Important Notices

The following important notices are presented in English, French, and German.

Important Notices
This guide is delivered subject to the following conditions and restrictions:
Copyright Radware Ltd. 2006–2011. All rights reserved.
The copyright and all other intellectual property rights and trade secrets included in this guide are owned by Radware Ltd.
The guide is provided to Radware customers for the sole purpose of obtaining information with respect to the installation and use of the Radware products described in this document, and may not be used for any other purpose.
The information contained in this guide is proprietary to Radware and must be kept in strict confidence.
It is strictly forbidden to copy, duplicate, reproduce or disclose this guide or any part thereof without the prior written consent of Radware.

Notice importante
Ce guide est sujet aux conditions et restrictions suivantes : Copyright Radware Ltd. 2006–2011. Tous droits réservés.
Le copyright ainsi que tout autre droit lié à la propriété intellectuelle et aux secrets industriels contenus dans ce guide sont la propriété de Radware Ltd.
Ce guide d'informations est fourni à nos clients dans le cadre de l'installation et de l'usage des produits de Radware décrits dans ce document et ne pourra être utilisé dans un but autre que celui pour lequel il a été conçu.
Les informations répertoriées dans ce document restent la propriété de Radware et doivent être conservées de manière confidentielle.
Il est strictement interdit de copier, reproduire ou divulguer des informations contenues dans ce manuel sans avoir obtenu le consentement préalable écrit de Radware.

Wichtige Anmerkung
Dieses Handbuch wird vorbehaltlich folgender Bedingungen und Einschränkungen ausgeliefert:
Das Urheberrecht und alle anderen in diesem Handbuch enthaltenen Eigentumsrechte und Geschäftsgeheimnisse sind Eigentum von Radware Ltd.
Die in diesem Handbuch enthaltenen Informationen sind Eigentum von Radware und müssen streng vertraulich behandelt werden.
Es ist streng verboten, dieses Handbuch oder Teile daraus ohne vorherige schriftliche Zustimmung von Radware zu kopieren, vervielfältigen, reproduzieren oder offen zu legen.
Copyright Notices

The following copyright notices are presented in English, French, and German.

Copyright Notices

This product contains work derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm.

RSA Data Security, Inc. makes no representations concerning either the merchantability of the MD5 Message-Digest Algorithm or the suitability of the MD5 Message-Digest Algorithm for any particular purpose. It is provided "as-is" without express or implied warranty of any kind.

This product contains code developed by the OpenSSL Project

This product includes software developed by the OpenSSL Project. For use in the OpenSSL Toolkit. (http://www.openssl.org/).

Copyright (c) 1998-2005 The OpenSSL Project. All rights reserved.

This product contains the Rijndael cipher

The Rijndael implementation by Vincent Rijmen, Antoon Bosselaers and Paulo Barreto is in the public domain and distributed with the following license:

@version 3.0 (December 2000)
Optimized ANSI C code for the Rijndael cipher (now AES)
@author Vincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>
@author Antoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>
@author Paulo Barreto <paulo.barreto@terra.com.br>

The OnDemand Switch may use software components licensed under the GNU General Public License Agreement Version 2 (GPL v.2) including LinuxBios and Filo open source projects. The source code of the LinuxBios and Filo is available from Radware upon request. A copy of the license can be viewed at:

http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

This code is hereby placed in the public domain.

This product contains code developed by the OpenBSD Project


The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This product includes software developed by Markus Friedl
This product includes software developed by Theo de Raadt
This product includes software developed by Niels Provos
This product includes software developed by Dug Song
This product includes software developed by Aaron Campbell
This product includes software developed by Damien Miller
This product includes software developed by Kevin Steves
This product includes software developed by Daniel Kouril
This product includes software developed by Wesley Griffin
This product includes software developed by Per Allansson
This product includes software developed by Nils Nordman
This product includes software developed by Simon Wilkinson

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

ALL THE SOFTWARE MENTIONED ABOVE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.

IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Notice traitant du copyright

Ce produit renferme des codes développés dans le cadre du projet OpenSSL.

Copyright (c) 1998-2005 Le projet OpenSSL. Tous droits réservés. Ce produit inclut la catégorie de chiffre Rijndael.

L'implémentation de Rijndael par Vincent Rijmen, Antoon Bosselaers et Paulo Barreto est du domaine public et distribuée sous les termes de la licence suivante :

@version 3.0 (Décembre 2000)

Code ANSI C code pour Rijndael (actuellement AES)
@author Vincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>
@author Antoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>
@author Paulo Barreto <paulo.barreto@terra.com.br>.

Le commutateur OnDemand peut utiliser les composants logiciels sous licence, en vertu des termes de la licence GNU General Public License Agreement Version 2 (GPL v.2), y compris les projets à source ouverte LinuxBios et Filo. Le code source de LinuxBios et Filo est disponible sur demande auprès de Radware. Une copie de la licence est répertoriée sur :

http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

Ce code est également placé dans le domaine public.

Ce produit renferme des codes développés dans le cadre du projet OpenSSL.

Les membres du conseil de l'Université de Californie. Tous droits réservés.
La distribution et l'usage sous une forme source et binaire, avec ou sans modifications, est autorisée pour autant que les conditions suivantes soient remplies :

1. La distribution d'un code source doit inclure la notice de copyright mentionnée ci-dessus, cette liste de conditions et l'avis de non-responsabilité suivant.

2. La distribution, sous une forme binaire, doit reproduire dans la documentation et/ou dans tout autre matériel fourni la notice de copyright mentionnée ci-dessus, cette liste de conditions et l'avis de non-responsabilité suivant.

3. Le nom de l'université, ainsi que le nom des contributeurs ne seront en aucun cas utilisés pour approuver ou promouvoir un produit dérivé de ce programme sans l'obtention préalable d'une autorisation écrite.

Ce produit inclut un logiciel développé par Markus Friedl
Ce produit inclut un logiciel développé par Theo de Raadt
Ce produit inclut un logiciel développé par Niels Provos
Ce produit inclut un logiciel développé par Dug Song
Ce produit inclut un logiciel développé par Aaron Campbell
Ce produit inclut un logiciel développé par Damien Miller
Ce produit inclut un logiciel développé par Kevin Steves
Ce produit inclut un logiciel développé par Daniel Kouril
Ce produit inclut un logiciel développé par Wesley Griffin
Ce produit inclut un logiciel développé par Per Allansson
Ce produit inclut un logiciel développé par Nils Nordman
Ce produit inclut un logiciel développé par Simon Wilkinson.

La distribution et l'usage sous une forme source et binaire, avec ou sans modifications, est autorisée pour autant que les conditions suivantes soient remplies :

1. La distribution d'un code source doit inclure la notice de copyright mentionnée ci-dessus, cette liste de conditions et l'avis de non-responsabilité suivant.

2. La distribution, sous une forme binaire, doit reproduire dans la documentation et/ou dans tout autre matériel fourni la notice de copyright mentionnée ci-dessus, cette liste de conditions et l'avis de non-responsabilité suivant.

LE LOGICIEL MENTIONNÉ CI-DESSUS EST FOURNI TEL QUEL PAR LE DÉVELOPPEUR ET TOUTE GARANTIE, EXPLICITE OU IMPLICITE, Y COMPRIS, MAIS SANS S’Y LIMITER, TOUTE GARANTIE IMPLICITE DE QUALITÉ MARCHANDE ET D’ADÉQUATION À UN USAGE PARTICULIER EST EXCLUE.

EN AUCUN CAS L'AUTEUR NE POURRA ÊTRE TENU RESPONSABLE DES DOMMAGES DIRECTS, INDIRECTS, ACCESSOIRES, SPÉCIAUX, EXEMPLAIRES OU CONSÉCUTIFS (Y COMPRES, MAIS SANS S’Y LIMITER, L’ACQUISITION DE BIENS OU DE SERVICES DE REMPLACEMENT, LA PERTE D’USAGE, DE DONNÉES OU DE PROFITS OU L’INTERUPTION DES AFFAIRES), QUELLE QU’EN SOIT LA CAUSE ET LA THÉORIE DE RESPONSABILITÉ, QU’IL S’AGISSE D’UN CONTRAT, DE RESPONSABILITÉ STRICTE OU D’UN ACTE DOMMAGEABLE (Y COMPRES LA NÉGLIGENCE OU AUTRE), DÉCOULANT DE QUELLE QUE FAÇON QUE CE SOIT DE L’USAGE DE CE LOGICIEL, MÊME S’IL A ÉTÉ AVERTI DE LA POSSIBILITÉ D’UN TEL DOMMAGE.

Copyrightvermerke
Dieses Produkt enthält einen vom OpenSSL-Projekt entwickelten Code
Dieses Produkt enthält vom OpenSSL-Projekt entwickelte Software. Zur Verwendung im OpenSSL Toolkit. (http://www.openssl.org/).
Copyright (c) 1998-2005 The OpenSSL Project. Alle Rechte vorbehalten. Dieses Produkt enthält die Rijndael cipher
Die Rijndael-Implementierung von Vincent Rijndael, Anton Bosselaers und Paulo Barreto ist öffentlich zugänglich und wird unter folgender Lizenz vertrieben:
@version 3.0 (December 2000)
Optimierter ANSI C Code für den Rijndael cipher (jetzt AES)
@author Vincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>
@author Antoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>
@author Paulo Barreto <paulo.barreto@terra.com.br>

Der OnDemand Switch verwendet möglicherweise Software, die im Rahmen der DNU Allgemeine Öffentliche Lizenzvereinbarung Version 2 (GPL v.2) lizensiert sind, einschließlich LinuxBios und Filo Open Source-Projekte. Der Quellcode von LinuxBios und Filo ist bei Radware auf Anfrage erhältlich. Eine Kopie dieser Lizenz kann eingesehen werden unter:
http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

Dieser Code wird hiermit allgemein zugänglich gemacht.

Dieses Produkt enthält einen vom OpenBSD-Projekt entwickelten Code
The Regents of the University of California. Alle Rechte vorbehalten.

Die Verbreitung und Verwendung in Quell- und binärem Format, mit oder ohne Veränderungen, sind unter folgenden Bedingungen erlaubt:

3. Weder der Name der Universität noch die Namen der Beiträger dürfen ohne ausdrückliche vorherige schriftliche Genehmigung verwendet werden, um von dieser Software abgeleitete Produkte zu empfehlen oder zu bewerben.


Dieses Produkt enthält von Simon Wilkinson entwickelte Software

Die Verbreitung und Verwendung in Quell- und binärem Format, mit oder ohne Veränderungen, sind unter folgenden Bedingungen erlaubt:


SÄMTLICHE VORGENANNTE SOFTWARE WIRD VOM AUTOR IM IST-ZUSTAND ("AS IS") BEREITGESTELLT. JEGLICHE AUSDRÜCKLICHEN ODER IMPLIZITEN GARANTIEN, EINSCHLIESSLICH, DOCH NICHT BESCHRÄNKT AUF DIE IMPLIZIERTEN GARANTIEN DER MARKTGÄNGIGKEIT UND DER ANWENDBARKEIT FÜR EINEN BESTIMMTEN ZWECK, SIND AUSGESCHLOSSEN.

UNTER KEINEN UMSTÄNDEN HAFTET DER AUTOR FÜR Direkte oder Indirekte Schäden, für bei Vertragsregelung entstandene Schäden, für besondere Schäden, für schadensersatz mit strafcharakter, oder für Folgeschäden einschliesslich, doch nicht beschränkt auf, erwerb von ersatzgütern oder ersatzleistungen; verlust an Nutzung, Daten oder Gewinn; oder geschäftsunterbrechungen gleich, wie sie entstanden sind, und für jegliche Art von Haftung, sei es verträge, gefäß
HRDUNGSCHAFTUNG, ODER DELIKTISCHE HAFTUNG (EINSCHLIESSLICH FAHRLÄSSIGKEIT ODER ANDERE), DIE IN JEGLICHER FORM FOLGE DER BENUTZUNG DIESER SOFTWARE IST, SELBST WENN AUF DIE MÖGLICHKEIT EINES SOLCHEN SCHADENS HINGEWIESEN WURDE.
# Document Conventions

The following describes the conventions and symbols that this guide uses:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Description (French)</th>
<th>Beschreibung (German)</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Example]</td>
<td>An example scenario</td>
<td>Un scénario d’exemple</td>
<td>Ein Beispielszenarium</td>
</tr>
<tr>
<td>![Caution:]</td>
<td>Possible damage to equipment, software, or data</td>
<td>Endommagement possible de l’équipement, des données ou du logiciel</td>
<td>Mögliche Schäden an Gerät, Software oder Daten</td>
</tr>
<tr>
<td>![Note:]</td>
<td>Additional information</td>
<td>Informations complémentaires</td>
<td>Zusätzliche Informationen</td>
</tr>
<tr>
<td>![To]</td>
<td>A statement and instructions</td>
<td>Références et instructions</td>
<td>Eine Erklärung und Anweisungen</td>
</tr>
<tr>
<td>![Tip:]</td>
<td>A suggestion or workaround</td>
<td>Une suggestion ou solution</td>
<td>Ein Vorschlag oder eine Umgehung</td>
</tr>
<tr>
<td>![Warning:]</td>
<td>Possible physical harm to the operator</td>
<td>Blessure possible de l’opérateur</td>
<td>Verletzungsgefahr des Bedieners</td>
</tr>
</tbody>
</table>
Table of Contents

Important Notices ............................................................................................................. 3
Copyright Notices ............................................................................................................. 4
Document Conventions ..................................................................................................... 9

Chapter 1 – Preface ...................................................................................................... 19
Who Should Use this Guide ............................................................................................ 19
What You Will Find in this Guide .................................................................................... 19
Related Documentation ................................................................................................. 19
Symbols, Commands, and Keys ....................................................................................... 20

Chapter 2 – Server Load Balancing ............................................................................. 21
Configuring a Juniper Networks Device for Server Load Balancing ......................... 21
  Loading the ADC Software onto a Device ................................................................. 21
  Upgrading the ADC Software ..................................................................................... 24
  Installing a License for the ADC Software ................................................................. 25
  Auto-Configuration in the ADC Software ................................................................. 25
  ADC Instances ............................................................................................................... 26
  Virtual Routers and Unit .............................................................................................. 28
  Anycast and Dynamic Routing .................................................................................... 34
Understanding Server Load Balancing ........................................................................ 36
  Identifying Your Network Needs .................................................................................. 36
  How Server Load Balancing Works ............................................................................. 36
  Server Load Balancing on the Juniper Networks Router Introduction ..................... 37
Implementing Server Load Balancing .......................................................................... 38
  Network Topology Requirements ................................................................................ 39
  Server Load Balancing Configuration Basics ............................................................ 41
Server Load Balancing Options .................................................................................... 43
  Supported Services and Applications ......................................................................... 44
  Disabling and Enabling Real Servers ......................................................................... 44
  Health Checks for Real Servers ................................................................................... 45
  Configuring Multiple Services in the Same Group ....................................................... 46
  Load-Balancing Methods for Real-Server Groups ....................................................... 46
  Weights for Real Servers .............................................................................................. 48
  Connection Timeouts for Real Servers ........................................................................ 49
  Maximum Connections for Real Servers ...................................................................... 49
  Unlimited Connections to Real Servers ....................................................................... 50
  Backup Servers and Overflow Configuration ............................................................. 50
  Backup-Only Server ..................................................................................................... 52
  Backup Preemption ....................................................................................................... 52
  Server Warm-Up Time ................................................................................................. 53
  Direct Server Return .................................................................................................... 54
Chapter 3 – Load-Balancing Special Services .......................................................... 65

FTP Server Load Balancing ....................................................................................... 65
  FTP Network Topology Restrictions ........................................................................ 65
  Configuring FTP Server Load Balancing ................................................................. 66
TFTP Server Load Balancing ....................................................................................... 67
  Configuring TFTP Server Load Balancing ................................................................. 67
Lightweight Directory Access Protocol Server Load Balancing ....................... 68
  Configuring LDAP Server Load Balancing ............................................................. 69
  LDAP Operations and Server Types ........................................................................ 70
  LDAP Write ................................................................................................................ 70
Domain Name System Server Load Balancing ....................................................... 71
  Preconfiguration Tasks ............................................................................................. 73
  Configuring UDP-Based DNS Load Balancing ......................................................... 73
  Configuring TCP-Based DNS Load Balancing ......................................................... 74
  Content-Based DNS Server Load Balancing ............................................................ 74
Real-Time Streaming Protocol Server Load Balancing ........................................ 77
  How RTSP Server Load Balancing Works ................................................................. 78
  Supported RTSP Servers .......................................................................................... 78
  Configuring RTSP Load Balancing ......................................................................... 79
  Content Intelligent RTSP Load Balancing ............................................................... 82
Secure Sockets Layer Server Load Balancing ....................................................... 88
  How SSL Session ID-Based Server Load Balancing Works ..................................... 89
  Configuring SSL Server Load Balancing ................................................................. 90
Wireless Application Protocol Server Load Balancing ....................................... 91
  WAP SLB with RADIUS Snooping .......................................................................... 92
  WAP Server Load Balancing with RADIUS/WAP Persistence .............................. 95
Session Initiation Protocol Server Load Balancing .............................................. 96
  SIP Processing .......................................................................................................... 96
  Configuring SIP Server Load Balancing ................................................................. 96
  Session Persistency Using the Refer Method ......................................................... 98
HTTP Server Load Balancing ................................................................. 99
  When to Disable Persistence to the Server Listening Ports ..................... 99
  Configuring HTTP Server Load Balancing .............................................. 99
  Connection Pooling ............................................................................. 101
HTTP Persistency ..................................................................................... 102
  HTTP and HTTPS Persistence Based on Client IP ................................. 102
  Cookie-Based Persistence ...................................................................... 102
  Content Intelligent HTTP Server Load Balancing .................................... 113
Windows Terminal Server Load Balancing ............................................. 127
  Configuring Windows Terminal Server Load Balancing ........................... 127
IP (Plain) Server Load Balancing ............................................................ 128

Chapter 4 – Filtering ................................................................................ 131
  Filter Overview .................................................................................... 131
    Router Interfaces ............................................................................. 132
    Filtering Benefits ............................................................................ 132
  Filter Configuration ............................................................................ 132
    ADC Filter terms ............................................................................ 132
    Stacking Filters ............................................................................ 134
    Overlapping Filters ....................................................................... 134
    The Default Filter .......................................................................... 135
    Optimizing Filter Performance ......................................................... 135
    Filter Logs ...................................................................................... 135
    Using Per-Packet Load Balancing with Filters ................................... 136
  Tunable Hash for Filter Redirection ....................................................... 137
  Filter-Based Security .......................................................................... 137
  Matching TCP Flags ........................................................................... 143
  Deny Filter Based on Layer 7 Content .................................................. 147
    Denying HTTP URL Requests ......................................................... 147
    Denying HTTP Headers .................................................................. 148
  Cache Server Load Balancing ............................................................. 149
    Cache Server Load Balancing Overview ......................................... 150
    Cache Redirection Environment ...................................................... 150
    RTSP Cache Redirection ................................................................. 155
    Excluding Noncacheable Sites ......................................................... 157
    Network Address Translation Options .............................................. 157
    Content Intelligent Cache Redirection ............................................ 158
  HTTP Redirection ................................................................................ 172
    HTTP Redirection Overview ........................................................... 172
    IP-Based HTTP Redirection ............................................................ 173
    TCP Service Port-Based HTTP Redirection ...................................... 175
    MIME Type Header-Based Redirection .......................................... 177
    URL-Based Redirection ................................................................. 179
## ADC Software User Guide

### Table of Contents

Source IP from HTTP Header and Host Header-Based Redirection ........................................ 180
HTTP to HTTPS Redirection ...................................................................................................... 182

**Chapter 5 – Health Checking** .............................................................................................. 183
Real-Server Health Check Configuration .................................................................................. 184
Health Check Source IP Address .............................................................................................. 185
TCP Health Checks .................................................................................................................. 185
Ping Health Checks .................................................................................................................. 186
Application-Based Health Checks ............................................................................................ 186
  HTTP Health Checks ............................................................................................................... 187
  DNS Health Checks ............................................................................................................... 189
  TFTP Health Check .............................................................................................................. 190
  SNMP Health Check .............................................................................................................. 190
  FTP Server Health Checks .................................................................................................... 191
  POP3 Server Health Checks ................................................................................................. 192
  SMTP Server Health Checks ............................................................................................... 193
  IMAP Server Health Checks ............................................................................................... 193
  NNTP Server Health Checks ............................................................................................... 194
  RADIUS Server Health Checks .......................................................................................... 195
  SSL Server Health Checks .................................................................................................... 197
  WAP Gateway Health Checks .............................................................................................. 198
  LDAP Health Checks ............................................................................................................ 202
  Windows Terminal Server Health Checks ............................................................................. 203
  RTSP Health Check ............................................................................................................. 203
SIP Health Checks .................................................................................................................... 204
  Configuring the SIP Health Checks ....................................................................................... 204
Script-Based Health Checks .................................................................................................... 205
  Configuring Script-Based Health Checks ............................................................................. 205
  Script Formats ..................................................................................................................... 206
  Scripting Commands ............................................................................................................ 208
  Scripting Guidelines ............................................................................................................ 208
  Adding the Script to a Group ............................................................................................... 208
  Script Configuration Examples ............................................................................................ 209
Direct Server Return Health Checks .......................................................................................... 213
Server-Based Group Health Check ............................................................................................ 213
Buddy Server Health Checks .................................................................................................... 214
Failure Types .......................................................................................................................... 217
  Service Failure ..................................................................................................................... 217
  Server Failure ....................................................................................................................... 217

**Chapter 6 – High Availability** .............................................................................................. 219
RMS Introduction ..................................................................................................................... 219
RMS Support ........................................................................................................................... 220
## Table of Contents

Connection Synchronization ................................................................. 222

### Chapter 7 – Content String Handling .............................................. 225
- Avoidance HTTP String Matching for Real Servers .......................... 225
  - Configuring Avoidance URL String Matching ................................. 226
- Regular Expression Matching .............................................................. 227
  - Standard Regular Expression Characters ........................................ 227
  - Configuring Regular Expressions .................................................... 227
- Using Variables in Strings for HTTP .................................................. 228
- Content Precedence Lookup ............................................................... 229
  - Using the or / and Operators ......................................................... 230
  - Assigning Multiple Strings ............................................................ 231
- String Case Sensitivity ....................................................................... 233
- Configurable HTTP Methods ............................................................. 233
- Pattern-Based Content-Match ............................................................ 234
  - Pattern Criteria .............................................................................. 234

### Appendix A – ADC Software Load Command Set ............................. 237
- Command Set for Loading the ADC Software Onto a Device ............ 237

### Index .............................................................................................. 239
# Table of Figures

- Figure 1: Anycast Example .............................................................. 34
- Figure 2: How Server Load Balancing Works ........................................ 37
- Figure 3: Static DNS Traffic Increases ............................................. 38
- Figure 4: Implementing Server Load Balancing .................................. 39
- Figure 5: Direct Server Return Path .................................................. 40
- Figure 6: Load-Balancing to a DNS group ........................................ 41
- Figure 7: How Direct Server Return Works ....................................... 54
- Figure 8: DoS SYN Attack Example ................................................... 57
- Figure 9: Repelling DoS Attacks ...................................................... 58
- Figure 10: Client NAT Example ........................................................ 63
- Figure 11: LDAP Query Example ..................................................... 71
- Figure 12: Load-Balancing UDP and TCP Queries ............................. 72
- Figure 13: DNS Server Group .......................................................... 75
- Figure 14: RTSP Load Balancing ...................................................... 79
- Figure 15: Media Cache Servers in Reverse Proxy Mode .................. 83
- Figure 16: SSL Session ID-Based SLB.............................................. 89
- Figure 17: WAP Server Load Balancing ........................................... 92
- Figure 18: SIP Server Load Balancing .............................................. 97
- Figure 19: Cookie-Based Persistence ............................................... 103
- Figure 20: Insert Cookie Mode ........................................................ 105
- Figure 21: Passive Cookie Mode ..................................................... 106
- Figure 22: Rewrite Cookie Mode ...................................................... 107
- Figure 23: URL-Based Server Load Balancing ................................. 114
- Figure 24: URL Hashing for Server Load Balancing ....................... 124
- Figure 25: Filter-Based Security ..................................................... 138
- Figure 26: TCP Flag Filter .............................................................. 144
- Figure 27: Cache Redirection Environment ....................................... 151
- Figure 28: Cache Redirection Using Proxy Servers ......................... 152
- Figure 29: RTSP Cache Redirection ................................................ 155
- Figure 30: URL-Based Cache Redirection ........................................ 160
- Figure 31: Hashing on the URL ....................................................... 169
- Figure 32: Forward Proxy Mode ...................................................... 170
- Figure 33: Network Topology .......................................................... 215
- Figure 34: Using or/and Operators ................................................... 230
- Figure 35: Assigning Multiple Strings ............................................. 232
Chapter 1 – Preface

Juniper Networks® Application Delivery Controller (ADC) for the MX Series 3D Universal Edge Router offers advanced router-integrated ADC functions that enables service providers and enterprises to efficiently scale service capacity and increase service performance. Routers are already ubiquitously deployed throughout the network: at the network edge, in the network core, and in the data center. Integrating the advanced ADC with the carrier-grade MX3D router promotes network consolidation and reduces the number of network elements that providers must rack, power, cool, maintain, and upgrade. Furthermore, the ADC software, which is optionally licensed, improves service resiliency by monitoring server and application health and by automatically bypassing failures.

