Globally Distributed Content
(Using BGP to Take Over the World)

Horms (Simon Horman)
`horms@valinux.com`
Senior Software Engineer
VA Linux Systems

December 2000

`http://supersparrow.org/`

> I love a sunburnt country,
> A land of sweeping plains,
> Of ragged mountain ranges,
> Of droughts and flooding rains.
> I love her far horizons,
> I love her jewel-sea,
> Her beauty and her terror-
> The wide brown land for me!
>
> *Dorothea McKellar – My Country*

Acknowledgement

I would like to acknowledge my employer VA Linux Systems, without whose help this work would not have been possible. Special thanks goes to Ben Buxton for his valuable assistance.

Licence

Version Information

Version	Date	Description
0.0.0	5th December 2000	Initial release

Presented

Linux.Conf.Au
18th – 20th January 2001
University of New South Wales
Sydney, New South Wales, Australia

LinuxWorld Conrerence and Expo
30th January - 2nd February 2001
Jacob Javits Center
New York, New York, United States of America

Abstract

Electronic content made available over the Internet is becoming increasingly important for providers and users alike. To provide the best possible service to end users it is desirable for content to be network-wise as close to client hosts as possible.

Static mirrors of sites are one means of distributing traffic between sites and giving users the opportunity to connect to a site that will give them a fast response. However, manually selecting sites, which may or may not be available, from a list of mirrors is a tedious process. The sites at the top of the list are a tempting choice — economy of choice in lieu of the possibility of faster access.

Instead of expecting users to manually select a mirror, it makes sense for the service provider to automatically direct clients to a site that will offer them good performance, that is to have a global load balancing algorithm in place. One such algorithm is to use BGP to select which site has the least cost path to a given client. This paper will examine the implementation of such an load balancing scheme.

Contents

1 Introduction

Electronic content made available over the Internet is becoming increasingly important for providers and users alike. To provide the best possible service to end users it is desirable for content to be network-wise as close to client hosts as possible.

Static mirrors of sites are one means of distributing traffic between sites and giving users the opportunity to connect to a site that will give them a fast response. However, manually selecting sites, which may or may not be available, from a list of mirrors is a tedious process. The sites at the top of the list are a tempting choice — economy of choice in lieu of the possibility of faster access.

Instead of expecting users to manually select a mirror, it makes sense for the service provider to automatically direct clients to a site that will offer them good performance, that is to have a global load balancing algorithm in place.

There are a number of factors that may be taken into account when developing such a load balancing algorithm. Load and number of connections are common choices when designing an algorithm to run on a Local Areal Network. When examining global load balancing, other factors, such as the relative speed and bandwith between the client and different possible servers come into play. The emphasis of the discussion in this paper will be on enabling clients to connect to servers that minimise network delays. The assumption is that servers are locally load balanced as required to cope with traffic.

Given that the path that traffic takes on the Internet is governed by the BGP, it would seem that this may provide an interesting basis for a global load balanacing algorithm. BGP has information on the best path to any point on the Internet from where the BGP-speaker is connected. As this information is memory-resident in the BGP-speaker any queries should be fast. As it turns out BGP also provides for failure recovery as the protocol is designed to adapt to changing networks. These factors make BGP an attractive choice as the basis for a global load balancing algorithm.

An algorithm which intelligently selects the server a client should connect to is only useful if this information can be made transparently available to clients. For a load balancer to be useful in the context of the Internet it must work seemlessly with existing protocols. The mechanism to communicate information to clients should, ideally, be independent of any particular host, a globally redundant system is most desirable. DNS is a well established protocol that is used by clients on the Internet. DNS also has a measure of redundancy built in, as a domain may have multiple name servers and clients will attempt to find an active name server for a domain before returning an error. These two charactaristics make communicating global load balancing information over DNS an attractive option.

It is also possible to convey load balancing information using HTTP redirects. While using HTTP redirects is only useful in the context of HTTP it is of note that web sites are an application that benefit well from global load balancing. The advantage of using HTTP redirects is that a much finer granularlity may be achieved, per URL as opposed to per host as for DNS.

This paper will examine designing and implementing a global load balancer using BGP as the basis for the underlying algorithm. Results may be communicated by DNS or HTTP redirects and the implemenation has the flexibility to allow other applications to tie into the load balancer.

2 Selecting Servers

An important component of global load balancing traffic is to determine which POP[1] should handle an incoming connection. There are many criteria that can be taken into account including: Relative Capacity of different POPs; Relative Load of different POPs; Latency between POPs and client; Network distance between POPs and client.

It is important that when a new connection is received, allocating the connection to the apropiate POP is done quickly. A long and involved process to determine which POP to allocate a connection to will, at best, cause the connection establishment time for the client to be slow. At worst the client may time-out.

