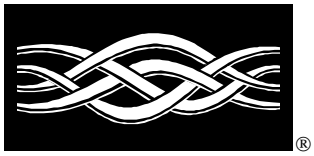

White Paper

Cache Array Routing Protocol and Microsoft Proxy Server 2.0

© 1997 Microsoft Corporation. All rights reserved.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Microsoft trademarks: Microsoft, BackOffice, the BackOffice logo and MSN are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries";

Other products and company names mentioned herin may be the trademarks of their respective owners.

Microsoft Corporation • One Microsoft Way • Redmond, WA 98052-6399 • USA

Microsoft® BackOffice™

Abstract

Microsoft® Proxy Server 2.0 uses the Cache Array Routing Protocol (CARP) to provide seamless scaling and extreme efficiency when using multiple proxy servers arrayed as a single logical cache. CARP uses hash-based routing to provide a deterministic "request resolution path" through an array of proxies. The request resolution path, based upon a hashing of proxy array member identities and uniform resource locators (URLs), means that for any given URL request, the browser or downstream proxy server will know exactly where in the proxy array the information will be stored – whether already cached from a previous request, or making a first Internet hit for delivery and caching.

CARP provides two powerful benefits:

1) Because CARP provides a deterministic request resolution path, there is none of the query messaging between proxy servers that is found with conventional Internet Cache Protocol (ICP) networks, a process that creates a heavier congestion of queries the greater the number of servers.

2) CARP eliminates the duplication of contents that otherwise occurs on an array of proxy servers. With an ICP network, an array of five proxy servers can rapidly evolve into essentially duplicate caches of the most frequently requested URLs. The hash-based routing of CARP keeps this from happening, allowing all five proxy servers to exist as a single logical cache. The result is a faster response to queries and a far more efficient use of server resources.

CONTENTS

INTRODUCTION

Microsoft® Proxy Server 2.0 introduces the Cache Array Routing Protocol (CARP) to greatly expand the scalability and efficiency of proxy servers networked into an array.

To fully appreciate the Microsoft Proxy Server 2.0 solution to caching, it is helpful to consider the growing need for the efficiencies that are gained by caching web sites. The World Wide Web (WWW) has become so popular, and the traffic so heavy, that frustration with response times has led to jokes about the WWW standing for the World Wide Wait.

Proxy servers, originally developed for security as extensions of firewalls, soon proved to have additional value – the speed with which their cached URLs were returned to users. Because it is the most requested URLs that remain stored in cache, proxy servers provide great efficiency:

- Less network traffic. Once an object has been downloaded from the Internet, subsequent users will retrieve that object from the cache instead of having to request the same object across a remote network link.
- Better user performance. Typically Proxy caches reside much closer to an end user than the remote web server. Additionally, while requests across the Internet are serviced at lowest common denominator WAN speeds (typically 64 kbps - 1.5 Mbps), requests to the cache are serviced at LAN speeds (10 Mbps - 100 Mbps).
- Better perceived network reliability. Having cached copies of objects compensates for transient Internet "brownouts" and loss of service due to heavy congestion on remote segments.

Caching proved so efficient that the need was soon seen for deploying multiple proxy servers that could communicate and work together to create a more robust system. In 1995 the Internet Cache Protocol (ICP) was developed to allow Individual proxies to "query" manually configured neighboring proxies in order to find cached copies of requested objects. If all queries failed to find a cached object, the proxy would then use HTTP to request the object from the Internet.

Although ICP allows proxy servers to be networked together, certain problems emerge when using the protocol. These include:

- ICP arrays must conduct queries to determine the location of cached information, an inefficient process that generates extraneous network traffic.
- ICP arrays have "*negative scalability*" in that the more proxy servers added to the array, the more querying required between servers to determine location.
- ICP arrays over a period of time tend to become highly redundant, with each server containing largely the same information – the URLs of the most frequently used sites.

Now Microsoft has significantly enhanced the efficiency of using multiple proxy servers with the introduction of Cache Array Routing Protocol (CARP), a series of algorithms that are applied on top of HTTP. CARP yields several new and interesting benefits to cache users and network operators – without introducing new wire protocols. Microsoft will present the finalized version of the CARP protocol to the Internet Engineering Task Force (IETF) for consideration as an Internet Standard protocol.