This guide describes how to configure and use the ADC software.

Who Should Use this Guide

This Application Guide is intended for network installers and system administrators engaged in configuring and maintaining a network. The administrator should be familiar with Ethernet concepts, IP addressing, and SNMP configuration parameters.

What You Will Find in this Guide

This guide helps you to plan, implement, and administer the ADC software. Where possible, each section provides feature overviews, usage examples, and configuration instructions.

- **Server Load Balancing**—Describes how to configure the ADC software to balance network traffic among a group of available servers for more efficient and robust network services.
- **Load-Balancing Special Services**—Describes how to extend server load-balancing configurations to load-balance services including source IP addresses, FTP, RTSP, DNS, WAP, IDS, and Session Initiation Protocol (SIP).
- **Filtering**—Describes how to configure and optimize network traffic filters for security and Network Address Translation (NAT).
- **Health Checking**—Describes how to configure the ADC software to recognize the availability of the various network resources used with the various load-balancing and application redirection features.
- **High Availability**—Describes how to use the redundant Multiservices PIC (RMS) to ensure that network resources remain available if one NPU is offline.
- **Content String Handling**—Discusses two tools for troubleshooting your ADC software: monitoring ports and filtering session dumps.

Related Documentation

- **ADC Software Release Notes**
- **ADC Software Reference Guide**
- **ADC Software Troubleshooting Guide**
Symbols, Commands, and Keys

Table 1 on page 20, is a list of commands to help you use the application.

<table>
<thead>
<tr>
<th>Symbol/Command/Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>This symbol on the command line indicates you are in Configuration mode.</td>
</tr>
<tr>
<td>?</td>
<td>Shows information pertaining to the commands at the level of the hierarchy you are currently viewing.</td>
</tr>
<tr>
<td>&gt;</td>
<td>This symbol on the command line indicates you are in Operational mode.</td>
</tr>
<tr>
<td>Clear</td>
<td>Clears the gathered statistics. Only available in Operational mode.</td>
</tr>
<tr>
<td>Commit</td>
<td>Saves and applies the changes made. The device in question begins using the edits immediately upon a successful commit. You can set commit times for the future by entering a value. For example, “commit 10” would save and apply the edits in 10 minutes.</td>
</tr>
<tr>
<td>Configure or Edit</td>
<td>Enters Configuration mode. The symbol preceding your command entries will change to show a “#”. Otherwise the prompt shows a “&gt;”, indicating you are in Operational mode.</td>
</tr>
</tbody>
</table>
| Examples:          | “adc@juniper6 #” Configuration mode  
|                    | “adc@juniper6 >” Operational mode |
| Request            | Executes specific actions. Only available in Operational mode. |
| Run                | Executes a command from anywhere within the hierarchy. For example, “run show extensions adc status” will execute the command ‘show extensions adc status’ and display the result from any level of the hierarchy. |
| Set                | Adds a command in Configuration mode. |
| Show               | Displays the current configuration of the command hierarchy you are currently viewing. |
| Status             | Shows information pertaining to the CPU currently being used. |
| Tab or Space key   | Auto-completes an entry. If you start to type a command and press Tab, the system completes the command if enough letters are present for it to determine which command you were typing. If there is not enough information for the system to determine which command you wanted, it will display an error message and list the commands that fit the partial entry you typed. |
| Top                | Moves you to the top, or first, level of the command hierarchy. |
| Up                 | Moves you up one step on the command hierarchy. |
Chapter 2 – Server Load Balancing

You can configure the ADC software within your router to balance user session traffic among a group of available servers that provide shared services.

The following topics are addressed in this chapter:

- **Configuring a Juniper Networks Device for Server Load Balancing**—This section describes how to set up the device, load the ADC software, and configure Multiservices-DPCs for use.
- **Understanding Server Load Balancing**—This section discusses the benefits of server load balancing (SLB) and its operation.
- **Implementing Server Load Balancing**—This section describes how implementing SLB provides reliability, performance, and ease of maintenance on the network.
- **Direct Server Return**—This section discusses Direct Server Return, mapping real to virtual ports, monitoring real server ports, and delayed binding.
- **Session Timeout per Service**—This section describes the configuration of the session timeout per service feature.
- **Persistency**—This section explains the general use of persistence and how it relates to SLB.
- **NAT IP Addresses**—This section describes how Network Address Translation works.

For additional information on SLB commands, see the *ADC Software Command Reference*.

Configuring a Juniper Networks Device for Server Load Balancing

This section describes the steps for configuring the ADC software on the Multiservices-DPC. In order to perform these steps, you must be a user with super-user permissions. Before configuration, make sure you have performed the steps required to load the ADC software.

Loading the ADC Software onto a Device

This section describes the steps for loading the ADC software on a device. In the following procedure, many of the SLB options are left at their default values. See *Server Load Balancing Options* for more options. Before you start configuring, you must be connected to the router CLI as the administrator.

- **Note**: These commands also can be used to install other software add-ons.

- **To load the ADC software**
  1. Use FTP to transfer the package to the router.

- **Note**: A “package” is a compressed file containing one or more software modules, add-ons, or additional files for installation on the router.
2. Allow the package to run on the router.

```
[edit]
system {
    extensions {
        providers {
            radware;
            license-type strategic deployment-scope commercial;
        }
    }
}
```

3. Commit and quit to enact the changes.

```
user@host# commit and-quit
```

4. Add the new software to the router.

```
user@host> request system software add <package>
```

5. Install the ADC software on the Multiservices-DPC NPUs. Repeat as needed for all relevant NPUs.

```
[edit]
chassis {
    fpc <number> {
        pic <number> {
            adaptive-services {
                service-package {
                    extension-provider {
                        control-cores 1;
                        data-cores 7;
                        object-cache-size 512;
                        wired-process-mem-size 256;
                        data-flow-affinity {
                            hash-key layer-3;
                        }
                        package adc-ctrl;
                        package adc-data;
                    }
                }
            }
            }
        }
    }
```
6. Configure the commit and op scripts used by the ADC software.

```
[edit]
system {
    scripts {
        commit {
            allow-transients;
            file radware-conf.slax;
        }
        op {
            file radware-op.slax;
        }
    }
}
```

7. Configure the device to use ECMP hash on Layer 3 with IP address only.

```
[edit]
forwarding-options {
    hash-key {
        family inet {
            layer-3 {
                destination-address;
                source-address;
            }
        }
    }
    enhanced-hash-key {
        services-loadbalancing {
            family inet {
                layer-3-services {
                    source-address;
                }
            }
        }
    }
}
```

**Note:** The destination-address, source-address, and services-loadbalancing commands are hidden in Junos OS version 11.1. As a result, there is no CLI auto-completion for these statements.
8. Configure the device to capture logs.

```plaintext
[edit]
system {
syslog {
    file device-log {
        any any;
    }
    file adc-log {
        any any;
        match adc;
    }
}
}
```

9. Commit the changes and quit configuration mode.

```
user@host# commit and-quit
```

Notes:

>> After committing the configuration as listed above, the Multiservices-DPC NPUs become active and the ms-x/y/0 interfaces appear in the configuration.

>> To view a full command set for loading the ADC software onto a device, see ADC Software Load Command Set.

Upgrading the ADC Software

In order to upgrade the ADC software, the old ADC software must be fully removed from the device. Only after this is done can the new software be loaded.

To remove the existing ADC software

1. Save the configuration for later use, if needed.

```
[edit]
user@host# save <filename>
```

2. Delete the ADC configuration.

```
[edit]
user@host# delete extensions adc
```

3. Delete the commit and op scripts used by the ADC software.

```
[edit]
user@host# delete system scripts
```

4. Delete the ADC software from the Multiservices-DPC NPUs. Repeat for all relevant NPUs.
5. Commit the changes and quit configuration mode.

```
[edit]
user@host# commit and-quit
```

6. Remove the ADC software components in the following order:

```
user@host> request system software delete adc-data
user@host> request system software delete adc-ctrl
user@host> request system software delete adc-mgmt
```

7. Install the new ADC software onto the device. See [Loading the ADC Software onto a Device](#).

### Installing a License for the ADC Software

You must purchase a suitable license in order to run the ADC software. Each license is for one Multiservices-DPC (two NPUs per license). You should purchase licenses according to the number of Multiservices-DPCs that you have in your device.

#### To install a license in your device

1. Use FTP to transfer the license file to the device.
2. Install the license on the device.

```
user@host> request system license add <license filename>
```

#### To verify the licenses that are installed in your device

```
user@host> show system adc license-info
```

Or

```
user@host> show extensions adc license-info
```

### Auto-Configuration in the ADC Software

As part of the ADC software integration into the Juniper Networks Junos OS system, the ADC software is using an internal configuration. The internal configuration is done in two ways: using a commit script and using the Junos OS internal API.
The internal configuration is used for the following reasons:
1. To direct the relevant traffic from the client-facing interfaces into the Multiservices-DPCs running the ADC software.
2. To direct the relevant traffic from the Multiservices-DPCs running the ADC software to the server-facing interfaces, and to the servers themselves.
3. To enable some internal communication between the Multiservices-DPCs running the ADC software.

As part of the auto-configuration, the ADC software uses Junos OS firewall filters, Junos OS routing-instances of type forwarding-instance, and Junos OS IFL and IFA (units and addresses) defined on the Multiservices-DPCs running the ADC software.

Any configuration that is static by nature is done using a commit-script configuration. This configuration is easily shown as part of the overall router configuration (Viewing the Auto-Configuration). An example configuration is routing-instances.

Any configuration that is dynamic by nature is done using Junos OS API. This configuration is changing over time and cannot be statically configured using a commit script. An example of such a configuration is a route to an active VIP.

Viewing the Auto-Configuration

In the Junos OS, you can see the configuration done by the commit-script using the command:

```
user@host> show configuration | display commit-script
```

Auto-configuration done with the Junos OS API is not shown in the command above. In order to see the auto-configuration done with the Junos OS API, use the following command:

```
user@host> show extensions adc internal generated-configuration
```

Using Units

As stated in this chapter, as part of the auto-configuration, the ADC software defines IFLs and IFAs (units and addresses) on the Multiservices-DPC. These IFLs require a unique unit number that is used later in auto-configured filters to direct traffic.

By default, the units used by the ADC software for automatic configuration are in the range of 10,000 to 11,032.

Change this unit range with the following configuration:

```
[edit extension adc]
internal-unit-range 8000 to 9032;
```

Use this command only when you really need to. When changing this range, you should make sure the range is big enough; the range should be at least as big as the amount of servers defined in the ADC software configuration plus 32 units that are used internally and with no relation to the current configuration.

ADC Instances

An adc-instance is an instance of Application Delivery Software running on one or more Multiservices-DPC interfaces of a Juniper Networks device. An adc-instance includes a complete set of ADC definitions: real-servers, groups of servers, virtual servers using virtual IP addresses, and virtual services accessed by clients.
Using multiple instances on a single device allows you to create completely separate ADCs running on the same machine. Using different instances for different traffic guarantees computation power, guaranteeing no interruption between services. This can be used, for example, to load-balance traffic from different applications, where complete separation is required.

**To configure an adc-instance**

```plaintext
[edit]
extensions {
ad
    adc {
        adc-instance <name> {
            ... instance configuration ...
        }
    }
}
```

**Note:** Names of configuration objects, such as real servers, virtual servers and services, and so on, must be unique within the complete ADC configuration, across ADC instances. For example, a real server called real1 can appear only once throughout the entire ADC configuration and cannot be used in two ADC instances.

**Service Interface Use in ADC Instances**

You must specify router interfaces that are bound to an adc-instance.

- **Multiservices interfaces** — The physical multiservices interfaces of a device that are used to run the load-balancing instance application. The more multiservices interfaces used for a load-balancing instance, the more capacity and processing power the instance has. At least one MS interface must be specified for each adc-instance, up to eight interfaces can run the same instance.

  A multiservices interface is associated exclusively to a single load-balancing instance (it cannot be shared between instances).

  For more information, see:
  - [RMS Introduction, page 219.](#)
  - [Configuring a Juniper Networks Device for Server Load Balancing, page 21](#)

- **Client-facing interfaces** — The device interfaces where client traffic is received. Traffic arriving on these interfaces is handled by the ADC software and destined to be routed to the virtual IP addresses and filter destination addresses configured in the instance.

  At least one client-facing interface must be specified for each adc-instance. A client-facing interface can be shared between instances.

- **Server-facing interfaces** — The device interfaces where servers are connected, usually through switches or routers. Traffic to the servers is routed to these interfaces.

  At least one server-facing interface must be specified for each load-balancing instance; a server-facing interface can be shared between instances.

  The same device interface can be used as a client-facing interface in one (or more) adc-instances, and as a server-facing interface in other instances.
To configure server and client logical interfaces

```
[edit extensions adc]
adc-instance <name> {
    router-interfaces {
        ms-interfaces {
            ms-0/1/2;
        }
        client-facing {
            ge-0/1/1.0;
        }
        server-facing {
            ge-0/1/0.0;
        }
    }
}
```

**Note:** Multiservices interfaces are the physical interface (IFD) MS-x/y/0, while client-facing and server-facing interfaces are logical interfaces with units (IFL), for example, ge-x/y/z.

**Virtual Routers and Unit**

Using virtual routers allows you to create "routing zones" in the router. These routing zones are logically separated routing tables. A change in one routing zone does not change another zone. When a packet hits an interface that is attached to a virtual router, a routing lookup is performed on the routing table of the specific virtual router only. For more information about virtual routers, see your Juniper Networks documentation.

**Virtual Router Basic Configuration**

The basic configuration of a virtual router includes interfaces. Each interface belongs to only one virtual router. This allows a simple configuration where a packet that comes from this interface will perform a routing lookup in the routing table of the specific virtual router only. The interfaces used here are logical interfaces; for example, ge-0/1/1.40.

**Example Attaching a logical interface to a virtual router**

```
[edit extension adc]
routing-instances {
    HTTP-side {
        instance-type virtual router;
        interface ge-0/1/1.40;
    }
}
```
Support for Virtual Routers

ADC software support for virtual routers is merged into the product using logical interfaces. The ADC software also supports servers in different virtual routers. The support is done by attaching a unit to each of the following configuration entities: real-servers, groups, and adc-instances.

- **Client and Server Interfaces**
- **Traffic to Servers**

Client and Server Interfaces

All server- and client-facing interfaces are logical interfaces and the ADC software only handles traffic coming from these interfaces, as defined by you. Since the ADC software uses filters to catch traffic, a parallel virtual router configuration is possible when attaching the interface to a client virtual router.

**Example Configuring server- and client-facing logical interfaces**

```plaintext
[edit extensions adc]
adc-instance demo1 {
  router-interfaces {
    client-facing {
      ge-0/1/1.0;
    }
    server-facing {
      ge-0/1/0.0;
    }
  }
}
```

Traffic to Servers

In general, the unit is used when the traffic is going out from the ADC software to the server. To support virtual routers on the server side, each server is assigned a unit. When the traffic is going out from the ADC software to this server, the traffic will go out from the matching Multiservices-DPC NPU IFL (ms-x/y/z.#, where # is the unit). This allows you to attach the relevant IFL to a virtual router and attach the server to this virtual router. If the unit is not configured on the server, the unit is taken from the group configuration. If the unit is not configured in the group, the unit is taken from the adc-instance configuration. If no unit is configured, the ADC software uses the default unit (unit 0).

**Examples**

- **A** Configuring a unit on a server

  ```plaintext
  [edit extensions adc adc-instance demo1]
  real-servers {
    r3 {
      address 200.0.0.13 unit 40;
    }
  }
  ```

- **B** Configuring the unit on a group
This will set all servers inside this group to use unit 40, unless a unit is configured on a specific server inside the group.

C Configuring the unit on an adc-instance

This sets all servers inside this adc-instance to use unit 40, unless a unit is configured on a specific server or group inside the instance.

Additional Virtual Router Configuration

In order for a server with subunits to work, additional configuration changes are needed on the ADC software.

The ADC software does health checking on each defined server (see Health Checking, page 183). In order for the traffic to get from the ADC software to the server, a source IP with the same subunit as the server must be defined. Usually all subunits that are in use in a certain adc-instance must have a matching IP address with the same subunit defined in the instance.

Example Configuring a health-check source address with unit 40

Another configuration that is per-unit is the NAT address (see NAT IP Addresses, page 61). As each server has its own unit, it should also have its own NAT address in this unit. When a NAT address is chosen for a server, the relevant NAT address pool contains only addresses that are defined on the same unit as this server. The NAT address is configured at the service-interface hierarchy level, as it is different for each service-interface. All service-interfaces inside an adc-instance should have a NAT address for the server unit in order for the NAT action to work properly.
Example Configuring a NAT address range on unit 40 in MS-1/0/0

```
[edit extensions adc]
adc-instance <name> {
    service-interfaces {
        ms-1/0/0 {
            unit 40 {
                family {
                    inet {
                        nat-address 200.0.0.100;
                    }
                }
            }
        }
    }
}
```

In order to support a virtual router, you should define a logical interface on all multiservices interfaces that are part of the adc-instance and attach them to the desired virtual router. You should attach the relevant server’s facing interface (IFL) to the same virtual router so that traffic continuity can be maintained.
Configuring Virtual Router Support

The following steps show how to configure virtual router support in various ways.

To configure virtual router support on a Multiservices-DPC interface

1. Configure a logical interface on the relevant Multiservices-DPC interface.
2. Attach it to a virtual router together with the relevant server interface.

```
[edit]
interfaces {
    ge-0/1/1 {
        unit 40 {
            family inet {
                address 200.0.0.10/16;
            }
        }
    }
    ms-1/0/0 {
        unit 40;
    }
}
routing-instances {
    server-side {
        instance-type virtual router {
            interface ge-0/1/1.40;
            interface ms-1/0/0.40;
        }
    }
}
```
To configure the unit in the adc-instance

1. Configure the health-check IP for unit 40.
2. Configure the group of servers to unit 40. This will affect all servers inside this group.

```plaintext
[edit extensions adc]
adc-instance demo1 {
    health-check {
        unit 40 {
            family {
                inet {
                    address {
                        200.0.0.1;
                    }
                }
            }
        }
    }
    real-servers {
        r1 {
            address 200.0.0.11;
        }
        r2 {
            address 200.0.0.12;
        }
        r3 {
            address 200.0.0.13;
        }
    }
    groups {
        g1 {
            real-servers [ r1 r2 r3 ];
            group-unit 40;
        }
    }
}
```

In this example, all traffic going to servers on group g1 will hit "server-side" virtual routers only.

**Virtual Router Limitations**

There are a few limitations to using virtual routers:

- Multiple virtual routers are supported on the server side only.

  The ADC software supports only one virtual router on the client side, and all traffic from the ADC software to the clients will go out using IFL 0 of the relevant Multiservices-DPC NPU.

- The ADC software does not support overlapping IP addresses.

  All IP addresses used in regard to the ADC software, including server addresses, virtual server addresses, clients (UAs) addresses, and so on, must be unique throughout the router.
Anycast and Dynamic Routing

Anycast allows you to serve requests in the closest location to the client. Using Anycast, the ADC software can take into account the service availability in each location and suggest to the client the closest location to handle the request, according to the routing dynamic protocol.

Anycast is based on dynamic routing protocol algorithms. In Figure 1 on page 34, a client is using a web browser to view the website of Example Corporation at IP address 10.1.1.1. The Example Corporation has two websites, one in San Jose and one in Denver, each with identical content and available services. Both websites have the MX Series device running the ADC software with virtual service configured as 10.1.1.1.

Figure 1: Anycast Example

Finding the right route involves the following procedure:

1. The user sends a request to 10.1.1.1.
2. The user’s router forwards the request to its default gateway, that forwards the request to its default gateway, until an explicit route to 10.1.1.1 is found. The explicit route to 10.1.1.1 is based on dynamic routing protocols that calculate the best path to 10.1.1.1 according to route algorithms.
3. The first router that has an explicit route to 10.1.1.1 forwards the requests to the San Jose site.
4. If all servers at the San Jose site fail health checks, the route to 10.1.1.1 is redrawn from San Jose location. Using dynamic routing logic, the routers deduce that the best route for 10.1.1.1 is now in Denver and forward all requests designated to 10.1.1.1 to the Denver site.

Dynamic Protocol Integration

Anycast is based on dynamic routing protocols. Such dynamic protocols are Border Gateway Protocol (BGP) and Open Shortest Path First (OSPF), but can include any other dynamic protocol that is supported in the MX Series router.

The ADC software is integrated with the dynamic protocols by preparing a routing-instance of type forwarding with the virtual IP address (VIP) routes inside it. When a certain VIP is available, the route to this VIP will exist in the routing-instance. This allows the dynamic protocol to publish the VIP as owned by the router.

When the virtual IP address is not available (i.e., all the servers for this VIP are down), the route is redrawn using the routing-instance. This will cause the routing protocol to redraw the route to this IP from its publications. In turn, traffic to this VIP will no longer be routed to this specific router.
To do this, attach the relevant routing-instance to the dynamic protocol configuration according to the desired dynamic protocol.

**Note:** The routing-instance is created by the commit-script automatically and also is used to direct traffic. You should not change the routing-instance or any other automatically created configuration.

