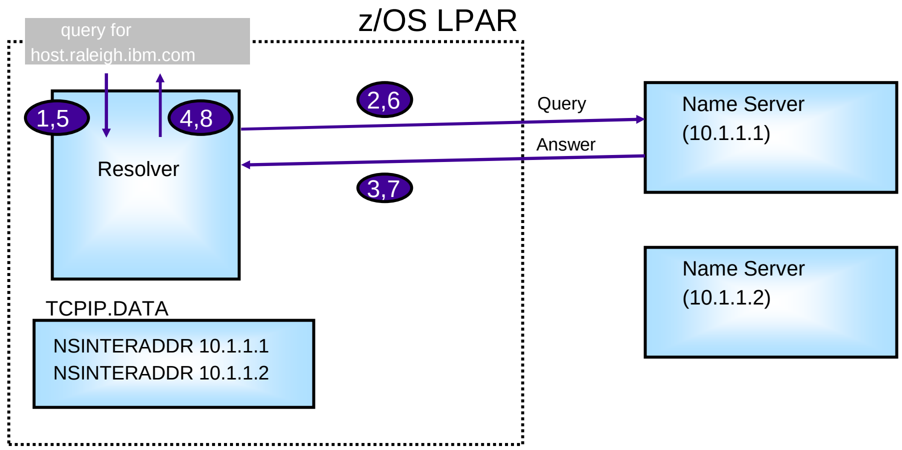
This presentation describes enhancements to the z/OS Resolver to cache DNS responses. This is one of the enhancements in z/OS V1R11 Communications Server for scalability, performance, constraint relief, and acceleration. This theme is a major area of enhancements in z/OS V1R11 Communications Server.

Existing resolver logic always contacts name server

- No memory in resolver of output from previous requests
- Specified DNS name servers contacted on each request



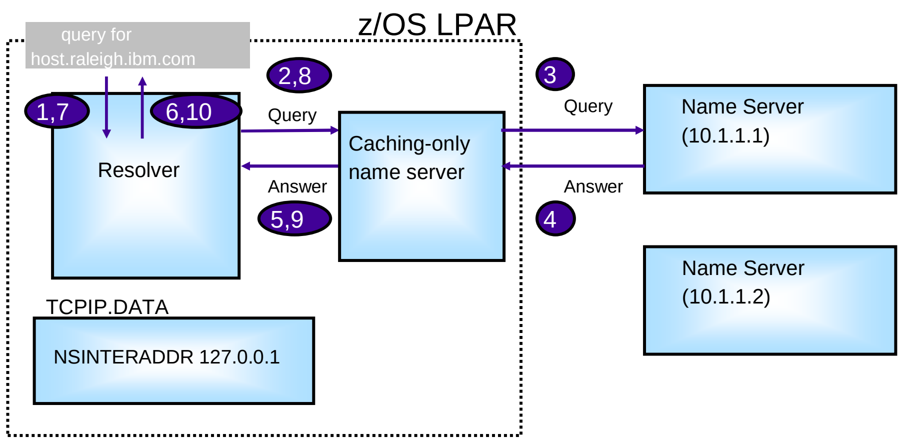
z/OS Resolver processing handles each API request for translation independently. Before V1R11, that meant the results from one request were never used to satisfy later requests, because the results of the first request were never saved. This resulted in queries continually being sent to the name servers specified in the TCPIP.DATA dataset.

Steps one through four in this diagram provide an example of this processing. An application delivers a request to translate the host name host.raleigh.ibm.com into an IP address. The resolver forwards the request to the first DNS name server specified in the list of name servers in the TCPIP.DATA dataset. When the response arrives from the name server, the resolver provides the response data to the application. If the first DNS name server does not respond in time, the resolver forwards the request to the second name server in the list.

What happens when another application, or even the same application, sends a second request to translate host.raleigh.ibm.com? The same sequence of actions occurs, as indicated by steps five through eight. If the first DNS name server does not respond in time to this second request, the resolver again forwards the request to the second name server in the list. If only a limited set of resources can be targets for resolution requests, this repetitive processing can become very costly.

Caching-only name server provided some relief

- Each request directed to local caching-only name server, which retains the information
- Still requires building a DNS request for each resolution attempt



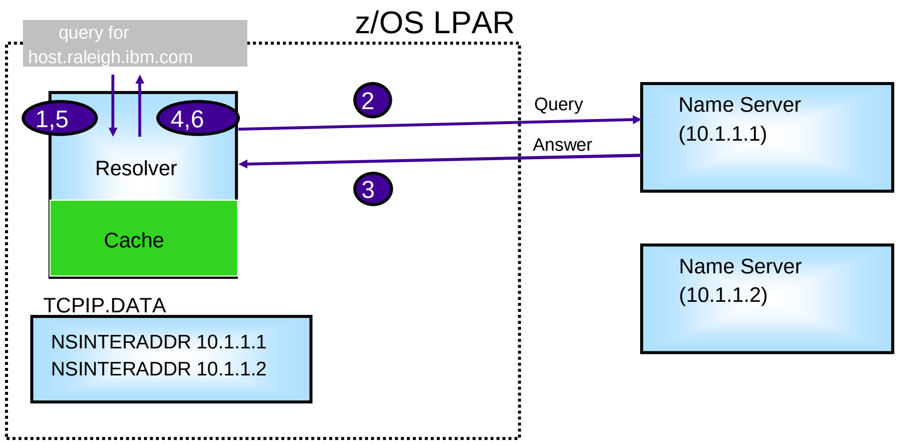
One alternative that provided some improvement was configuring a caching-only name server. Consider the same sequence from the previous slide, but this time with a caching-only name server defined on the z/OS LPAR. Now, in step two, the z/OS resolver forwards the request to the local name server, and the name server communicates with other name servers to translate the host name. The caching-only name server, when it receives the response in step four, first caches the information, and then returns the response to the resolver. The resolver then forwards the response to the requesting application in step six.