Microsoft Proxy Server 2.0, with the power of CARP, allows for "*queryless*" dis-

tributed caching. This is especially important because the vast resources of the Internet, as well as its huge potential for marketing, sales, and other business activities, make the Internet an increasingly essential element for an organization's communications infrastructure. Furthermore, distributed caching helps alleviate network administrator concerns about bottlenecks arising from "push" technologies to client desktops.

The queryless distributed caching allowed by Microsoft Proxy Server 2.0 provides strengths including:

- CARP doesn't conduct queries. Instead it uses hash-based routing to provide a deterministic "request resolution path" through an array of proxies. The result is single-hop resolution. The web browser, such as Microsoft Internet Explorer, or a downstream proxy, will know exactly where each URL would be stored across the array of servers.
- CARP has positive scalability. Due to its hash-based routing, and hence, its freedom from peer-to-peer pinging, CARP becomes faster and more efficient as more proxy servers are added.
- CARP protects proxy server arrays from becoming redundant mirrors of content. This vastly improves the efficiency of the proxy array, allowing all servers to act as a single logical cache.
- CARP automatically adjusts to additions or deletions of servers in the array. The hashed-based routing means that when a server is either taken off line or added, only minimal reassignment of URL cache locations is required.
- CARP provides its efficiencies without requiring a new wire protocol. It simply uses the open standard HTTP. One advantage of this is compatibility with existing firewalls and proxy servers.
- CARP can be implemented on clients using the existing, industry-standard client Proxy Auto-Config file (PAC). This extends the systemic benefits of single hop resolution to clients as well as proxies. By contrast, ICP is only implemented on Proxy servers.

The seamless scalability, freedom from ICP-type querying, protection against redundant caching, client integration, and ability to automatically adjust to server array membership, combine to make Proxy Server 2.0 the platform of choice for bringing the power of proxy server efficiencies into the enterprise.

TAKING A CLOSER LOOK AT QUERYLESS DISTRIBUTED CACHING

The Cache Array Routing Protocol allows Proxy Server 2.0 to be used for queryless distributed caching. The beauty of queryless distributed caching is that as many proxy servers as desired can be joined into an array, without incurring the inefficiency of ICP queries. For example, a single URL request from an ICP-based array of six proxy servers could result in a process of:

- Query being sent to the client's default proxy server.
- If URL not found, default server sends queries to the other five servers.
- No-hit messages are returned from the servers without the URL.
- A hit message confirming presence of URL, if currently cached.
- If a hit message is received, a copy of the cached information is transferred to the client's default proxy server -- a practice which over time can lead to redundant copies across the other servers.

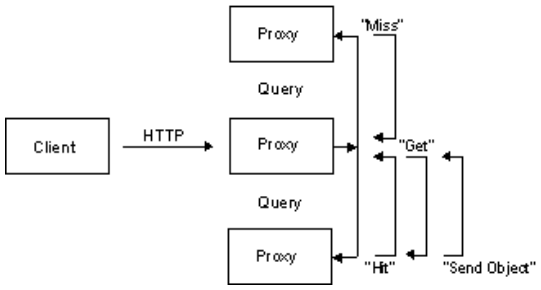

Figure 1. ICP-based arrays generate extraneous traffic because they must query for cache locations.

With CARP, the web browser or downstream proxy server handling the request performs a hash function based upon the "array membership list" and URL to provide the exact cache location of an object, or where it will be cached upon downloading from the Internet – resulting in single-hop resolution.

How CARP Works

A closer look at the routing algorithm is provided later in this paper, but the basic mechanism is:

- All proxy servers are tracked through an "array membership list", which is automatically updated through a time-to-live (TTL) countdown function that regularly checks for active proxy servers.
- A hash function is computed for the name of each proxy server.
- A hash function is computed for the name of each requested URL.
- The hash value of the URL is combined with the hash value for each proxy. Whichever URL+Proxy Server hash comes up with the highest value, becomes "owner" of the information cache.

The result is a deterministic location for all cached information, meaning that the web browser or downstream proxy server can know exactly where a requested URL either already is stored locally, or will be located after caching. Because the hash

functions used to assign values are so great – 2^32 = 4294967296 – the result is a statistically distributed load balancing across the array.

The deterministic request resolution path that CARP provides means that there's no need to maintain massive location tables for cached information. The browser simply runs the same math function across an object to determine where it is .