**Dynamic Protocol Configuration**

You should configure the routing protocol to advertise routes from the ADC software routing-instance. The relevant routing-instance name is `rdwr-<adc-instance-name>-fe-ri`. For an `adc-instance` with the name "demo", the routing-instance name is `rdwr-demo-fe-ri`. The "fe" stands for front-end, which represents the client-side routes (as opposed to the back-end which represents the server-side routes).

The ADC software uses a few routing instances for internal purposes. For example, for each routing-instance, there is also a back-end routing-instance. In the example above, the back-end routing-instance name is `rdwr-demo-be-ri`. These routing-instances are for internal use only and should not be published with routing protocols. The only routing-instance that should be published is the front-end routing-instance.

Once the routing-instance is identified, you must make sure the relevant routes are indeed published. This is done by importing the relevant routes with a tag and making sure routes with this tag are published. Later, the tag should be removed. This configuration is a Junos OS configuration and does not involve newly-added SLB related configurations, other than the `rdwr-demo-fe-ri` reference.

This is done using the following configuration:

```
[edit]
policy-options {
  policy-statement Policy-To-Add-Tag-To-LBI-Route {
    from instance rdwr-demo-fe-ri;
    then {
      tag add 1;
      accept;
    }
  }
  policy-statement Policy-To-Dynamically-Advertise-LBI-Routes {
    from tag 1;
    then {
      tag subtract 1;
      accept;
    }
  }
}
```

The policy to add the routes and tag should be imported using the following configuration:

```
[edit]
routing-options {
  instance-import Policy-To-Add-Tag-To-LBI-Route;
}
```

The policy to advertise and remove the tag should be used in the desired protocol.
For OSPF, use the following configuration:

```plaintext
[edit]
protocols {
  ospf {
    export Policy-To-Dynamically-Advertise-LBI-Routes;
  }
}
```

For BGP, use the following configuration:

```plaintext
[edit]
protocols {
  bgp {
    export Policy-To-Dynamically-Advertise-LBI-Routes;
  }
}
```

### Understanding Server Load Balancing

Server load balancing (SLB) benefits your network in a number of ways:

- **Increased efficiency for server use and network bandwidth**
  
  With SLB, the ADC software is aware of the shared services provided by your server group and can then balance user session traffic among the available servers. Important session traffic gets through more easily, reducing user competition for connections on overused servers. For even greater control, traffic is distributed according to a variety of user-selectable methods.

- **Increased reliability of services to users**
  
  If any server in a server group fails, the remaining servers continue to provide access to vital applications and data. The failed server can be brought back up without interrupting access to services.

- **Increased scalability of services**
  
  As users are added and the server group’s capabilities are saturated, new servers can be added to the group transparently.

### Identifying Your Network Needs

Server load balancing (SLB) can be the right option for addressing these vital network concerns:

- A single server no longer meets the demand for its particular application.
- When servers hold critical application data and must remain available even in the event of a server failure.
- You want to use multiple servers or hot-standby servers for maximum server uptime.
- You must be able to scale your applications to meet client request capacity.
- You cannot afford to continue using an inferior load-balancing technique, such as DNS round-robin or a software-only system.

### How Server Load Balancing Works

In an average network that employs multiple servers without server load balancing (SLB), each server usually specializes in providing one or two unique services. If one of these servers provides access to applications or data that is in high demand, it can become overused. Placing this kind of
strain on a server can decrease the performance of the entire network as user requests are rejected by the server and then resubmitted by the user stations. Ironically, overuse of key servers often happens in networks where other servers are actually available.

The solution to getting the most from your servers is SLB. With SLB, the ADC software is aware of the services provided by each server. The ADC can direct user session traffic to an appropriate server, based on a variety of load-balancing algorithms. Figure 2 on page 37, illustrates traditional versus SLB network configurations.

**Figure 2: How Server Load Balancing Works**

To provide load balancing for any particular type of service, each server in the group must have access to identical content, either directly (duplicated on each server) or through a back-end network (mounting the same file system or database server).

The ADC software acts as a front-end to the servers, interpreting user session requests and distributing them among the available servers.

**Server Load Balancing on the Juniper Networks Router Introduction**

To deploy server load balancing (SLB), you must first run the ADC software on Multiservices-DPCs on the Juniper Networks router.

On these Multiservices-DPCs, you configure a "load-balancing instance," which is a complete load balancer. You may want to use multiple load-balancing instances, resulting in independent load-balancing systems, each with its set of servers and services.
Within a load-balancing instance, you configure the following basic entities:

- **Real servers**—These are your application servers.
- **Group**—Multiple servers with the same content, so that client requests can be load-balanced between them.
- **Virtual server**—A virtual IP address that accepts client requests.
- **Virtual service**—A service that is being load-balanced across the servers in the group; for example, `dns-virtual-service`. The service belongs to a virtual server, that defines the IP address through which the service is accessible to the client. The service is accessed through one or more predefined application ports (TCP or UDP).

### Implementing Server Load Balancing

Consider a situation where customer DNS traffic is relatively static and is kept on an NFS server for easy administration. As the customer base increases, the number of simultaneous DNS requests also increases, as shown in [Figure 3 on page 38](#).

![Figure 3: Static DNS Traffic Increases](image)

Such a customer has three primary needs:

- Increased server availability
- Server performance scalable to match new demands
- Easy administration of network and servers
All these issues can be addressed by adding the ADC software on your Juniper Networks router, as shown by Figure 4 on page 39.

- Reliability is increased by providing multiple paths from the clients to the ADC software and by accessing a group of servers with identical content. If one server fails, the others can take up the additional load.
- Performance is improved by balancing the request load across multiple servers. More servers can be added at any time to increase processing power.
- For ease of maintenance, servers can be added or removed dynamically, without interrupting shared services.

**Network Topology Requirements**

When deploying server load balancing (SLB), there are a few key aspects to consider:

- In standard SLB, all client requests to a virtual-server IP address and all responses from the real servers must pass through the router, as shown in Figure 5 on page 40. If there is a path between the client and the real servers that does not pass through the router, Direct Server Return mode can be used (see Direct Server Return). Alternately, client Network Address Translation (NAT) IP addresses can be used to enforce the return path via the router.
• Identical content must be available to each server in the same group. Either of the following methods can be used:
  — Static applications and data are duplicated on each real server in the group.
  — Each real server in the group has access to the same data through use of a shared file system or back-end database server.

• Clients and servers can be connected through the same router port. Each port in use on the router can be configured to process client requests, server traffic, or both:
  — Client-facing interfaces—Router ports through which client requests to the virtual server are received.
  — Server-facing interfaces—Router ports to which servers are connected (directly or through routing). Responses to clients are received on the router through these ports.
Consider the network topology in Figure 6 on page 41:

**Figure 6: Load-Balancing to a DNS group**

The device load-balances traffic to a Domain Name System (DNS) server group. The device interfaces connected to the DNS server group (interfaces ge-0/0/1, ge-0/0/2, and ge-0/0/3) are server-facing and interface ge-0/1/1 is client-facing.

**Server Load Balancing Configuration Basics**

This section describes the steps for configuring the ADC software. In the following procedure, many of the server load balancing (SLB) options are left to their default values. See Server Load Balancing Options for more options. Before you start configuring, you must be connected to the router CLI as the administrator.

**To configure server load balancing**

1. Configure a load-balancing instance.
   a. Define service-interfaces (Multiservices-DPC NPUs to run this instance of the load-balancing module).

   **Note:** You must set the required units on each Multiservices-DPC NPU interface that runs the load-balancing module; for example, "unit 0;".

   b. Define client-facing interfaces.
c. Define interfaces.

**Note:** Service interfaces use the interface naming structure (ms-1/1/0), while both client- and server-interfaces show the interface with the unit (ge-1/1/1.0).

```
[edit extensions]
adc {
    adc-instance <instance name> {
        router-interfaces {
            ms-interfaces {
                <ms-x/y/z | rmsX>;  
            }
            client-facing {
                <ge-x/y/z.#>;  
            }
            server-facing {
                <fe-x/y/z.#>;  
            }
        }
    }
}
```

2. Define each real server. Make sure the servers are connected via a router interface that is defined as a server-facing interface for the adc-instance.

   For each real server, you must assign a real-server name and specify its actual IP address.

```
[edit extensions adc adc-instance demo1]
real-servers {
    real1 {
        address 100.100.100.1;
    }
    real2 {
        address 100.100.100.2;
    }
}
```

3. Define the group and assign real servers to it.

   The real servers in any given group must have an IP address accessible to the module that performs the SLB functions. This IP routing is most easily accomplished by placing the servers on a network local to the router. Routing to the server can be used as long as it does not violate the topology rules outlined in Network Topology Requirements.

```
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        real-servers [real1 real2];
    }
}
```

4. Define the virtual server and virtual service.
The virtual server defines the IP address to which client requests are sent. The virtual service
defines a destination port within the virtual-server IP address. The virtual service configuration
includes parameters relevant to the processing of client requests to this service. The service is
actually provided by the real servers in the group defined in the virtual service.

```
[edit extensions adc adc-instance demo1]
virtual-server virt1 {
    address 120.10.10.10;
    dns-virtual-service dns1 {
        groups group1;
    }
}
```

5. In order to send health checks to the real servers, you must define an IP address to be used as
the source IP address for the health check traffic.

```
[edit extensions adc adc-instance demo1]
health-check-source {
    unit 0 {
        family {
            inet {
                address { 200.0.0.1; }
            }
        }
    }
}
```

6. Run a commit to save all changes to the router and begin using the new software package.

```
user@host> commit
```

Server Load Balancing Options

In the procedures discussed in Configuring a Juniper Networks Device for Server Load Balancing,
many of the server load balancing (SLB) options are left at their default values. Use the following
configuration options to customize SLB in the ADC software:

- Supported Services and Applications
- Disabling and Enabling Real Servers
- Health Checks for Real Servers
- Configuring Multiple Services in the Same Group
- Load-Balancing Methods for Real-Server Groups
- Weights for Real Servers
- Maximum Connections for Real Servers
- Backup Servers and Overflow Configuration
- Backup Preemption
- Server Warm-Up Time
• Direct Server Return
• Fast Load Balancing
• Session Timeout per Service
• SYN Protection

Supported Services and Applications

Each virtual server can be configured to support up to 8 service ports and is limited to a total of 1023 services per router. If more than eight service ports are required for a virtual address, you can define multiple virtual servers with the same address. The `protocol` command specifies whether this virtual service is a TCP or UDP application. The `port` command specifies the application port for this application.

In order to load-balance traffic to IP address 1.100.100.100 to UDP port 777, use the following configuration.

![Example Protocol configuration](image)

Table 2 on page 44 shows some common application ports and the applications that use them.

<table>
<thead>
<tr>
<th>Number</th>
<th>TCP/UDP Applications</th>
<th>Number</th>
<th>TCP/UDP Application</th>
<th>Number</th>
<th>TCP/UDP Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>ftp-data</td>
<td>80</td>
<td>http</td>
<td>389</td>
<td>ldap</td>
</tr>
<tr>
<td>21</td>
<td>ftp</td>
<td>109</td>
<td>pop2</td>
<td>443</td>
<td>https</td>
</tr>
<tr>
<td>22</td>
<td>ssh</td>
<td>110</td>
<td>pop3</td>
<td>554</td>
<td>rtsp</td>
</tr>
<tr>
<td>23</td>
<td>telnet</td>
<td>119</td>
<td>nntp</td>
<td>1812</td>
<td>RADIUS authentication</td>
</tr>
<tr>
<td>25</td>
<td>smtp</td>
<td>143</td>
<td>imap</td>
<td>1813</td>
<td>RADIUS accounting</td>
</tr>
<tr>
<td>69</td>
<td>tftp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Disabling and Enabling Real Servers

If you need to reboot a server, make sure that new sessions are not sent to the real server and that current sessions are not discarded before shutting down the server.

`user@host> request extensions adc real-server disable <server name>`
When the current session count on your server falls to zero, you can reboot your server.

- When using cookie persistency on the real server, you can use the command with the `allow cookie` option to allow new connections with existing cookie information to be assigned to a disabled server:

```bash
user@host> request extensions adc real-server disable <server name> allow-cookie
```

When the current session count on your server falls to zero and persistent sessions for the real server have aged out (see the persistence parameters you have set for this real server), you can shut down your server. For more information, see Persistency.

- When you want to disable the server immediately and without allowing existing connections to time out, use the following command:

```bash
user@host> request extensions adc real-server disable <server name> force
```

- When maintenance is complete, use the following command to enable the real server:

```bash
user@host> request extensions adc real-server enable <server name>
```

The router resumes assignment of connections to this real server immediately.

## Health Checks for Real Servers

Determining the health of each real server is a necessary function for server load balancing (SLB). By default for all services, the router pings servers to determine their status.

The router checks each service on each real server every five seconds. If the real server is busy processing connections, it may not respond to a health check. By default, if a service does not respond to four consecutive health checks, the load-balancing module declares the service unavailable. As shown below, the health check interval and the number of retries can be changed:

### Example Sample health check configuration

```bash
[edit extensions adc adc-instance demo1]
real-servers {
    <real-server-name> {
        health-check {
            interval <number> seconds;
            failure-retries <number>;
            recovery-retries <number>;
        }
    }
}
```

For more complex health checking strategies, see Real-Server Health Check Configuration in the Health Checking chapter.
Configuring Multiple Services in the Same Group

When you configure multiple services in the same group, their health checks are dependent on each other. If a real server fails a health check for a service, then the status of the real server for the second service appears as “blocked.”

- **Independent services**—If you are configuring two independent services such as FTP and SMTP—where the real-server failure on one service does not affect other services that the real server supports—then configure two groups with the same real servers, but with different services. If a real-server configured for both FTP and SMTP fails FTP, the real-server is still available for SMTP. This allows the services to act independently even though they are using the same real servers.

- **Dependent services**—If you are configuring two dependent services such as HTTP and HTTPS—where the real-server failure on one service blocks the real server for other services—then configure a single group with multiple services. If a real-server configured for both HTTP and HTTPS fails for the HTTP service, then the server is blocked from supporting any HTTPS requests. The load-balancing module blocks HTTPS requests, (even though HTTPS has not failed) until the HTTP service becomes available again. This helps in troubleshooting so you know which service has failed.

Load-Balancing Methods for Real-Server Groups

Load-balancing methods are used for selecting which real-server in a group receives the next client connection. The available metrics include hash, least connections, round-robin, response (response time), and bandwidth.

Example Group metric configuration

```plaintext
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    load-balance-method {
      round-robin;
    }
  }
}
```

**Hash**

The hash load-balancing method uses IP address information in the client request to select a server. For virtual-services, the client source IP address is used. All requests from a specific client are sent to the same server. This is useful for applications where client information must be retained between sessions.

When selecting a server, a mathematical hash of the relevant IP address information is used as an index into the list of currently available servers. Any given IP address information will always have the same hash result, providing natural persistence, as long as the server list is stable. When a configured server becomes unavailable, clients bound to operational servers will continue to be bound to the same servers for future sessions and clients bound to unavailable servers are rehashed to select an operational server.

Some services allow you to hash using the client-ip and port. This is done using the source-port-in-hash parameter. There are more hash options in filters, that are set using the load-balancing-hash parameter. For more information on filters, see Filtering, page 131.
**Least Connections**

With the least-connections load-balancing method, the number of connections currently open on each real server is measured in real time. The server with the fewest current connections is considered to be the best choice for the next client connection request.

This option is the most self-regulating, with the fastest servers typically getting the most connections over time.

**Round-Robin**

With the round-robin load-balancing method, new connections are issued to each server in turn; that is, the first real server in the group gets the first connection, the second real server gets the next connection, followed by the third real server, and so on. When all the real servers in this group have received at least one connection, the issuing process starts over with the first real server.

**Response Time**

The response-time load-balancing method uses real-server response time to assign sessions to servers. The response time between the servers and the load-balancing module is used as the weighting factor. The router monitors and records the amount of time it takes for each real server to reply to a health check to adjust the real-server weights. The weights are adjusted so they are inversely proportional to a moving average of response time. In such a scenario, a server with half the response time as another server receives a weight twice as large.

**Note:** The effects of the response-time or bandwidth weighting apply directly to the real servers and are not necessarily confined to the group. When response-time or bandwidth-metered real servers are also used in other groups that use the least-connections, round-robin, or hash methods, the response-time or bandwidth weights are applied on top of the method calculations for the affected real servers. Since the response-time or bandwidth weight changes dynamically, this can produce fluctuations in traffic distribution for the groups that use the least-connections, round-robin, or hash load-balancing methods.

By default, the value for each server in groups that use the load-balancing method response-time or bandwidth is updated every 60 seconds. To change the frequency of these checks, use the following adc-instance command:

```
[edit extension adc adc-instance demo1]
group-updates-interval <1-256 seconds>
```

**Bandwidth**

The bandwidth load-balancing method uses real-server octet counts to assign sessions to a server. The load-balancing module monitors the number of octets sent between the server and the module. Then, the real-server weights are adjusted so they are inversely proportional to the number of octets that the real server processes during the last interval.

Servers that process more octets are considered to have less available bandwidth than servers that have processed fewer octets. For example, the server that processes half the amount of octets over the last interval receives twice the weight of the other servers. The higher the bandwidth used, the smaller the weight assigned to the server. Based on this weighting, the subsequent requests go to the server with the highest amount of free bandwidth. These weights are automatically assigned.
The bandwidth metric requires identical servers with identical connections.

**Note:** The effects of the response-time or bandwidth weighting apply directly to the real servers and are not necessarily confined to the group. When response-time or bandwidth-metered real servers are also used in other groups that use the least-connections, round-robin, or hash methods, the response-time or bandwidth weights are applied on top of the method calculations for the affected real servers. Since the response-time or bandwidth weight changes dynamically, this can produce fluctuations in traffic distribution for the groups that use the least-connections, round-robin, or hash load-balancing methods.

By default, the value for each server in groups that use the load-balancing method `response-time` or `bandwidth` is updated every 60 seconds. To change the frequency of these checks, use the following `adc-instance` command:

```
[edit extension adc adc-instance demo1]
group-updates-interval <1-256 seconds>;
```

### Weights for Real Servers

Weights can be assigned to each real server. These weights can bias load balancing to give the fastest real servers a larger share of connections. Weight is specified as a number from 1 to 48. Each increment increases the number of connections the real server gets. By default, each real server is given a weight setting of 1. A setting of 10 assigns the server roughly 10 times the number of connections as a server with a weight of 1.

#### Example Weight configuration

```
[edit extensions adc adc-instance demo1]
real-servers {
    real1 {
        weight 10;
    }
}
```

**Note:** The effects of the response-time or bandwidth weighting apply directly to the real servers and are not necessarily confined to the group. When response-time or bandwidth-metered real servers are also used in other groups that use the least-connections or round-robin methods, the response-time or bandwidth weights are applied on top of the least-connections or round-robin calculations for the affected real servers. Since the response-time or bandwidth weight changes dynamically, this can produce fluctuations in traffic distribution for the groups that use the least-connections or round-robin load-balancing methods.

### Readjusting Server Weights Based on SNMP Health Check Response

The ADC software can be configured to dynamically change weights of real servers based on a health check response using the Simple Network Management Protocol (SNMP).
To enable dynamic assignment of weights based on the response to an SNMP health check

```plaintext
[edit extensions adc adc-instance demo1]
group MyGroup {
    health-check {
        snmp {
            adjust-server-weight;
        }
    }
}
```

For more information on configuring SNMP health checks, see [SNMP Health Check](#).

### Connection Timeouts for Real Servers

In some cases, open TCP/IP sessions might not be closed properly (for example, the ADC receives the SYN for the session, but no FIN is sent). If a session is inactive for 10 minutes (the default), it is removed from the connection table in the load-balancing module.

**Example Timeout configuration**

```plaintext
[edit extensions adc adc-instance demo1]
real-servers {
    real1 {
        connection-timeout 4;
    }
}
```

### Maximum Connections for Real Servers

You can set the number of open connections each real server is allowed to handle.

**Example Connection limit configuration**

```plaintext
[edit extensions adc adc-instance demo1]
real-servers {
    real1 {
        max-connections 1600;
    }
}
```
Values average from approximately 500 HTTP connections for slower servers to 1500 for quicker, multiprocessor servers. The appropriate value also depends on the duration of each session and how much CPU capacity is occupied by processing each session. Connections that use a lot of Java or CGI scripts for forms or searches require more server resources and thus a lower max-connections limit. You may wish to use a performance benchmark tool to determine how many connections your real servers can handle.

A server reaches overflow when current connections to the server reach the max-connections limit, at which time the load-balancing module no longer sends new connections to the server. When the server drops back below the max-connections limit, and the overflow is relieved, new sessions are again allowed.

**Note:** When a backup server is set for a real server, the backup server can be activated when the real server reaches an overflow state.

**Unlimited Connections to Real Servers**

This feature allows an unlimited number of connections to be allocated to traffic accessing a real server. The CLI specifies a range of 0 to 200,000 connections per real server. When the max-connections parameter is not set for a real server, the specified real server can handle up to its maximum connection capacity, or that of the load-balancing module.

**Backup Servers and Overflow Configuration**

A real server can be configured to back up other real servers or server groups in case of server failure and can be configured to handle overflow traffic when the maximum connection limit is reached. Each backup real server must be assigned a real-server number and real-server IP address. Finally, the backup server must be assigned to each real server or server group that it will back up.

**Note:** When a server is configured to back up a group, the backup server is always used in cases where all servers in the group are in overflow.
Example

A Define real server 4 as a backup/overflow for real servers 1 and 2.

```
[edit extensions adc adc-instance demo1]
real-servers {
    real4 {
        ip 4.1.1.1;
    }
    real1 {
        backup-real-server real4 {
            use-when-overflow;
        }
    }
    real2 {
        backup-real-server real4 {
            use-when-overflow;
        }
    }
}
```

B A backup/overflow server can be assigned to a group. If all real servers in a group fail or overflow, the backup is activated for this group.

```
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        backup-real-server real4;
    }
}
```

C Groups also can use another group for backup/overflow:

```
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        backup-real-server real1 {
            backup-group group2;
        }
    }
}
```
Groups also can use a secondary group for backup/overflow:

```bash
[edit extensions adc adc-instance demo1]
groups {
group1 {
    backup-group group2;
    secondary-backup group3;
}
}
```

**Backup-Only Server**

Unlike a backup/overflow server, a backup-only server is used to back up real servers only and does not provide an overflow capability. This provides for the enforcement of maximum session capacity while still providing resiliency. In this configuration, if the primary server reaches its maximum session capacity, the backup server does not take over sessions from the primary server. The backup server only comes into play if the primary server fails.

To define real server 4 as a backup-only server for real servers 1 and 2

```bash
[edit extensions adc adc-instance demo1]
real-servers {
    real4 {
        address 4.1.1.1;
    }
    real1 {
        backup-real-server real4 {
            use-when-overflow;
        }
    }
    real2 {
        backup-real-server real4 {
            use-when-overflow;
        }
    }
}
```

**Note:** A group backup is always both for overflow and for failure.

**Backup Preemption**

The ADC software provides support to control preemption of backup when a primary server becomes active.
By default, preempt is enabled. When the primary server becomes active, it displaces the backup server and takes control. When `do-not-preempt` is used, the backup server continues processing requests sent by the ADC software even if the primary server becomes active. During this process, the primary server is operationally disabled and becomes active only if the backup server goes down.