2.1 Load and Capacity

Information on the relative capacity and load of different POPs may be collected and stored locally and used to quickly determine the best site for an incoming connection based on these criteria. However, the internal load of a POP doesn't necessarily correlate to its apropriateness. An underloaded POP, 23 hops away may well provide inferior service to a more loaded POP, 2 hops away. It is certainly not accurate to say that communicating to all POPs is of equal cost for a given client. This is in contrast to load balancing over a LAN where communication to all servers can — reasonably be expected to be equal.

---

[1] POP: Point Of Presance: A host or group of hosts connected to the Internet, usually in a single physical location.

2.2 Network Latency

It would seem that information about the state of the network may provide a valuable means for determining the best POP to handle a client's connection. A readily available network metric is ping-time, a meausre of the round trip time for a packet from one host to another and back again. Unfortunately, ping-times are not a good measure of network performance. It is quite possible for a high latency link to be a high bandwidth link. Satellite and DSL are good examples of this. It is of note that much Internet traffic, including HTTP, SMTP, FTP and streaming media are bandwidth, rather than latency sensitive. The usefulness of ping-times is further compromised if asymetric routing is being used, as the ping-time will not provide information on any differences in the forward and return trip times.

The collection of ping-times is also problematic as having POPs ping a client is inherently slow. It is possible to cache the results but avoiding a delay when a client first connects is difficult. A ping may also be blocked by a packet filter for some reason. If ping packets are being blocked, at best the ping-time data will be unavailable and at worst the ping may time-out, adding to the delay in establishing a connection for the client.

2.3 Network Address Allocations

APNIC[2], ARIN[3] and RIPE[4] are responsible for, amongst other things, allocating IP addresses to network providers and the like. If assumptions are made about the location of different providers then it is possible to construct a loose map of the Internet with some creative use of the `whois` command and trawling of online documents provided by these organisations. This map may be used as a heuristic to determine which POP is closest to a given client.

While simple, this method has many drawbacks. Constucting such a map is only pratcical if POPs are on very distinct networks. Another problem is that static maps are not adaptive to network changes and failures. It is also true to say that such a map may not accurately reflect network topology — geographic locality is no gaurantee of netowrk-wise closeness. Furthermore, to reduce the complexity and size of the map it may be neccessary to make generalisations about networks by aggregating small allocations into one larger allocation — this may reduce the accuracy of the map.

It may be possible to suplement the map by inspecting the hostname. For instance, while a `.com` may be located anywhere, it is reasonable to assume that a `.au`

---

[2] APNIC: Asia Pacific Network Information Centre: `http://www.apnic.net/`

[3] ARIN: American Registry for Internet Numbers: `http://www.arin.net/`

[4] RIPE: Réseaux IP Européens: `http://www.ripe.net/`

is located in Australia. However, doing a reverse lookup on the IP addresses of connecting clients may create a significant performance problem.

Though the approach of manually mapping the Internet is somewhat cumbersome it is simple once set up. No special information or services are required from upstream providers. And the map is static, so lookups should be fast. The map aproach certainly has its merits, but a more dynamic and automated mechanism would be a significant improvement.

2.4 Routing Information

Routing information determines the path that packets will take. This information changes as network topology changes. It would seem that this may provide useful information in determining the best POP for a client. BGP is the protocol that is used to determine routes on the Internet. The following section discusses how BGP works and how it may be used to find the network-wise closest POP for a given client.

3 BGP

The Border Gateway Protocol version 4 (BGP)[9] is a routing protocol, as defined in RFC 1773[20]. BGP is used to communicate routing information between different providers on the Internet and for this reason reflects the path that traffic will take from a given point on the Internet.

3.1 Routing Protocols

A *route* is a a set of addresses and the next hop used to send traffic to the addresses. A *router* is nominally a host that has more than one network interface and makes decisions about to which interface a given packet should be sent. As network topologies become more complex, the number of different routes increases, as does the frequency of routes changing. For this reason it is useful for routers to have a method for dynamically updating routes as the network topology changes, that is, as other routers come and go from the network either because of administrative changes or failures of routers or links between routers.

Routing protocols are a mechanism for routers to communicate routes with each other. Routers that communicate routes with each other are referred to as *peers*. When routes are sent between routers they contain information in addition to the addresses that the route covers and the next hop for this traffic. The additional information may be used to expire the route and to determine the cost of the route relative to other routes. When a router sends such a route it is said to be *advertising*. A route advertisement can be seen as a promise to deliver traffic for a given set of addresses. Advertising routes that cannot be satisfied leads to either *routing loops* or *black-holing*. A routing loop refers to traffic bouncing between routers until the maximum hop count is reached. Black-Holing refers to receiving traffic and then discarding it. In either case the addresses covered by the route is effectively removed from the network.

A *prefix* is a set of network addresses that a given route covers. In routing protocols this is given as either a classful network or in the case of more recently developed routing protocols a CIDR network. Classless Inter-Domain Routing (CIDR) is defined in RFC 1519 [10]. CIDR networks allow networks to be defined as a network address and a netmask, enabling more flexible division of networks than classful routing.