Steps seven through ten show the changed processing for a subsequent request for the same resource. When the next request to translate this same host name is received, the resolver again forwards the request to the caching-only name server, as shown in step eight. This time, however, the caching-only name server does not communicate with other name servers, but instead returns the cached data to the resolver. You can see that this reduces the network flows significantly, but it still requires repeated communication with some name server to obtain the cached information.

IBM Software Group Enterprise Networking Solutions

z/OS V1R11 introduces resolver caching

- Resolver cache queried for each request
- Communication with name server only if cache information not available



Page 4

z/OS V1R11 provides the capability for the resolver to now cache information from the name servers, bypassing the need for a caching-only name server. This diagram shows the same query and answer sequence, but this time with the resolver providing the cache functions.

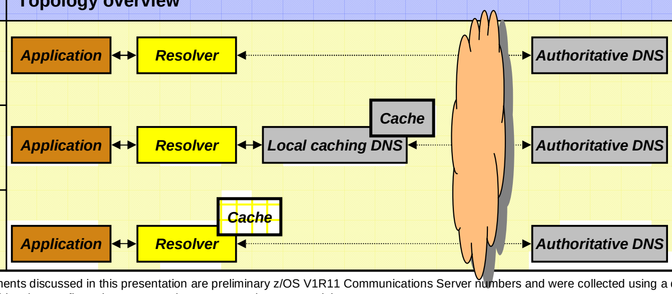
As before, steps one through four require the resolver to build and send a query to the name server in order to retrieve the information about host.raleigh.ibm.com initially. As part of step four, however, the resolver now saves the information into the local resolver cache. When the second request for the host name translation is received, the resolver queries the local cache for data about the host name. In this example, the information is there, and is still valid, so the resolver returns the response data immediately to the application. No query or answer sequence to a name server is required for this second request, providing significant savings in time and processing.

Resolver DNS cache benefits

- The performance benefits of local name caching depend on
  - Amount of calls to the resolver in general
    - Examples are client application workload, Web services workload, and services that do reverse resolution of client IP address
  - Amount of repetitive resolutions of the same host names or addresses
    - The more repetitive resolutions, the more cache hits
  - The time-to-live (TTL) values that are returned by the name server
    - TTL values of zero cannot be cached

Setup	Throughput	CPU
No caching	1	100
Caching-only DNS	4.1	81
Resolver caching	7.7	58

Topology overview



Note: The performance measurements discussed in this presentation are preliminary z/OS V1R11 Communications Server numbers and were collected using a dedicated system environment. The results obtained in other configurations or operating system environments might vary

Caching of name server replies is especially beneficial for environments that generate a high rate of resolver calls, where a high percentage of those calls are repetitive resolutions, and the DNS information does not change very frequently.

Before z/OS V1R11, the only way to provide name serving performance benefits was to configure and run a local name server in caching-only mode. With z/OS V1R11, name server caching is built into the z/OS system resolver.

Some preliminary performance testing of resolver caching has been completed. For the resolver performance runs, all calls were Gethostbyname invocations. One thousand different host names were used for the test, and repetitive resolver calls for those host names were performed. The first query for a particular name obtained information from an external Linux® DNS server, and those results were cached by the z/OS resolver. All subsequent lookups were resolved by the resolver cache. The preliminary performance results shown in the chart indicate that resolver caching provides significant performance improvements. Resolver caching resulted in higher throughput, and less processor use, than the same sequence when caching was done by a locally defined caching-only name server.

Configuring resolver caching … it's all optional!!

- Resolver caching started automatically
  - Turn off using NOCACHE statement
- Use CACHESIZE to adjust maximum storage limits
  - If modifying limit, select value that is 50% larger than anticipated needs
- Use MAXTTL to adjust maximum entry retention time value

```
F RESOLVER,DISPLAY

EZZ9298I DEFAULTTCPIPDATA - None
EZZ9298I GLOBALTCPIPDATA - SYS1.TCPPARMS(TCPDATA)
EZZ9298I DEFAULTIPNODES - USER1.ETC.IPNODES
EZZ9298I GLOBALIPNODES - None
EZZ9304I COMMONSEARCH
EZZ9304I CACHE
EZZ9298I CACHESIZE - 200M
EZZ9298I MAXTTL – 214748364
EZZ9293I DISPLAY COMMAND PROCESSED
```