**Example Disabling preemption configuration**

```
[edit extensions adc adc-instance demo1]
real-servers {
    real4 {
        ip 4.1.1.1;
    }
    real1 {
        backup-real-server real4 {
            do-not-preempt;
        }
    }
}
```

**Server Warm-Up Time**

Server warm-up time is an optional service that can be implemented on new real servers. The primary purpose of this service is to avoid sending a high rate of new connections to a new server. When the warm-up time is entered, traffic is throttled and increased gradually until server initialization is complete. Server warm-up time is controlled by setting a time limit that determines the length of the warm-up time period.

Server warm-up time begins when any of the following situations occur:

- Server comes online
- A new real server is added and comes online
- Multiple real servers are in a warm-up time mode

Server warm-up time ends when any of the following situations occur:

- The server warm-up time limit expires
- New real-server load-balancing method weight reaches its target value

Server warm-up time is set at the group level and applies to all real servers in that group.

**Example Enable warm-up time for group1 with a setting of 10 seconds**

```
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        server-warm-up-time 10;
    }
}
```
To remove the warm-up time configuration, remove the `server-warm-up-time` parameter from the group configuration.

Be aware of the following limitations when using this feature:

- Server warm-up time is supported for Layer 4 load balancing only. Content-match configurations are not supported.
- Server warm-up time is only supported for the least-connections load-balancing method.

**Direct Server Return**

Some clients may need the Direct Server Return (DSR) feature, which allows the server to respond directly to the client. This capability is useful for sites where large amounts of data flows from servers to clients, such as with content providers or portal sites that typically have asymmetric traffic patterns.

DSR and content-intelligent Layer 7 routing cannot be performed at the same time because content intelligent routing requires that all frames return to the router for connection splicing.

DSR requires that the server be set up to receive frames that have a destination IP address that is equal to the virtual-server IP address.

**How Direct Server Return Works**

The sequence of steps that are executed in this scenario are shown in [Figure 7 on page 54](#).

**Figure 7: How Direct Server Return Works**

1. A client request is forwarded to the ADC software.
2. No IP address is changed and the router forwards the request to the best server, based on the configured load-balancing method.
3. The server responds directly to the client, bypassing the router, and using the virtual-server IP address as the source IP address.
Example DNS configuration

```bash
[edit extensions adc adc-instance demo1]
groups {
group1 {
direct-server-return;
}
}
```

**Fast Load Balancing**

Traffic to virtual services is managed using the connection table. Each connection is recorded in the table. Usually, the connection table is used both for the request processing and for reply processing. In request processing, the ADC software looks for a corresponding entry to check persistency information, finds the appropriate real-server address and listening port, and uses it to send the request to the server. In reply processing, the ADC software looks for a corresponding entry to know how to change the source address from a real-server address and listening port back to the virtual-server address and service port.

In some cases, faster traffic processing can be achieved by not checking the connection table for the response path, but by using another, more efficient, mechanism for the address and port translation. Using fast load balancing does not completely eliminate connection table use. To work without a connection table, see Per-Packet Load Balancing, page 56.

Fast load balancing is set for virtual services. It is available for many application-specific virtual services.

### Example Setting fast load balancing for plain-virtual-service

```bash
[edit extensions adc adc-instance demo1 virtual-server v1]
plain-virtual-service fast-service {
  fast-load-balancing;
}
```

**Note:** For some applications, fast load balancing cannot be used, due to the required behavior of that application. For example, FTP requires application-level processing for the replies. For such applications, the fast-load-balancing parameter is not available in the corresponding virtual service.

The translation between the server address and listening port to the virtual server address and service port is done according to a static table. This means that the same server with a listening port cannot be related to more than one service. Therefore, there are cases where fast load balancing cannot be used for a service. Such configurations are blocked upon commit.

These cases include:

1. When the service is using a group that is used in other services with the same service port.
2. When the service is using a group that has servers that participate in other groups, that are used in services with the same service port.
3. When the service is using a group that has real servers using explicit listening ports, and the group is used in other services.

4. When some clients require direct access to real servers in conjunction to access to real servers via the service.

5. When delayed bind is used. This includes content match, cookie based persistency, SYN protection, and other application-specific services, such as HTTP connection pooling and LDAP write servers support.

Per-Packet Load Balancing

Traffic to virtual services is usually managed using the connection table. For some cases, it may be required to not use the connection table. This may be the case for simple request-response protocols, typically UDP-based, where no connection persistency is required.

As suggested by its name, in per-packet load balancing each packet arriving to the ADC software is load-balanced separately with no relation to other packets.

The per-packet-load-balancing parameter must be configured with the fast-load-balancing parameter. Since no connections are kept for the group, the default load-balancing method of least-connections cannot be used. You must actively change the load-balancing method in the group to a different method.

To configure per-packet load balancing

```
[edit extensions adc adc-instance demo1 virtual-server v1]
dns-virtual-service UDP-service {
    per-packet-load-balancing;
    fast-load-balancing;
}
```

Per-Packet Load-Balancing Benefits

1. Improved memory use. No memory is required for connection entries for services that use per-packet load balancing.

2. Improved traffic distribution to servers when there are few client addresses and the same source port is used.

Per-Packet Load-Balancing Limitations

There are cases where per-packet load balancing cannot be used for a service. These cases include:

1. When a session consists of more than one packet that must get to the same server.

2. When fast load balancing cannot be used.

3. When Network Address Translation (NAT) usage is needed. NAT cannot be used in conjunction with per-packet load balancing.

   Services that use per-packet load balancing do not use NAT when forwarding traffic to servers, even if client nat was configured for a real server.

Session Timeout per Service

The ADC software implements a feature that allows for the configuration of session timeout based on a service timeout instead of the real-server timeout.
With this feature, by default the timeout value for the service is not set. When the value is not specified, the service uses the real-server timeout value. Once the timeout value for the service is configured, that value is used instead. This is useful in cases where sessions need to be kept alive after their real-server configured timeout expires. For example, an FTP session could be kept alive after its server defined timeout period.

The following is an example of how a timeout of 10 minutes is configured for HTTP on virtual server 1:

**Example Configure http-virtual-service timeout for 10 seconds**

```plaintext
[edit extensions adc adc-instance demo1]
virtual-server <name> {
    http-virtual-service <name> {
        service-timeout 10;
    }
}
```

**SYN Protection**

The SYN protection feature in the load-balancing module prevents SYN denial-of-service (DoS) attacks on the server. DoS attacks occur when the server or router is denied servicing the client because it is saturated with invalid traffic.

Typically, a three-way handshake occurs before a client connects to a server. The client sends out a synchronization (syn) request to the server. The server allocates an area to process the client requests, and acknowledges (ack) the client by sending a syn ack. The client then acknowledges the syn ack by sending an acknowledgement back to the server, thus completing the three-way handshake.

**Figure 8 on page 57** illustrates a classic type of SYN DoS attack. If the client does not acknowledge the server’s syn ack with a data request (req) and, instead, sends another syn request, the server gets saturated with syn requests. As a result, all of the server’s resources are consumed and it can no longer service legitimate client requests.

**Figure 8: DoS SYN Attack Example**
Using ADC software with SYN protection, as illustrated in the above diagram, the ADC software intercepts the client syn request before it reaches the server. The ADC software responds to the client with a syn ack that contains embedded client information. The ADC software does not allocate a session until a valid syn ack is received from the client or the three-way handshake is complete.

**Repelling DoS SYN Attacks with Delayed Binding**

Figure 9 on page 58 illustrates both a normal and a denial of service server response using delayed binding.

**Figure 9: Repelling DoS Attacks**

### Normal Request with Delayed Binding

- Client sends a SYN request
- Device responds with special SYN ACK
- Client sends an ACK or DATA REQ
- Device recognizes valid three-way handshake
- Device sends a SYN request to server
- Server responds with SYN ACK
- Device sends ACK or DATA REQ
- Server responds with DATA and the Device splices connection to client

### DoS SYN Attack with Delayed Binding

- Client sends a SYN request
- Device responds with special SYN ACK
- Client sends new SYN requests
- Device responds with another SYN ACK
- No session entry is made until a valid three-way handshake is complete
- Device and server resources are protected for legitimate requests

Once the ADC software receives a valid ack or data req from the client, the ADC software sends a syn request to the server on behalf of the client, waits for the server to respond with a syn ack, and then forwards the client’s data req to the server. The ADC software delays binding the client session to the server until the proper handshakes are complete.

Thus, with SYN protection, two independent TCP connections span a session: one from the client to the ADC and the second from the ADC to the selected server. The ADC temporarily terminates each TCP connection until content has been received, thus preventing the server from being inundated with syn requests.

**Note:** SYN protection is automatically enabled when content intelligent switching features are used. However, if you are not parsing content, you can explicitly enable SYN protection if desired.
Configuring SYN Protection

To configure your load-balancing module for SYN protection

```plaintext
[edit extensions adc adc-instance demo1]
virtual-server <name> {
    xxx-virtual-service <name> {
        syn-protection;
    }
}
```

Notes:

>>> Enable SYN protection without configuring any HTTP SLB processing or persistent binding types.

>>> SYN protection is only available for content-based services.

To configure SYN protection for cache redirection, see SYN Protection.

Persistency

The ADC software persistence feature ensures that all connections from a specific client session reach the same real server, even when server load balancing (SLB) is used.

Overview of Persistency

In a typical SLB environment, traffic comes from various client networks across the Internet to the virtual-server IP address in the ADC software. The module then load-balances this traffic among the available real servers.

In any authenticated Web-based application, it is necessary to provide a persistent connection between a client and the content server to which it is connected. Because HTTP does not carry any state information for these applications, it is important for the browser to be mapped to the same real server for each HTTP request until the transaction is complete. This ensures that the client traffic is not load-balanced mid-session to a different real server, forcing you to restart the entire transaction.

Persistence-based SLB enables the network administrator to configure the network to redirect requests from a client to the same real server that initially handled the request. Persistence is an important consideration for administrators of e-commerce websites, where a server may have data associated with a specific user that is not dynamically shared with other servers at the site.

In the ADC software, persistence can be based on the following characteristics: source IP address, cookies, and Secure Sockets Layer (SSL) session ID.

Using Source IP Address

TCP/IP session persistence can be achieved by using the source IP address as the key identifier for the connection. Client IP-based persistence is not dependent on the load-balancing method.

There are two major conditions that cause problems when session persistence is based on a packet’s IP source address:
Many clients sharing the same source IP address (proxied clients)—Proxied clients appear to the ADC as a single source IP address and do not take advantage of traffic distribution. When many individual clients behind a firewall use the same proxied source IP address, requests are directed to the same server, without the benefit of load-balancing the traffic across multiple servers. Persistence is supported without the capability of effectively distributing traffic load.

Also, persistence can be broken when there are multiple proxy servers behind the application router performing SLB. The router changes the client’s address to different proxy addresses as attempts are made to load-balance client requests.

Single clients sharing a pool of source IP addresses—When individual clients share a pool of source IP addresses, persistence for any given request cannot be assured. Although each source IP address is directed to a specific server, the source IP address itself is randomly selected, thereby making it impossible to predict which server will receive the request. SLB is supported, but without persistence for any given client.

Cross-Services Source IP Persistency
Sometimes it is required to maintain persistence for both HTTP and Secure Sockets Layer (SSL) sessions.

The persistency client-IP cross-services configuration maintains persistence for the same service across multiple sessions from the same client, or maintains persistence between different services (for HTTP and SSL traffic only) from the same client to map to the same real server, as long as the same group is configured for both services.

Example Cross-services persistency

```
[edit extensions adc adc-instance demo1 virtual-server v1]
http-virtual-service HTTP1 {
    persistency client-ip cross-services;
}
ssl-virtual-service SSL1 {
    persistency client-ip cross-services;
}
```

Using Cookies
Cookies are strings passed via HTTP from servers to browsers. Based on the mode of operation, cookies are inserted by either the ADC module or the server. After a client receives a cookie, a server receives this cookie on the next GET command issued by the client, that allows the server to positively identify the client as the one that received the cookie earlier.

The cookie-based persistence feature solves the proxy server problem and gives better load distribution at the server site. In the ADC module, cookies are used to route client traffic back to the same real server to maintain session persistence.

For more information, see Cookie-Based Persistence.

Using SSL Session ID
The Secure Sockets Layer (SSL) Session ID is effective only when the server is running SSL transactions. Because of the heavy processing load required to maintain SSL connections, most network configurations use SSL only when necessary. Persistence based on the SSL session ID ensures completion of complex transactions in proxy server environments. However, this type of persistence does not scale on servers because of their computational requirements.

For more information, see How SSL Session ID-Based Server Load Balancing Works.
NAT IP Addresses

In complex network topologies, the ADC software functions can be managed using a client Network Address Translation (NAT) IP address on the server-facing interfaces traffic.

When the client requests services from the ADC software virtual server, the client sends its own IP address for use as a return address. If a NAT IP address is configured for the Multiservices-DPC NPU, the ADC software replaces the client’s source IP address with the ADC software NAT IP address before sending the request to the real server. This process is called client NAT.

The real server uses the NAT IP address as the destination address for any response. Load-balancing traffic is forced to return through the ADC software and through the same Multiservices-DPC NPU, regardless of alternate paths. Once the ADC software receives the translated IP address, it puts the original client IP address into the destination address and sends the packet to the client. This process is transparent to the client.

Note: Because requests appear to come from the ADC software NAT IP address rather than the client source IP address, the network administrator should be aware of this behavior during debugging and statistics collection.

The NAT IP address cannot be used with per-packet load balancing.

NAT IP Address Configuration

Network Address Translation (NAT) IP addresses are bound to a Multiservices-DPC NPU. However, you can also specify using the NAT mechanism per server.

Example NAT IP address configuration

```plaintext
[edit extensions ]
adc {
    adc-instance <instance-name> {
        service-interfaces {
            <ms-x/y/z | rmsX> {
                unit <interface-unit-number> {
                    family inet {
                        nat-address <ip>;
                    }
                }
            }
        }
    }
}
```

Range of NAT Addresses to Multiservices-DPC NPU

You can add more than one Network Address Translation (NAT) IP address to a Multiservices-DPC NPU to achieve high concurrent sessions. These NAT IPs are selected in round-robin for the incoming connections.

A maximum of 32 NAT IP addresses can be added.
Example Range of NAT IP address configuration

```
[edit extensions]
adc {
    adc-instance <instance-name> {
        service-interfaces {
            <ms-x/y/z | rmsX> {
                unit <interface-unit-number> {
                    family inet {
                        nat-address-range <ip> to <ip>;
                    }
                }
            }
        }
    }
}
```

Client NAT Configuration per Server

After configuring the Network Address Translation (NAT) addresses on all Multiservices-DPC NPUs, you should configure the servers so that traffic going to them will have client NAT processing. By default, traffic to servers will not use the client NAT mechanism. You should specifically configure each relevant servers to use client NAT processing so that traffic to them will use the NAT mechanism.

Example Configuring a server to use the client NAT addresses

```
[edit extensions]
adc {
    adc-instance <instance-name> {
        real-servers {
            <name> {
                client-nat;
            }
        }
    }
}
```

Client NAT Configuration per Filter

On filter load balancing, the default is a transparent load balancing. This means that when filters are used, traffic going to a server with client NAT enabled will not do client NAT processing and will leave the router with the original client IP as a source IP.

To override this, you should specifically configure the filter to use client NAT.
Note: For TCP-related filters, client NAT can be configured per filter. For UDP-related filters (or when the protocol is not specified), you must configure client NAT and server NAT in order to enable NAT processing. For more information, see Filtering.

**Client NAT Example**

Figure 10 on page 63 and Table 3 on page 63 illustrate a common client NAT setup.

<table>
<thead>
<tr>
<th>Servers</th>
<th>Client NAT Needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>NO</td>
</tr>
<tr>
<td>R2</td>
<td>NO</td>
</tr>
<tr>
<td>R3</td>
<td>NO</td>
</tr>
<tr>
<td>R4</td>
<td>YES</td>
</tr>
</tbody>
</table>

Table 3: NAT Servers Example

If NAT is not used for server R4, traffic from that server is sent directly to the client, while the source IP is the server IP. The client that sent the session to the destination IP VIP will not recognize the session and the session is broken.
Example Configuring the Multiservices-DPC NPU to have NAT addresses

```yaml
[edit extensions adc adc-instance <instance name>]
router-interfaces {
    ms-interfaces {
        <ms-x/y/z | rmsX> {
            unit <interface-unit-number> {
                family inet {
                    nat-address 200.200.200.68;
                }
            }
        }
    }
}
```

Example Configuring the server named R4 to require client NAT processing

```yaml
[edit extensions adc adc-instance demo1]
real-servers {
    R4 {
        client-nat;
    }
}
```

The NAT addresses and client NAT processing are transparent.

**NAT IP Limitations**

Client Network Address Translation (NAT) cannot work together with the `per-packet-load-balancing` parameter. If a service is configured with per-packet load balancing, the NAT configuration is ignored and traffic reaches the server using the original client IP.
Chapter 3 – Load-Balancing Special Services

This chapter discusses server load balancing (SLB) based on special services, such as FTP, TFTP, LDAP, DNS, RTSP, WAP, SIP, SSL, and HTTP, SSL, as well as basic SLB. For information on how to configure your network for SLB, refer back to Server Load Balancing.

The following topics are addressed in this chapter:

- FTP Server Load Balancing
- TFTP Server Load Balancing
- Domain Name System Server Load Balancing
- Real-Time Streaming Protocol Server Load Balancing
- Secure Sockets Layer Server Load Balancing
- Wireless Application Protocol Server Load Balancing
- Session Initiation Protocol Server Load Balancing
- Secure Sockets Layer Server Load Balancing
- HTTP Server Load Balancing
- Windows Terminal Server Load Balancing
- IP (Plain) Server Load Balancing

For additional information on SLB commands, see the ADC Software Reference Guide.

FTP Server Load Balancing

As defined in RFC 959, FTP uses two connections: one for control information and another for data. Each connection is unique. Unless the client requests a change, the server always uses TCP port 21 (a well-known port) for control information, and TCP port 20 as the default data port.

FTP uses TCP for transport. After the initial three-way handshake, a connection is established. When the client requests any data information from the server, it issues a PORT command (such as ls, dir, get, put, mget, and mput) via the control port.

There are two modes of FTP operation, active and passive:

- **Active FTP** — The FTP server initiates the data connection.
- **Passive FTP** — The FTP client initiates the data connection. Because the client also initiates the connection to the control channel, passive FTP mode does not pose a problem with firewalls and is the most common mode of operation.

The ADC software supports both active and passive modes of FTP operation. You can change from active to passive or vice versa in the same FTP session.

This section includes the following topics:

- FTP Network Topology Restrictions
- Configuring FTP Server Load Balancing

FTP Network Topology Restrictions

FTP control and data channels must be bound to the same real server.
Configuring FTP Server Load Balancing

To configure parameters for FTP server load balancing

The following is a list of default parameters for the FTP SLB. These parameters do not need to be configured or set up when using the FTP SLB feature.

- Protocol—TCP
- Port—21
- Data-port—20

1. Set the group parameter to the group of real servers used for this virtual service.
2. Make sure the group used here has the FTP health check parameter set up. For more details, see FTP Server Health Checks.

```
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 1.100.100.100;
    description vs1;
    ftp-virtual-service ftp-vsvc1 {
        group MyGroup;
    }
}
```

To configure additional parameters for FTP server load balancing

1. Configure the service-timeout parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.

2. Configure the persistent-timeout parameter to the amount of time that persistency information is kept, even if no new relevant connections are detected, in minutes. By default, there is no special timeout for persistent connections. The usual service timeout is used.

3. Select the server-listening-port parameter to reflect the real-server listening port for FTP control connections. When this parameter is set, the destination port of client requests is changed before traffic is forwarded to the server. Valid selections are 0 to 65534. A value of 0 indicates using the explicit configuration of listening ports at the real server. The default is 21.

4. Determine if SYN protection is required for this service. If it is, set the syn-protection parameter.
TFTP Server Load Balancing

As defined in RFC 1350, the Trivial File Transfer Protocol (TFTP) can only read and write files from or to a remote server. TFTP uses UDP datagrams to transfer data. A transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes.

Each data packet contains one block of data, and must be acknowledged by an acknowledgment (ack) packet before the next packet can be sent. A data packet of less than 512 bytes signals termination of a transfer.

TFTP SLB server is similar to other types of SLB. It uses a configured SLB method to select the TFTP server. No additional commands are required to load-balance to TFTP servers.

This section includes the following topics:

- Configuring TFTP Server Load Balancing

Configuring TFTP Server Load Balancing

This section describes how to configure the parameters for TFTP server load balancing.
To configure parameters for TFTP server load balancing

**Note:** The following is a list of default parameters for the TFTP SLB. These parameters do not need to be configured or set up when using the TFTP SLB feature.

- Protocol—UDP
- Port—69

1. Set the group parameter to the group of real servers used for this virtual service.
2. Make sure the group used here has the TFTP health check parameter set up. For more details, see TFTP Health Check.

```
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 1.100.100.100;
    tftp-virtual-service tftp-vsvc1 {
        group MyGroup;
    }
}
```

To configure additional parameters for TFTP server load balancing

1. Configure the service-timeout parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.

```
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 1.100.100.100;
    tftp-virtual-service tftp1 {
        group MyGroup;
        service-timeout 10;
    }
}
```

**Lightweight Directory Access Protocol Server Load Balancing**

As defined in RFC 2251, the Lightweight Directory Access Protocol (LDAP) is an application-level protocol between LDAP clients and servers, allowing clients to retrieve LDAP directory entries via the Internet. The client sends a protocol operation request to the server and the server returns a response. If based on TCP, port 389 is used. Once a connection is set up between client and server,
the client issues operations to the server, and the server returns responses to the client. Before LDAP directory operations can be issued, in general a bind operation is issued, in which authorization is sent also.

This section includes the following topics:

- Configuring LDAP Server Load Balancing
- LDAP Operations and Server Types
- LDAP Write

**Configuring LDAP Server Load Balancing**

**To configure LDAP server load balancing**

**Note:** The following is a list of default parameters for the LDAP server load balancing. These parameters do not need to be configured or set up when using the LDAP server load balancing feature.

- Protocol—TCP
- Port—389

1. Configure the four real LDAP servers and their real IP addresses.
2. Configure group 1 for LDAP.
3. Configure the load-balancing method and health-check type to use.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check ldap;
    real-servers [ldap1 ldap2 ldap3 ldap4];
    load-balancing-method round-robin;
  }
}
```

4. Set the **group** parameter to the group of real servers used for this virtual service.
5. Make sure the group used here has the LDAP health check parameter set up. For more details, see **LDAP Health Checks**.

```plaintext
[extensions adc adc-instance demo1 ]
virtual-server virt1 {
  address 20.20.20.20;
  ldap-virtual-service ldap1 {
    group group1;
  }
}
```
To configure additional parameters for LDAP server load balancing

1. Configure the fast-load-balancing parameter, that is the connection table used for requests only.

2. Configure the service-timeout parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.

3. Select the server-listening-port parameter to reflect the real-server listening port for LDAP control connections. When this parameter is set, the destination port of client requests is changed before traffic is forwarded to the server. Valid selections are 0 to 65534. A value of 0 indicates using the explicit configuration of listening ports at the real server. The default is 389.

4. Set the allow-write-servers parameter. This sends LDAP write requests to a dedicated server. This is only required in special cases. See LDAP Operations and Server Types.

5. Determine if SYN protection is required for this service. If it is, set the syn-protection parameter.

For more details, see SYN Protection.

