Content Networks

Christophe Deleuze

The Internet is constantly evolving, both in usage patterns and in underlying technologies. Last few years, there has been a growing interest on “content networking” technologies. Various differing systems can be labelled under this name, but they all share the ability to access objects in a location independant manner. Doing so implies a shift in the way communications take place in the Internet.

Classic Internet model

The Internet protocols stack comprises three layers and is shown in Figure 1. The network layer is implemented by IP (Internet Protocol) and various routing protocols. Its job is to bring datagrams hop by hop to their destination host, as identified by the destination IP address. IP is “best-effort”, meaning that no guarantee is made about the correct delivery of datagrams to the destination host.

The transport layer provides an end to end communication service to applications. Currently two services are available, a reliable ordered byte stream transport, implemented by TCP (Transmission Control Protocol), and an unreliable message transport, implemented by UDP (User Datagram Protocol).

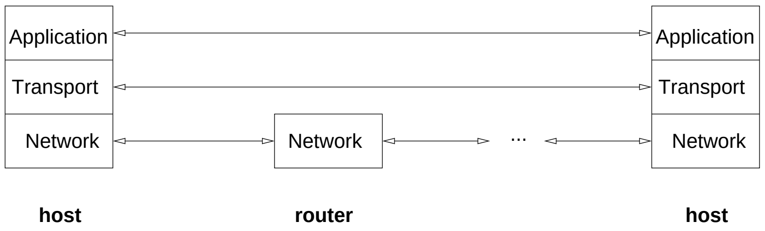

Figure 1: The three layers of the Internet protocols stack

Above the transport layer lies the application layer, which defines application message formats and communication semantics. The web uses a client server application protocol called Hypertext Transfer Protocol (HTTP) [10].

A design principle of the Internet architecture is the “end-to-end principle”, stating that everything that can be done in the end hosts should be done there, and not in the network itself [8]. That’s why IP provides so crude a service, and transport and application layer protocols are implemented only in the end hosts.

Application objects, such as web pages, files, etc. (we’ll simply call those “objects”) are identified by uniform resource locators (URLs)[1]. URLs for web objects have the form `http://host:port/path`. That means that the server application lives on host with `host` hostname (or possibly IP address) on port `port` (with default value of 80), and knows the object under the name `path`. Thus URLs, as their name implies, tell *where* the object can be found. To access such an object, a TCP connection is open to the server running on the specified host and port and the object named `path` is requested.

[1]Actually URLs identify “resources” that can be mapped to different objects called “variants”. A variant is identified by an URL and a set of request header values, but in order to keep things simple, we’ll not consider this in the following.

Content networks

Content networks (CNs) aim to provide location independant access to an object, most commonly because they handle some kind of (possibly dynamic) replication of the objects. Object replication can be used to improve availability in the face of network or server failure, to make possible large numbers of concurrent accesses, and to allow users to access to close copies of objects, thus limiting the effects of network congestion. By design, URLs are not suited to identify objects available on several places on the network.

Handling such replication and location independant access usually involves breaking the end to end principle at some point. Communication is no more managed end to end: intermediate network elements operating at the application layer (whose most common type are “proxies”) are involved in the communication. (CNs are not the only case where this principle is violated).

In the same way that IP routers relay IP datagrams (i.e. network layer protocol data units), routing them to their destination according to network layer information, those application layer nodes relay application messages, using application layer information (such as content URLs) to decide where to send them. This is often called *content routing*.

So the goal of a CN is to manage replication, handling two different tasks: *distribution* ensures the copying and synchronization of the instances of an object from an *origin server* to various *replica servers*[2], and *redirection* allows users to find one instance of the object (possibly the one closest to them.) This is illustrated in Figure 2.

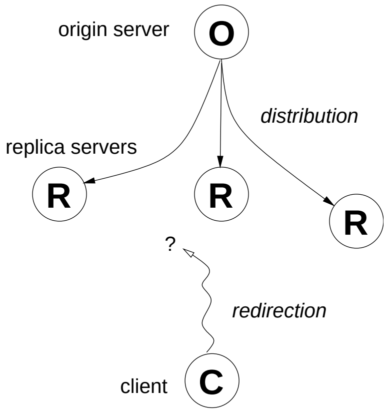

Figure 2: Elements of a content network

There exist various kinds of CNs, differing in the extend to which they handle these tasks and in the mechanisms they use to do so. There are many possible ways to classify them. In this article, we use a classification based on who owns and administrates the CN. We thus find three categories: CNs owned by network operators, content providers, and users.

Network operators’ CNs

Network operators (also called Internet Service Providers or ISPs) often install caching proxies in order to save bandwidth [11]. Clients send their requests for objects to the proxy instead of the origin server. The proxy keeps copies of popular objects in its cache and can answer directly if it has the requested object in cache[3]. If this is not the case, it gets the object from the origin server, possibly stores a copy in its cache, and sends it back to the client.