Improving Client Performance with Distributed Caching

CARP allows for distributed caching, allowing hierarchical proxy arrays, such as might exist with a branch office, to forward requests to the central office proxy array. Distributed caching offers several benefits, including:

- Distribution of server loads, with downstream proxies (such as with a branch office) offloading cache hits from upstream proxies.
- Improvement of client performance by bringing caches closer to client (such as caching information at the workgroup level, rather than at the enterprise level, or making use of a regional internet service provider (ISP) point of presence (POP) rather than a central ISP POP).
- Improvement of cache hit rates due to increase in cache space.

Distributed caching also provides value in environments such as:

- Corporate branch office with Internet connectivity provided by central office.
- Consolidated ISP POPs – with multiple, geographically distributed POPs routed into a central POP which has Internet connectivity
- Corporate proxies operating behind proxy-type firewalls
- Corporations/ISP POPs that are too big to operate with a single proxy server and need additional robustness.

Distributed caching also can be combined with the IPX-capabilities of Winsock Proxy to support mixed network environments and allow pockets of IPX-only clients access to IP-based intranet and Internet sites.

Charting a Request with Distributed Caching

The clean logic of distributed caching can best be seen with a basic flowchart illustrating the decision path for a request-forwarding decision in the distributed caching scheme.

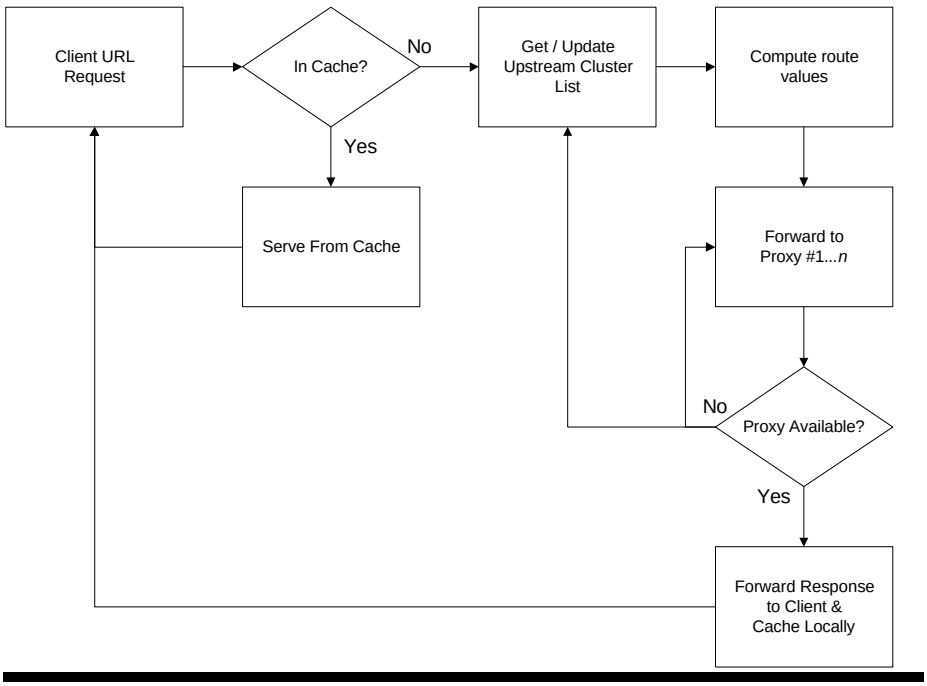

Figure 2. Flowchart for a request-forwarding decision in distributed caching.

Note that "upstream" proxies can be either other proxy arrays or other members of the local array.

Understanding the Routing Algorithm

As noted earlier, the power of the CARP algorithm is that it is deterministic – identical scores will be calculated for identical sets of inputs. Given a list of all the proxies in an upstream array, Proxy Server 2.0 computes a hash of the requested URL, and combines it with a hash of the available upstream proxy names. The result of this combination of a single URL plus *n* proxies results in *n* 'scores'. The proxy with the highest 'score' will be the first proxy to which Proxy Server 2.0 will attempt to forward its requests. If that request fails, Proxy Server 2.0 routes to the next highest score and so on. This provides the advantage of coordinated caching (higher cache hit rates due to sorted requests) without incurring the liability of expensive coordination mechanisms such as ICP.