```plaintext
[extensions adc adc-instance demo1]
virtual-server virt1 {
  address 1.100.100.100;
  ldap-virtual-service ldap1 {
    group MyGroup;
    fast-load-balancing;
    service-timeout 200;
    allow-write-servers;
    syn-protection;
  }
}
```

LDAP Operations and Server Types

There are two kinds of LDAP operations: read and write. Clients use read operations to browse a directory on the server, and use write operations to modify a directory on the server. LDAP servers are categorized into two kinds: read and write servers. Read servers only conduct read operations, and write servers perform both read and write operations.

**LDAP Write**

An LDAP connection is set up and bound to a read server. After that, operation frames received by the ADC software are checked (at Content-string) to determine if there are any write operations. The bind and write operation data frames are stored for potential later use. When a write operation arrives, the ADC software disconnects the connection to the read server and reinitiates another connection with the write server without the client’s knowledge. Once the connection is set up with the write server, all the later requests goes to the write server until an unbind request is received by the ADC software. All of these operations occur within one TCP connection.

After the reset is sent to the old server, a connection is set up to the new server. Stored data frames are forwarded to the server. After the write operation is forwarded to the server, the connection is spliced.
Figure 11 on page 71 shows four LDAP servers load-balancing LDAP queries.

**Figure 11: LDAP Query Example**

---

**Domain Name System Server Load Balancing**

In the ADC software, Domain Name System (DNS) load balancing allows you to choose the service based on the two forms of DNS queries: UDP and TCP. This lets the ADC software send TCP DNS queries to one group of real servers and UDP DNS queries to another group of real servers. The requests are then load-balanced among the real servers in that group.

DNS requests and replies are short and frequent. Each DNS request and reply usually consist of one packet. In typical DNS scenarios, the ADC software expects to see many requests and reply events. If the ADC software treats each request and reply as a connection in the connection table, the connection table limit will be quickly reached, and it will not allow the ADC software to load balance more DNS requests. Since there is usually only one request and one reply, the session persistency is not required in most DNS cases.

**Tip:** To load-balance more DNS traffic, you should configure DNS load balancing with the fast-load-balance and per-packet-load-balance parameters, unless content-match load balancing is needed. For more information, see Fast Load Balancing and Per-Packet Load Balancing.
Figure 12 on page 72 shows four real servers load-balancing UDP and TCP queries between two groups.

This section includes the following topics:

- Preconfiguration Tasks
- Configuring UDP-Based DNS Load Balancing
- Configuring TCP-Based DNS Load Balancing
- Content-Based DNS Server Load Balancing
Preconfiguration Tasks

1. Configure the four real servers and their real IP addresses.

   [edit extensions]
   adc {
       adc-instance demo1 {
           real-servers {
               real20 {
                   address 10.10.10.20;
               }
               real21 {
                   address 10.10.10.21;
               }
               real22 {
                   address 10.10.10.22;
               }
               real26 {
                   address 10.10.10.26;
               }
           }
       }
   }

   2. Configure group 1 for UDP and group 2 for TCP.

   [edit extensions adc adc-instance demo1 ]
   groups {
       group1 {
           real-servers [ real20 real21 ];
           health-check dns host <host-name>;
       }
       group2 {
           real-servers [ real22 real26 ];
           health-check dnstcp host <host-name>;
       }
   }

   For more information on configuring health check, see DNS Health Checks.

Configuring UDP-Based DNS Load Balancing

1. Configure a virtual server.
2. Configure the real server and group, and make sure that the group is using the DNS UDP health check.
   For more details, see DNS Health Checks.
3. Set up the DNS service for the virtual server, and add real-server group 1.
By default, the protocol UDP is used.

```
[edit extensions adc adc-instance demo1]
virtual-server virt1 {
    address 20.20.20.20;
    dns-virtual-service dns1 {
        group group1;
        per-packet-load-balancing;
        fast-load-balancing;
    }
}
```

### Configuring TCP-Based DNS Load Balancing

1. Configure a virtual server.
2. Configure a real server and a group, and make sure that the group is using the DNS UDP health check.
   
   For more details, see [DNS Health Checks](#).
3. Set up the DNS service for the virtual server, and select real-server group 2.
4. As this is TCP-based load balancing, make sure to enable TCP DNS queries.

```
[edit extensions adc adc-instance demo1]
virtual-server virt1 {
    address 20.20.20.20;
    dns-virtual-service dns1 {
        group group2;
        protocol tcp;
        fast-load-balancing;
    }
}
```

### Content-Based DNS Server Load Balancing

The Internet name registry has become so large that a single server cannot keep track of all the entries. This is resolved by splitting the registry and saving it on different servers.

If you have large DNS server groups, the ADC software allows you to load-balance traffic based on DNS names. To load-balance DNS names, the hostname is extracted from the query, processed by the regular expressions engine, and the request is sent to the appropriate real server.

For example, consider in [Figure 13 on page 75](#) a DNS server group load balancing DNS queries based on DNS names. Requests with DNS names beginning with A through G are sent to Server 1, DNS names beginning with H through M are sent to Server 2, DNS names beginning with N through T are sent to Server 3, and DNS names beginning with U through Z are sent to Server 4.
To configure DNS server load balancing

1. Before you can configure DNS server load balancing, ensure that the module is configured for basic SLB.
   For information on how to configure your network for SLB, see Server Load Balancing
2. Enable DNS server load balancing.

```
[edit extensions adc adc-instance demo1]
virtual-server virt1 {
    address 20.20.20.20;
    dns-virtual-service dns1 {
        group group2;
    }
}
```
3. Define the hostnames.

```
[edit extensions adc adc-instance demo1]
content-match {
  string a-to-g {
    text-search {
      url-string "[abcdef]+\..com";
    }
  }
  string h-to-m {
    text-search {
      url-string "[hijklm]+\..com";
    }
  }
  string n-to-t {
    text-search {
      url-string "[nopqrst]+\..com";
    }
  }
  string u-to-z {
    text-search {
      url-string "[uvwxyz]+\..com";
    }
  }
}
```

4. Add the defined strings to the real server.

```
[edit extensions adc adc-instance demo1]
real-servers {
  real1 {
    content-strings a-to-g;
  }
  real2 {
    content-strings h-to-m;
  }
  real3 {
    content-strings n-to-t;
  }
  real4 {
    content-strings u-to-z;
  }
}
```

**Note:** If you do not add a defined string (or add the defined string **any**), the server handles any request.
To configure additional parameters for DNS server load balancing

1. Configure the protocol parameter. This is the protocol used for DNS. The default is UDP.
2. Configure the port parameter. This is the application port for the DNS service. The default is 53.
3. Configure the per-packet-load-balancing parameter. This enables per-packet load-balancing with no connection table.
4. Configure the service-timeout parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.
5. Configure the fast-load-balancing parameter, which is the connection table used for requests only.
6. Configure the select-by-content parameter. This sets DNS content-based load balancing.
   a. url—Select the server according to content-strings associated to the servers.
   b. hash-url—Select the server according to the hash of the url string, ensuring URL-to-server persistency.
7. Determine if SYN protection is required for this service. If it is, set the syn-protection parameter.
   For more details, see SYN Protection.
8. Select the server-listening-port parameter to reflect the real-server listening port for DNS control connections. When this parameter is set, the destination port of client requests is changed before traffic is forwarded to the server. Valid selections are 0 to 65534. A value of 0 indicates using the explicit configuration of real-server listening ports. The default is 389.

```
[edit extensions adc adc-instance demo1]
virtual-server virt1 {
  address 20.20.20.20;
  dns-virtual-service dns1 {
    protocol tcp;
    port 53
    group MyGroup;
    per-packet-load-balancing;
    service-timeout 200;
    fast-load-balancing;
    select-by-content;
    syn-protection;
    server-listening-port 80;
  }
}
```

Real-Time Streaming Protocol Server Load Balancing

Real-Time Streaming Protocol (RTSP) is an application-level protocol for control over the delivery of data with real-time properties, as documented in RFC 2326. RTSP is the proposed standard for controlling streaming data over the Internet. RTSP uses the Real-Time Transport Protocol (RTP) to format packets of multimedia content. RTSP is designed to efficiently broadcast audio-visual data to large groups.
Typically, a multimedia presentation consists of several streams of data (for example, video stream, audio stream, and text) that must be presented in a synchronized fashion. A multimedia client such as Real Player or Quick Time Player downloads these multiple streams of data from the multimedia servers and presents them on the player screen.

RTSP is used to control the flow of these multimedia streams. Each presentation uses one RTSP control connection and several other connections to carry the audio/video/text multimedia streams. In this document, the term RTSP server refers to any multimedia server that implements the RTSP protocol for multimedia presentation.

This section includes the following topics:

- How RTSP Server Load Balancing Works
- Supported RTSP Servers
- Configuring RTSP Load Balancing
- Content Intelligent RTSP Load Balancing

### How RTSP Server Load Balancing Works

The objective of RTSP SLB is to intelligently change an RTSP request, and the other media streams associated with a presentation, to a suitable RTSP server based on the configured load-balancing method. Typically, an RTSP client establishes a control connection to an RTSP server over TCP port 554 and the data flows over UDP or TCP. This port can be changed, however.

ADC software supports two content-string methods (URL hashing and URL pattern matching) and all Layer 4 load-balancing methods. This section discusses load-balancing RTSP servers for Layer 4; for information on load-balancing RTSP servers for content-string, see Content Intelligent RTSP Load Balancing.

For information on using RTSP with cache redirection, see RTSP Cache Redirection.

**Note:** This feature is not applicable if the streaming media (multimedia) servers use HTTP protocol to tunnel RTSP traffic. To ensure that RTSP SLB works, make sure the streaming media server is configured for RTSP.

### Supported RTSP Servers

In a typical scenario, the RTSP client issues several sequences of commands to establish connections for each component stream of a presentation. There are several variations to this procedure, depending upon the RTSP client and the server involved. For example, there are two prominent RTSP server and client implementations.

The RTSP stream setup sequence is different for these two servers, and the ADC software handles each differently as shown below:

- **Real Server**
  
  Real Server from RealNetworks Corporation supports both UDP and TCP transport protocols for the RTSP streams. The actual transport is negotiated during the initialization of the connection. If TCP transport is selected, then all streams of data flow in the TCP control connection itself. If UDP transport is chosen, the client and server negotiate a client UDP port, which is manually configurable.
  
  The real media files that the Real Server plays have the extension ".rm", ".ram", or ".smil".

- **QuickTime Streaming Server**
  
  QuickTime Streaming Server from Apple Incorporated supports a QuickTime presentation that typically has two streams and therefore uses four UDP connections exclusively for transport and one TCP control connection. QuickTime clients use a UDP port, which is manually configurable.
  
  The QuickTime files have the extension ".mov".
ADC Software can also support other RTSP-compliant applications such as Microsoft Windows Media Server 9.

**Configuring RTSP Load Balancing**

In this example, the device is load-balancing RTSP traffic between two media server farms, as shown in Figure 14 on page 79. One group of media servers consists of QuickTime servers and the other group of servers consists of RealNetworks servers. Each group has its own virtual-server IP address. For example, three Real Networks servers host media files for Alteon News; similarly, another three QuickTime servers host media files for AlteonNews. The content is duplicated among the servers in each group. Depending on the client request type, the device is configured to load-balance in the following way:

**Figure 14: RTSP Load Balancing**

To configure RTSP server load balancing

The following is a list of default parameters for the RTSP server load balancing. These parameters do not need to be configured or set up when using the RTSP SLB feature.

- **Protocol** — TCP

1. At the device, before you start configuring RTSP load balancing:
   - Connect each QuickTime server to the Layer 2 ADC.
   - Connect each RealNetworks server to the Layer 2 ADC.
   - Configure the IP addresses on all devices connected to the device.
   - Configure the IP interfaces on the ADC.
2. Configure IP addresses for the real servers.

```plaintext
[edit extensions]
adc {
    adc-instance demo1;
    real-servers {
        real1 {
            address 30.30.30.10;
        }
        real2 {
            address 30.30.30.20;
        }
        real3 {
            address 30.30.30.30;
        }
        real4 {
            address 40.40.40.10;
        }
        real5 {
            address 40.40.40.20;
        }
        real6 {
            address 40.40.40.30;
        }
    }
}
```

3. Create a group to support RealNetworks servers.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
    group100 {
        real-servers [ real1 real2 real3 ];
    }
}
```

4. Create a group to support QuickTime servers.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
    group200 {
        real-servers [ real4 real5 real6 ];
    }
}
```

5. Create a virtual server for the RealNetworks servers.

6. Make sure the group used here has the RTSP health check parameter set up.
For more details, see RTSP Health Check.

```
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 30.30.30.100;
    rtsp-virtual-service rtsp1 {
        group group100;
    }
}
```

7. Create a virtual server for the QuickTime servers.
8. Make sure the group used here has the RTSP health check parameter set up.
   For more details, see RTSP Health Check.

```
[extensions adc adc-instance demo1]
virtual-server virt2 {
    address 40.40.40.100;
    rtsp-virtual-service rtsp2 {
        group group200;
    }
}
```

To configure additional parameters for RTSP server load balancing

1. Configure the port parameter. This is the application port for the RTSP service. Valid entries are 9 to 65534. The default port is 554.
2. Configure the service-timeout parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.
3. Configure the persistent-timeout parameter to the amount of time that persistency information is kept, even if no new relevant connections are detected, in minutes. By default, there is no special timeout for persistent connections. The usual service timeout is used.
4. Determine if SYN protection is required for this service. If it is, set the `syn-protection` parameter.
   For more details, see **SYN Protection, page 57.**

```plaintext
[extensions adc adc-instance]
virtual-server virt2 {
  address 40.40.40.100;
  domain-name <name>;
  rtsp-virtual-service rtsp2 {
    port 554;
    group group200;
    service-timeout 20;
    persistent-timeout;
    syn-protection;
  }
}
```

**Content Intelligent RTSP Load Balancing**

The ADC software supports RTSP load balancing based on the URL hash method or string matching to load-balance media servers that contain multimedia presentations. Because multimedia presentations consume a large amount of Internet bandwidth, and their correct presentation depends upon the real-time delivery of the data over the Internet, several media servers contain the same multimedia data.

- Retrieving files from the RealNetworks server group

- Retrieving files from the QuickTime server group
  RTSP://www.AlteonNews.com/*.mov is load-balanced among the Quick Time media servers using virtual IP address 40.40.40.100.

For more conceptual information on RTSP, see **Real-Time Streaming Protocol Server Load Balancing.**

**Figure 15 on page 83** shows two groups of media servers: Group 1 is configured for URL hashing and group 2 is configured for string matching. The media servers are cache servers configured in reverse proxy mode.
URL Hash

Use the URL hash method to maximize client requests to hash to the same media server. The original servers push the content to the cache servers ahead of time. For example, in Figure 15 on page 83, an ISP is hosting audio-video files for Alteon News on media servers 1, 2, 3, and 4. The domain name adcnews.com associated with the virtual IP address 120.10.10.10 is configured for URL hash.

The first request for http://adc.com/saleswebcast.rm hashes to media server 1. Subsequent requests for http://adc.com/saleswebcast.rm from other clients or from client 1 will hash to the same server 1. Similarly, another request for http://adc.com/marketingwebcast.rm may hash to media server 2, provided saleswebcast and marketingwebcast media files are located in the origin servers.

Typically, a set of related files (audio, video, and text) of a presentation are usually placed under the same directory (called container directory). ADC software URL hashing ensures that the entire container is cached in a single cache by using the entire URL to compute the hash value and omitting the extension (for example, .ram, .rm, and .smil) occurring at the end of the URL.
String Matching
Use the string matching option to populate the RTSP servers with content-specific information. For example, you have clients accessing audio-video files on ADC1 and clients accessing audio-video files on ADC2. You can host the ADC1 media files on media servers 5 and 6 and host ADC2 media files on media servers 7 and 8.

To configure content-intelligent RTSP server load balancing
1. Before you start configuring RTSP server load balancing, configure the ADC software for standard SLB, as described in Loading the ADC Software onto a Device, page 21.
2. Configure IP addresses for the real servers.

```plaintext
[edit extensions]
adc {
    adc-instance demo1 {
        real-servers {
            real1 {
                address 10.10.10.1;
            }
            real2 {
                address 10.10.10.2;
            }
            real3 {
                address 10.10.10.3;
            }
            real4 {
                address 10.10.10.4;
            }
            real5 {
                address 10.10.10.5;
            }
            real6 {
                address 10.10.10.6;
            }
            real7 {
                address 10.10.10.7;
            }
            real8 {
                address 10.10.10.8;
            }
        }
    }
}
```

3. Create a group to support RealNetworks servers.
4. Create a group to support QuickTime servers.

```
[edit extensions adc adc-instance demo1]
groups {
    group100 {
        real-servers [real1 real2 real3 real4];
    }
}
```

5. Create a virtual server for group 1 media servers.

Configure a virtual server and select rtsp or port 554 as a service for the virtual server.

```
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 120.10.10.10;
    rtsp-virtual-service rtsp1 {
        group group100;
    }
}
```

6. Configure URL hash-based RTSP load balancing for group 1 servers.

URL hashing maintains persistency by enabling the client to hash to the same media server.

```
[extensions adc adc-instance demo1]
virtual-server virt2 {
    address 40.40.40.100;
    domain-name <name>;
    rtsp-virtual-service rtsp2 {
        port 554;
        group group1;
        select-by-content {
            first-content-term {
                hash-url;
            }
        }
    }
}
```

7. Create another virtual server for group 2 media servers.

Configure a virtual server and select rtsp as a service for the virtual server.

```
[extensions adc adc-instance demo1]
virtual-server virt3 {
    address 40.40.40.101;
    domain-name <name>;
    rtsp-virtual-service rtsp3 {
        port 554;
        group group2;
        select-by-content {
            first-content-term {
                hash-url;
            }
        }
    }
}
```
8. Configure string matching-based RTSP load balancing for group 2 servers.
   
   — Enable content-string pattern matching.

```
[extensions adc adc-instance demo1]
virtual-server virt2 {
  address 120.10.10.20;
  rtsp-virtual-service rtsp2 {
    group group200;
  }
}
```

— Add URL strings.

```
[extensions adc adc-instance demo1]
virtual-server virt2 {
  address 40.40.40.100;
  domain-name <name>;
  rtsp-virtual-service rtsp2 {
    port 554;
    group group1;
    select-by-content {
      url;
    }
  }
}
```

— Add the defined strings to the real servers.

```
[extensions adc adc-instance demo1]
content-match {
  text-search {
    string adc1 {
      url-string "[adc1]+\.mov";
    }
  }
  text-search {
    string adc2 {
      url-string "[adc2]+\.mov";
    }
  }
}
```
Clients retrieving \texttt{RTSP://adcnews.com/saleswebcast.rm} hash to the same media server—1, 2, 3, or 4.

A client request of the form \texttt{RTSP://120.10.10.20/../adc1.mov} is load-balanced between RTSP servers 5 and 6 using string matching. A client request of the form \texttt{RTSP://120.10.10.20/../adc2.mov} is load-balanced between RTSP servers 7 and 8.

\begin{verbatim}
[extensions adc adc-instance demo1]
real-servers {
    [ real1 real2 real3 real4 ] {
        content-strings any;
    }
}
real-servers {
    [ real5 real6 ] {
        content-strings adc1;
    }
}
real-servers {
    [ real7 real8 ] {
        content-strings adc2;
    }
}
\end{verbatim}

To configure additional parameters for content-intelligent RTSP server load balancing

1. Configure the \texttt{port} parameter. This is the application port for the RTSP service. Valid entries are 9 to 65534. The default port is 554.

2. Configure the \texttt{select-by-content} parameter. This sets RTSP content-based load balancing.
   a. \texttt{url}—Select the server according to content-strings associated to the servers.
   b. \texttt{hash-url}—Select the server according to the hash of the url string, ensuring URL-TO-SERVER persistency.

3. Configure the \texttt{service-timeout} parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.

4. Configure the \texttt{persistent-timeout} parameter to the amount of time that persistency information is kept, even if no new relevant connections are detected, in minutes. By default, there is no special timeout for persistent connections. The usual service timeout is used.

5. Set the \texttt{source-port-in-hash} parameter based on the following:
   a. When the group load-balancing method is set to \texttt{hash}, by default the client address is hashed to select a server. This maintains client-IP based persistency.
   b. When client-IP persistency is \texttt{not} required, and to achieve better distribution of connections between servers, set the \texttt{source-port-in-hash} parameter. When set, the client-IP and port are used with the hash function to select a server.

6. Select the \texttt{server-listening-port} parameter to reflect the real-server listening port for RTSP control connections. When this parameter is set, the destination port of client requests is changed before traffic is forwarded to the server. Valid selections are 0 to 65534. A value of 0 indicates using the explicit configuration of listening ports at the real server. The default is 20.
7. Determine if SYN protection is required for this service. If it is, set the `syn-protection` parameter.

For more details, see [SYN Protection, page 57](#).

```plaintext
[extensions adc adc-instance demo1]
virtual-server virt2 {
    address 40.40.40.100;
    domain-name <name>;
    rtsp-virtual-service rtsp2 {
        port 554;
        group group200;
        select-by-content {
            [hash url | url ];
        }
    service-timeout 20;
    persistent-timeout;
    source-port-in-hash;
    server-listening-port 20;
    syn-protection;
}
}
```

## Secure Sockets Layer Server Load Balancing

Secure Sockets Layer (SSL) is a set of protocols built on top of TCP/IP that allows an application server and client to communicate over an encrypted HTTP session, providing authentication, non-repudiation, and security. The SSL protocol **handshake** is performed using clear (unencrypted) text. The content data is then encrypted (using an algorithm exchanged during the handshake) prior to being transmitted.

Using the SSL session ID, the ADC software forwards the client request to the same real server to which it was bound during the last session. Because the SSL protocol allows many TCP connections to use the same session ID from the same client to a server, key exchange must be done only when the session ID expires. This reduces server overhead and provides a mechanism, even when the client IP address changes, to send all sessions to the same real server.

**Note:** The SSL session ID can only be read by the ADC software after the TCP three-way handshake. In order to make a forwarding decision, the ADC software must terminate the TCP connection to examine the request.

Some versions of Web browsers allow the session ID to expire every 2 minutes, thereby breaking the SSL ID persistence. To resolve this issue, use persistency with metric **hash** or **pbind clientip**.

**Note:** The destination port number to monitor for SSL traffic is user-configurable.

This section includes the following topics:

- **How SSL Session ID-Based Server Load Balancing Works**
- **Configuring SSL Server Load Balancing**
How SSL Session ID-Based Server Load Balancing Works

- All SSL sessions that present the same session ID (32 random bytes chosen by the SSL server) are directed to the same real server.
- New sessions are sent to the real server based on the metric selected (hash, round-robin, leastconns, minmisses, response, and bandwidth).
- If no session ID is presented by the client, the ADC software picks a real server based on the metric for the real-server group and waits until a connection is established with the real server and a session ID is received.
- The session ID is stored in a session hash table. Subsequent connections with the same session ID are sent to the same real server. This binding is preserved even if the server changes the session ID mid-stream. A change of session ID in the SSL protocol will cause a full three-way handshake.
- Session IDs are kept until an idle time equal to the configured server timeout (a default of 10 minutes) for the selected real server has expired. Figure 16 on page 89, is an example of persistence based on SSL session ID, as follows:
  a. An SSL Hello handshake occurs between Client 1 and Server 1 via the ADC software.
  b. An SSL session ID is assigned to Client 1 by Server 1.
  c. The ADC software records the SSL session ID.
  d. The ADC software selects a real server based on the existing SLB settings.
  e. As a result, subsequent connections from Client 1 with the same SSL session ID are directed to Server 1.

Figure 16: SSL Session ID-Based SLB

- Client 2 appears to the ADC software to have the same source IP address as Client 1 because they share the same proxy firewall. However, the ADC software does not automatically direct Client 2 traffic to Server 1 based on the source IP address. Instead, an SSL session ID for the new traffic is assigned. Based on SLB settings, the connection from Client 2 is spliced to Server 3.
- As a result, subsequent connections from Client 2 with the same SSL session ID are directed to Server 3.
Configuring SSL Server Load Balancing

To configure SSL server load balancing

The following is a list of default parameters for the Secure Sockets Layer (SSL) SLB. These parameters do not need to be configured or set up when using the SSL SLB feature.

- Protocol — TCP

1. Set the group parameter to the group of real servers used for this virtual service.
2. Make sure the group used here has the SSL health check parameter set up.

For more details, see HTTP Health Checks.