[2] By “replica”, we mean any server of any kind other than the origin that is able to serve an instance of the object. This term often has a narrower meaning, not applying, for example, to caching proxies.

[3] To be really precise, such a caching proxy does not cache objects but server responses.

This caching proxy scheme can be used recursively, making those proxies contact parent proxies for requests they can't fullfill from their local store. Such hierarchies of caching proxies actually lead to constructing content distribution trees. This makes sense if the network topology is tree-like, although there are some drawbacks, including the fact that unpopular objects (not found in any cache) experience delays increasing with the depth of the tree. Another problem is with origin servers whose closest tree node is not the root.

The Squid caching proxy [5] can be configured to choose the parent proxy to query for a request based on the domain-name of the requested URL (or to get the object directly for the origin server). This allows setting up multiple logical trees on the set of proxies, a limited form of content routing. Such manual configuration is cumbersome especially since domain names do not necessarily (and actually mostly not) match network topology. Thus the administrator must know where origin servers are in the network to use this feature effectively.

The same effects can be achieved, to some extent, in an automatic and dynamic fashion using ICP, the Internet Cache Protocol [16, 15]. ICP allows a mesh of caching proxies to cooperate by exchanging hints about the objects they have in cache, so that a proxy missing an object can find a close proxy that has it. One advanced feature of ICP allows to select among a mesh of proxies the one that has the smallest RTT to the origin server.

One design flaw of ICP is that it identifies objects with URLs. We have mentionned previously that an URL actually identify a resource, that can be mapped to several different objects called variants. Thus information provided by ICP is of little use for resources that have multiple variants. However, in practice most resources have only one variant so this weakness doesn't do much harm.

Users normally configure their browsers to use a proxy, but automatic configuration is sometimes possible. Multiple proxies can be used by a client with protocols such as the *Cache Array Routing Protocol* (CARP) [14]. To avoid configuration issues a common trend is for ISPs to deploy *interception proxies*. Network elements such as routers running Cisco's *Web Cache Coordination Protocol* (WCCP) [6, 7] redirect HTTP traffic to the proxy, without the users knowing. The proxy then answers client requests pretending being the origin server. This poses a number of problems, as discussed in [12].

Caching proxies have limited support for ensuring object consistency. Either the origin server gives an expiration date or the proxy estimates the object lifetime based on the last modification time, using an heuristic known as "adaptive TTL".

Content providers' CNs

Contrary to ISPs whose main goal is to save bandwidth, content providers want to make their content widely available to users, while staying in control of the delivery (including ensuring that users are not delivered stale objects). We can again roughly classify such content networks in three sub-categories:

servers farms are locally deployed CNs aimed at providing more delivery capacity and high availability of content;

mirrors sites are distributed CNs making content available in different places thus allowing users to get the content from a close mirror;

CDNs (Content Delivery Networks) are mutualized CNs operated for the benefit of a number of content providers, allowing them to get their content replicated to a large number of servers around the world at lower cost.

Server farms

Server farms are made of a load balancing device (we'll call it "switch") receiving client requests and dispatching them to a series of servers (the "physical" servers). The whole system appears to the outside world a as single "logical" server. The goal of a server farm is to provide scalable and highly available service. The switch monitors the physical servers and uses various load metrics in its dispatching algorithm. Since the switch is a single point of failure, a second switch is usually set up in a hot failover standby mode, as shown on Figure 3.

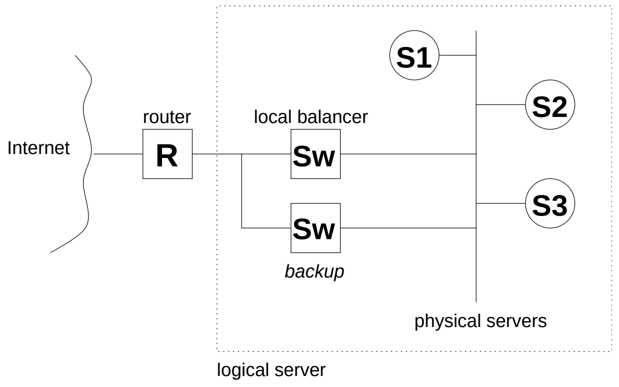

Figure 3: Server farm

Some switches are called "L4 switches" (4 is the number of the transport layer in the OSI reference model) meaning they look at network and transport layer information in the first packet of a connection to decide to which physical server the incoming connection should be handed. They establish a state associating the connection with the chosen physical server and use it to relay all packets of the connection. The exact way the packets are sent to the physical servers vary. It usually involves a form of mangling of IP and TCP headers in the packets (like NAT – Network Address Translation – does) or IP encapsulation. These tricks are not not necessary if all the physical servers live on the same LAN (local area network).