MAXTTL defaults to name server supplied value

CACHESIZE defaults to 200M of storage, about 80K entries

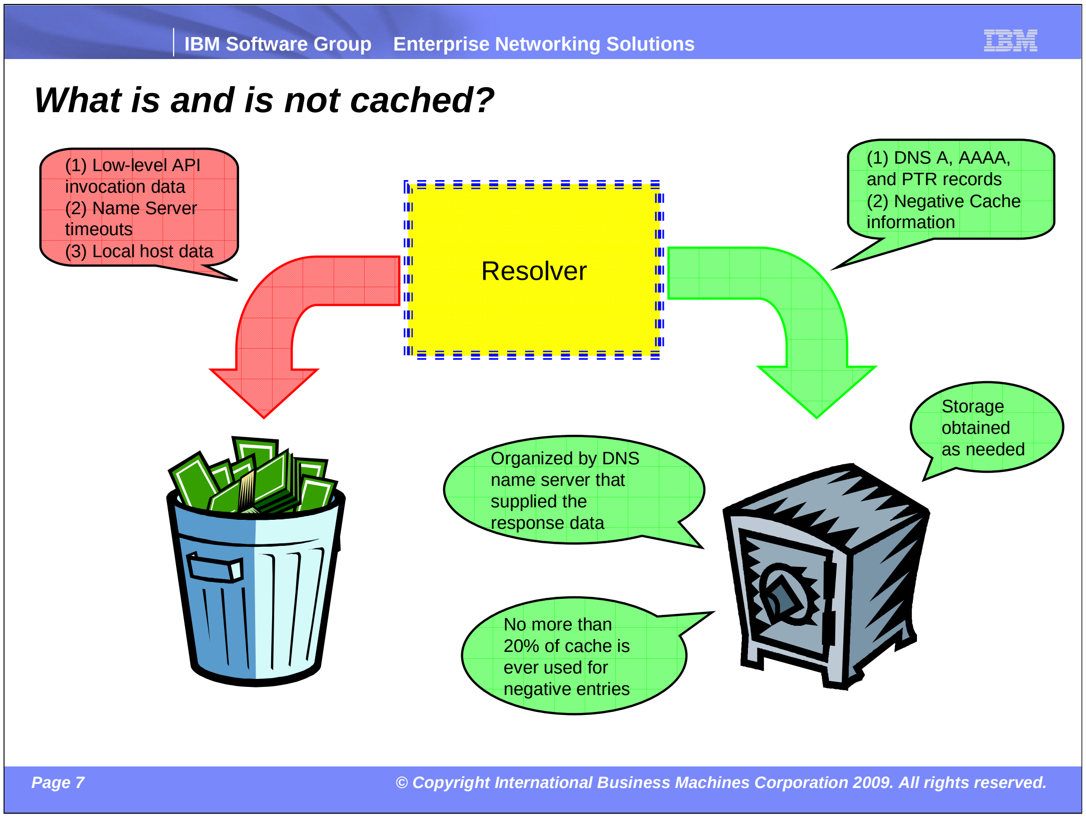
You do not have to make any configuration changes to use the Resolver DNS Cache function. It is started by default when the resolver is activated.

There are two additional optional configuration statements you can use to tailor the caching function. The first statement is CACHESIZE, which specifies the maximum amount of storage that can be used by the resolver to maintain cache information. The default is 200 megabytes of storage, which translates into roughly 80,000 cache entries, or approximately 400 entries per one megabyte of storage. If you choose to modify the value of CACHESIZE, first estimate the number of entries that you expect to have in the cache. After converting that number to a value in megabytes, add an additional 50% to the size to allow for more efficient resolver use of storage.

The second configuration statement is MAXTTL, which specifies the maximum duration of time, in seconds, that a cache entry can remain usable. The default is the largest value that can be returned by a DNS name server.

You can display the current resolver setup values at any time using the MODIFY RESOLVER,DISPLAY command, as shown here.

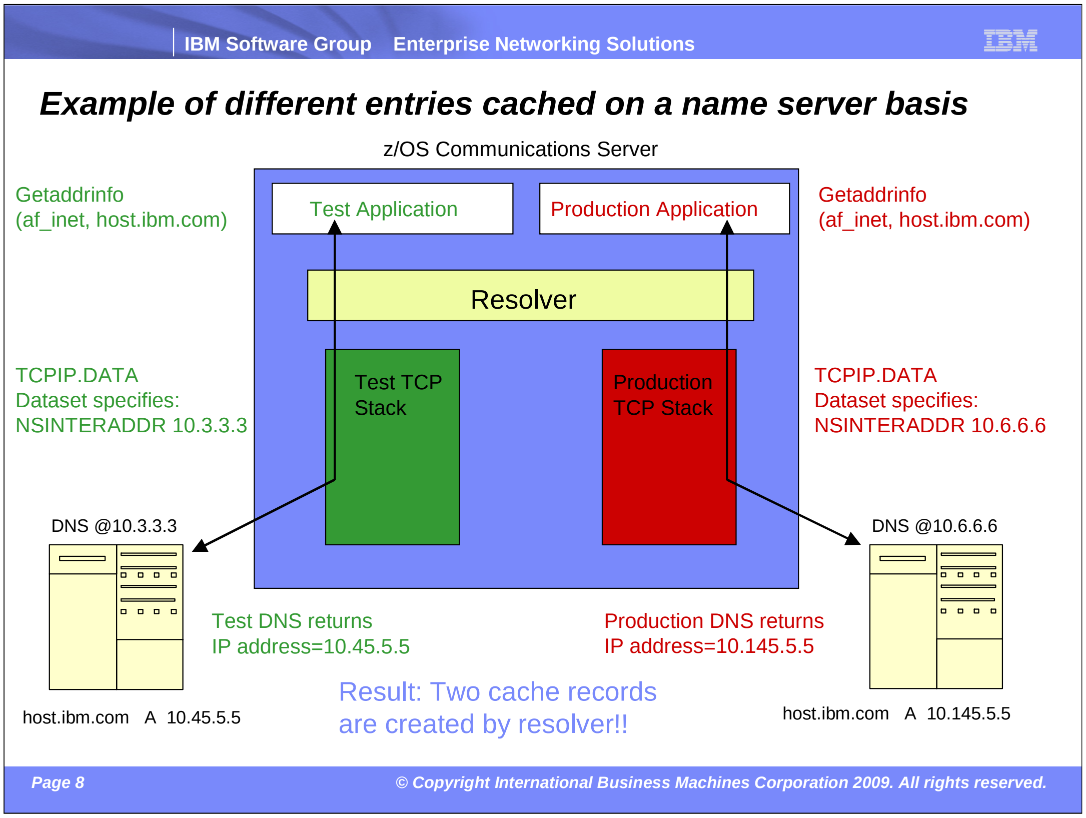
You can, if you so choose, disable the function using the NOCACHE statement. Caching can be disabled either on a system-wide basis or only for a subset of users on your system.