```plaintext
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 1.100.100.100;
    ssl-virtual-service SSL1 {
        group group1;
    }
}
```

To configure additional parameters for SSL server load balancing

1. Configure the port parameter. This is the application port for the DNS service.
2. Set the persistency parameter. This sets the server persistency scheme. By default, no persistency scheme is used. Supported values are:
   a. **client-ip** — The client IP address is used as an identifier, and all connections from the same client are associated with the same real server until the client becomes inactive and the connection is timed out of the persistency binding table. The connection timeout value (set for the real server) is used to control how long these inactive but persistent connections remain associated with the real servers. When the client resumes activity after the connection has timed out, the client is connected to the most appropriate real server based on the load-balancing method.
   b. **client-ip cross-services** — Similar to the client-ip value, this option also guarantees cross-services persistency for HTTP and SSL services when the same group is used. HTTP and SSL traffic coming from the same client IP maps to the same real server irrespective of the load-balancing method used since the services are related.
   c. **ssl-id** — Persistency is based on SSL session IDs.
3. Configure the fast-load-balancing parameter, which is the connection table used for requests only.
4. Configure the service-timeout parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.
5. Set the source-port-in-hash parameter based on the following:
   a. When the group load-balancing method is set to hash, by default the client address is hashed to select a server. This maintains client-IP based persistency.
b. When client-IP persistency is not required, and to achieve better distribution of connections between servers, set the source-port-in-hash parameter. When set, the client-IP and port are used with the hash function to select a server.

6. Configure the persistent-timeout parameter to the amount of time that persistency information is kept, even if no new relevant connections are detected, in minutes. By default, there is no special timeout for persistent connections. The usual service timeout is used.

7. Determine if SYN protection is required for this service. If it is, set the syn-protection parameter.
   
   For more details, see SYN Protection.

8. Select the server-listening-port parameter to reflect the real-server listening port for HTTP control connections. When this parameter is set, the destination port of client requests is changed before traffic is forwarded to the server. Valid selections are 0 to 65534. A value of 0 indicates using the explicit configuration of listening ports at the real server. The default is 20.

```plaintext
extensions adc adc-instance demo1
virtual-server virt1 {
    address 1.100.100.100;
    domain-name <name>;
    ssl-virtual-service ssl1 {
        port 80;
        group group1;
        persistency [client-ip [cross-services] | ssl-id];
        fast-load-balancing;
        service-timeout 10;
        source-port-in-hash;
        persistent-timeout 20;
        syn-protection;
        server-listening-port 777;
    }
}
```

### Wireless Application Protocol Server Load Balancing

The Wireless Application Protocol (WAP) is an open, global specification for a suite of protocols designed to allow wireless devices to communicate and interact with other devices. It empowers mobile users with wireless devices to easily access and interact with information and services instantly by allowing non-voice data, such as text and images, to pass between these devices and the Internet. Wireless devices include cellular phones, pagers, Personal Digital Assistants (PDAs), and other hand-held devices.

WAP supports most wireless networks and is supported by all operating systems—with the goal of interoperability. A WAP gateway translates Wireless Markup Language (WML)—which is a WAP version of HTML—into HTML/HTTP so that requests for information can be serviced by traditional Web servers.

To load-balance WAP traffic among available parallel servers, the ADC software must provide **persistency** so that the clients can always go to the same WAP gateway to perform WAP operation. **Figure 17 on page 92** shows that the user is first authenticated by the remote access server. In this example, the RADIUS servers are integrated with the WAP gateways.
ADC software allows you to configure the device to select a WAP gateway for each client request based on one of the following three methods: static session entry via TPCP, RADIUS snooping, or RADIUS/WAP persistence.

This section includes the following topics:

- WAP SLB with RADIUS Snooping
- WAP Server Load Balancing with RADIUS/WAP Persistence

**WAP SLB with RADIUS Snooping**

RADIUS snooping allows the ADC software to examine RADIUS accounting packets for client information. This information is needed to add to or delete session entries from the module’s session table so that it can perform the required persistency for load balancing. A session entry does not age out. Such an entry, added using RADIUS snooping, will only be deleted using RADIUS snooping. The module load-balances both the RADIUS and WAP gateway traffic using the same virtual-server IP address.

**How WAP Server Load Balancing Works with RADIUS Snooping**

Before the RAS allows the WAP traffic for a user to pass in and out of the gateway, it sends a RADIUS Accounting Start message to one of the RADIUS servers. The ADC software then snoops on the packet to extract the required information. It must know the type of the RADIUS Accounting message, the client IP address, the caller ID, and the user's name. If it finds this information, the ADC software adds a session entry to its session table. If any of this information is missing, the ADC software does not take any action to handle the session entry.
When the client ends the WAP connection, RAS sends a RADIUS Accounting Stop packet. If the ADC software finds the needed information in a RADIUS Accounting Stop packet, it removes the corresponding session entry from its table. The following steps occur for RADIUS snooping:

1. The user is authenticated on dialing.
2. The RAS establishes a session with the client and sends a RADIUS Accounting Start message with the client IP address to the RADIUS server.
3. The ADC software snoops on the RADIUS accounting packet and adds a session entry if it finds enough information in the packet.
4. The ADC software load-balances the WAP traffic to a specific WAP gateway.
5. When the client terminates the session, the RAS sends an Accounting Stop message to the RADIUS server, and the session entry is deleted from the ADC software.

Consider the following items before configuring RADIUS snooping:

— The same virtual-server IP address must be used when load-balancing both the RADIUS accounting traffic and WAP traffic.
— All the RADIUS servers must use the same UDP port for RADIUS accounting services.
— Before a session entry is recorded, WAP packets for a user can go to any of the real WAP gateways.
— If a session entry for a client cannot be added because of resource constraints, the subsequent WAP packets for that client will not be load-balanced correctly. The client will need to drop the connection and then reconnect to the wireless service.
— The persistence of a session cannot be maintained if the number of healthy real WAP gateways changes during the session. For example, if a new WAP server comes into service or some of the existing WAP servers are down, the number of healthy WAP gateway changes and, in this case, the persistence for a user cannot be maintained.
— Persistence cannot be maintained if the user moves from one ISP to another, or if the base of the user's session changes (that is, from CALLING_STATION_ID to USER_NAME, or vice versa). For example, if a user moves out of a roaming area, it is possible that the user's CALLING_STATION_ID is not available in the RADIUS accounting packets. In this case, the ADC software uses USER_NAME to choose a WAP server instead of CALLING_STATION_ID. Thus, persistence cannot be maintained.

Configuring WAP Server Load Balancing Using RADIUS Snooping

To configure parameters for WAP server load balancing

Note: The following is a list of default parameters for the WAP server load balancing. These parameters do not need to be configured or set up when using the WAP server load balancing feature.

- Protocol—UDP

1. Set the radius-group parameter to the group of real servers used for this virtual service. RADIUS-group servers use port 1813.
2. Make sure the group used here has the WAP health check parameter set up. For more details, see WAP Gateway Health Checks, page 198.

```plaintext
[extensions adc adc-instance demo1]
virtual-server virt1 {
  address 25.25.25.25;
  description wap1;
  wap-virtual-service wap1 {
    groups {
      radius-group RADgroup1;
    }
  }
}
```

To configure additional parameters for WAP server load balancing

1. Configure the `wsp-group` parameter. This is the group of real servers for wts load balancing using port 9200.
2. Configure the `wtp-group` parameter. This is the group of real servers for wtp load balancing using port 9201.
3. Configure the `wtls-group` parameter. This is the group of real servers for wts load balancing using ports 9202 and 9203.
4. Configure the `radius-authentication` parameter when RADIUS authentication traffic should be load-balanced to the RADIUS group, in addition to the RADIUS accounting traffic.
5. Configure the `radius-legacy-ports` parameter. This sets the default ports to 1645 for authentication and 1646 for accounting.
6. Configure the `service-timeout` parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.
7. Configure the `persistency wap-radius` parameter. This enables WAP RADIUS persistence on the filter by binding both WAP and RADIUS sessions to the same server. Select a value from 0 to 32768.

For more detail, see [WAP Server Load Balancing with RADIUS/WAP Persistence](#).

```
[extensions adc adc-instance demo1]
virtual-server virt2 {
    address 40.40.40.100;
    description wap1;
    wap-virtual-service wap1{
        groups {
            wsp-group WSPgroup1;
            wtp-group WTPgroup1;
            wtls-group WTLSgroup1;
            radius-group RADgroup1;
        }
        radius-authentication;
        radius-legacy-ports;
        service-timeout 200;
        persistency wap-radius;
    }
}
```

### WAP Server Load Balancing with RADIUS/WAP Persistence

This feature allows for RADIUS and WAP persistence by binding both (RADIUS accounting and WAP) sessions to the same server.

A WAP client is first authenticated by the RADIUS server on UDP port 1812. The server replies with a RADIUS Accept or Reject frame. The ADC software forwards this reply to the RAS. After the RAS receives the RADIUS accept packet, it sends a RADIUS accounting start packet on UDP port 1813 to the bound server. The ADC software snoops on the RADIUS accounting start packet for the "framed IP address" attribute. The "framed IP address" attribute is used to rebind the RADIUS accounting session to a new server.

The following steps occur for RADIUS/WAP persistence:

1. The user is authenticated on dialing.

   The RAS sends a RADIUS authentication request on UDP port 1812 to one of the servers. The ADC software receives the authentication request. If there is no session corresponding to this request, a new session is allocated and the client is bound to a server. The ADC software then relays the authentication request to the bound server.

2. The RAS establishes a session with the client and sends a RADIUS Accounting Start message to the RADIUS server on UDP port 1813.

3. The ADC software snoops on the RADIUS accounting start packet for the "framed IP address" attribute.

   This attribute in a RADIUS accounting packet contains the IP address of the specific client (the IP address of the wireless device).

   **Note:** The RADIUS accounting packet and the RADIUS accounting service must share the same listening port.

4. The "framed IP address" attribute is used to rebind the RADIUS session to a new server.
The ADC software hashes on the framed IP address to select a real server for the RADIUS accounting session. If the "framed IP address" is not found in the RADIUS accounting packet, then persistence is not maintained for the RADIUS/WAP session. The load-balancing method of the real-server group must be hash for RADIUS/WAP persistence.

5. When the client begins to send WAP requests to the WAP gateways on ports 9200 through 9203, a new session is allocated and a server is bound to the WAP session. The RADIUS session and the WAP session are now both bound to the same server because both sessions are using the same source IP address.

Session Initiation Protocol Server Load Balancing

The Session Initiation Protocol (SIP) is an application-level control (signaling) protocol for Internet multimedia conferencing, telephony, event notification, and instant messaging. The protocol initiates call setup, routing, authentication, and other feature messages to endpoints within an IP domain. The SIP is used to locate users (where the caller and called parties are at), determine user capability (what type of protocol TCP, UDP, and other capabilities the user can support), user availability, call setup (how to create the call), and call handling (how to keep the call up and how to bring down the call).

This feature load-balances UDP-SIP proxy servers such as Nortel MCS (Multimedia Communications Server).

This section includes the following topics:

- **SIP Processing**
- **Configuring SIP Server Load Balancing**

SIP Processing

SIP over UDP processing provides the capability to scan and hash calls based on an SIP Call-ID header to an SIP server. The Call-ID uniquely identifies a specific SIP session. Future messages from the same Call-ID is switched to the same SIP server. This involves stateful inspection of SIP messages.

SIP is a text-based protocol with header lines preceding the content. Like HTTP, the first header line has the method specification, followed by other header lines that specify other parameters such as Call-ID and so on.

Configuring SIP Server Load Balancing

**Figure 18 on page 97**, is an example of a device performing UDP-based SIP server load balancing. In this example, three SIP proxy servers are configured in real-server Group 100. The ADC software is configured for SIP service (port 5060) for virtual server 40.40.40.100.
Follow this procedure to configure the topology as illustrated in Figure 18 on page 97:

1. At the device, before you start configuring SIP server load balancing:
   - Connect each SIP proxy server to the device.
   - Configure the IP addresses on all devices connected to the application module.

2. Configure IP addresses for the SIP proxy servers and increase the timeout for idle sessions.
   SIP sessions are quite long and data may be flowing while the signaling path is idle. Because the ADC software resides in the signaling path, it is recommended to increase the real-server session timeout value to 30 minutes (the default value is 10 minutes).

```plaintext
[edit extensions]
adc {
    adc-instance demo1 {
        real-servers {
            real1 {
                address 10.10.10.1;
                connection-timeout 30;
            }
            real2 {
                address 10.10.10.2;
                connection-timeout 30;
            }
            real3 {
                address 10.10.10.3;
                connection-timeout 30;
            }
        }
    }
}
```

3. Create a group to load balance the SIP proxy servers.
4. Define the group method for the server group.
   A UDP-based SIP load-balancing method must be set to hash.

5. Define the health check for the group.
   The health check is UDP for UDP-based SIP load balancing.

   ```
   [edit extensions adc adc-instance demo1]
   groups {
     group100 {
       real-servers [real1 real2 real3];
       load-balance-method hash;
       health-check sip;
     }
   }
   ```

   **Note:** The following is a list of default parameters for the SIP SLB. These parameters do not need to be configured or set up when using the SIP SLB feature.
   - Protocol—UDP and SIP enabled
   - Port—5060

6. Set the group parameter to the group of real servers used for this virtual service.

7. Make sure the group used here has the SIP health check parameter set up. For more details, see DNS Health Checks.

   ```
   [extensions adc adc-instance demo1]
   virtual-server virt1 {
     address 40.40.40.100;
     sip-virtual-service my-sip-service {
       group group100;
     }
   }
   ```

**To configure additional parameters for SIP server load balancing**

1. Configure the `service-timeout` parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.

2. Configure the `persistent-timeout` parameter to the amount of time that persistence information is kept, even if no new relevant connections are detected, in minutes. By default, there is no special timeout for persistent connections. The usual service timeout is used.

**Session Persistency Using the Refer Method**

The refer method of load-balancing SIP servers is required for "call transfer" services. The refer method indicates that the recipient should contact a third party using the contact information provided in the request.
To maintain session persistency, the new request from the recipient to the third party should also hash the same real server. To maintain persistency, whenever the ADC software receives a session configured for the refer method, it creates a persistent session.

When creating a session for a new request, the ADC software looks up the session table and selects the correct real server. If there is a persistent session, then the real server specified in the session entry is used if that real server is up, otherwise the normal min/miss method is used to select the real server.

HTTP Server Load Balancing

The ADC software allows you to use the client IP address to maintain load balancing for both HTTP and HTTPS sessions only. The `persistency client-ip` command maintains load balancing for the same service across multiple sessions from the same client, or maintains it between different services (for HTTP and HTTPS traffic only) from the same client to map to the same server, as long as the same group is configured for both services. In the ADC software, when the load-balancing method is hash, persistence may also be maintained to the real-server port (real-server listening-ports), in addition to the real server.

This section includes the following topics:

- **When to Disable Persistence to the Server Listening Ports**
- **Cookie-Based Persistence**
- **Content Intelligent HTTP Server Load Balancing**

**When to Disable Persistence to the Server Listening Ports**

In cases where two different services, such as TCP and UDP, must both maintain persistence to the same real server.

**Configuring HTTP Server Load Balancing**

To configure HTTP server load balancing

The following is a list of default parameters for the HTTP server load balancing. These parameters do not need to be configured or set up when using the HTTP SLB feature.

- **Protocol—TCP**
  
  1. Set the `group` parameter to the group of real servers used for this virtual service.
  2. Make sure the group used here has the HTTP health check parameter set up.

For more details, see [HTTP Health Checks](#).

```plaintext
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 1.100.100.100;
    http-virtual-service http1 {
        group group1;
    }
}
```
To configure additional parameters for HTTP server load balancing

1. Configure the port parameter. This is the application port for the HTTP service. Valid entries are 9 to 65534. The default is 80.

2. Set the persistency parameter. This sets the server persistency scheme. By default, no persistency scheme is used. Supported values are:
   a. client-ip—The client IP address is used as an identifier, and all connections from the same client are associated with the same real server until the client becomes inactive and the connection is timed out of the persistency binding table. The connection timeout value (set for the real server) is used to control how long these inactive but persistent connections remain associated with the real servers. When the client resumes activity after the connection has timed out, the client is connected to the most appropriate real server based on the load-balancing method.
   b. client-ip cross-services—Similar to the client-ip value, this option also guarantees cross-services persistency for HTTP and Secure Sockets Layer (SSL) services when the same group is used. HTTP and SSL traffic coming from the same client IP maps to the same real server irrespective of the load-balancing method used since the services are related.
   c. ssl-id—Persistency is based on SSL session IDs.

   **Note:** Client-IP persistency is supported only in http-virtual-service and in ssl-virtual-service. For other services, in order to achieve client-ip persistency, use a load-balancing method of hash.

3. Configure the service-timeout parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.

4. Configure the persistent-timeout parameter to the amount of time that persistency information is kept, even if no new relevant connections are detected, in minutes. By default, there is no special timeout for persistent connections. The usual service timeout is used.

5. Configure the hostname parameter. This is the HTTP hostname, which is used with health checks.

6. Set the source-port-in-hash parameter based on the following:
   a. When the group load-balancing method is set to hash, by default the client address is hashed to select a server. This maintains client-IP based persistency.
   b. When client-IP persistency is not required, and to achieve better distribution of connections between servers, set the source-port-in-hash parameter. When set, the client-IP and port are used with the hash function to select a server.

7. Set the add-x-forwarded-for parameter. This parameter inserts an x-forwarded-for header to client requests.

8. Determine if SYN protection is required for this service. If it is, set the syn-protection parameter.
   For more details, see SYN Protection.

9. Select the server-listening-port parameter to reflect the real-server listening port for HTTP control connections. When this parameter is set, the destination port of client requests is changed before traffic is forwarded to the server. Valid selections are 0 to 65534. A value of 0 indicates using the explicit configuration of listening ports at the real server. The default is to use the service port, typically 80.
10. Set the `http-parsing-depth` parameter. This sets the depth of a search within HTTP requests for content strings.

11. Configure the `fast-load-balancing` parameter, which is the connection table used for requests only.

12. Set the `efficient-memory-use` parameter. This accepts a client HTTP request only after the handshake is complete.

   The default behavior provides faster client-response time, but higher memory use. When setting the `efficient-memory-use` parameter, memory use is lower but client-response time is slower. It is recommended to use this configuration only when there are known memory issues.

```plaintext
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 1.100.100.100;
    domain-name <name>;
    http-virtual-service http1 {
        port 80;
        group group1;
        persistency [client-ip [cross-services] |cookie];
        service-timeout 10;
        persistent-timeout 20;
        hostname <name>
        source-port-in-hash;
        add-x-forwarded-for;
        syn-protection;
        server-listening-port 777;
        http-parsing-depth <#>;
        fast-load-balancing;
        efficient-memory-use;
    }
}
```

**Connection Pooling**

The ADC software supports connection pooling to the servers. This feature multiplexes client and server connections and improves the throughput of server load balancing. It also helps the real server to establish and terminate fewer TCP connections.

In a connection-pooled environment, the ADC software supports and maintains a pool of server connections for servicing client connections. When a client requests a connection, the ADC software selects and uses an unused connection from the server pool to service the request. When the client request is complete, the ADC software returns the server connection to the pool and terminates the client connection.

The ADC software only supports this feature when the `syn-protection` parameter is enabled. For more details, see [SYN Protection](#).
To enable connection pooling for an HTTP service