More complex, "L7 switches", (7 is the number of the application layer in the OSI reference model) look at application layer information, such as URL and HTTP request headers. They are sometimes called "content switches". On a TCP connection, application data is available only after the connection has been opened. A proxy application on the switch must thus accept the connection from the client, receive the request and then open another connection with the selected physical server and forward the request. When the response comes back, it must copy the bytes from the server connection to the client connection.

Such a splice of TCP connections consumes much more resources in the switch than the simple packet mangling occuring in L4 switches. Bytes arrive at one connection, are handed to the proxy application which copy them to the other connection, all of this involving multiple kernel mode to user mode memory copy operations and CPU context switches. Various optimizations are implemented in commercial products. The simplest one is to put the splice in kernel mode. After it has sent the request to the physical server, the proxy application asks the kernel to splice the two connections, and forgets about them. Bytes are then copied between the connections directly by the kernel, instead of being given to the proxy application and back to the kernel.

It is even possible to actually *merge* the two TCP connections, that is, simply relaying packets at the network layer to establish a direct TCP connection between the client and the physical server. This requires mangling TCP sequence numbers (in addition to addresses and ports) when relaying packets, since the two connections won't have used the same initial sequence numbers. This can be much more complex to perform (or even impossible) if TCP options differ in the two connections.

Mirror sites

In such a CN, a set of servers are installed in various places in the Internet, and they are defined as “mirrors” of the master server. Synchronization is most commonly performed on a periodic basis (often every night), using FTP or specialized tools such as rsync [4].

Redirection is performed by the user itself for most sites. The master server, to which the user initially connects, displays a list of mirrors with geographic information and suggests the user to choose a mirror close to him, by simply clicking on the associated link.

This process can be sometimes automated. One trick is to store the user’s choice in a cookie, such that the next time the user connects to the master site, the information provided in the cookie will be used to issue an “HTTP redirect” (an HTTP server response asking the client to retry the request on a new URL) to the previously selected site.

Other schemes involve trying to find which of the mirrors is closest to the user based on information provided in the user request (such as prefered language) or indicated by network metrics. Such schemes were not very common for simple mirror sites, but today many commercial products allowing for this kind of “global load balancing” are available.

In any case (except if redirection is automatic and DNS based – we discuss this in the next section) the URLs of objects change across mirrors.

Content Delivery Networks

Most content providers can’t afford owning a number of mirror sites. Having servers in different places around the world costs lots of money. Operators of CDNs own a large replication infrastructure (Akamai, the biggest one, claims to have 15000 servers) and get paid by content providers to distribute their content. By mutualizing the infrastructure, CDNs are able to provide very large reach at affordable costs.

CDN servers do not store entire sites of all the content providers, but rather cache a subset according to local client demand. Such servers are called *surrogates*. They manage their disk store like proxies do, and serve content to client like mirrors do (i.e. contrary to proxies they act as authoritative source for the content they deliver).

Due to the fact that the number of surrogates can be so large, and to the attractivity of a “no user configuration is necessary” argument, CDNs typically include complex redirection systems allowing to perform automatic and user-transparent redirection to the selected surrogate. The selection is based on information about surrogates load and on network metrics collected by various ways such as routing protocol information, RTTs (round trip times) measured by network probes etc. The client is made to connect to the selected surrogate either by sending it an HTTP redirect message, or by using the DNS system: when the client try to resolve the hostname of the URL in an IP address to connect to, it is given back the address of the selected surrogate instead. Using the DNS ensures that the URL is the same for all object copies. In this case, CDNs actually turn URLs into location independant identifiers.

In addition to proxy-like on-demand distribution, content can also be “pushed” in surrogates in a proactive way. Synchronization can be performed by sending invalidation messages (or updated objects) to surrogates.

CDN principles are also being used in private intranets for building Entreprise Content Delivery Networks (ECDNs).

Users’ CNs

Users operated CNs are better known under the name peer to peer (P2P) networks. In these networks, the costly replication infrastructure of other CNs is replaced by the users’ computers who make some of their storage and processing capacities available to the P2P network. Thus, no big money is needed, and no one has control over the CN.

One advantage P2P networks have over other CNs in that they are usually built as overlay networks and do not strive for seamless integration with the current web. Thus they are free to build new distribution (some of them allow downloading files from multiple servers in parallel) and redirection mechanisms from scratch, and even to use their own namespace instead of being stuck with HTTP and URLs.

P2P networks basically handle the distribution part of replication in a straightforward way: the more popular an object is, the more users will have a copy of it, thus the more copies of the object will be available on the network. More complex mechanisms can be involved but this is the basic idea.

The redirection part of replication is more problematic with most current P2P networks. It can be handled by a central directory as in Napster: every user first connects to a central server, updates the directory for locally available objects, then looks up the directory for locations of objects the user wants to access. Of course, such a central directory poses a major scalability and robustness problem.