The resolver does not cache all DNS name server response data. Only information obtained using Getaddrinfo, Gethostbyname, Getnameinfo, or Gethostbyaddr resolver API calls is saved. This includes both DNS queries that return valid response data and DNS queries that return “negative cache” responses. “Negative cache” responses are responses that definitively indicate that the target resource does not exist. DNS information obtained using other resolver API calls is not saved, nor is information saved that is obtained from local host files. Also, the resolver does not save any information regarding name server time-out conditions.

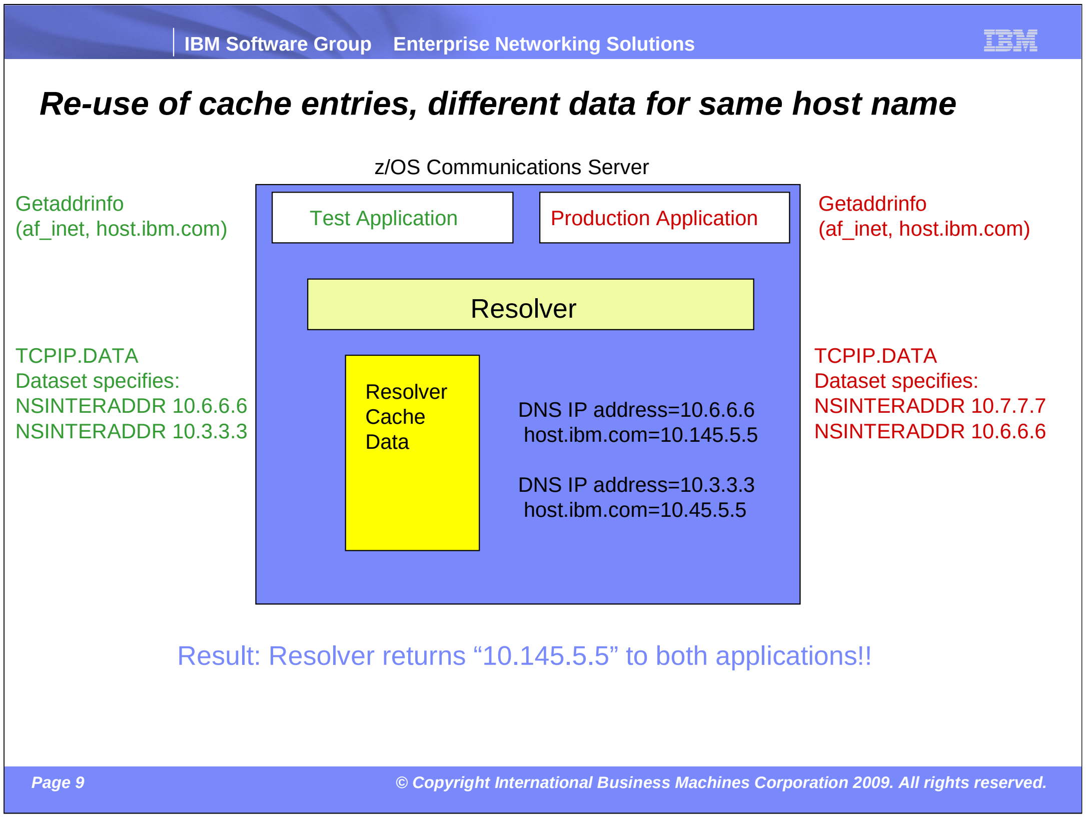
The resolver only allocates storage for entries on an “as-needed” basis, and only 20% of the maximum storage available (as specified by CACHESIZE) can be used for saving “negative cache” information. This limits the resolver’s exposure to “denial of service” attacks that might be designed to fill the cache with unusual entries.

The information is organized according to the IP address of the DNS name server that provided the cache information. The next two slides elaborate on this topic.



Consider an installation that has two applications: a test application and a production application. The two applications have different TCPIP.DATA datasets. Within the TCPIP.DATA dataset is the NSINTERADDR list of name servers to be used for the application. Each application has a different name server defined as the primary name server to be queried.

Assume that each application issues a resolver API call for the same resource, which in this example is a Getaddrinfo API call to resolve “host.ibm.com”. As indicated by the NSINTERADDR values, the resolver directs the requests to different name servers. In order to segregate the test environment from the production environment, the test name server returns a different IP address for “host.ibm.com” than the production name server returns. When caching the responses, the resolver creates two different cache records. Each cache record is associated with the IP address of the name server providing the information. This is true even if the two name servers provided the exact same response information. This allows the resolver, on subsequent requests for “host.ibm.com”, to return the correct resource information based on the name servers to be contacted to process the request.



Continuing with the previous example, what happens if the application TCPIP.DATA datasets are modified to include different NSINTERADDR statements? In the case of the test application, the IP address of the production name server has been inserted at the front of the list. In the case of the production application, an IP address of a new name server has been inserted as the first address. How does this affect what is returned?