When peers are configured to communicate routes with each other they are said to have a *session* running. When the session is established the routers advertise routes to each other and each router uses this information to determine the best route for each prefix that has been advertised to it or is advertised by it. When a session goes down, either administratively or because of a timeout, the prefixes advertised by the peer in question are removed, enabling the network to adapt to failures.

Routing protocols are divided into two types: Interior Gateway Protocol (IGP) and Exterior Gateway Protocol (EGP). An IGP is concerned with managing routes within a single network, ensuring that each point of the network is able to get to all other points in the network. An EGP communicates information about which addresses are within a network or may be accessed through a network. When BGP is used to communicate routes between different networks on the Internet, it is being used as an EGP.

3.2 Autonomous Systems

When networks communicate routes using BGP, individual networks are identified using an Autonomous System (AS) Number as defined in RFC 1930[11]. Each route communicated using BGP contains an *AS path*, an ordered list of ASes that the route has been advertised by.

As an example suppose that there are three networks, imaginatively named Network A, B and C, as per figure 1. These Networks have the AS numbers 64600, 64601 and 64602 respectively. Networks A and C are each directly connected to B. A *border router* is a router on the edge of a network that communicates directly with routers on other networks. BGP peering sessions are run between border routers in Networks A and B and Networks B and C. There is no direct link

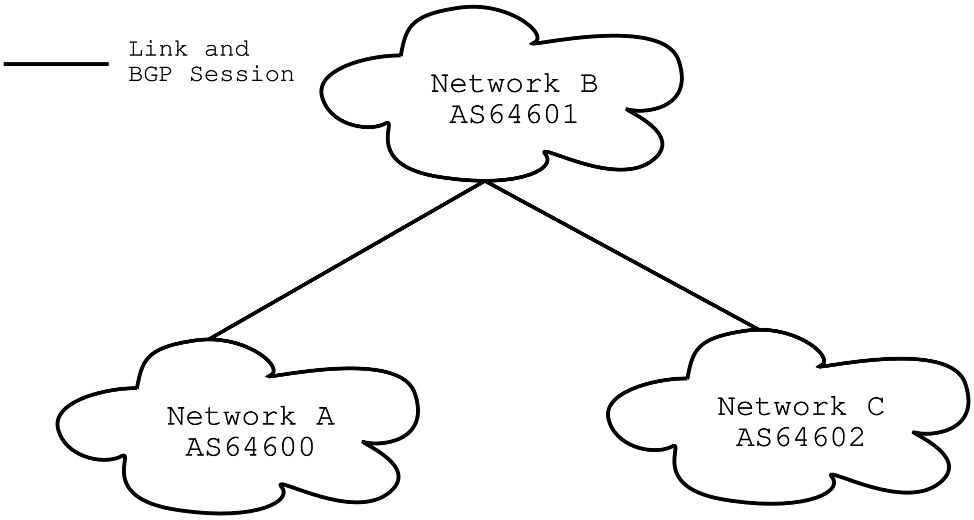

Figure 1: Network Diagram – Transit

between Networks A and C, rather these networks see routes to each other that transit through Network B. This given, the AS path on a router in Network A for a prefix advertised by Network C would be 64601 64602, showing that the route originated from AS64602 and was transited through AS64601. In other words, traffic will travel through Network B to get to its ultimate destination, Network C.

3.3 Finding The Peer Closest to a Client

Now suppose that a provider has two POPs, one on each of Network A and C, called POP X and Y respectively. This is shown in figure 2. By obtaining BGP information from upstream networks and the other POP it is possible for a POP to determine which POP is closest to a given IP address. That IP address could be that of a client wanting to access a service available on both POPs. The result could be used to determine which POP the client should connect to.

To do this each POP has an AS Number, this may be from the range 64512 to 65535, which is reserved for private use by the Internet Assigned Numbers Authority (IANA) as described in RFC 1930. All ASes used in theses examples are from this range. All IP addresses used in examples are from ranges reserved for private use as per RFC 1918[21].

Each POP sets up a BGP session with its upstream network or networks. In this example POP X will have a BGP session with a router in Network A. Similarly for POP Y and Network C. As the POPs will not be originating any valid routes it