Gnutella and Freenet, for example, use a distributed searching strategy instead of a centralized directory. A node query neighors that themselves query neighbors and so on until either one node with the requested object is found or a limit on the resources consumed by the search has been hit. Although there is no single point of failure, such a scheme is no more scalable that the central directory. It seems easy to perform denial of service attacks by floodind the network with requests. Additionnaly, you can never be sure to find the object even if someone has it.

The above examples are primitive and have serious flaws, but much research work is being performed on this topic, see [13] for a summary.

Although they are currently mainly used for very specific file sharing applications, P2P networks do provide new and valuable concepts and techniques. For example *Edge Delivery Network* is a commercially-available software based ECDN inspired by Freenet. Various projects use a “scatter/gather” distribution scheme, useful for very large files: users download several file chunks in parallel from other currently downloading users, thus refraining to use server resources for long periods of time.

Some projects attempt to integrate P2P principles in the current web architecture and protocols. Examples are [3] and [1].

Conclusion

Current CNs have been designed and deployed as ad-hoc solutions of specific problems occuring in the current architecture of the network. Caching proxies lack proper means to ensure consistency, CDNs tricks DNS to turn URLs into location independant identifiers. P2P networks are mostly limited to file sharing applications.

CNs implement mechanisms to ensure distribution of content to various locations, and redirection of users to a close copy. They often have to break the end to end principle in order to do so, mainly because current protocols assume each object is available in only one statically defined location.

Probably the first step in building efficient distribution and redirection mechanisms for providing an effective replication architecture is the setting up of a proper replication aware namespace. Applications would pass an object name to a name resolution service and be given back one or more locations for this object. The need for such a location independant namespace has long ago been anticipated. URLs are actually defined as one kind of URIs (Uniform Resource Identifiers), another one being URNs (Uniform Resource Names) intended to provide such namespaces. An URN IETF working group [2] has been active for a long time and recently published a set of RFCs (3401 to 3406).

Work on the topic of content networking has also been performed by the now closed “Web Replication and Caching” (WREC) IETF working group which issued a taxonomy in [9]. An interesting survey of current work on advanced content networks is [13].

References

[1] BitTorrent. `http://bitconjurer.org/BitTorrent/`

[2] IETF URN working group. `http://www.ietf.org/html.charters/urn-charter.html`

[3] Open content network. `http://www.open-content.net`

[4] Rsync. `http://rsync.samba.org`

[5] Squid internet object cache. `http://www.squid-cache.org`

[6] M. Cieslak and D. Forster. Web cache coordination protocol v1.0. Expired Internet draft `draft-forster-wrec-wccp-v1-00.txt`, Cisco Systems, July 2000.

[7] M. Cieslak, D. Forster, G. Tiwana, and R. Wilson. Web cache coordination protocol v2.0. Expired Internet draft `draft-wilson-wrec-wccp-v2-00.txt`, Cisco Systems, July 2000.

[8] David D. Clark. The design philosophy of the DARPA Internet protocols. *Computer Communication Review*, 18(4):106–114, August 1988. Originally published in Proc. SIGCOMM'88.

[9] Ian Cooper, Ingrid Melve, and Gary Tomlinson. *Internet Web Replication and Caching Taxonomy*. RFC 3040, IETF, January 2001. `http://www.ietf.org/rfc/rfc3040.txt`

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616, June 1999. `http://www.ietf.org/rfc/rfc2616.txt`

[11] Geoff Huston. Web caching. *Internet Protocol Journal*, 2(3):2–20, September 1999.

[12] Geoff Huston. The middleware muddle. *Internet Protocol Journal*, 4(2):22–27, June 2001.

[13] H.T Kung and C. H. Wu. Content networks: Taxonomy and new approaches, 2002. `http://www.eecs.harvard.edu/~htk/publication/2002-santa-fe-kung-wu.pdf`

[14] Vinod Valloppillil and Keith W. Ross. Cache array routing protocol v1.0. Expired Internet draft `draft-vinod-carp-v1-03.txt`, Microsoft corporation, February 1998.

[15] D. Wessels and K. Claffy. *Application of Internet Cache Protocol (ICP), version 2*. RFC 2187, September 1997. `http://www.ietf.org/rfc/rfc2187.txt`

[16] D. Wessels and K. Claffy. *Internet Cache Protocol (ICP), version 2*. RFC 2186, September 1997. `http://www.ietf.org/rfc/rfc2186.txt`

CHRISTOPHE DELEUZE holds a Ph.D. degree in computer science from Université Pierre et Marie Curie, Paris. He worked on quality of service architectures in packet networks, then spent three years in a start-up company designing CDN systems. He has also been a teacher. E-mail: `christophe.deleuze@free.fr`