When caching is active, before contacting any DNS name server, the resolver first checks the contents of the cache database. For the test application request, the resolver starts by examining the set of cache information provided by the first DNS name server in the list, which is now the production name server. Since there is information cached from this name server about the target resource, that information is used. This means that the test application is now given 10.145.5.5, instead of the address 10.45.5.5 that had been returned previously. This illustrates the importance of the order of the name servers in the NSINTERADDR list.

What if the production application issued the request? In this case, the resolver first examines the set of cached DNS information provided by the newly defined name server, but there is no information from that name server in the cache. As part of the same resolver cache query, the set of cached DNS information provided by the production name server is examined next. There is cached information from this name server, so that cache data is used. This is an example of how the entire list of name servers is examined for cached information before any searches to any name servers are attempted.

Using the cached information

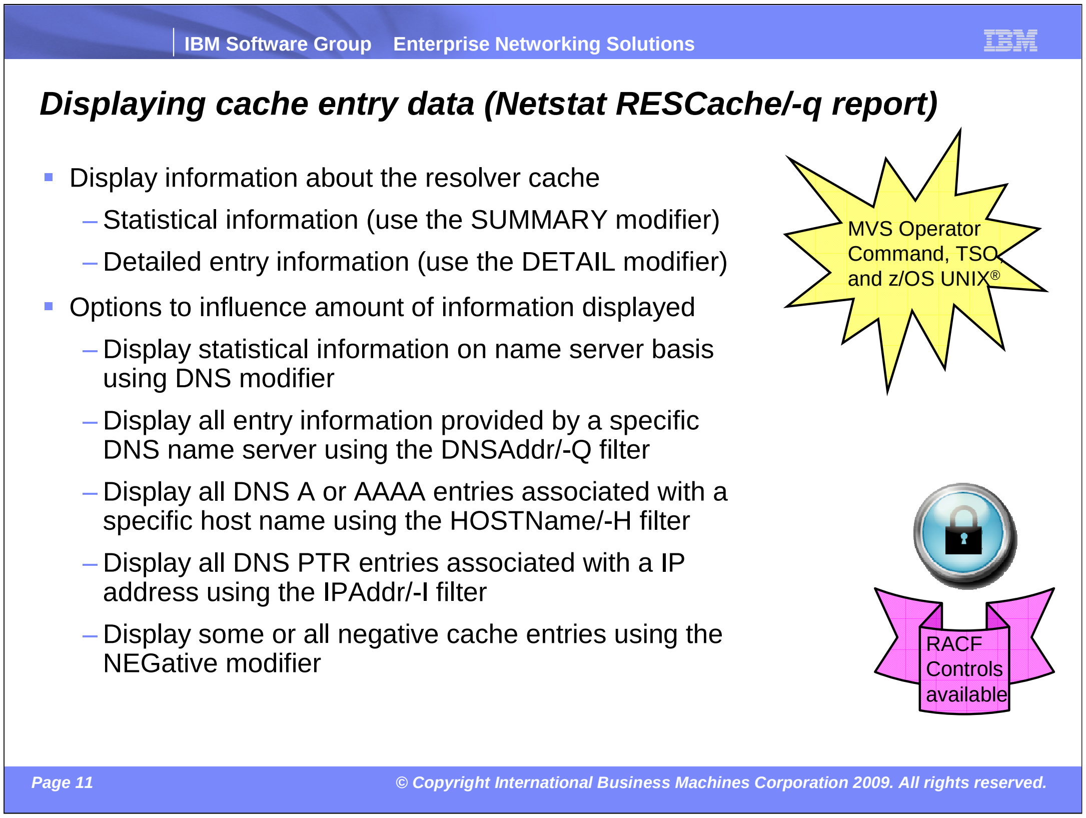
- No change to the resolver API's
- Data saved independent of API used to acquire cache entry
  - Data cached by Getaddrinfo can be retrieved using Gethostbyname, and vice versa
  - Data cached by Getnameinfo can be retrieved using Gethostbyaddr, and vice versa
  - Usable by both EBCDIC and ASCII applications

- No "round robin" algorithm applied to cached data before delivery to application
  - Sorted by Getaddrinfo automatically
  - SORTLIST directive applies to IPv4 addresses

The resolver caching function does not impact the data that is presented to the application across the resolver APIs. The same control block structures are used for returning the information. Applications invoking the resolver should not detect any difference between data supplied from the cache and data that had to be retrieved from a name server.

Furthermore, the cache function is designed to allow resource information to be re-used by compatible API calls. For instance, if Getaddrinfo is used to obtain IPv4 addresses for a host name, that same cached information can be retrieved later using Gethostbyname. The same capability exists for Getnameinfo and Gethostbyaddr calls in terms of host names obtained from an IPv4 address. IPv6 processing is only available using Getaddrinfo and Getnameinfo, so IPv6 information cannot be shared in this manner. In addition, the resolver translates the cache information from EBCDIC to ASCII, or vice versa, so cached information is available using either protocol.

