The Middleware Dilemma

Geoff Huston
March 2001

It is not often that an entire class of technology generates an emotive response. But, somehow, middleware has managed to excite a lot of folk. For some ISPs middleware, in the form of web caches, is not only useful, its 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 folk middleware is an integral part of the network. For others, middleware is seen as something akin to network heresy. Not only does middleware often break the semantics of the internet protocol, it is also in direct contravention to the end to end architecture of the Internet. Middleware breaks the operation of certain applications.

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 through to being immoral and possibly illegal. Strong stuff indeed for an engineering community.

So what is middleware all about and why the fuss? One definition of middleware is a network device which does something other than forward an IP packet onward along the best path to the packet's destination address. In other words, anything other than a router. 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 was addressed to the middleware device itself.

Why would a network go to all this bother to trap and process certain packet? Surely its 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.

Lets look at this in a bit more detail, using a specific form of middleware. A common form of middleware is the so-called "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 (an 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 which matches the destination address of the original request, assuming the identity of the 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 correctly configured, 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 slowly under extreme load, while the local cache continues to operate at a consistent service level. The combination of the potential for improved performance and lower overall cost is certainly one which looks enticing: the result is the same set of web transactions delivered to customers, but cheaper and faster.

But not everything is perfect in this transparent caching world. What if the web server used a security model which 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 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 server's status, 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. Given that the middleware cache is sitting inside a network somewhere on the path between the client and the service, 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 differnet requestors, based on some inference gained from the source IP address of the requestor, or the time of day, or some other variable. A transparent cache will not detect such variations in the server's response 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 percieved 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.

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 are 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 server.

The common theme of these issues is that there are a set of inconsistent assumptions at play here. One 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.

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. Network Address Translators can reduce the demand for IP address space. Firewalls can be reasonably effective security policy agents. 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. But middleware comes at a steep long term price.

The leverage 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 applications that are resident on the connected computer. Within this architecture adding new services to the network is as simple as distributing new applications. 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 passes the entire responsibility for service control to the network's users. The real strength of the Internet lies in its architectural simplicity and lack of interdependencies within the network.

Middleware cuts across this model by inserting directly into the network functionality which alters packets on the fly, or, as with a transparent cache, intercepts traffic, interprets the upper level service request associated with the traffic and generates responses by acting as a proxy for the intended recipient. With middleware present in an internet 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 address remote device. You may instead be communicating with a middleware box, or have had the middleware box alter your traffic in various ways. Now its not just the end user applications which define an Internet service. Middleware also is becoming part of the service. To change the behaviour of a service which has middleware deployed requires the network's middleware to be changed. A new service may not be deployed until the network's 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.

All this middleware overhead makes applications more complex, makes the network more complex, and makes networking more expensive and less flexible. From this perspective middleware is an unglamorous hack; nasty, brutish and, hopefully, short-lived.