

CSS 11000 Caching Configuration and Compatibility Considerations

Document ID: 12574

Introduction

Prerequisites

Requirements

Components Used

Conventions

Cache Configuration

Layer 2 Switch Versus Flow Switch

Routing

Services

Layer4 Versus Layer 5

Asymmetry

Balance

Failover

EQL Bypass

Cache Bypass

URL Parameters Bypass Feature

Pre-Fetching Feature

Related Information

Introduction

This document is a guideline for determining caching and compatibility considerations when implementing Cisco Content Services Switches (CSS) 11000/11500.

Prerequisites

Requirements

Readers of this document should be knowledgeable of the following:

- The cache's default gateway must be the CSS.

Components Used

This information applies to all Cisco CSS (11000 & 11500) running WebNS Software Release 3.0x or higher.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, make sure that you understand the potential impact of any command.

Conventions

For more information on document conventions, see the Cisco Technical Tips Conventions.

Cache Configuration

Layer 2 Switch Versus Flow Switch

The CSS is a flow switch, as opposed to a Layer 2 (L2) switch. A flow switch maps flows to ports based on IP information, as opposed to MAC layer information as in a L2 switch. The flow switch functionality is what allows the CSS to be content smart and look at the TCP synchronize/start (SYN) packet or HTTP GET and make content decisions, as opposed to just forwarding frames. In caching environments, the flow switch requires that you route all of the packets coming from the cache destined to the clients.

Routing

Since the CSS must be the default gateway for the cache, and route all of the packets from the cache, the CSS must have a complete routing table.

Services

When you configure a service as type transparent-cache, the CSS rewrites the destination MAC address with the cache's MAC and forwards the IP packets to the cache in order to preserve the destination IP address. If the cache cannot listen promiscuously for all port 80 traffic, then you must configure the cache as type proxy-cache; however, proxy-cache services cannot use the failover method of bypass. Since you MAC forward the packets to the cache, the cache should be directly connected to the box. If this cannot be accomplished, then it must be connected through a L2 device that cannot be shared with the client or Internet connection.

When a service is configured as either proxy-cache or transparent-cache, there is an automatic invisible access list (ACL) that is created that lets the source IP of the cache bypass all content rules so that it can go get pages from the origin server. Starting in WebNS version 4.0, you can enter the **no cache-bypass** command for transparent-cache and proxy-cache services in order to disable the automatic bypass of content rules.

You cannot have two live CSS, each with one cache directly attached, load balance the two caches when configured as type transparent-cache, unless each cache has a connection to each switch directly. One switch will get the request from the client and then maybe choose to serve the request to the cache connected to the other switch, which would then see the request come in and match it against its rule and perhaps decide to serve it to the cache connected to the first switch. Since these packets are MAC forwarded, there is no way to construct an ACL to do this correctly.

Layer4 Versus Layer 5

A rule is considered to be Layer 5 (L5) when the information required to match the rule or make the load balancing decision is not in the TCP SYN, but is in the HTTP GET; therefore, when you add a URL, a balance method of domain, domain hash, or URL hash, the rule is automatically L5 and induces spoofing. Spoofing is an advantage even when not a requirement because it protects the cache against SYN flood attacks. A L5 configuration allows for both the Extension Qualifier List (EQL) and failover bypass, which causes the box to spoof a connection with the origin server, as opposed to the cache. When a content rule is considered L5, this means that the CSS spoofs the connections with the clients.

Asymmetry

If there is a route back to the clients from the origin server that does not pass through the CSS, then you have asymmetry, otherwise known as triangulation. If you are using L5 rules and there is a bypass, the box spoofs a

connection with the origin server, and the return path goes directly to the client, where it is then very quickly dropped since the client does not recognize the sequence number. You need to either fix your asymmetry, send everything to the cache and use failover linear, or use Network Address Translation (NAT) peering, by NATing your clients IPs with source groups.

Balance

It is recommended that you use the balance method of domain hash and URL hash. Domain hash does a hash across the entire host tag and then determines which server should services that domain, allowing for an evenly distributed load. The URL hash balance method works similarly to the domain hash method, but uses the URL as the data source as opposed to the host tag.

When using a service type proxy-cache, we balance based on the first GET of a persistent connection, and as long as the second GET matches the same rule, we do not rebalance, which could cause for some sites to be duplicated across the caches (refer to the pre-fetching section below). In WebNS 4.0, you can configure the CSS to break the persistent connection and rebalance by issuing the **no persistent** command on the content rule. Whenever you issue the **no persistent** command, you want to use the WebNS 4.0 **persistence reset remap** command as well. The **persistence reset** command is used to determine how persistent connections should be broken. The default is persistence reset redirect, which causes the CSS to send a TCP reset and an HTTP 302 redirect to the client, forcing the client to establish a new connection. Internet Explorer (IE) 5.0 has known problems with receiving multiple redirects back to the same domain. Persistence reset remap preserves the client connection and moves the connection on the backside from one server to another.