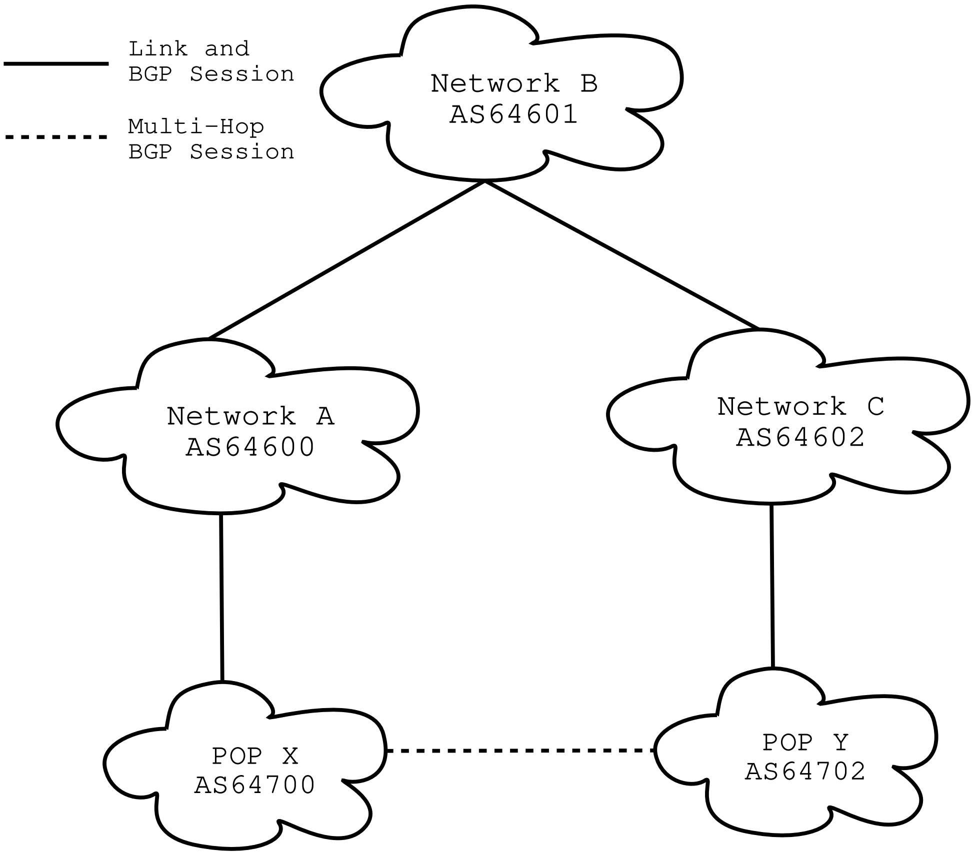

Figure 2: Network Diagram – Points of Presence

is important that the POPs are configured not to send any routes to the upstreams and the upstreams are configured not to accept any routes from the POPs. This is referred to as *filtering*. Once these BGP sessions are established each POP has a view of all the routes that its respective upstream has. This is extendible to POPs with multiple upstreams by the POP in question establishing BGP sessions with each of its upstreams. By establishing a *multi-hop BGP Session* between POPs X and Y it is possible for each POP to see the view of the network that POP has, and in turn the view that POP's upstream has.

If the router running the BGP sessions to Network A from POP X is queried for the prefix used to route traffic to an address in Network C then there are two probable answers; A prefix with the AS path 64600 64601 64602 as learned through the BGP session with Network A, or a prefix with AS path 64702 64602 as learned through the multi-hop BGP session with POP Y. The latter prefix should be preferred as it has a shorter AS path, though it is possible to change this using weights. As the preferred path contains the AS number of POP Y, this must be closer to the the queried address in terms of the BGP routing topology. This means that if the AS number for one of the POPs appears in the AS path for a preferred prefix then the corresponding POP must be closer to the addresses covered by the prefix that the POP making the request. If the AS numbers of multiple POPs appear in the AS path then the last POP in the AS path must be closest, as AS numbers at the end of the AS path are closer to the origin than those at the beginning.

4 Directing Traffic

While it is important to devise a workable algorithm to load balance traffic, it is also important to make this information transparently available to users.

Technologies such as The Linux Virtual Server[6] that implement Layer 4 Switching[5] while very effective for load balancing traffic on a Local Area Network (LAN), do not extend well to load balancing over a Wide Area Network (WAN) such as the Internet. These technologies rely on all in-bound packets and, often, all return packets passing through a single point. While this is acceptable on a LAN where all packets must pass through a limited number of switches and routers, the fundamental problem with using this in the context of a WAN is that having all traffic pass through one site, only to be sent to another, has the potential to significantly increase latency. This also has the potential to reduce reliability as packets are traversing more hops across potentially uncontrolled networks.

A step forward would be to provide a mechanism for connections to be redirected to another site, such that once the redirection has been made clients communicate directly to the site they have been redirected to. It also makes sense to allow any participating site to make this redirection. Thus, no site would be a single point of failure for establishing or maintaining connections for the network presence as a whole. Two ways of achieving this are by using DNS and HTTP redirects.

4.1 DNS

Typically DNS[18][19][16][6] servers are set up to statically map a given query to a reply or list of replies. In the case of a hostname lookup, an IP address or list of IP addresses will be returned. Generally, the result changes infrequently if at

---

[5] Layer 4 Switching: Determining the path of packets based on information available at layer 4 of the OSI 7 layer protocol stack. In the context of the Internet, this implies that the IP address and port are available as is the underlying protocol, TCP/IP or UDP/IP. This is used to effect load balancing by keeping an affinity for a client to a particular server for the duration of a connection.