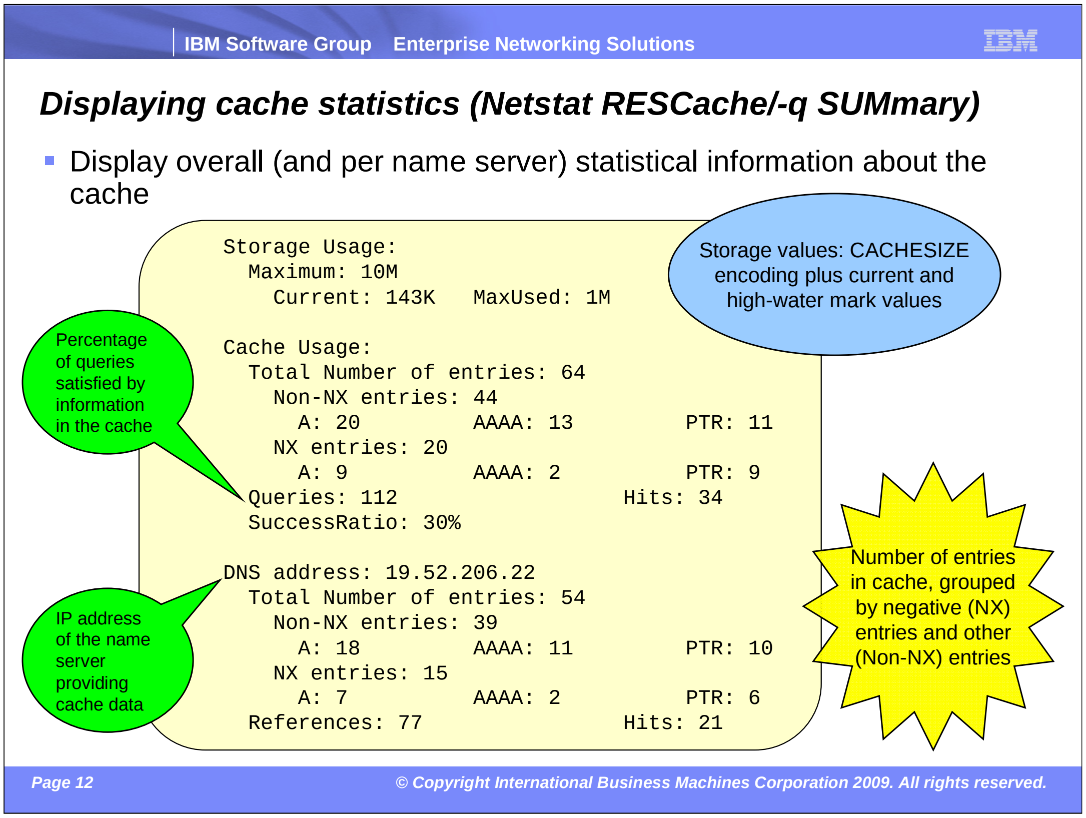
One function not provided by resolver caching is the ability to return the cached IP addresses in a different, or "round robin", order than they were received from the name server. The resolver returns the addresses in the same order all the time. It should be noted that Getaddrinfo processing sorts the list of addresses already, eliminating some advantages of a round robin approach. Similarly, if you have SORTLIST definition statements coded, the list of addresses are re-ordered into a more predictable pattern.



A new Netstat report, RESCACHE, is available for displaying information regarding the resolver cache. Two main types of information can be displayed: statistical information, and actual resource information. The next slides show sample displays of both types of cache information.

The Netstat RESCACHE report is available in the TSO, z/OS UNIX, and MVS operator command environments. The RESCACHE report is no different from other Netstat reports in terms of RACF® requirements.

You can specify additional modifiers or filters to influence the amount of cache data that is displayed. For statistical information, you can add the DNS modifier to have the overall statistics broken into statistical information on a name server IP address basis. You have even more filtering options when displaying detailed resource information. You can filter the information by the IP address of the name server that provided the information. You can filter the information so that only entries related to a specific host name value, or specific IP address value, are displayed. You can display only negative cache information, either all entries or subsets of entries based on name server IP address, host name value, or IP address value.

