Towards a Platform for Intelligent Activity at the Edge

Hilarie Orman
Novell, Inc.
horman@novell.com

Abstract

Proxy caches for content on the Internet are high-performance platforms with complex software services. Because they understand application semantics, and because they have a great deal of memory, they are the natural place for new services that are tailored closely to site or user preferences and requirements. The engineering aspects of caches and how they contribute to a new network infrastructure for highly capable or intelligent services are examined in this paper.

1. Introduction

From the beginning of the networking era, there have been expectations of intelligent software as integral part of applications. The concept of distributed software agents followed quickly on the heels computer networking, as evidenced by the early work on actors with distributed control [Hew77]. This work assumed widespread, inexpensive methods for communication and control, but no large-scale system emerged to meet the hopes of the intelligent agent researchers. Instead, the Internet thrived on applications with simple point-to-point semantic. As the Internet entered its more recent wildly successful phase, more thinking was invested in having the network software itself perform complex tasks. Two extreme viewpoints in this space have been intelligent agents and active networks. However, in both cases, the infrastructural support for the concepts has lagged far behind the thinking. What we have seen is the success of an Internet architecture that presents simple end-to-end semantics but with an implementation that has many more layers "in the middle". The thesis of this paper is that an emerging middleware layer, implemented on edge devices, provides the foundation for the addition of greater capability to large-scale networks, and this is part of the enablement of a new class of services.

2. Background and Overview of Proxy Architectures

As the Internet evolved from a small research experiment into a global communication infrastructure, the uniform architecture began to separate into two pieces: the high bandwidth core and the edge organizations (collections of LANs). Separate routing protocols evolved for the two regimes, and some differences in addressing and naming were smoothed over by translating gateways. Finally, security concerns led to a fairly complete separation enforced by firewalls.

The World-Wide Web brought about a revolution in the Internet by making it easy and inexpensive to publish and consume high-grade documents (visually pleasing, organized, etc.). Most Internet users came to believe that the Internet and the WWW were synonymous. Within a matter of a few years, the problem of scaling the web became a critical matter. To solve the problems of reliability and latency, engineers have been building a new infrastructure on the old frame.

The first important advances in the network were load balancing for WWW servers and caching to reduce latency in delivery. The platforms that assist in the first category are called "reverse proxies", and the platforms in the second category are "proxies." Both are critical in avoiding Internet meltdowns. For example, proxies are an essential component in quality of service, because the latency of Internet accesses depends on the square of the number "hops" [Rob98]. Proxy caches reduce the number of hops and thus contribute to improved latency; they also reduce the congestion in the core of the Internet and make other services run faster.

It was natural to add proxy caches to firewalls or security gateways, and this "platform sharing" has contributed to widespread adoption of both forward and reverse proxies. This has also provided a way to unify security services by collocating two different functions, both having common security dependencies, in one box.

The placement of such services at the point joining an organization is also an architectural "chokepoint" where engineering for high performance has a very good payoff. Competitions for high-speed cache services have been intense and have bred a new class of engineers who are expert in network stack scaling and optimization. Web caching workshops and competitive bakeoffs have become part of the Internet engineering landscape.

The second stage of development is going on today, as a middleware layer of cooperating content repositories and/or caches is deployed by content distribution services. In this phase, there is movement of content to distribution sites, again, to reduce latency due to network transmission times but also to reduce the latency due to the load on the origin servers.

Caches can be the platform for adding intelligent services to the network infrastructure. As we will see, they differ from switches and routers because they have much more RAM and disk space. Because so much storage is available on very fast, inexpensive machines, there are opportunities for adding much more in the way of application level services, invoked both explicitly and implicitly, and for keeping the state information that the services require.

The backing store for caching systems presents opportunities for optimization that are novel with respect to traditional network file systems such as NFS or AFS. The problem might be presented as "caching all the way down", because the disk system is primarily for holding the working set of the cache, and the consistency requirements for backing store are much relaxed from those of a file system. This flexibility offers more opportunities for minimizing disk access time by careful selection of cylinder groups and at the same time imposes fewer "housekeeping" requirements. One might consider the backing store to be similar to a file system that has frequent deletes with no processing cost for the deletion.

There are research areas open in the management of disk backing stores, keeping multiple copies of objects, coping with dynamic content, tighter synchronization with origin servers, and moving to a hybrid system that supports network file systems with efficient caching store.