The following is a simplified, step-by-step representation of how the light-weight on-the-fly routing algorithm works, based upon an array of four proxy servers named Jericho1-4:

(1) Get upstream array membership list and compute hashes on proxy names:

Proxy	Hash
Jericho1	13
Jericho2	8
Jericho3	5
Jericho4	28

(2) Get URL to route upstream, and compute a hash of the URL:

www.microsoft.com
19

(3) Combine hashes. The hash combination algorithm takes into account a *load factor* assigned to each proxy (Proxies with ability to handle more HTTP requests should be routed more traffic):

		www.microsoft.com
Proxy	Hash	19
Jericho1	13	5
Jericho2	8	9
Jericho3	5	7
Jericho4	28	4

(4) Find the highest “score” and forward the URL request to that proxy (in this case Jericho2):

		www.microsoft.com
Proxy	Hash	19
Jericho1	13	5
Jericho2	8	9
Jericho3	5	7
Jericho4	28	4

(5) Compute for route for other URLs (this shows the natural load balancing that occurs as the hash functions result in distribution across the array):

Proxy	Hash	www.microsoft.com	www.yahoo.com	www.msn.com	www.ibm.com
		19	14	5	2
Jericho1	13	5	6	10	4
Jericho2	8	9	2	7	5
Jericho3	5	7	4	3	10
Jericho4	28	4	7	8	1

Incremental Scaling – Easy to Add or Subtract Servers

The CARP algorithm is especially good at accommodating changes in array membership. With other proxy systems, adding or subtracting a server from the proxy array can result in reassigning all cached URLs. With CARP, only a fraction are reassigned – 1/*n*, in which *n* is the number of proxy servers. For example, adding a fifth server would result in about 1/5th of the cache to be reassigned – as URLs are assigned to the server with the next highest hashing score. The effect of this is that proxies can be added and subtracted from a proxy array with minimal cache invalidation of existing proxies' caches.

Here's an illustration of adding another proxy – Jericho5 – to the array:

Proxy	Hash	www.microsoft.com	www.yahoo.com	www.msn.com	www.ibm.com
		19	14	5	2
Jericho1	13	5	6	10	4
Jericho2	8	9	2	7	5
Jericho3	5	7	4	3	10
Jericho4	28	4	7	8	1
Jericho5	14	2	9	4	6

Notice that the only 'highest scoring' route that changes is that of www.yahoo.com, which is routed to Jericho5 instead of Jericho4. More specifically, a change in the table membership from *n* member to *x* members results in a reshuffling of *1/x* of the routes. Because the machine name is hashed, as well as the URL,

the algorithm produces a “route stickiness” which makes additions to and removals from the array a very efficient process.

In addition to easy scalability, this allows for excellent fail-over. If a server fails, its URLs are automatically re-routed to the servers with the next highest scores. In the above example, a failure of “Jericho2” would result in www.microsoft.com being re-routed to Jericho3. Because hash functions are deterministic, all fail-over reassignments are made in a consistent, reliable manner.

ROUTING OPTIONS

The CARP algorithm provides two routing options – hierarchical and distributed. Hierarchical routing involves forwarding requests from a single proxy up to an array of upstream proxies. Distributed routing involves resolving requests received by one member of the array via another member of the same array. Distributed routing extends the benefits of CARP to legacy downstream clients which aren't capable of implementing CARP.

It's important to note that these two routing options aren't mutually exclusive and can be arbitrarily combined. (For example, a request might be first resolved in the distributed manner within an array and then if a cached copy still can't be found, it is forwarded upstream hierarchically). The only real technical differences between these two methods are:

- Whether the request is forwarded to the local or upstream array
- Whether responses from the forwarding are cached by the local proxy

