The Middleware Muddle

July 2001
Geoff Huston

This occasional column is an individual soapbox on views of various aspects of the Internet. The views stated here are intended to be mildly provocative, and, if backed to the wall, the author will rapidly disclaim any responsibility for them whatsoever!

It is not often that an entire class of technology can generate an emotive response. But, somehow, middleware has managed to excite many strong reactions. For some *Internet Service Providers* (ISPs), middleware (in the form of *Web caches*) is not only useful, it's critical to the success of their enterprise. For many corporate networks, middleware (in the form of *firewalls*) is the critical component of their network security measures. For such networks, middleware is an integral part of the network. Other networks use middleware, in the form of *Network Address Translators* (NATs), as a means of stretching a limited number of Internet public addresses to provide connectivity services to a much larger local network. For others, middleware is seen as something akin to network heresy. For them, not only does middleware often break the basic semantics of the Internet Protocol, it is also in direct contravention to the end-to-end architecture of the Internet. Middleware, they claim, breaks the operation of entire classes of useful applications, and this makes the Internet a poorer network as a result.

Emotions have run high in the middleware debate, and middleware has been portrayed as being everything from absolutely essential to the operation of the Internet as we know it, to being immoral and deceptive. Strong stuff indeed from an engineering community, even one as traditionally opinionated as Internet engineers.

So what is middleware all about and why the fuss?

It may be helpful to start with a definition of middleware. One definition of middleware is that of anything in the network that functions at a level in a network reference model above that of end-to-end transport (TCP/IP), and below that of the application environment (the *Application Programming Interface* [API]) [1]. Of course, this definition encompasses a very broad class of services that covers everything from *Authentication, Authorization, and Accounting* (AAA) servers and *Domain Name System* (DNS) servers through to various forms of information discovery services and resource management.

Another possible definition of middleware adopts the perspective of the integrity of the end-to-end model of Internet architecture [2]. From this perspective, middleware is a class of network devices that do something other than forward or discard an IP packet onward along the next hop to the destination address of the packet - in other words, anything other than a packet-switching element that sits in the transmission path of the packet.

With such an end-to-end definition of middleware, these middleware units may intercept the packet and alter the header or payload of the packet, redirect the packet to be delivered to somewhere other than its intended destination, or process the packet as if it were addressed to the middleware device itself. From this perspective, AAA, the DNS, and related services from our first definition are simply applications that traverse the network.

There's nothing like confusion over definitions to fuel a debate, and this area is no exception. However, a debate over definitions is too often a dry one. So, in the interest of adding a little more incendiary material to the topic, let's simply use this second definition of middleware to look further at the issues.

Why would a network go to all this bother to trap and process certain packets? Surely it's easier and cheaper to simply forward the packet onward to its intended destination? The answer can be "yes" or "no", depending on how you feel about the role of middleware in TCP/IP.

An Example: Cache Middleware

Let's look at this in a bit more detail, using a specific flavour of middleware to illustrate the middleware dilemma. A common form of middleware is the *Transparent Web Cache*. Such a Web cache is constructed using two parts, an *interceptor* and a *cache system* . The interceptor is placed into the network, either as a software module added to a router or as a device, which is spliced into a point-to-point link. The interceptor takes all incoming TCP traffic addressed to port 80 (a *Hypertext Transfer Protocol* (HTTP) session) and redirects it across to the cache system. All other traffic is treated normally. The cache system accepts all such redirected packets as if they were directly addressed to the cache itself. It responds to the HTTP requestor as if it were the actual intended destination, using a source address that matches the destination address of the original request, assuming the identity of then actual intended content server. If the requested Web object is located in the local cache, it will deliver the object to the requestor immediately. If it is not in the cache, it will set up its own session with the original destination, send it the original request, and feed the response back to the requestor, while also keeping a copy for itself in its cache.

Caching of content works well in the Web world simply because so much Web traffic today is movement of the same Web page to different recipients. It is commonly reported that up to one half of all Web traffic in the Internet is a duplicate transmission of content. If an ISP locally caches all Web content as it is delivered, and checks the cache before passing through a content request, then the ISP's upstream Web traffic volume may be halved. Even a moderately good cache will be able to service about one quarter of the Web content from the cache. That amount of local caching can be translated into a significant cost saving for the ISP.