3. Smart Middleware Boxes

3.1. Performance Management

Web caching applications are probably the most stressful environment for Internet protocols. Almost any performance anomaly will be magnified in such an environment, and engineers who understand the details of a successful networking implementation are being honed in greater numbers than ever before. To meet the demands of holding the working set of web data, the configuration of a caching server or proxy uses anywhere from 256M to 4G bytes of RAM and from a gigabyte to nearly a terabtye of disk storage.

The point of caching is, of course, primarily to reduce latency. The Internet suffers from great variations in latency because it does not reserve resources in advance of communication. The latency variation can be two or more orders of magnitude in common cases Kal[99], though particular point-to-point connections may show almost no variation for weeks at a time. The caches serve to minimize the latency and its variation, leading to a much more predictable service model.

Web caches that are placed near the end user (the browser) are frequently configured as "forward proxies", and they typically have very large working sets (millions of objects) in order to achieve a cache hit rate of over 50%. The latency variation can become much smaller when the cache is near the user, because the number of buffering points between the user and the data is probably only 2 or 3 (e.g., a router and a proxy cache).

Caches can also reduce latency by distributing the load from the original content site to the high performance cache. This offloading of network communication and object retrieval leaves the original content servers free to perform non-cacheable computations in support of their web service; the computations are usually hidden in scripts that are privy to confidential information and/or large databases. Caches that reduce latency by offloading the content have working sets which are are typically much smaller than the ones for forward proxies. In fact, at times all of the site's content fits into RAM on its reverse proxy; in contrast, a forward proxy may have trouble fitting the index for its disk-cached content into the same amount of memory.

Web caching systems have the latitude to implement efficient networking systems in ways that are difficult for general purpose operating systems. For a server machine to handle 10K connections and delivery of up to a 10^10 objects per hour is a difficult task for most network stacks. Such systems must have very fast TCP message dispatching, space for large amounts of connection state (the TCB's), and efficient handling of storage for network data. Work in the research community over the last decade has repeatedly demonstrated methods that help meet the goals; the use of polling in preference to interrupts [Min93], "no copy" I/O for network data, and the ability to communicate with applications without copying or re-encoding data [Dru93]. Because web caching systems do not have the burden of supporting general purpose processing as in standard operating systems, they can make use of the research results for very high throughput. Web caching systems today can handle a sustained rate of up to 2400 requests/second, which far exceeds the rates considered maximal 10 years ago.

Name lookups, for the Domain Name System (DNS), can be a major bottleneck for high performance boxes, and most of these systems maintain huge caches of name-to-IP-address mappings, in effect caching much of the root server information and many common hostnames (because www.xxxzzzy.com is the usual form of a name presented for resolution). A cache of 20K domain names usually results in a 99% hit rate for web proxy systems; the number of instructions for a DNS lookup is little more than the time to hash the name and do the lookup. Cache misses are infrequent, occurring only once every few seconds, and this keeps UDP connections down to a very few. A 20K cache could be usefully configured to serve all DNS lookup needs for an entire organization.

3.2 Expanded functionality

The middleware layer of web caching boxes is rapidly expanding. As web caching appliances become more pervasive, more uses get loaded onto them. Thus, we see the next stage of the Internet architecture revolution: the movement of a service infrastructure into the network.

Some services already exist in the net. These include many kinds of web-based mail services, homepage hosting, file storage, messaging. However, for the most part, these are not distributed services, and they represent small points or islands of service, themselves separated by firewalls. Web caches, on the other hand, sit in the infrastructure itself, and their services apply to the applications riding above them.

New services that are emerging through this middleware are centered on web functions. We note several areas where current research and products are centered: object-specific functionality, user-specific functionality, and authentication and access control services.

3.2.1 Name Translation and Content Transcoding Intelligence

There has been no simple way to have content available from multiple sites in the Internet. Replication strategies, meant to reduce latency and to distribute server load, often founder on single points of provisioning (such as DNS servers). This is rapidly changing, and users now find that loading a web page from one location can mysteriously turn into loading a page from any of a number of servers seemingly unrelated to the target.

One way of doing this involves translating the content, turning an embedded name (URL) into a new name, perhaps by changing the hostname, based on the user's location, current conditions, or other factors. In essence, multiple path routing results from URL translation.

The ability to select a new server location and to translate the embedded names very rapidly is a level of intelligence not normally associated with applications. The devices that do this have streamlined ability to parse HTTP and HTML, to do text substitution, and to deliver the results to the network interface with minimal overhead. This is an edge service that requires some understanding of the application language, and it turns protocol layering concepts on their heads.

Novell's approach to web proxies uses translation for two applications. One is to assist in load balancing and another is to secure the transmission of confidential data. In each case, the translation is done by pattern matching on URL's.