[6] DNS: Domain Names Service: Maps hostnames to IP addresses and vice versa on the Internet.

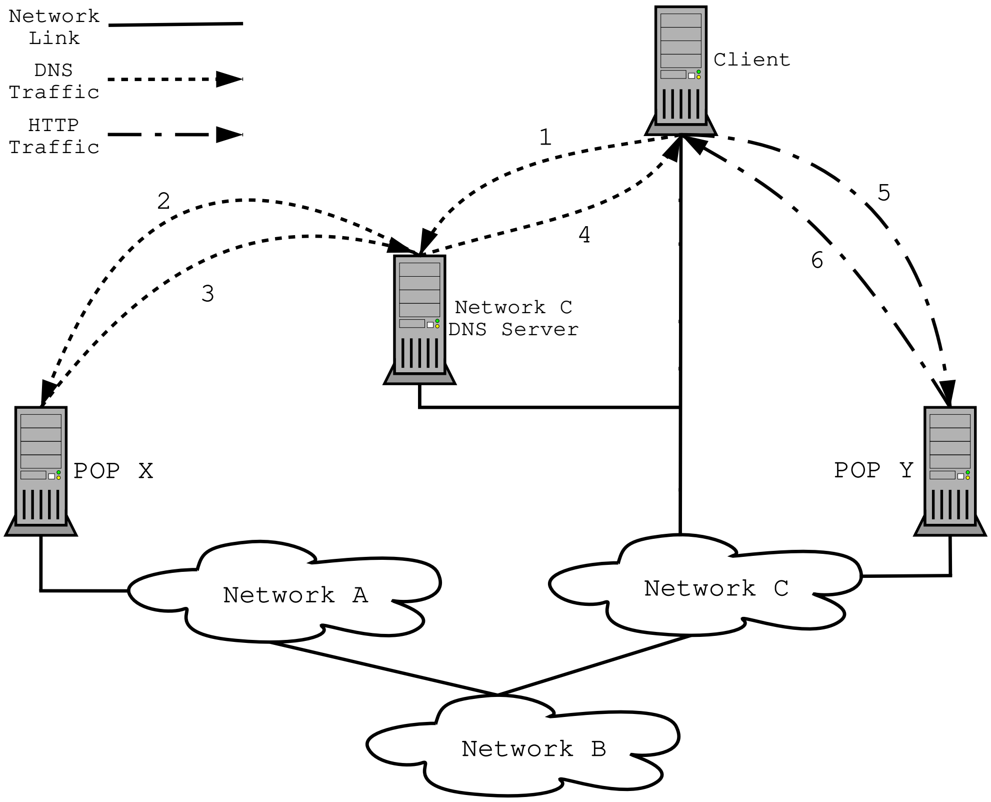

Figure 3: Redirecting Connections Using DNS

all. It is, however, possible to have a DNS server that returns results based on the output of some algorithm. This allows results to be determined dynamically. In this way the results of DNS lookups may be used the communicate the results of a load balancing algorithm to clients. DNS is a fair choice for this application as the DNS protocol is designed with some measure of redundancy. A domain may have multiple DNS servers and if one fails others may handle requests without the client being notified of any problems.

As an example, suppose that *www.slarken.org.au* is mirrored between POP X and Y and DNS is being used to distribute traffic between these two POPs as shown in figure 3.

1. Client Makes DNS Request to local DNS Server in Network C for *www.slarken.org.au*

2. The DNS Server makes a recursive query on behalf of the Client. In doing so it queries POP X for the IP address of *www.slarken.org.au*. Both POP X and

Y are authorative for the *slarken.org.au* domain, the Network C DNS Server happens to query POP X this time around. POP Y would do equally well.

3. The DNS server in POP X is able determine the best POP for a given connection. Note that the best POP for the Network C DNS Server is queried and not the best route to the Client, as the IP address of the Client is not known to the DNS server in POP X. This assumes that Clients use DNS servers that are network-wise close to them. The IP address of the web server or farm in POP Y is returned in response to the DNS query by the Network C DNS Server.

   If POP X had been down then the Network C DNS Server would have queried POP Y and returned the IP address of the web server or farm within itself, thus the result would be the same.

   If POP Y was down then the query to the route server in POP X would have shown that POP X was the closest POP to the Network C DNS Server and the IP address of the web server or farm in POP X would be returned.

   If both POPs were down then there would be no result and the DNS lookup would fail.

4. Network C DNS Server responds to the Client's DNS request with the answer obtained from POP X.

5. The client has the IP address of a server in POP Y as the IP address of www.slarken.org.au and makes an HTTP request to this server.

