Scenera Research LLC thanks Julie Tomlinson for her research, writing, and editing.

References

1. The IETF Request for Comments Web site: <http://www.ietf.org/rfc.html>.
2. M. Day, J. Rosenberg, and H. Sugano. 2000. A model for presence and instant messaging. Internet Engineering Task Force, RFC 2778; <http://www.ietf.org/rfc/rfc2778.txt>.
3. M. Day, S. Aggarwal, G. Mohr, and J. Vincent. 2000. Instant messaging/presence protocol requirements. Internet Engineering Task Force, RFC 2779; <http://www.ietf.org/rfc/rfc2779.txt>.
4. M. Perry, C. Delporte, F. Demi, A. Ghosh, and M. Luong. 2001. MQSeries Publish/Subscribe Applications. IBM Corporation, Hampshire, UK; <http://www.redbooks.ibm.com/redbooks/pdfs/sg246282.pdf>. 244pp.
5. R.P. Morris. 2007. HTTP publish/subscribe communication protocol. U.S. Patent Application Publication No. 20070192325. 19pp.

Contact

Contact Scenera Research at info@sceneraresearch.com.