Hierarchical Routing

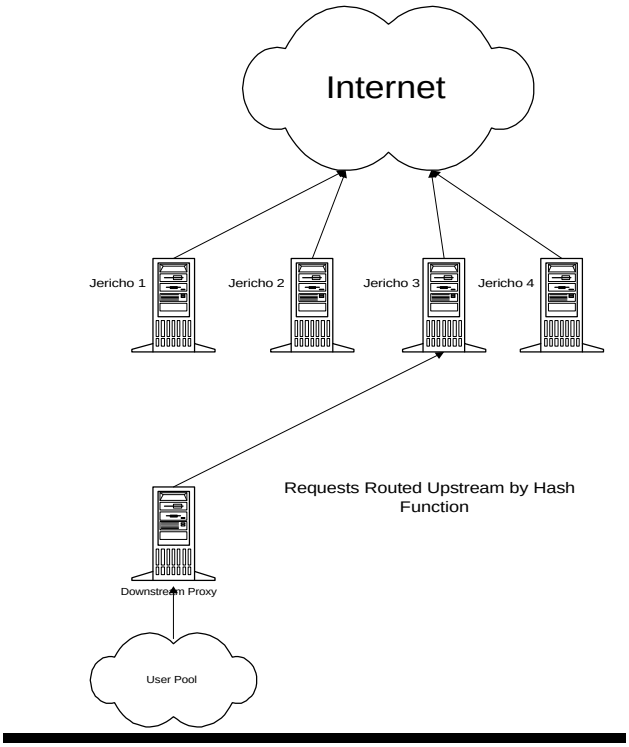

Figure 3. Hierarchical routing, in which requests are forwarded to upstream proxies.

Hierarchical routing involves a downstream proxy that has *n* upstream proxies to which it can forward requests. The downstream proxy uses the array membership list of the upstream proxies and hash-based routing to intelligently determine which upstream proxy to forward the request to.

Because all downstream proxies are constructing their route table with the same inputs, they will all route their requests to the same upstream proxy thereby maximizing potential cache hit rates.

Distributed Routing

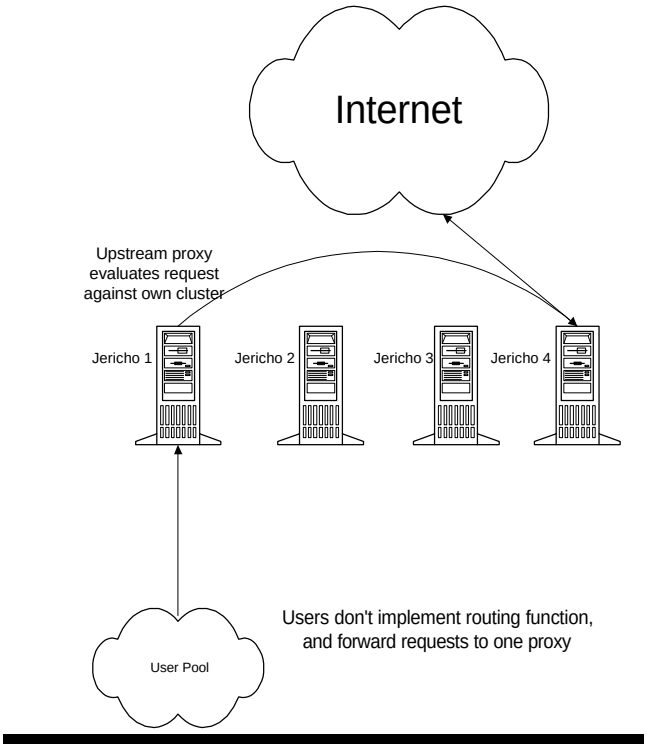

Figure 4. Distributed routing, in which requests are forwarded laterally to the highest scoring proxy.

Distributed routing uses hash-based routing to intelligently process requests within an array of proxies. In this scenario, a proxy with full knowledge of the members of it's own array determines that the request is not ideally processed by himself. Proxy 1 then forwards the request to the 'highest scoring' proxy – in this case Proxy 4. Because Proxy 1 forwarded the request within its own array, he won't cache the returned response since a cacheable response will be held in Proxy 4. This provides maximum efficiency in cache usage, protecting the efficiency of a single coordinated disk cache spread out across all machines.

Combination Routing

Distributed and hierarchical caching can be easily combined. For example, all requests within a workgroup or branch office might first be resolved within its own array of proxies and then forwarded to the enterprise or ISP proxy array as needed. Examining the case of client → local proxy array → ISP proxy array → Internet, there may be up to four routing calculations per request:

1. The client forwards to local proxy #1. That proxy applies the routing algorithm against its own array and determines that local proxy #2 in its own array should handle the request and forwards it.
2. Local_proxy #2 applies the routing algorithm against its array and decides that it is the proper proxy to handle this request.