6. The server responds to the Client's HTTP request.

4.2 HTTP

As an alternative to using DNS to communicate load balancing information to clients HTTP[3][7][7] redirects may be used. The advantage of this is that as the redirection is done by an HTTP server much finer granularity may be achieved. Whereas DNS has a granularity of per-hostname, HTTP may have a per URL[8] granularity. For instance all `.jpeg`, `.jpg` and `.png` URLs may be redirected while all other URL may be handled locally. That is, images are load balanced while HTML pages are handled by a central server.

The key disadvantage of using HTTP redirection is that once a client is redirected to a site there may be no way of directing the client to another site. This may be a problem if all URLs are directed to another site and this site fails. Except by than

---

[7] HTTP: Hypertext Transfer Protocol. Protocol used to transfer data on the World Wide Web

[8] URL: Universal Resource Locator

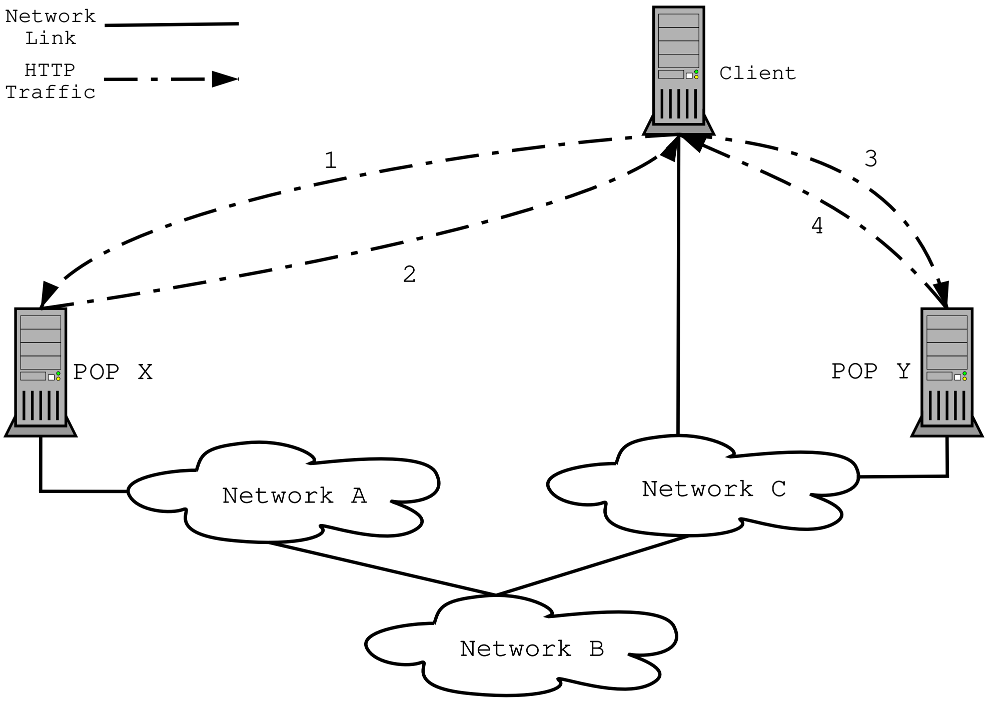

Figure 4: Redirecting Connections Using HTTP

manually going back or reloading the initial URL, the client has no way to access an active site.

Suppose once again that *www.slarken.org.au* is mirrored between POP X and Y and that HTTP redirects are being used to distribute traffic between these two POPs. A sample request may work as follows:

1. Web servers or farms in POP X and Y are both listed as IP addresses for *www.slarken.org.au*. The client happens to send an HTTP request to POP X this time around. POP Y would do equally well.
2. The HTTP server in POP X is able to redirect clients to the network-wise closest POP. Note that the IP address of the Client is used here, whereas when using DNS the IP address of the Client's DNS server is used. The HTTP server then sends an HTTP redirect to a URL that resolves to the IP address of a web server or farm in POP Y, *www.y.slarken.org.au*.

   If POP X had been down then the client would most likely have stalled. A reload would probably have cased an HTTP request to be sent to the other IP address for *www.slarken.org.au* and POP Y would handle the request from there.

If POP Y was down then the query to the route server in POP X would have shown that POP X was the closest POP to the Client and the Apache server would send an HTTP redirect to a URL that resolves to the IP address of a web server or farm in POP X, *www.x.slarken.org.au*.

If both POPs were down then the connection would fail, as would a reload.

3. The Client has *www.y.slarken.org.au*, the URL of a server in POP Y and makes an HTTP request to this server.
4. The server responds to the Client's HTTP request.

5 Implementation: Super Sparrow

Super Sparrow is an implementation of global load balancing written in C[15] and released under the GNU General Public Licence and GNU Lesser General Public Licence[8]. Super Sparrow is available from *http://supersparrow.org/* and is supplied with source and RPM[2][9] packages with comprehensive online documentation. Super Sparrow is able to load balance traffic by using BGP to find the peer closest to a client.