A different mechanism retains state between object name references, establishing a user-specific mapping with a short lifetime. For example, an image and the site with information about the image might be linked implicitly. When the user browser requests the image, a computation at the web cache, acting as a proxy, determines the image and its related site. This makes the content of the page itself a function of the proxy, and this is an especially rich service model.

A more complex processing model, for services that are very computationally expensive or only available on non-proxy platforms, uses nearby processors that are in the same administrative domain as the proxy. These might be thought of as "lollipop" processors, because the processing flow, normally straight through the proxy between the client and server, now takes a tour sideways.

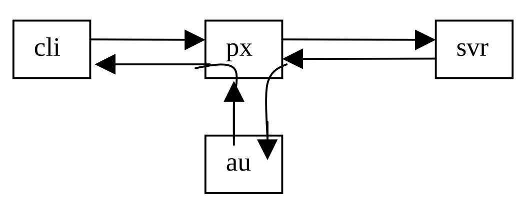

Figure 1: Example data flow, auxiliary processing using ICAP

The proposed protocol between the proxy ("px") and the auxiliary processor ("au") is built on HTTP and is known as the Internet Content Adaptation Protocol – ICAP [ICAP00].

Examples of complex services that might be performed by an auxiliary processor include: having HTML text converted to different human languages, audio content transcoded to achieve a service level based on current resource availability (network quality of service), negotiation with advertisers for micropayments for accepting advertising and offering demographics, access to service information that is proprietary to the owner of the auxiliary processor box, personalization of web pages, hosting user processes for directed personalization, management of “persona” information based on user policy and profiles, analysis of access patterns to remote or local web servers (correlation, clustering, neural network creation), access to trusted financial services, high assurance security services, etc.

3.2.2 Object Policy

If, by abuse of language, we consider URL's to be object names, we can say that edge devices are delivering a form of object intelligence by analyzing object names and mapping them to attributes. These attributes may be simple, such as content ratings based on external evaluations, or quite complicated, such as automatically generated content classification based on word or image analysis, cluster analysis based on access patterns by other users, etc.

Architecturally, the ability to map names to attributes and evaluators is an important advance in the capability of edge devices. It entails maintenance of state, caching, and parsing of names at a chokepoint between the end consumers and their network access. The mechanics that allow this are a general solution for several classes of problems, and can be the foundation for performing such functions as selecting the image resolution to be used for low-bandwidth or low-power devices in the future.

Access control based on content rating is a discretionary policy generally set at a forward proxy by users or on their behalf by the proxy administrator in accordance with a site policy. Content rating services typically depend on a very large list of URL's and a vector of their attributes. By building a state machine that is the compilation of the strings, a proxy can very quickly match URL requests against the rating and the content-based policy for the user and/or the site.

The performance cost of the policy check is quite small; the number of instructions executed is approximately equal to the number of bytes in the URL name. The main performance penalty comes from the amount of space used by the state machine; that memory is not available for caching of content, and this leads to the biggest performance penalty in terms of content delivery. When the policy database occupies 10% of the memory, observed performance on web benchmarks decreases by about 5%, even if the database lookups are not executed. If the lookups are executed, they only use about 5% of the total machine cycles.

The efficiency of a cache depends on the available memory. For a forward proxy, the ratio of disk to RAM can be as high as 10K to 1. The index for the disk cache can use nearly as much memory as the objects cached in RAM, in extreme cases. The RAM requirements for classification of hundreds of thousands of URLs is a noticeable fraction of the memory, and this mandates that any auxiliary data associated with URL sets be represented as compactly as possible. It is possible to maintain auxiliary data for hundreds of thousands URLs that are non-resident in the cache and to access the data quickly when making a decision about whether or not to fetch a page, what quality of service to use for its transfer, or what origin server to use for its retrieval.

The dependence of performance on memory demonstrates a point of importance for intelligent edge devices: in general, proxy caches are not CPU limited, running idle as much as 90% of the time in normal operation. This leaves a lot of cycles for running computational tasks on the caching system itself.

3.2.3 Identity and Security

Edge devices are the injection point for basing access control decisions on Internet-wide identities and for management of identity information. Most large organizations have fine-grained management of access control internally, but only the coarsest management of objects presented externally. A common policy for many organizations is that the only bi-directional information flow is through SMTP, and the only HTTP access is for information that is cleared for public access. In the latter case, the information goes onto a web server that is primarily "read only".