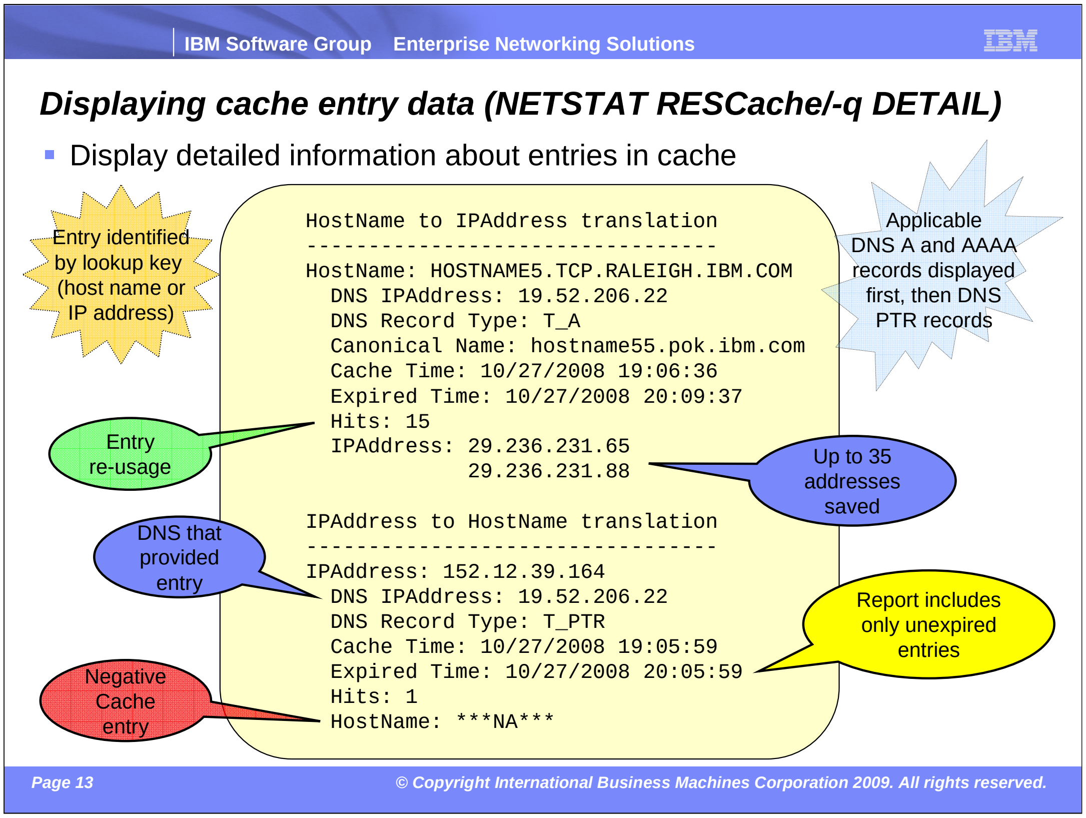


This is an example of a partial Netstat report showing cache statistical information.

Three components of storage usage information are displayed. One is the maximum amount of storage permitted, or CACHESIZE. Another is the current amount of storage in use. The last value is the maximum amount of storage the resolver has used for caching since the resolver was started.

Cache usage statistics include the total number of entries in the cache and the volume of cache activity. The number of entries is differentiated between negative cache entries and non-negative cache entries. Within each of these main categories, the number of DNS A, AAAA, and PTR records is indicated. These same subsets of entries are displayed for individual name servers.

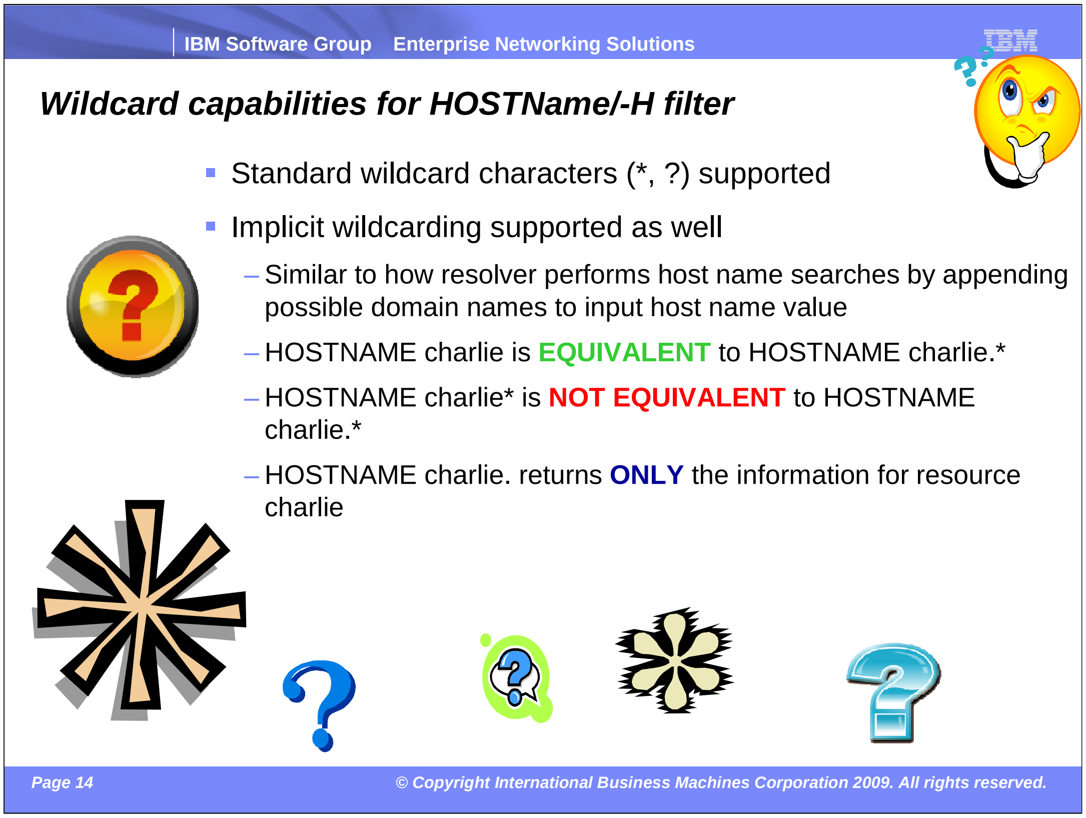
The number of resolver cache requests and how often usable data was returned by the cache gives you a sense of the efficiency of your cache operations. Note that a single resolver API call can generate multiple cache queries. For instance, a Getaddrinfo request for both IPv6 and IPv4 addresses generates two cache queries. On an individual name server level, the “References” value indicates the number of times the set of cache information provided by this name server was examined. Typically, the sum of the name server “References” values is greater than the total number of cache queries, since multiple sets of name server information can be examined as part of one cache query.



This is a partial example of a Netstat report showing detailed cache entry information. The reports are formatted such that DNS A and AAAA records are displayed as one group, and DNS PTR records are displayed as a second group. Negative cache entries can appear in either group, in any order.

For each record, the cache entry key, or the target resource that was searched for, is the first line of the entry. After that, the two types of entries are very similar. The IP address of the DNS name server that supplied this particular information is displayed, allowing you to see which values were provided by which name servers. In the case of DNS A and AAAA record entries, the host name used to create the record might really be an alias or nickname for the official name of the resource. For that reason, the display includes the official, or canonical, name, regardless of whether the names match or not. There is no canonical name concept for DNS PTR records.