The cached Web content is traffic that is not purchased as transit traffic from an upstream ISP, representing a potential saving on the cost of upstream transit services. This saving, in turn, can allow the ISP to operate at a lower price point in the retail market. The cache is also located closer to the ISP's customers, and with appropriate tuning, the cache can also deliver cached content to the customer at a consistently much faster rate than a request to the original content server. For very popular Web sites the originating server may be operating more slowly under extreme load, while the local cache continues to operate at a more consistent service level. The combination of the potential for improved performance and lower overall cost is certainly one that looks enticing: the result is the same set of Web transactions delivered to customers, but cheaper and faster.

End-to-End Issues with Cache Middleware

But not everything is perfect in this transparent caching world. What if the Web server used a security model that served content only to certain requestors, and the identity of the requestor was based on their IP address? This is not a very good security model, admittedly, but it's simple, and because of its simplicity this practice enjoys very common usage. With the introduction of a transparent cache, the Web client sees something quite strange. The Web client can ping the Web server, the client can communicate with any other port on the server, and if the client were to query the status of the server, the Web server would be seen to be functioning quite normally. But, mysteriously, the client cannot retrieve any Web content from the server, and the server does not see any such request from the client. The middleware cache is sitting inside a network somewhere on the path between the client and the service, but it may well be the case that neither the end client or the end server are aware of the deployment of the middleware unit. It is not surprising that this is a remarkably challenging operational problem for either the client or the server to correctly diagnose.

A similar case is where a Web server wishes to deliver different content to different requestors, based on some inference gained from the source IP address of the requestor, or the time of day, or some other variable derived from the circumstances of the request. A transparent cache will not detect such variations in the response of the server and will instead deliver the same version of the cached content to all clients whose requests pass through the transparent cache. Variations of this situation of perceived abnormal service behaviour abound, all clustered around the same concept that it is unwise in such an environment for a server to assume that it is always communicating with the end client. Indeed the situation is common enough that the Web application has explicit provision for instructing cache servers about whether the content can be cached and replayed in response to similar subsequent requests.

More subtle vulnerabilities also are present in such a middleware environment. A client can confidently assert that packets are being sent to a server, and the server appears to be responding, but the data appears to have been corrupted. Has the server been compromised? It may look like this is the case, but when middleware is around, looks can be deceiving. If the integrity of the cache is compromised, and different pages are substituted in the cache, then to the clients of the cache it appears that the integrity of original server has been compromised. The twist with transparent cache middleware is that the clients of the cache may be unaware that the cache exists, let alone that their requests are being redirected to the cache server. Any abnormalities in the responses they receive are naturally attributed to problems with the security of the server and the integrity of the associated service.

The common theme of these issues is that there are sets of inconsistent assumptions at play here. On the one hand, the assumption of an end-to-end architecture leads an application designer to assume that an IP session opened with a remote peer will indeed be with that remote peer, and not with some intercepting network-level proxy agent attempting to mimic the behaviour of that remote peer. On the other hand, is the assumption that transactions adhere to a consistent and predictable protocol, and transactions may be intercepted and manipulated by middleware as long as the resultant interaction behaves according to the defined protocol.

Middleware Architecture

Are transparent caches good or bad? Is the entire concept of middleware good or bad?

There is no doubt that middleware can be very useful. Cache systems can create improved service quality and reduced cost. NATs can reduce the demand for public IP address space. Firewalls can be effective as security policy agents. Middleware can perform load balancing across multiple service points for a particular class of applications, such as a Web server farm. Middleware can dynamically adjust the Internal Protocol parameters of a TCP session to adapt to particular types of networks, or various forms of network service policies. Middleware can provide services within the network that relieve the end user of a set of tasks and responsibilities, and middleware can improve some aspects of the service quality. Middleware can make an Internet service faster, cheaper, more flexible, and more secure, although probably not all at the same time. But middleware comes at a steep long-term price.