5.1 Route Servers

To avoid the need for yet another BGP implementation, Super Sparrow accesses BGP information by querying route servers. A route server is a router, or host running a routing daemon, that may be queried for the preferred prefix for a IP address. The current implementation supports three different route servers: GNU Zebra[14] and GateD[17], routing daemons that run on a variety of platforms including Linux[10], and Cisco IOS[4], the operating system that runs on Cisco routers and is synonymous with BGP and routing in general.

The current implementation uses telnet access to route servers to query the preferred prefix for an IP address. The tenet interface to route servers is generally intended for manual interaction with the route server but provides a reasonably fast and portable way of other programmes accessing the route server. A sample session with GNU Zebra to determine the preferred prefix for `192.168.193.15` from a route server running on `192.168.192.13` is given in figure 5.

The results shown in the figure indicate that there are two prefixes for the query made and that the second prefix listed is preferred. The AS path for the preferred

---

[9] RPM: Red Hat Package Manager

[10] Linux is a trademark of Linus Torvalds

```
$ telnet 192.168.192.13 bgpd
Trying 192.168.192.13...
Connected to 192.168.192.13.
Escape character is '^]'.

Hello, this is zebra (version 0.89.horms.pre.2)
Copyright 1996-2000 Kunihiro Ishiguro


User Access Verification

Password:
jasmine> sho ip bgp 192.168.193.15
BGP routing table entry for 192.168.193.0/24
Paths: (2 available, best #2, table Default-IP-Routing-Table)
  64600 64601 64602
    192.168.192.12 from 192.168.192.12 (192.168.192.12)
      Origin IGP, metric 1, localpref 100, valid, external
      Last update: Fri Oct  6 15:47:28 2000

  64702
    192.168.193.11 from 192.168.193.11 (192.168.193.11)
      Origin IGP, metric 1, localpref 100, valid, external, best
      Last update: Fri Oct  6 15:44:05 2000


jasmine> exit
Connection closed by foreign host.
```

Figure 5: Determining The Prefered Prefix Using GNU Zebra

prefix is `64702`. It is the AS path of the preferred prefix that Super Sparrow uses to determine if a point of presence in the Super Sparrow network is closer to the address begin queried than the route server being queried as described in section 3.3.

5.2 libsupersparrow

The core functionality of Super Sparrow, including the ability to communicate with route servers is encapsulated as a library, *libsupersparrow*. Breaking this code out into a library allows flexibility to make the algorithm available to a variety of applications.

The library is able to manage connections to multiple route servers and multiple connections to a single route server. In the latter case, connections are cached such that if multiple connections to a route server are requested a single connection will be opened and shared. This avoids the possibility of exceeding the maximum number of connections a route server will accept.

Results read from route servers may be cached to increase performance and avoid placing excessive load on route servers. The implementation of this is quite simple. Results are stored in a self reordering linked list. Queries to the cache search through the list sequentially and if a result is found it is moved to the front of the list. The number elements and the timeout for elements in the cache is configurable. Though simple, the cache yields a significant performance increase, if successive queries are received for the same IP address.

The library also manages the relationship between the AS numbers of POP and their IP addresses or hostnames. This enables the library to determine if the prefered prefix for an address, as returned by a route server, includes the AS number of a peer and if so the IP addresses or hostnames that should be returned accordingly.

Two aplications that link against libsupersparrow have been written: *mod_supersparrow* a driver module for Dents to allow Super Sparrow to be tied to the DNS for a domain, and *supersparrow* a standalone utility that may be used for testing and tying Super Sparrow to aplications that can communicate over standard I/O. The latter may be used in conjunction with Apache's *mod_rewrite* to tie Super Sparrow directly to Apache.

5.3 mod_supersparrow (DNS)

Dents[5] is a modular DNS server that is intended as a drop in replacement for BIND[13].

Dents allows zones to be mounted in the name space much in the same way that UNIX allows partitions to be mounted in a directory structure. Just as different mounted partitions in a directory structure may have different file systems controlled by different portions of code in the kernel, Dents allows different zones types, controlled by driver modules.

Access to a the root name server is analogous to the root (/) directory in a UNIX directory structure. Dents allows this zone to be mounted and resolved using the driver module mod_recursive. BIND for one uses RFC 1035[19] style zone files. This is supported in Dents by mounting a zone using the *mod_stddb* driver module.

One advantage of being able to use different driver modules is that arbitrary modules may be defined. Driver modules that access zone information stored in a relational data base or produce standard mappings from an IP address to a hostname for dialup pools are two example applications.

Super Sparrow implements a Dents driver module, *mod_supersparrow*, that allows Dents to return results based on information from BGP speaking route-servers. In this way the IP address returned for a hostname lookup may be governed by the BGP-based global loadbalancing algorithm implemented by Super Sparrow. Details of how client-server interactions work in such a setup are described in section 4.1.