3. Local_proxy #2 doesn't have the object in its cache, so it applies the routing algorithm against the upstream array and forwards the request to ISP_Proxy #1
4. ISP_Proxy #1 applies the routing algorithm against its array and discovers that its the correct proxy to handle this request. Since it doesn't have it in its cache, it forwards the request to the Internet.

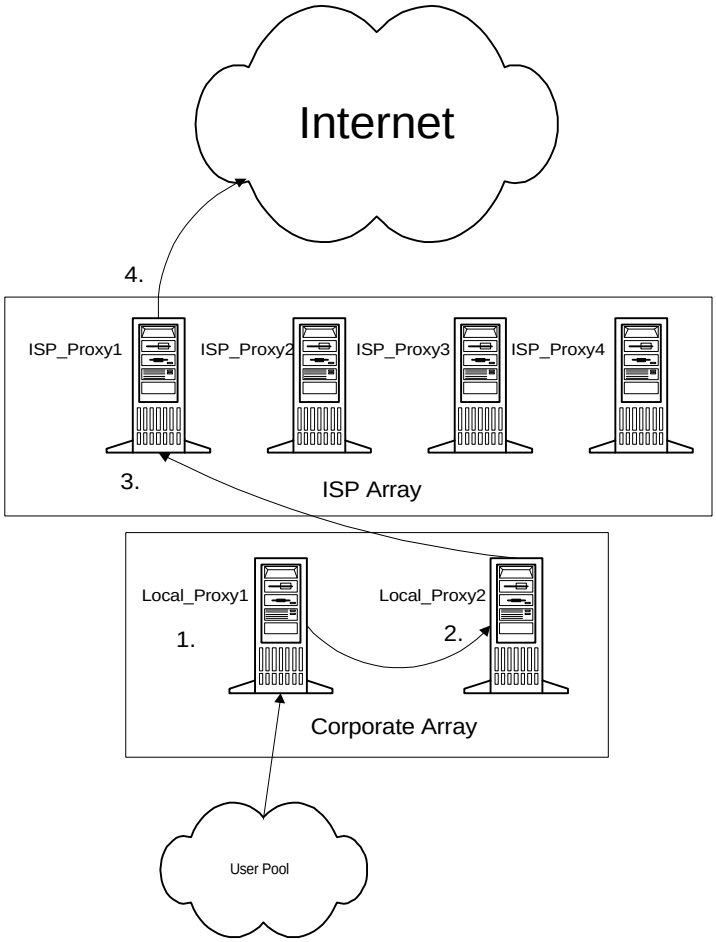

Figure 5. Combination routing uses distributed and hierarchical caching.

Preventing Routing Loops

Routing loops within an array are prevented by the deterministic scoring mechanism. For any given URL, the sequence of proxy preferences within the array is fixed. Since all proxies always forward requests to a higher scoring proxy and never backtrack track to a lower scoring proxy, loops will never occur.

AUTOMATIC UPDATING OF MEMBERSHIP LIST

Proxy Server 2.0 uses an "array manager" to maintain a current list of the members of a particular proxy array, and to make that list available to other systems which request it, such as downstream clients and proxies.

Communications between array managers are handled via HTTP and remote procedure calls (RPC). RPC interfaces are used to handle modifications to the array table -- such as membership, status, and parameters. HTTP is used to publish array information. Publishing via HTTP allows the array table to be consumed by any product supporting the HTTP protocol. The array manager is designed to provide "one-stop shopping" – any one member of the array will have current information about every other member of the array. Therefore, a client need only query one, randomly selected, array member in order to properly route into the array.

The membership lists contains information including:

- The URL that a array manager should call in order to get the array information from a remote manager.
- Load factors to allow a different proportion of the requests for an array to be sent to different machines. Load factors could be a factor if new machines have been added to an array that have larger hard drives, or significantly greater processing power.
- Time-to-Live (TTL) countdown until array members are checked again for status.
- Global parameters such as how often any member of the array table should ask other members for an update in array membership.