The SSL3 protocol [Fre96] is the *de facto* standard for secure connections for web browsers today, but while it is sufficient for encrypting TCP connections, there are few systems that integrate it into authorization or access control. By clever use of the HTTP protocol, it is possible for edge devices to interpose access control mechanisms based on SSL privacy and authentication.

Reverse proxies, sitting near the authoritative sources for content, are natural enforcement point for access control at the granularity of a web page. They can present an access policy for the outside world that is a consistent extension of the internal controls, without changing any of the legacy software for the origin site. The reverse proxy can implement identity services, authentication services, and access control enforcement on behalf of the origin server organization, using SSL as the common security protocol.

Consider an organization with a public web server that wants to enforce a simple security model: information on its server www.myorg.com is public (no authentication), while information on www.internal.myorg.com is accessible to any user authenticated by the network authority for myorg.com. Novell's reverse proxy solution to implementing this policy allows the administrator of the reverse proxy to enter the control rules as initial configuration schema for the proxy, to specify a certificate for authenticating the server to the clients, and to designate an authentication server for trusted communication.

When the reverse proxy receives a request for an item in www.myorg.com, it fetches the object from the origin server if need be (communication between the proxy and the origin server is trusted) and delivers the page to the requestor. If the client request is for an object in www.internal.myorg.com, perhaps '/plan1.html', then it redirects the client to use https as the access protocol. The client browser will comply with an SSL encrypted connection to the reverse proxy, and the proxy will redirect the client to an authentication server, and that server may either use SSL mutual authentication to establish an authenticated identity or it may redirect the client through a longer login dialogue. At the end of the dialogue, the reverse proxy will receive an indication from the authentication server that the login was either successful or unsuccessful. In the former case, it will deliver the originally requested object to the client over their SSL connection; the latter case it will respond to the client with an error message.

The use of SSL requires no changes to either the client or server and illustrates how a new service is easily built on a proxy cache foundation. Obviously, more complicated and finer-granularity access control policies are easily added as the requirements of the organization evolve.

If authentication intelligent edge devices are the directors of the authentication process, they must have the intelligence to deal with identity information. This is an attractive notion, because a trustworthy platform in the network avoids the many problems associated with maintaining security information on end-user machines or bound to particular organizations. Internet users are beginning to desire an Internet persona that is under their own control, even when their employment or Internet service provider changes.

An even more open model would allow the objects to have access control lists with pairs consisting of the distinguished name and certifying authority of authorized users. This allows organizations to extend their fine-grained control methods to the growing class of Internet users with credentials backed by public keys. While it may be difficult to retrofit legacy operating systems with such controls, proxies that have been extended to understand identity, authentication, and access controls can easily enforce access rules that are extended far beyond the legacy model.

4. Next stage: the Intelligent Infrastructure

Adding more intelligent services to the network edges is a key part of creating new network services and of scaling the performance requirements to encompass global services. We will continue to see more software added to edge devices in the future. Routers, switches, proxy services, and other edge dwellers will creep "outwards" towards application services. The way to keep the network healthy during this transition is to work towards an architectural model that supports programming at the network edges.

Edge devices today are specialized creatures stepping gingerly towards open standards, common application interfaces, and extensibility. The edge world can take a giant step forward by defining some common execution environments and library services for unifying the services. A programmability model could take a two or three step approach by implementing a series of execution environments that have engineered tradeoffs of expressibility vs. ease of use vs. security.

The first tier programming language would define a language based on the keywords or tags and a set of transformation rules based on regular expression replacement operations. API's for common protocols such HTTP, HTML, SNMP, RTP, etc. would be available as an open standard. These rules would be dynamically loadable and compilable on receipt.

The second tier of languages would be bounded time execution languages with variables and assignments and conditionals, each with a security policy enforced by a policy engine in the execution environment. These languages would also be dynamically loadable and amenable to just-in-time (JIT) compilation.

The third tier languages would be general purpose and dynamically loadable; Java is a good candidate for a third-tier language. Third tier languages present security challenges, but their expressive power is a compelling force, and operating systems with effective process isolation mechanisms will be able to run them at the network edges with minimal risk.

With a programmable network edge, application semantics that operate on network data “flows” are possible, and they can assure consistent semantics for applications even when the applications are resident in the edge infrastructure, rather than at a central site.

The next challenge is to move from explicit programming of the network edge to programming that flows transparently to the points in the network that can most effectively execute the application.

The original concept of the Internet’s end-to-end semantics for data transport and complex semantics for applications, is yielding to a richer model that allows edge devices to play a part in routing, caching, and executing application semantics. This is an evolving model, moving through a logical series of steps towards more complicated distributed execution models.