```plaintext
[extensions adc adc-instance dem01]
virtual-server virt1 {
    address 1.100.100.100;
    http-virtual-service http1 {
        connection-pooling;
    }
}
```

## HTTP Persistency

The ADC software supports the following persistency schemes for your HTTP applications:

- **HTTP and HTTPS Persistence Based on Client IP**
- **Cookie-Based Persistence**
  - Insert Cookie Mode
  - Inspect Cookie Mode
  - Rewrite Cookie Mode
- **Content Intelligent HTTP Server Load Balancing**

### HTTP and HTTPS Persistence Based on Client IP

The ADC software allows you to use the client IP address to maintain persistence for both HTTP and Secure Sockets Layer (SSL) (HTTPS) sessions only. The `persistency client-ip` command maintains persistence for the same service across multiple sessions from the same client, or maintains persistence between different services (for HTTP and HTTPS traffic only) from the same client to map to the same server, as long as the same group is configured for both services. In the ADC software, when the load-balancing method is hash, persistence can also be maintained to the real-server port (real-server listening ports), in addition to the real server.

#### When to Use Cross-Services Persistency

There are two instances when you want to use cross-service persistence:

- In cases where two different services, such as TCP and UDP, must both maintain persistence to the same real server.
- When client IP-based persistence is not dependent on the load-balancing metric.

### Cookie-Based Persistency

Cookies are a mechanism for maintaining state between clients and servers. When the server receives a client request, the server issues a **cookie**, or token, to the client, which the client then sends to the server on all subsequent requests. Using cookies, the server does not require authentication, the client IP address, or any other time-consuming mechanism to determine that the user is the same user that sent the original request.

In the simplest case, the cookie may be just a "customer ID" assigned to the user. It might be a token of trust, allowing the user to skip authentication while his or her cookie is valid. It might also be a key that associates the user with additional state data that is kept on the server, such as a
In a more complex application, the cookie might be encoded so that it actually contains more data than just a single key or an identification number. The cookie might contain the user's preferences for a site that allows their pages to be customized. Figure 19 on page 103, is an example of how cookie-based persistence works.

Figure 19: Cookie-Based Persistence

1. User registers to buy an item.
2. Device completes 3-way handshake with client and then forwards HTTP request to cookie server.
3. Web server designated to server cookies records information and sends the client a cookie.
4. Device records or rewrites cookie information based on configuration.
5. Cookie stored on client machine.
6. Client decides to buy item. Cookie information is sent as part of the HTTP request.
7. Based on cookie information, the device redirects to same server or hashes on cookie value.

The following topics discussing cookie-based persistence are detailed in this section:

- Permanent and Temporary Cookies
- Cookie Formats
- Cookie Properties
- Maintaining Persistency for Client Browsers that Do Not Accept Cookies
- Cookie Modes of Operation
- Configuring Cookie-Based Persistence

**Permanent and Temporary Cookies**

Cookies can either be permanent or temporary. A **permanent cookie** is stored on the client's browser, as part of the response from a website's server. It is sent by the browser when the client makes subsequent requests to the same site, even after the browser has been shut down. A **temporary cookie** is only valid for the current browser session. Similar to a Secure Sockets Layer (SSL) session-based ID, the temporary cookie expires when you shut down the browser. Based on RFC 2109, any cookie without an expiration date is a temporary cookie.

**Cookie Formats**

A cookie can be defined in the HTTP header (the recommended method) or placed in the URL for hashing. The cookie is defined as a "Name=Value" pair and can appear along with other parameters and cookies. For example, the cookie "SessionID=1234" can be represented in one of the following ways:

- In the HTTP header:
  
  Cookie: SessionID=1234
  
  Cookie: ASP_SESSIONID=POIUHKJHLKHD
  
  Cookie: name=john_smith

  The second cookie represents an Active Server Page (ASP) session ID. The third cookie represents an application-specific cookie that records the name of the client.

- Within the URL, as a URL parameter:
http://www.mysite.com/reservations?SessionID=1234

**Cookie Properties**
Cookies are configured in the ADC software by defining the following properties:

- **Cookie names of up to 20 bytes.**
- **The offset of the cookie value within the cookie string.**
  For security, the real cookie value can be embedded somewhere within a longer string. The offset directs the ADC to the starting point of the real cookie value within the longer cookie string.
- **Length of the cookie value.**
  This defines the number of bytes to extract for the cookie value within a longer cookie string.
- **Whether to find the cookie value in the HTTP header (the default) or the URL.**
- **Cookie values of up to 64 bytes for hashing.**
  Hashing on cookie values is used only with the passive cookie mode (Inspect Cookie Mode), using a temporary cookie. The ADC mathematically calculates the cookie value using a hash algorithm to determine which real server should receive the request.
- **An asterisk (*) in cookie names for wildcards.**
  For example, Cookie name = ASPsession*

**Maintaining Persistency for Client Browsers that Do Not Accept Cookies**
Under normal conditions, most browsers are configured to accept cookies. However, if a client browser is not configured to accept cookies, you must use **hash** or **persistency client-ip** (for client IP persistence) as the load-balancing method to maintain session persistence.

With cookie-based persistence enabled, session persistency for browsers that do not accept cookies is based on the source IP address. However, individual client requests coming from a proxy firewall will appear to be coming from the same source IP address. Therefore, the requests is directed to a single server, resulting in traffic being concentrated on a single real server instead of load-balanced across the available real servers.

**Cookie Modes of Operation**
The ADC software supports the following modes of operation for cookie-based session persistence: **insert**, **inspect**, and **rewrite** mode. Table 4 on page 104, shows the differences among the modes:

<table>
<thead>
<tr>
<th>Cookie Mode</th>
<th>Configuration Required</th>
<th>Cookie Location</th>
<th>Uses ADC Session Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert cookie</td>
<td>ADC software only</td>
<td>HTTP header</td>
<td>No</td>
</tr>
<tr>
<td>Inspect cookie</td>
<td>Server and ADC software</td>
<td>HTTP header or URL</td>
<td>Yes</td>
</tr>
<tr>
<td>Rewrite cookie</td>
<td>Server and ADC software</td>
<td>HTTP header</td>
<td>No</td>
</tr>
</tbody>
</table>

**Insert Cookie Mode**
In the insert cookie mode, the ADC generates the cookie value on behalf of the server. Because no cookies are configured at the server, the need to install cookie server software on each real server is eliminated.

In this mode, the client sends a request to visit the website. The ADC software performs load-balancing and selects a real server. The real server responds without a cookie. The ADC software inserts a cookie and forwards the new request with the cookie to the client.
Figure 20 on page 105 is an example of insert cookie mode.

Figure 20: Insert Cookie Mode

1. Configure Router Integrated ADC to insert cookie values.
2. Client visits the Web site via HTTP request.
3. Router Integrated ADC selects server via SLB.
4. Server responds without any cookie.
5. Router Integrated ADC inserts a cookie and forwards the request to the client.

Cookie Insert Configuration Options
Currently the configuration options provided are:

- **Cookie-name**—There is no default.
- **Expiration**—If configured, the client sends a cookie only until the expiration date and time. Otherwise, the cookie expires after the current session.
- **Domain-name**—If configured, the cookie scope applies to this domain. When you send GET requests to any page in this domain, the cookie is sent along.
- **Path**—Enter the subset of URLs on the origin server to which this cookie applies. The default is "/", which means the cookie is used for all requests to the hostname.
- **Secure**—When set, the client is required to use a secure connection to obtain content associated with the cookie.
- **When-cookie-missing <select-server | keep-server>**—Determines how to handle subsequent requests with no cookie in a TCP session where a server was already selected. Select-server means a new server is selected for new requests with no cookie. Keep-server means the server that was selected earlier for request in this connection is used. The default value is keep-server.

```plaintext
[edit extensions adc adc-instance demo1 virtual-server virt1]
http-virtual-service my-service {
    persistency cookie {
        cookie-name customer-ID;
        type insert {
            expiration <date MM/dd/yy [@hh:mm] | duration days [:hours [:min]] >;
            domain-name <domain name>;
            path <path>;
            secure;
            when-cookie-missing <select-server|keep-server>;
        }
    }
}
```
Inspect Cookie Mode
In inspect cookie mode, when the client first makes a request, the ADC software selects the server based on the configured load-balancing method. The real server embeds a cookie in its response to the client. The ADC records the cookie value and matches it in subsequent requests from the same client.

**Note:** The inspect cookie mode is recommended for temporary cookies. However, you can use this mode for permanent cookies if the server is embedding an IP address. In this case, a cookie must be eight characters long and every two characters represent one byte of IP address encoded in hexadecimal.

Figure 21 on page 106 is an example of passive cookie mode operation.

**Figure 21: Passive Cookie Mode**

1. Configure server to send cookies.
2. Configure Router Integrated ADC to record cookie values.
3. Client visits the Web site.
4. ADC selects cookie server.
5. Server returns cookie value.
6. ADC registers cookie value and forwards it to the client for use in subsequent requests.

Subsequent requests from Client 1 with the same cookie value is sent to the same real server.

Rewrite Cookie Mode
In rewrite cookie mode, the ADC software generates the cookie value on behalf of the server, eliminating the need for the server to generate cookie values for each client.

Instead, the server is configured to return a special persistence cookie that the ADC software is configured to recognize. The module then intercepts this persistence cookie and rewrites the value to include server-specific information before sending it on to the client. Subsequent requests from the same client with the same cookie value are sent to the same real server.

Rewrite cookie mode requires at least eight bytes in the cookie header.

**Note:** Rewrite cookie mode only works for cookies in the HTTP header, not cookies that appear as URL parameters.
Figure 22 on page 107 is an example of rewrite cookie mode operation.

**Figure 22: Rewrite Cookie Mode**

1. Configure Router Integrated ADC to rewrite cookie values.
2. Client visits the Web site via HTTP request.
3. ADC selects server via SLB.
4. Server HTTP response includes empty cookie.
5. ADC rewrites the cookie to contain a server ID for hashing on subsequent requests.

**Configuring Cookie-Based Persistence**

1. Before you can configure cookie-based persistence, you need to configure the ADC software for basic SLB.
   For information, see [Server Load Balancing](#).
2. Select the appropriate load-balancing method for the real-server group.
   You may want to use hash to allow client IP-based persistency for browsers that do not accept cookies. Otherwise, we recommend the least-connections load-balancing method for HTTP traffic.
3. Enable cookie-based persistence on the virtual server service.
   In this example, cookie-based persistence is enabled for service 80 (HTTP).
   - Cookie-based persistence mode: *insert, inspect, or rewrite*
   - Cookie name
   - Starting point of the cookie value
   - Number of bytes to be extracted
   - Look for cookie in the URI [e | d]
     To look for cookie name/value pair in the URI, enter e to enable this option. To look for the cookie in the HTTP header, enter d to disable this option.
Setting Expiration Timer for Insert Cookie

If you configure for insert cookie persistence mode, then you are prompted for the cookie expiration timer. The expiration timer specifies a date string that defines the valid lifetime of that cookie. The expiration timer for insert cookie can be of the following types:

- **Absolute timer**
The syntax for the absolute timer is `MM/dd/yy[@hh:mm]`. The date and time are based on RFC 822, RFC 850, RFC 1036, and RFC 1123, with the variations that the only legal time zone is GMT. Once the expiration date is met, the cookie is not stored or given out. For example:

```
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 1.100.100.100;
    http-virtual-service http1 {
        port 80;
        group MyGroup;
        persistency cookie {
            cookie-name CookieSession1;
            type {
                insert {
                    expiration 12:31:12@11:59;
                }
            }
        }
    }
}
```

- **Relative timer**
  
  This timer defines the elapsed time from when the cookie was created. The syntax for the relative timer is `days[:hours[:minutes]]`. For example:

```
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 1.100.100.100;
    http-virtual-service http1 {
        port 80;
        group MyGroup;
        persistency cookie {
            cookie-name CookieSession1;
            type {
                insert {
                    expiration 29:11:59;
                }
            }
        }
    }
}
```

The ADC software adds or subtracts hours according to the time zone settings. When the relative expiration timer is used, make sure the `tzone` setting is set correctly. If NTP is disabled, the `tzone` setting will still apply to the cookie mode.

**Note:** If the cookie expiration timer is not specified, the cookie will expire when the user's session ends.
New Configuration options
The new configuration options provided are:

- **Cookie path**
  If configured, the cookie is sent only for URL requests that are a subset of the path; the path defaults to "/".

- **Secure flag**
  If secure flag is set, the client is required to use a secure connection to obtain content associated with the cookie.

**Example 1: Setting the cookie location**
In this example, the client request has two different cookies labeled "UID." One exists in the HTTP header and the other appears in the URI:

```
GET /product/switch/UID=12345678;ck=1234...
Host: www.adc.com
Cookie: UID=87654321
```

- Look for the cookie in the HTTP header (default).
  If you do not set the **look-in-uri** parameter, the ADC software defaults to looking in the header. Without this parameter set, the ADC software will use `UID=87654321` as the cookie.
- Look for the cookie in the URI.

```
user@host# set look-in-uri
```

When the **look-in-uri** parameter is set, the ADC software uses `UID=12345678` as the cookie.

**Example 2: Parsing the cookie**
This example shows three configurations where the ADC software uses the hashing key or wildcards to determine which part of the cookie value should be used for determining the real server. For example, the value of the cookie is defined as follows:

```
>> Cookie: sid=0123456789abcdef; name1=value1;...
```

- Select the entire value of the **sid** cookie as a hashing key for selecting the real server:

```
[edit extensions adc adc-instance demo1 virtual-server virt1]
http-virtual-service {
    persistency cookie {
        cookie-name sid;
        inspect {
            cookie-value-offset 1;
            cookie-value-length 16;
        }
    }
}
```

This command directs the ADC to use the **sid** cookie, starting with the first byte in the value and using the full 16 bytes.
• Select a specific portion of the sid cookie as a hashing key for selecting the real server:

```plaintext
[edit extensions adc adc-instance demo1 virtual-server virt1]
http-virtual-service {
    persistency cookie {
        cookie-name sid;
        inspect {
            cookie-value-offset 8;
            cookie-value-length 4;
        }
    }
}
```

This command directs the ADC to use the sid cookie, starting with the eighth byte in the value and using only four bytes. This uses 789a as a hashing key.

• Using wildcards for selecting cookie names:

```plaintext
[edit extensions adc adc-instance demo1 virtual-server virt1]
http-virtual-service {
    persistency cookie {
        cookie-name ASPSESSIONID*;
        inspect {
            cookie-value-offset 1;
            cookie-value-length 16;
        }
    }
}
```

With this configuration, the ADC will look for a cookie name that starts with ASPSESSIONID. ASPSESSIONID123, ASPSESSIONID456, and ASPSESSIONID789 will all be seen by the ADC as the same cookie name. If more than one cookie matches, only the first one is used.

**Example 3: Using passive cookie mode**

If you are using passive cookie mode, the ADC software examines the server’s Set-Cookie: value and directs all subsequent connections to the server that assigned the cookie.

For example, if Server 1 sets the cookie as "Set-Cookie: sid=12345678," then all traffic from a particular client with cookie sid=12345678 is directed to Server 1.

The following command is used on the ADC software:

```plaintext
[edit extensions adc adc-instance demo1 virtual-server virt1]
http-virtual-service {
    persistency cookie {
        cookie-name sid;
        inspect {
            cookie-value-offset 1;
            cookie-value-length 8;
        }
    }
}
```
Example 4: Using rewrite cookie mode

- Select whether the cookie-value-length should be 8 bytes or 16 bytes.

  If the cookie-value-length parameter is configured to be eight bytes, the ADC software will rewrite the placeholder cookie value with the encrypted real-server IP address.
  
  If the cookie-value-length parameter is configured to be 16 bytes, the ADC software will rewrite the cookie value with the encrypted real-server IP address and virtual-server IP address.
  
  All subsequent traffic from a specific client with this cookie is directed to the same real server and virtual server.

```
[extensions adc adc-instance demo1]
virtual-server virt1 {
  address 1.100.100.100;
  http-virtual-service http1 {
    port 80;
    group MyGroup;
    persistency cookie {
      cookie-name CookieSession1;
      rewrite {
        cookie-value-length <8|16>;
        look-in-uri;
        response-count <1-16>;
        when-cookie-missing <select server|keep server>;
      }
    }
  }
}
```

Server-Side Multi-Response Cookie Search

Cookie-based persistence requires the ADC software to search the HTTP response packet from the server and, if a persistence cookie is found, sets up a persistence connection between the server and the client. The ADC software looks through the first HTTP response from the server. While this approach works for most servers, some customers with complex server configurations might send the persistence cookie a few responses later. In order to achieve cookie-based persistence in such cases, ADC software allows the network administrator to configure the device to search through multiple HTTP responses from the server.

In the ADC software, the network administrator can modify a response counter to a value from 1 to 16. The ADC software will look for the persistence cookie in this number of responses (each of them can be multi-frame) from the server.

Configuring Server-Side Multi-Response Cookie Search

Configure the server-side multi-response cookie search by using the following command:

```
user@host# set response-count <1-16>
```
Proxy Support for Insert Cookie
When the insert cookie persistence mode is used, the ADC software will parse through every HTTP requests within the same TCP connection to look for the configured cookie name to use for persistency. If the client request arrives without a cookie, then the request is forwarded to the existing binded server. When cookie insert persistence mode is used, the ADC software must insert a cookie in the server-returned response for those client requests without a cookie.

If the client request arrives with a cookie, then the cookie is used to check against the persistence binding table.

Content Intelligent HTTP Server Load Balancing
The ADC software allows you to load-balance HTTP requests based on different HTTP header information, such as a "Cookie:" header for persistent load balancing, a "Host:" header for virtual hosting, or a "User-Agent" for browser-smart load balancing.

Note: When content-string load balancing is configured, an ADC software-enabled device does not support IP fragments. If IP fragments were supported in this mode, the device would have to buffer, reassemble, and inspect packets before making a forwarding decision.

Content intelligent HTTP load balancing supports up to two methods for an http-virtual-service with a logical AND or OR between them. It supports the following methods:

- URL-Based Server Load Balancing
- Virtual Hosting
- Cookie-Based Preferential Load Balancing
- URL Hashing for Server Load Balancing
- Header Hash Load Balancing

URL-Based Server Load Balancing
URL-based server load balancing (SLB) allows you to optimize resource access and server performance. Content dispersion can be optimized by making load-balancing decisions on the entire path and filename of each URL.

Note: Both HTTP 1.0 and HTTP 1.1 requests are supported.

For URL matching, you can configure up to 1024 strings comprised of 40 bytes each. Each URL request is then examined against the URL strings defined for each real server. URL requests are load-balanced among multiple servers matching the URL, according to the load-balancing metric configured for the real-server group (hash is the default).

Consider an example like Figure 23 on page 114, where the following criteria are specified for content load balancing:

- Requests with ".cgi" in the URL are forwarded to real servers 3 and 4.
- Requests with the string "images" in the URL are sent to real servers 1 and 2.
- Requests with URLs starting with "/product:" are sent to real servers 2, 3, and 5.
- Requests containing URLs with anything else are sent to real servers 1, 2, 3, and 4. These servers have been defined with the "any" string.
Configuring URL-Based Server Load Balancing

**To configure URL-based server load balancing**

1. Before you can configure string-based server load balancing (SLB), ensure that the ADC software is configured for basic SLB.
   
   For information on how to configure your network for SLB, see [Server Load Balancing](#).

2. Define the content-strings to be used for URL load balancing.

   ```
   [edit extensions adc adc-instance demo1]
   real-servers {
     <name> {
       content-strings [ string-name string-name ];
     }
   }
   ```

   A default string **any** indicates that the particular server can handle all URL or cache requests. See the following examples.
Example 1: String with the forward slash (/)

A string that starts with a forward slash (/), such as "/images," indicates that the server processes requests that begins with the "/images" string only.

The /images string allows the server to process these requests:

- /images/product/b.gif
- /images/company/a.gif
- /images/testing/c.jpg

This string will not allow the server to process these requests:

- /company/images/b.gif
- /product/images/c.gif
- /testing/images/a.gif

Example 2: String without the forward slash (/)

A string that does not start with a forward slash (/) indicates that the server will process any requests that contain the defined string.

The images string allows the server to process these requests:

- /images/product/b.gif
- /images/company/a.gif
- /images/testing/c.jpg
- /company/images/b.gif
- /product/images/c.gif
- /testing/images/a.gif

Example 3: String with the forward slash (/) only

If a server is configured with the load-balance string (/) only, it will only handle requests to the root directory.

The server will process any request to items in the root directory, such as the following:

- /
- /index.htm
- /default.asp
- /index.shtm
3. Configure the strings.

```
[edit extensions adc adc-instance demo1]
content-match {
    string index {
        text-search {
            url-string index.htm;
        }
    }
    string default {
        text-search {
            url-string default.asp;
        }
    }
    string index2 {
        text-search {
            url-string index.shtm;
        }
    }
}
```

4. Configure one or more real servers to support URL-based load balancing.
5. Add the defined strings to the real server.

```
[edit extensions adc adc-instance demo1]
real-server {
    real1 {
        content-strings [index default index2];
    }
}
```

**Note:** If you do not add a defined string (or add the defined string **any**), the server will handle any request.

A server can have multiple defined strings, such as:
- /images
- /sales
- .gif

With these defined strings, this particular server can handle requests that start with `/images` or `/sales`, and any requests that contain `.gif`. 

6. Use URL-based server load balancing on the virtual servers.

[edit extensions adc adc-instance demo1]
http-virtual-service http {
    group group1;
    select-by-content {
        first-content-term {
            url;
        }
    }
}

Statistics for URL-Based Server Load Balancing

To show the number of hits to the server load balancing or cache server

user@host> show extensions adc string-statistics [load-balancing-instance <name>]

Table 5 on page 117 shows sample statistics.

Table 5: String and Hits

<table>
<thead>
<tr>
<th>SLB String</th>
<th>Hits</th>
<th>SLB String</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>any</td>
<td>73881</td>
<td>/xitami</td>
<td>162102</td>
</tr>
<tr>
<td>.gif</td>
<td>0</td>
<td>/manual</td>
<td>0</td>
</tr>
<tr>
<td>/sales</td>
<td>0</td>
<td>.jpg</td>
<td>0</td>
</tr>
</tbody>
</table>

Virtual Hosting

The ADC software allows individuals and companies to have a presence on the Internet in the form of a dedicated website address. For example, you can have a "www.site-a.com" and "www.site-b.com" instead of "www.hostsite.com/site-a" and "www.hostsite.com/site-b."

Service providers, on the other hand, do not want to deplete the group of unique IP addresses by dedicating an individual IP address for each home page they host. By supporting an extension in HTTP 1.1 to include the host header, ADC software enables service providers to create a single virtual-server IP address to host multiple websites per customer, each with their own hostname.

Note: For server load balancing, one HTTP header is supported per virtual server.

The following list provides more detail on virtual hosting with configuration information:

- An HTTP/1.0 request sent to an origin server (not a proxy server) is a partial URL instead of a full URL.

Example

An example of the request that the origin server would see is as follows:

GET /products/2424/ HTTP/1.0User-agent: Mozilla/3.0Accept: text/html, image/gif, image/jpeg
The GET request does not include the hostname. From the TCP/IP headers, the origin server knows the requested hostname, port number, and protocol.

- With the extension to HTTP/1.1 to include the HTTP Host header, the above request to retrieve the URL www.radware.com/products/2424 would look like this:

```plaintext
GET /products/2424/ HTTP/1.1
Host: www.radware.com
User-agent: Mozilla/3.0
Accept: text/html, image/gif, image/jpeg
```

The Host header carries the hostname used to generate the IP address of the site.

- Based on the Host header, the ADC forwards the request to servers representing different customer websites.

- The network administrator must define a domain name as part of the 128 supported URL strings.

- The ADC performs string matching; that is, the string "radware.com" or "http://www.radware.com/" will match "http://www.radware.com/".

**Virtual Hosting Configuration Overview**

The sequence of events for configuring virtual hosting based on HTTP Host headers is as follows:

1. The network administrator defines a domain name as part of the 128 supported URL strings. Both domain names "www.company-a.com" and "www.company-b.com" resolve to the same IP address. In this example, the IP address is for a virtual server on the ADC.

2. "www.company-a.com" and "www.company-b.com" are defined as URL strings.

3. Server Group 1 is configured with Servers 1 through 8. Servers 1 through 4 belong to "www.company-a.com" and Servers 5 through 8 belong to "www.company-b.com."


5. The ADC software inspects the HTTP host header in requests received from the client.
   - If the host header is "www.company-a.com," the ADC directs requests to one of the Servers 1 through 4.
   - If the host header is "www.company-b.com," the ADC directs requests to one of the Servers 5 through 8.

**Configuring the Host Header for Virtual Hosting**

To support virtual hosting, configure the ADC for host header-based load balancing with the following procedure:

1. Before you can configure server load balancing (SLB) host header-based load balancing, ensure that the ADC software is configured for basic SLB.
   
   For information on how to configure your network for SLB, see [Server Load Balancing](#).
2. Turn on URL parsing for the virtual server for virtual hosting.

```
[edit extensions adc adc-instance demo1]
virtual-server <name> {
    http-virtual-service <name> {
        select-by-content {
            first-content-term {
                virtual-hosting;
            }
        }
    }
}
```

3. Define the hostnames.