Here's a sample array table:

```
Proxy Array Information/1.0
ArrayEnabled: 1
ConfigID: 866749230
ArrayName: Test Cluster
ListTTL: 900

CATNET07 157.55.98.140 80 http://CATNET07:80/array.dll MSProxy/2.0 171 Up 100 3000
CATNET09 157.55.98.140 80 http://CATNET09:80/array.dll MSProxy/2.0 171 Up 100 1500
GSFGROUP 157.55.98.140 80 http://GSFGROUP:80/array.dll MSProxy/2.0 171 Up 100 1500
GENACCTS2 157.55.98.140 80 http://GENACCTS2:80/array.dll MSProxy/2.0 171 Up 100 500
```

Membership Management

Once an array has been created, all members of the array manage their own local copies of the array list. There are two cases for table management: managing an upstream table, and managing a local table.

Upstream Table ManagementA Proxy manages its own "impression" of the upstream tables. Whenever the TTL countdown expires (usually set for several minutes), the proxy queries for a new array table.

Local Table Management In addition to the upstream table management protocol (reloading the table if the TTL has expired), a proxy within an array also watches all HTTP requests to any array member in order to determine the status of

that member. If a request fails, the local proxy marks that proxy member as down in its table for a given TTL period and doesn't forward requests to that member until the TTL expires, and the next table query shows it is active.

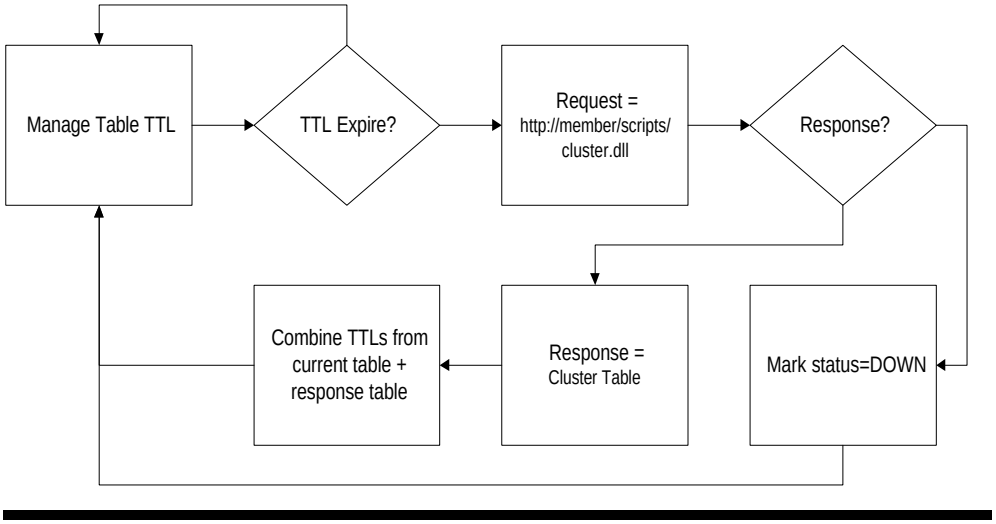

Figure 6. Each array member manages its own array table through queries triggered by TTL countdowns.

SUMMARY

The Cache Array Routing Protocol of Microsoft Proxy Server 2.0 allows organizations to gain much greater efficiencies from their proxy servers. The hash-based routing of CARP, with its deterministic request resolution path, provides single-hop resolution, while creating a single logical cache.

Proxy servers are spared the traffic congestion of ICP queries – a problem which increases which each server added to an ICP array. The efficiency of the array is preserved by avoiding duplication of content that can degrade a five-server ICP array into five independent caches holding much the same content.

Because server identities are hashed, in addition to the URLs, cached information has “stickiness”, meaning that array membership can be increased or decreased while causing minimal reassignment of currently stored information.

All of this results in the ability to deploy Proxy Server arrays that provide built-in load balancing, scalability, fault tolerance, ease of administration, and the efficiency of a single logical cache. And because CARP uses HTTP, it accomplishes this without introducing a new wire protocol.

CARP means faster response to queries, and a far more efficient use of server resources.

FOR MORE INFORMATION

To access information via the World Wide Web, go to http://www.microsoft.com and select Microsoft BackOffice®.

BackOffice info: http://www.microsoft.com/BackOffice
Proxy info: http://www.microsoft.com/Proxy
ICP Working Group: http://www.nlanr.net/Cache/ICP
Hierarchical HTTP Routing Protocol (a predecessor to CARP): http://www.nlanr.net/Cache/ICP/draft-vinod-icp-traffic-dist-00.txt