The advantage of the Internet lies in its unique approach to network architecture. In a telephone network, the end device, a telephone handset, is a rather basic device consisting of a pair of transducers and a tone generator. All the functionality of the telephone service is embedded within the network itself.

The architecture of the Internet is the complete opposite. The network consists of a collection of packet switches with basic functionality. The service is embedded within the protocol stack and the set of applications that are resident on the connected device. Within this architecture, adding new services to the network is as simple as distributing new applications among those end systems that want to use the application. The network makes no assumptions about the services it supports, and network services can be added, refined, and removed without requiring any change to the network itself. This results in a cheap, flexible, and basic network, and it passes the entire responsibility for service control to the network users. The real strength of the Internet lies in its architectural simplicity and lack of complex interdependencies within the network.

Middleware cuts across this model by inserting directly into the network functionality that alters the behaviour of the network. IP or TCP Packet Header fields may be altered on the fly, or, as with a transparent cache, middleware may intercept user traffic, use an application level interpreter to interpret the upper-level service request associated with the traffic, and generate a response, acting as an unauthorized proxy for the intended recipient. With middleware present in an IP network, sending a packet to an addressed destination and receiving a response with a source address of that destination is no guarantee that you have actually communicated with the addressed remote device. You may instead be communicating with a middleware box, or have had the middleware box alter your traffic in various ways that are not directly visible to the sender.

In such an environment, it's not just the end-user applications that define an Internet-deployed service, because middleware is also part of the Internet service architecture. Services may be deployed that are reliant on the existence of middleware to be effective. Streaming video services, for example, become far more viable as a scalable Internet service when the streaming video server content is replicated across a set of middleware streaming systems deployed close to end users of the service. To change the behaviour of a service that has supporting middleware deployed requires the network middleware to be changed. A new service may not be deployed until the network middleware is altered to permit its deployment. Any application requiring actual end-to-end communications may have to have additional functionality to detect if there is network middleware deployed along the path, and then explicitly negotiate with this encountered middleware to ensure that its actual communication will not be intercepted and proxied or otherwise altered.

Conclusion

The cumulative outcome is that such a middleware-modified Internet service model is not consistent with an end-to-end architecture. It represents the introduction of a more muddled service architecture where the network may choose to selectively intervene in the interaction between one device and another. Such a network architecture may not have stable scaling properties. Such an architecture may not readily support entire classes of new applications and new services. Such an architecture may not be sufficiently flexible and powerful to underpin a ubiquitous global data communications system. All this middleware overhead makes applications more complex, makes the network more complex, and makes networking more expensive, more limited, and less flexible.

From this perspective, middleware is an unglamorous hack. To adapt a 350-year-old quote from Thomas Hobbes, middleware is nasty, brutish, and short-sighted. It is, hopefully, a temporary imposition on an otherwise elegant, simple, and adequate Internet architecture. [3, 4]

References

[1] Aiken, B. et.al, "Network Policy and Services: A Report of a Workshop on Middleware," RFC 2768, February 2000.

[2] Carpenter, B. ed., "Architectural Principles of the Internet," RFC 1958, June 1996.

[3] Hobbes, Thomas (1588-1679), *Leviathan* , London, 1651. Available from many sources, including, ISBN 0140431950, Penguin Press, 1982.

[4] Hobbes, Thomas, Leviathan

GEOFF HUSTON holds a B.Sc. and a M.Sc. from the Australian National University. He has been closely involved with the development of the Internet for the past decade, particularly within Australia, where he was responsible for the initial build of the Internet within the Australian academic and research sector. Huston is currently the Chief Scientist in the Internet area for Telstra. He is also a member of the Internet Architecture Board, and is the Secretary of the Internet Society Board of Trustees. He is author of *The ISP Survival Guide* , ISBN 0-471-31499-4, *Internet Performance Survival Guide: QoS Strategies for Multiservice Networks* , ISBN 0471-378089, and coauthor of *Quality of Service: Delivering QoS on the Internet and in Corporate Networks* , ISBN 0-471-24358-2, a collaboration with Paul Ferguson. All three books are published by John Wiley & Sons. E-mail: gih@telstra.net