The client-server model, based on RPC [Whi76], has been a very successful way to build network services. The stateless semantics simplify the design of the client, and HTTP in large part relies on an RPC model. HTTP proxies are able to execute the semantics “parasitically”, i.e., in the case of transparent proxies, they are not the intended (“addressed”) parties to a transaction, but they can fulfill it nonetheless. Proxies have difficulty participating in connections where the server generates responses by executing local code with semantics that are unavailable to the proxy (e.g. cgi-bin scripts) or where client runs code from the server (e.g. “applets”). Active networks [Ten98], use an execution model that bases transport services on code that is executed through the infrastructure. Proxy platforms can enable a happy medium in which applications are either written with explicit proxy code that moves some of the application functionality onto proxy platforms (“proxylets” [Cro00]), or else the applications use servlets or applets with well-understood semantics that can be parasitically executed by the proxies. Both forward and reverse proxies can participate, based perhaps on the available of environmental information.

The logical extension of the distributed execution model may support mobile code, mobile agents, or other “untethered” applications that are not feasible today. The platforms for running these may resemble proxy platforms but they will have access to a broader range of standard semantics, directories, verifiable credentials, and most likely a computing base that supports higher assurances of trust than today’s machines.

A third stage of evolution that an intelligent infrastructure enables is third-party services that exist for the purpose of linking other services together on behalf of users. Today consumers can invoke shopping agents that attempt to maximize the value of information to the user, but the agents must execute in adversarial environments where their goals are thwarted. An intelligent service infrastructure will enable user agents to negotiate in good faith with brokering services in order to get a broad spectrum of information with higher value. These services will meet and execute “in the network”.

Engineering the next generation of network services will necessitate an intelligent infrastructure. Challenges exist in security, extensible service models, and interoperation in new areas such as the integration of telephony and Internet services with wireless handheld devices, the next generation of Internet video services, collaborative document management, and the instant messaging services hovering in our future. In order to make these services scalable at the same pace that we have seen the Internet and WWW services flourish, a programmable distributed service infrastructure is an obvious architectural requirement.

The infrastructure itself may likely yield a revolution much like we have seen in web services. This is likely to be a software agent infrastructure, with "safe enclaves" existing in the communications infrastructure itself. With this, users will be able to launch their software shopping agents into the network for untethered operation and trustworthy results.

5. Conclusions

Web caching and proxying systems are developing the platforms for an important new class of application accelerators and enhancers at the network perimeter. While the core of the network can be dedicated to moving bits over increasingly “fatter” pipes for more and more multimedia data, the need to dynamically manage information for individuals and businesses is driving the edge devices to a rich services model. This is ushering in a new notion of what constitutes the essential services of a global internet, of who engineers it and how, and it blurs the line between application and communications infrastructure. This trend is likely to continue, and it illustrates the mechanisms that will continue transforming the Internet through the next few decades.

Bibliography

[Cro00] Crowcroft, John, Self Organising Application-level Routing, Tech Report RN/00/23, University College London, Feb. 2000.

[Dru93] Peter Druschel and Larry L. Peterson, Fbufs: A High-Bandwidth Cross-Domain Transfer Facility (1993), ftp://ftp.cs.arizona.edu/reports/1993/TR93-05.ps

[Hew77] Hewitt, C. (1977), "Viewing Control Structures as Patterns of Passing Messages", Artificial Intelligence 8(3), 323-364.

[ICAP00], Elson, J. et al, ICAP the Internet Content Adaptation Protocol, http://www.i-cap.org/specification.txt

[Kal99] Kalidindi, Sunil, and Zekauskas, Matthew J., Surveyor: An Infrastructure for Internet Performance Measurements, INET '99 Proceedings

[Fre96] Freier, Alan O. and Karlton, Philip and Kocher, Paul C. The SSL Protocol Version 3.0, Mar. 96, http://www.netscape.com/eng/ssl3/ssl-toc.html,

[Min94] Minshall, Greg, and Major, Drew and Powell, Kyle, An Overview of the NetWare Operating System, Proceeding of Usenix Winter Technical Conference, 1994.

[Rob98] Roberts, Larry, Plenary Address, SIGCOMM '98, Vancouver, CA.

[Ten97] Tennenhouse, D, et al., A Survey of Active Network Research, IEEE Communications, Jan. 1997

[Whi76] White, J.E., A high-level framework for network-based resource sharing, Proceedings of the National Computer Conference, June 1976.