5.4 supersparrow

supersparrow is a stand-alone application that is linked against lib_supersparrow. The primary intention of this application is to act as a debugging tool during development of lib_supersparrow. As it turns out, supersparrow may be used in conjunction with applications that are able to commumicate with other programmes using standard I/O a useful example of which is Apache's mod_rewrite.

5.5 supersparrow with Apache (HTTP)

One of the most apealing aspects of The Apache HTTP Server[1] is its flexibility afforded to a large extent by its modular architecture. An excellent example of this is *mod_rewrite* which is part of the standard Apache distribition. mod_rewrite allows arbitary rewriting of requests received by Apache to other URLs at run time. The rewrite is done by a map and one of the map types supported is running an external programme.

mod_rewrite communicates with the external programme via standard I/O. The external programme is run once when apache starts, requests are written to the programme's standard in and results are read from the programme's starndard out. The *supersparrow* stand-alone application supports a batch mode, which allows it to be used as a map for mod_rewrite. In this way Apache may be tied directly to Super Sparrow to achieve the semantics described in section 4.2.

6 Conclusion

When implementing Global Load Balancing there is a need to take into account factors that are not apparent when load balancing traffic on a LAN. In particular there is a need to be completely independent of other sites. For this reasons methods such as DNS and HTTP Redirection are attractive for directing clients. This is in direct contrast to local load balancing where methods such as Layer 4 Switching offer superior control of traffic.

The BGP-based algorithm discussed provides a powerful mechanism for determining the network-wise POP for a client. It does not, however, take into account the relative capacity or load of the POPs that traffic is being directed to. The assumption made is that each POP has sufficient capacity to cope with the traffic that it is likely to receive. For this assumption to hold, local load balancing may need to be deployed at each POP. The Super Sparrow implementation has been designed with this in mind and will work with local load balancing technologies such as layer 4 switching technology. In particular, Super Sparrow is designed to work in conjunction with Ultra Monkey[12] which utilises the Linux Virtual Server to effect layer 4 switching.

It is anticipated that in the future the implementation of Super Sparrow will be expanded to allow other, non BGP-based algorithms that may take into account factors such as POP capacity and load. Ideally an algorithm that combines capacity and load information with BGP would provide a very flexible solution.

References

[1] The Apache Group. *Apache User's Guide*, apache 1.3 edition, 1999. http://www.apache.org.

[2] Edward C. Bailey. *Maximum RPM: Taking the Red Hat Package Manager to the Limit*. Red Hat Software, Inc., Research Triangle Park, NC, USA, 1.1 edition, June 1998. http://www.redhat.com/.

[3] T. Berners-Lee, R. Fielding, and H. Frystyk. Rfc 1945: Hypertext transfer protocol – http/1.0, 1996.

[4] Cisco Systems, Inc. *Network Protocols Command Reference, Part 1*, cisco ios 12.0 edition, 2000. http://www.cisco.com/.

[5] Johannes Erdfelt and Todd Lewis. Dents. http://www.dents.org/, 2000.

[6] Wensong Zhang et al. Linux virtual server project, 2000. http://www.linuxvirtualserver.org/.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Rfc 2068: Hypertext transfer protocol – http/1.1, 1997.

[8] Free Software Foundation. Gnu general public licence and lesser general public licence, 1991. http://www.fsf.org/.

[9] Avi Freedman. Bgp routing part i: Bgp and multi-homing, 1997. http://www.netaxs.com/ freedman/bgp/bgp.html.

[10] V. Fuller, T. Li, J. Yu, and K. Varadhan. Rfc 1519: Classless inter-domain routing (cidr): an address assignment and aggregation strategy, 1993.

[11] J. Hawkinson and T. Bates. Rfc 1930: Guidelines for creation, selection, and registration of an autonomous system (as)., 1996.

[12] Simon Horman. Ulttramonkey, 2000. http://ultramonkey.org/.

[13] Internet Software Consortium (ISC). *BIND Online Documentation*, bind version 8 edition, 1998. http://www.isc.org/.

[14] Kunihiro Ishiguro. *GNU Zebra Info*, gnu zebra version 0.88 edition, 2000. http://www.zebra.org/.

[15] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Languate*. Prentice Hall, 2 edition, 1988.

[16] Cricket Liu and Paul Albitz. *DNS and BIND*. O'Reilly and Associates, 3 edition, September 1998.

[17] Merit Gated Consortium. *Configuring Gated*, 1999. http://www.gated.org/.

[18] P. Mockapetris. Rfc 1034: Domain names - concepts and facilities, 1987.

[19] P. Mockapetris. Rfc 1035: Domain names - implementation and specification., 1987.

[20] Y. Rekhter and T Li. Rfc 1773: A border gateway protocol 4 (bgp-4), 1995.

[21] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J de Groot, and E. Lear. Rfc 1918: Address allocation for private internets., 1996.