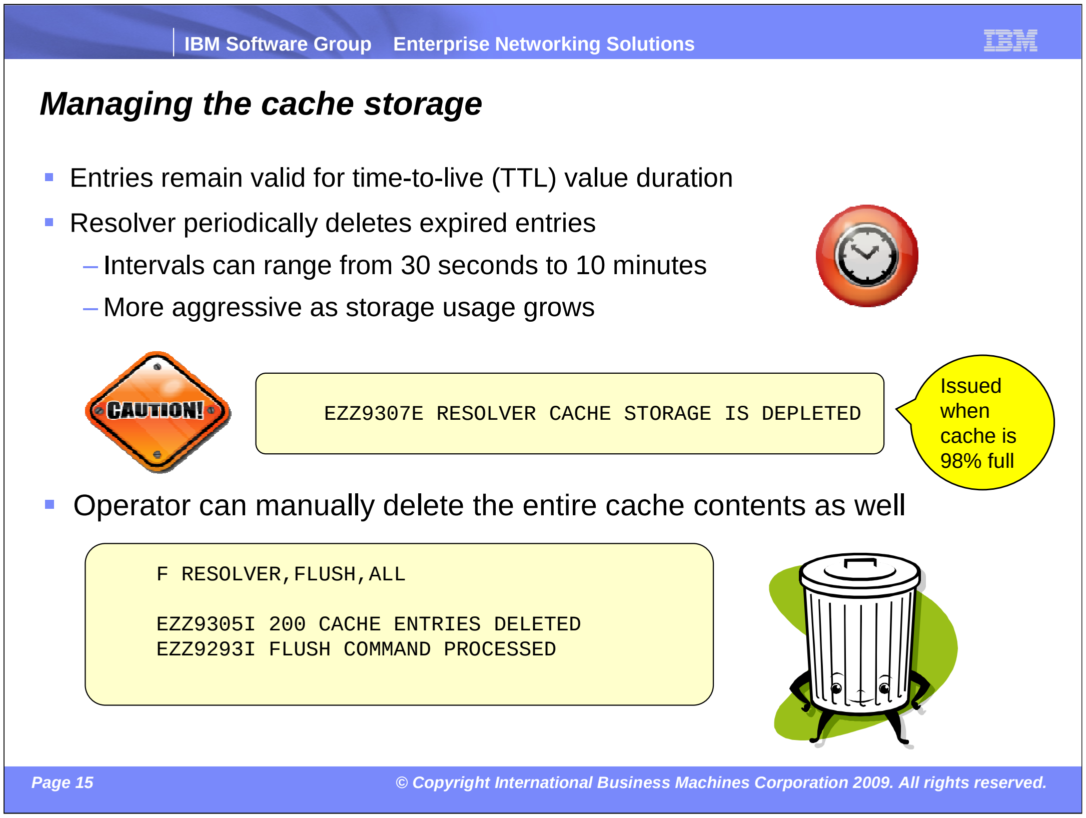
Two time values are displayed: one is the time and the date of cache entry creation. The other is the time and date when the entry expires, based on the name server supplied TTL or the MAXTTL setting. The Netstat RESCACHE report includes only resources that are in the cache which do not represent expired information. The number of times this entry has been re-used is displayed as the “Hits” value. Finally, for DNS A and AAAA entries, up to 35 IP addresses provided by the specified name server for the host name value are displayed. For DNS PTR entries, the one host name associated with the input IP address (either IPv4 or IPv6) is included.



As discussed previously, you can use the HOSTNAME filter to display all the cache entries created due to translation of a specific host name. HOSTNAME is not a new filter on the Netstat command, but some new capabilities were added when using HOSTNAME on the Netstat RESCACHE report. In particular, the standard wildcard characters asterisk and question mark can be used on the HOSTNAME filter for the RESCACHE report.

In addition to the standard wildcards, the value specified as the HOSTNAME filter is treated by the resolver as an “implicit wildcard” value. For example, if you specify **charlie** as the value for the HOSTNAME filter, it is treated as if you had specified **charlie.*** as the value. In both cases, host names like charlie.ibm.com and charlie.raleigh.ibm.com are displayed in the report, in addition to just the host name charlie. However, if you specify **charlie*** as the filter, that is not the same as if you had specified **charlie.*** as the filterv. In this case, host names such as charlie01.ibm.com match the first specification (**charlie***), but not the second value (**charlie.***).

If you place a period at the end of the host name value, that instructs the resolver to return information only for that name, without any trailing domain name information. Thus, in this final example, only records associated with the name **charlie** are displayed. You might still see multiple records in the output, if multiple name servers had provided information about **charlie**.



The resolver does not maintain the name server response information indefinitely. The name server provides a “time-to-live” (TTL) value for the resource, which indicates how long the information can be considered to be accurate. The resolver uses this TTL value to define an expiration time for the cache record. After this expiration time, the resolver no longer uses the information to satisfy application requests. You can set an upper limit on the TTL value using the MAXTTL configuration statement, as previously discussed.

Even though the resolver no longer uses the information, the record is not immediately deleted when the expiration time is reached. The resolver deletes the record as part of periodic cleanup processing. This periodic cleanup of the cache can occur anywhere in a range from 30 seconds to 10 minutes, based on the amount of storage currently being used to hold cache data. If the percentage of the maximum cache storage, as defined by CACHESIZE, reaches 98% capacity, the operator is alerted. Additional cache entries are created as needed until 99% of the cache maximum is reached. No new cache entries are created when capacity is 99% or higher.

A new FLUSH,ALL option on the MODIFY RESOLVER command is available for deleting all records from the resolver cache. The operator can use this command in response to the new cache depleted message as one pro-active way to handle the storage situation.

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

RACF z/OS

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.