Failover

When a content rule is composed of services that are of the transparent-cache type and using one of the caching balance methods, then the failover bypass method can be used to allow for the requests destined to a specific cache to be forwarded on to the origin server if the cache goes down. Refer to the asymmetry issue above.

EQL Bypass

The CSS can be configured with a list of file extensions to send to the cache. This list is called an EQL. If you append an EQL to the URL line in a content rule, then the rule only matches requests with the file extensions that are in the list. The CSS can also be configured to bypass an EQL by changing the application on the content rule to bypass. In WebNS 4.0, if you want the CSS to serve each request in a persistent connection to the appropriate destination based on an EQL, you need to issue the **no persistent** command on the content rule. Once the content rule has been bypassed, if you want the next get in the persistent connection to be sent to the cache if it is cacheable, then you need to issue the WebNS 4.0 **bypass persistence disable** global command. Whenever you disable persistence, you want to issue the 4.0 **persistence reset remap** global command to avoid issues with IE 5.0.

Cache Bypass

Some caches have the ability to configure a list of sites which should be bypassed by the cache. The CSS is incompatible with the bypass functionality of most caches. The CSS allows you to configure Network Qualifier Lists (NQLs), which allow you to configure a list of IP address or networks that you can add to a single clause in an ACL.

URL Parameters Bypass Feature

The CSS can be configured to automatically bypass any URLs that have parameters. These are requests that a cache can never satisfy (for example, stock quotes).

Pre-Fetching Feature

When providing a cache service to clients, the variable that creates the greatest delay in response time is when the cache does not contain the page being requested, because it then has to go and retrieve it. The goal of load balancing caches is to increase the probability of a cache hit. Some caches have a feature called pre-fetching, which allows the cache to parse the data returned from the initial get request and proactively go get objects on that page before the client explicitly requests them. At times this can cause extra utilization of the Internet backbone if the client were to stop the page from loading, but this is probably a negligible amount of traffic.

If you are using transparent caching, then the EQL bypass feature on the CSS breaks the pre-fetching algorithm because, by default, the EQL does not include the home page, which is simply a GET for "/." Without the main home page for a site, the cache does not have the main page to work off of for pre-fetching. Secondly, if we do not send dynamic requests to the cache, there will be duplicate requests going to the origin server, one from the CSS and one from the cache. The customer must decide what their goal is, and then determine which functionality from each product they want to combine

At sites where multiple caches are deployed, the load balancer should try to make a best effort attempt to ensure a cache hit. The CSS has several load balancing algorithms that were designed to optimize the number of cache hits. Since the switch is a L5 switch, it can look at information in the HTTP GET that other load balancers cannot. For instance, we can determine which cache to send a request to based on the domain name or the URL being requested. If you were to use the domain hash algorithm, all requests for content that had identical host tags would be sent to the same cache, therefore increasing the probability that the content would reside on that cache. When using pre-fetching with transparent caches, the cache is going to go get the request before the switch forwarding the request to it, which may cause the for the wrong cache to go get the page. In this instance, you may want to disable pre-fetching.

When discussing proxy caches, the IP header does not provide any truly valid information for load balancing in a manner to provide for optimizing the number of cache hits; the destination IP address does not change. The only information to go on would be the source IP address, which does not really mean anything when it comes to surfing the Internet. The switch can still view the host tag and URL information for making these decisions. Since proxy caches typically maintain a persistent connection with the client, the switch only makes the load balancing decision based on the first HTTP GET, and all subsequent GETS go to the same cache. Since we do not break the persistent connection, this works well with pre-fetching feature for all subsequent GETS on the same connection. Starting in WebNS Release 4.0, you can configure the CSS to break these connections if you so desire by issuing the **no persistent** command.

Related Information

- [CSS 11000 Series Content Services Switches Hardware Support](#)
- [CSS 11500 Series Content Services Switches Product Support](#)
- [Cisco Web Network Services Software Product Support](#)
- [Download CSS 11000 Software](#)
- [Download CSS 11500 Software](#)
- [Technical Support – Cisco Systems](#)

All contents are Copyright © 2006–2007 Cisco Systems, Inc. All rights reserved. Important Notices and Privacy Statement.

Updated: Jan 31, 2006

Document ID: 12574