```
[edit extensions adc adc-instance demo1]
content-match {
    text-search {
        string cust1 {
            url-string www.customer1.com;
        }
    }
    text-search {
        string cust2 {
            url-string www.customer2.com;
        }
    }
    text-search {
        string cust3 {
            url-string www.customer3.com;
        }
    }
}
```

4. Configure the real servers to handle the appropriate load-balancing strings.

To add a defined string:

```
[edit extensions adc adc-instance demo1]
real-servers {
    real1 {
        content-strings [cust1 cust2 cust3];
    }
}
```

Where:

Content-strings are defined for the load-balancing instance and associated to real servers.

**Note:** The server will handle any request if no string or the string any is defined.
Cookie-Based Preferential Load Balancing
Cookies can be used to provide preferential services for customers, ensuring that certain users are offered better access to resources than other users when site resources are scarce. For example, a Web server could authenticate a user via a password and then set cookies to identify them as "Gold," "Silver," or "Bronze" customers. Using cookies, you can distinguish individuals or groups of users and place them into groups or communities that are redirected to better resources and receive better services than all other users.

Cookie-based preferential services enable the following support:
- Redirect higher priority users to a larger server or server group.
- Identify a user group and redirect it to a particular server.
- Serve content based on user identity.
- Prioritize access to scarce resources on a website.
- Provide better services to repeat customers, based on access count.

Clients that receive preferential service can be distinguished from other users by one of the following methods:
- Individual user
  Specific individual users could be distinguished by IP address, login authentication, or permanent HTTP cookie.
- User communities
  Some set of users, such as "Premium Users" for service providers who pay higher membership fees than "Normal Users," could be identified by source address range, login authentication, or permanent HTTP cookie.
- Applications
  Users could be identified by the specific application they are using. For example, priority can be given to HTTPS traffic that is performing credit card transactions versus HTTP browsing traffic.
- Content
  Users could be identified by the specific content they are accessing.

Based on one or more of the criteria above, you can load-balance requests to different server groups.

Configuring Cookie-Based Preferential Load Balancing

To configure cookie-based preferential load balancing

1. Before you can configure cookie-based load balancing, ensure that the ADC software is configured for basic SLB.
   For information on how to configure your network for SLB, see Server Load Balancing.
2. Turn on URL parsing for the virtual server.

```plaintext
[edit extensions adc adc-instance demo1]
virtual-server <name> {
    http-virtual-service <name> {
        port <number>;
        select-by-content {
            first-content-term {
                cookie {
                    cookie-name <name>;
                    cookie-value-offset <1-64>;
                    cookie-value-length <1-64>;
                    look-in-url;
                }
            }
        }
    }
}
```

3. Define the cookie values.

```plaintext
[edit extensions adc adc-instance demo1]
content-match {
    text-search {
        string Gold {
            url-string "Gold";
        }
    }
    text-search {
        string Silver {
            url-string "Silver";
        }
    }
    text-search {
        string Bronze {
            url-string "Bronze";
        }
    }
}
```

Since a session cookie does not exist in the first request of an HTTP session, a default server or any server is needed to assign cookies to a None cookie HTTP request.

**Example**

- Real Server 1: **Gold** handles gold requests.
- Real Server 2: **Silver** handles silver request.
- Real Server 3: **Bronze** handles bronze request.
- Real Server 4: any handles any request that does not have a cookie or matching cookie.

With servers defined to handle the requests listed above, the following happens:
— Request 1 comes in with no cookie; it is forwarded to Real Server 4 to get a cookie assigned.
— Request 2 comes in with "Gold" cookie; it is forwarded to Real Server 1.
— Request 3 comes in with "Silver" cookie; it is forwarded to Real Server 2.
— Request 4 comes in with "Bronze" cookie; it is forwarded to Real Server 3.
— Request 5 comes in with "Titanium" cookie; it is forwarded to Real Server 4, since it does not have an exact cookie match (matches with "any" configured at Real Server 4).

4. Configure the real servers to handle the appropriate load-balance strings.

To add a defined string:

```plaintext
[edit extensions adc adc-instance demo1]
real-server {
 real1 {
    content-strings Gold;
 }
 real2 {
    content-strings Silver;
 }
 real3 {
    content-strings Bronze;
 }
 real4 {
    content-strings any;
 }
}
```

**Note:** If you do not add a defined string (or add the defined string **any**), the server will handle any request.

**Browser-Smart Load Balancing**

HTTP requests can be directed to different servers based on browser type by inspecting the "User-Agent" header. For example:

```
GET /products/2424/ HTTP/1.0
User-agent: Mozilla/3.0
Accept: text/html, image/gif, image/jpeg
```

**To allow the ADC software to perform browser-smart load balancing**

1. Before you can configure browser-based load balancing, ensure that the ADC software is configured for basic server load balancing (SLB).

For information on how to configure your network for SLB, see [Server Load Balancing](#).
2. Turn on URL parsing for the virtual server for the "User-Agent:" header.

```plaintext
[edit extensions adc adc-instance demo1]
virtual-server <name> {
  http-virtual-service <name> {
    port <port>;
    select-by-content {
      first-content-term {
        browser;
      }
    }
  }
}
```

3. Define the hostnames.

```plaintext
[edit extensions adc adc-instance demo1]
content-match {
  text-search {
    string ID {
      url-string "Mozilla";
    }
  }
  text-search {
    string ID2 {
      url-string "Internet Explorer";
    }
  }
  text-search {
    string ID3 {
      url-string "Netscape";
    }
  }
}
```

4. Configure the real servers to handle the appropriate load-balancing strings.

**Note:** If you do not add a defined string (or add the defined string any), the server will handle any request.

Use the following command to add a defined string:

```plaintext
[edit extensions adc adc-instance demo1]
real-servers {
  real1 {
    content-strings ID;
  }
  real2 {
    content-strings [ID2 ID3];
  }
}
```
URL Hashing for Server Load Balancing

By default, hashing algorithms use the IP source address and/or IP destination address (depending on the application area) to determine content location. The default hashing algorithm for server load balancing (SLB) is the IP source address. By enabling URL hashing, requests going to the same page of an origin server are redirected to the same real server or cache server.

Load-Balancing Nontransparent Caches

You can deploy a cluster of non-transparent caches and use the virtual server to load-balance requests to the cache servers. The client's browser is configured to send Web requests to a nontransparent cache (the IP address of the configured virtual server).

If hash is selected as the load-balancing algorithm, the ADC hashes the source IP address to select the server for SLB. Under this condition, the ADC cannot send requests for the same origin server to the same proxy cache server. For example, requests made from a client to http://radwarealteon.com from different clients can get sent to different caches, as seen in Figure 24 on page 124.

Figure 24: URL Hashing for Server Load Balancing

Configuring URL Hashing

You can direct the same URL request to the same cache or proxy server by using a virtual-server IP address to load-balance proxy requests. By configuring hash as the metric, the ADC uses the number of bytes in the URI to calculate the hash key.

If the host field exists and the ADC is configured to look into the Host header, the ADC uses the Host header field to calculate the hash key.

To configure URL hashing

1. Before you can configure URL hashing, ensure that the ADC is configured for basic server load balancing (SLB).

For information on how to configure your network for SLB, see Server Load Balancing.
2. Enable URL hashing.

```plaintext
[edit extensions adc adc-instance demo1]
virtual-server <name> {
    http-virtual-service <name> {
        port <port>;
        group <group>; # this is the only mandatory parameter
        select-by-content {
            first-content-term {
                url hash-length <1-255>;
            }
        }
    }
}
```

Hashing is based on the URL, including the HTTP Host header (if present), up to a maximum of 255 bytes.

3. Set the load-balancing method for the real-server group to hash.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
    <group-name> {
        load-balancing-method hash;
    }
}
```

Header Hash Load Balancing
ADC software allows you to hash on any selected HTTP header.

To configure the ADC software for load balancing based on header hash

1. Ensure that the ADC is configured for basic server load balancing (SLB).
   For information on how to configure your network for SLB, see Server Load Balancing.
2. Enable header hashing.

```plaintext
[edit extensions adc adc-instance demo1]
virtual-server <name> {
    http-virtual-service <name> {
        port <port>;
        group <group>;
        select-by-content {
            first-content-term {
                header-name User-Agent;
                hash-length <1-255>;
            }
        }
    }
}
```
3. Set the load-balancing method for the real-server group to hash.

```
[edit extensions adc adc-instance demo1]
  groups {
    <group-name> {
      load-balancing-method hash;
    }
  }
```

### Inserting the X-Forwarded-For Header in HTTP Requests

The ADC software can insert the inclusion of the X-Forwarded-For header in client HTTP requests in order to preserve client IP information. This feature is useful in proxy mode, where the client source IP information is replaced with the proxy IP address. However, it can also be used for all Layer 4 load balancing in both proxy and non-proxy mode, if there is a need to include the X-Forwarded-For header. This feature is supported at Layer 4 and Content-string.

![Tip Icon]

**To configure the ADC software to insert the X-Forwarded-For header**

1. Ensure that the ADC is configured for basic server load balancing (SLB).
   For information on how to configure your network for SLB, see Server Load Balancing.
2. Enable client Network Address Translation operation mode on the real servers used in load-balancing.

```
[edit extensions]
  adc {
    adc-instance <name>;
    real-servers {
      <real-server-name> {
        client-nat;
      }
    }
  }
```

3. On the virtual server attached to the real servers, enable the X-Forwarded-For header.

```
[edit extensions adc adc-instance demo1]
  virtual-server <name> {
    http-virtual-service <name> {
      add-x-forward-for;
    }
  }
```
Windows Terminal Server Load Balancing

Windows Terminal Services refer to a set of technologies that allow Windows users to run Windows-based applications remotely on a computer running as the Windows terminal server. The ADC software includes load balancing and persistence options aimed specifically at Windows terminal services.

In a load-balanced environment, a group of terminal servers has incoming session connections distributed in a balanced manner across the servers in the group. The Windows session director is used to keeping a list of sessions indexed by username. This allows a user to reconnect to a disconnected user session.

The session director provides functionality that allows a group of terminal servers to coordinate the reconnection of disconnected sessions. The session director is updated and queried by the terminal servers whenever users log on, log off, or disconnect their sessions while leaving their applications active.

The client can be reconnected to the terminal server where the user's disconnected session resides using the routing token information. The session director passes the routing token information to the client with the correct server IP address embedded. The client presents this routing token to the load balancer when it reconnects to the virtual IP address. The load balancer will decipher the token and send the client to the correct terminal server.

In some instances, a dedicated session director may not exist. If this is the case, enable the userhash functionality to perform the terminal server binding operation based on username hashing.

By default, Windows terminal server traffic uses TCP port 3389, but it can configured to work on any non-standard port.

For further information regarding Windows terminal services, see the Microsoft website.

This section includes the following topic:
- Configuring Windows Terminal Server Load Balancing

Configuring Windows Terminal Server Load Balancing

To configure WTS server load balancing

The following is a list of default parameters for the Windows terminal server load balancing. These parameters do not need to be configured or set up when using the WTS load-balancing feature.

- Protocol—TCP
- Port—3389

1. Set the group parameter to the group of real servers used for this virtual service.
2. Make sure the group used has the WTS health check parameter set up.

For more details, see Windows Terminal Server Health Checks.

```bash
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 1.100.100.100;
    wts-virtual-service wts1 {
        group MyGroup;
    }
}
```
To configure additional parameters for WTS server load balancing

1. Select the persistency scheme to use for the Windows terminal server virtual service:
   a. session-directory—Use to manage user assignments to servers.
   b. user-hash—Use when WTS session-director is not used.
2. Configure the service-timeout parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.
3. Select the server-listening-port parameter to reflect the real-server listening port for WTS control connections. When this parameter is set, the destination port of client requests is changed before traffic is forwarded to the server. Valid selections are 0 to 65534. A value of 0 indicates using the explicit configuration of listening ports at the real server. The default is 20.
4. Determine if SYN protection is required for this service. If it is, set the syn-protection parameter.
   For more details, see SYN Protection.

```plaintext
[extensions adc adc-instance demo1]
virtual-server virt1 {
   address 1.100.100.100;
   wts-virtual-service wts1 {
      port 3398;
      group MyGroup;
      persistency [session-directory | user-hash];
      service-timeout 10;
      server-listening-port 20;
      syn-protection;
   }
}
```

**IP (Plain) Server Load Balancing**

IP server load balancing allows you to configure your ADC software for server load balancing based on the client's IP address only. Typically, the client IP address is used with the client port number to produce a session identifier. When the Layer 3 option is enabled, the ADC software uses only the client IP address as the session identifier.

To configure IP server load balancing

1. Set the group parameter to the group of real servers used for this virtual service.
   Make sure the group used has the appropriate health check. For more details about supported health checks, see Health Checking.
2. Set the required protocol and port. The default protocol is TCP.

```plaintext
[extensions adc adc-instance demo1]
virtual-server virt1 {
    address 1.10.10.10;
    plain-virtual-service plain1 {
        protocol udp;
        port 80;
        group MyGroup;
    }
}
```

To configure additional parameters for IP server load balancing

1. Configure the protocol parameter. This is the protocol used for this service. The default is TCP.
2. Configure the port parameter. This is the application port for the service. Valid entries are 9 to 65534.
3. Configure the service-timeout parameter to the amount of time that idle connections should remain in the connection table before being removed, in minutes (0 to 32768). The default, when the parameter is not set, is to use the timeout configured for the real server, typically 10 minutes.
4. Select the server-listening-port parameter to reflect the real-server listening port for IP control connections. When this parameter is set, the destination port of client requests is changed before traffic is forwarded to the server. Valid selections are 0 to 65534. A value of 0 indicates using the explicit configuration of listening ports at the real server.
5. Set the source-port-in-hash parameter based on the following:
   a. When the group load-balancing method is set to hash, by default the client address is hashed to select a server. This maintains client-IP based persistency.
   b. When client-IP persistency is not required, and to achieve better distribution of connections between servers, set the source-port-in-hash parameter. When set, the client-IP and port are used with the hash function to select a server.
6. Determine if SYN protection is required for this service. If it is, set the syn-protection parameter. This parameter is available when the protocol is set to TCP. For more details, see SYN Protection.
7. Configure the fast-load-balancing parameter, which is the connection table used for requests only. Replies are sent to the client without a connection-table lookup. For more details, see Fast Load Balancing.
8. Configure the per-packet-load-balancing parameter. This enables per-packet load balancing with no connection table.

```plaintext
[extensions adc adc-instance demo1]
virtual-server virt1 {
  address 1.10.10.10;
  plain-virtual-service plain1 {
    protocol upd;
    port 27;
    group MyGroup;
    service-timeout 30;
    server-listening-port 20;
    source-port-in-hash;
    syn-protection;
    fast-load-balancing;
    per-packet-load-balancing;
  }
}
```
Chapter 4 – Filtering

This chapter provides a conceptual overview of filters and includes configuration examples showing how filter terms can be used for transparent server load balancing.

The following topics are addressed in this chapter:

- **Filter Overview** — This section describes the benefits and filtering criteria to allow for extensive filtering at the IP and TCP/UDP levels.
  - Filtering Benefits
  - Filter Configuration
  - ADC Filter terms
  - Stacking Filters
  - Overlapping Filters
  - The Default Filter
  - Optimizing Filter Performance
  - Filter Logs
  - Using Per-Packet Load Balancing with Filters
- **Tunable Hash for Filter Redirection** — Allows you to select any hash parameter for filter redirection.
- **Filter-Based Security** — This section provides an example of configuring filters for providing the best security.
- **Deny Filter Based on Layer 7 Content**
- **Cache Server Load Balancing** — This section discusses how filter terms can be created to intercept and load-balance traffic to cache and application servers.
  - Cache Server Load Balancing Overview
  - Cache Redirection Environment
  - RTSP Cache Redirection
  - Excluding Noncacheable Sites
  - Content Intelligent Cache Redirection
  - HTTP Redirection Overview
  - IP-Based HTTP Redirection
  - TCP Service Port-Based HTTP Redirection
  - MIME Type Header-Based Redirection
  - URL-Based Redirection
  - Source IP from HTTP Header and Host Header-Based Redirection
  - HTTP to HTTPS Redirection

**Filter Overview**

The load-balancing module is used to efficiently deliver content and secure your servers from unauthorized intrusion, probing, and denial-of-service (DoS) attacks. The ADC software includes extensive filtering capabilities at the Layer 2 (MAC), Layer 3 (IP), and Layer 4 (TCP/UDP) levels.
Router Interfaces

Traffic coming from client-facing interfaces is matched against filters. Servers must be connected to server-facing interfaces.

The order of the filter term matching process is according to the order the terms appear in the configuration. You can move terms around by using Juniper Networks CLI commands.

The order of matching filter terms between adc-instances is according to where the adc-instances appear in the configuration. Matches in one adc-instance are only compared with subsequent adc-instances in the configuration.

To configure client-facing and server-facing interfaces

```
[edit extensions adc]
router-instances {
    client-facing [ <instance> <instance> ];
    server-facing [ <instance> <instance> ];
}
```

For more information, see Server Load Balancing Configuration Basics.

Filtering Benefits

Filtering gives the network administrator a powerful tool with the following benefits:

- Filtering of Layer 2 non-IP frames—In the Alteon Application Switch Operating System, a filter can specify only source MAC and destination MAC addresses, and capture and apply an allow.
- Increased security for server networks
  Filtering gives the administrator control over the types of traffic permitted through the module. Filters can be configured to allow or deny traffic from Layer 2 to Layer 7: MAC address, IP address, protocol, Layer 4 port, and Layer 7 string or pattern content.
  You can also secure your module from further virus attacks by configuring the module with a list of potential offending string patterns. For more information, see Deny Filter Based on Layer 7 Content.
  Any filter can be configured to generate system log messages for increased security visibility.
- Used to map the source or destination IP addresses and ports
  Generic Network Address Translation (NAT) can be used to map the source or destination IP addresses and the ports of private network traffic to or from advertised network IP addresses and ports.

Filter Configuration

ADC filter terms are an ordered list of terms. Each filter term is composed from a match clause (ADC Filter Terms—"from" Clause) that defines the match criteria, and a then clause (ADC Filter Terms—"then" Clause) that defines the action and behavior with traffic that matches the term.

ADC Filter terms

The term name can contain letters, numbers, and hyphens (-) and can be up to 255 characters long. To include spaces in the name, enclose the entire name in quotation marks (" ").
Each term name must be unique within a filter.

You can specify multiple terms in the ADC filter, effectively chaining together a series of match-action operations to apply to the packets. You can also use the go-to action so that, when a match condition is met, the evaluation continues from the go-to term, rather than terminating.

ADC filter terms are evaluated in the order in which you specify them in the configuration. To reorder terms, use the configuration mode insert command. For example, the command insert term up before term start places the term up before the term start.

Up to 2048 filter terms can be configured on the module. Descriptive names can be used to define filter terms. Each filter can be set to perform from or then actions, based on any combination of the filter options.

**ADC Filter Terms—“from” Clause**

In the from statement in the ADC filter term, you specify conditions that the packet must match for the action in the then statement to be taken. All conditions in the from statement must match for the action to be taken. The order in which you specify match conditions is not important, because a packet must match all the conditions in a term for a match to occur.

If you specify no match conditions in a term, that term matches all packets.

In the from clause you can indicate Layer 4 information to match traffic:

- **source-address**—Source IP address or range.
- **destination-address**—Destination IP address or range (dip and dmask).
- **protocol <tcp | udp>**—Match using either TCP or UDP protocol. By default, both are matched.
- **source-port**—TCP/UDP application or source port or source port range (such as 31000 to 33000).

**Note:** The service number specified on the module must match the service specified on the server.

- **destination-port**—TCP/UDP application or destination port or destination port range (such as 31000 to 33000).

**Note:** Advanced filtering options such as TCP flags are available.

Using these filter criteria, you could create a single filter that blocks external Telnet traffic to your main server except from a trusted IP address. Another filter could warn you if FTP access is attempted from a specific IP address. Another filter could redirect all incoming e-mail traffic to a server where it can be analyzed for spam. The options are nearly endless.

**ADC Filter Terms—“then” Clause**

A filter term then statement instructs the filter what to do once the filtering criteria are matched. These actions are defined in the then clause of the filter term.

You can specify one of the following filter actions:

- **accept**—Allows the frame to pass (by default). It is processed according to its destination: either handled by ADC virtual services or by the router and sent to its destination.
- **discard**—Discards frames that fit this filter’s profile. They are not processed further.
• **go-to term**—Match to the specified term and continue classification from there.

  **Note:** The target term must appear further down the list than the currently evaluated term.

• **http-redirect**—Allows you to specify a target term name that the filter search should jump to when a match occurs. The http-redirect causes filter processing to jump to a designated filter, effectively skipping over a block of filter terms. Filter searching then continues from the designated filter term. To specify the new filter, use the http-redirect command. For more information, see HTTP Redirection.

• **load-balance**—Redirects frames that fit this filter’s profile, such as for web cache redirection. In addition, Layer 4 processing must be used.

• **content-term**—Traffic is further matched against content strings, when matched. The content term then clause is effective. When the content-term is not matched there is no further filter term matching.

• **log**—Generates system log messages when the filter term is hit. This option can be used in conjunction with other term actions.

• **per-packet-load-balancing**—To improve efficiency, by default, filter processing is performed only on the first frame in each session. Subsequent frames in the session are assumed to match the same criteria and are automatically treated in the same way as the initial frame. Sessions that match a filter term are logged in the connection table for immediate processing of subsequent frames, rather than a full search to find a matching term. Some types of filtering (such as TCP flag) require each frame in the session to be filtered separately. To set this behavior, set per-packet-load-balancing for the relevant filters.

### Stacking Filters

Filter terms are evaluated according to the order they appear in the configuration. Filters are used to match traffic incoming on client-facing ports. When traffic is encountered at a client-facing port of the adc-instance, if the filter term matches, its configured action takes place and the rest of the terms are ignored. If the filter term criteria do not match, the next filter term is tried.

As long as the filter terms do not overlap, you can improve filter performance by making sure that the most heavily used terms are applied first.

#### Example

Consider a filter system where the Internet is divided according to destination IP address:

<table>
<thead>
<tr>
<th>Filtering by Destination IP Address Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0 - - - - - - - - - - - - - - - - - - - - - - - - - 255.255.255.255</td>
</tr>
<tr>
<td>Deny</td>
</tr>
</tbody>
</table>

Filter 2 | Filter 3 | Filter 4 | Filter 1

Assuming that traffic is distributed evenly across the Internet, the largest area is the most used and is assigned to Filter 1. The smallest area is assigned to Filter 4.

### Overlapping Filters

Filters are permitted to overlap, although special care should be taken to ensure the proper order of precedence. When overlapping filters are present, the more specific filters (those that target fewer addresses or ports) should be applied before the generalized filters.
Example

Filtering by Destination IP Address Ranges

<table>
<thead>
<tr>
<th>Deny</th>
<th>Redirect</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0</td>
<td>--------</td>
</tr>
<tr>
<td>255.255.255.255</td>
<td></td>
</tr>
</tbody>
</table>

Filter 3  Filter 2  Filter 1

In this example, Filter 2 must be processed prior to Filter 3. If Filter 3 was permitted to take precedence, Filter 2 could never be triggered.

The Default Filter

Incoming traffic on client-facing interfaces that does not match any filter term is handled by the virtual server using its standard configuration.

Optimizing Filter Performance

Filter efficiency can be increased by placing filters that are used most often near the beginning of the filtering list.

Filter Logs

To provide enhanced troubleshooting and session inspection capability, packet source and destination IP addresses are included in filter log messages. Filter log messages are generated when a Layer 3/Layer 4 filter is triggered and has logging enabled. The messages are output to the console port, system host log (syslog).

Example

A network administrator has noticed a significant number of ICMP frames on one portion of the network and wants to determine the specific sources of the ICMP messages. The administrator uses the command-line interface (CLI) to create and apply the following filter:

```plaintext
[edit extensions adc adc-instance demo1]
filters {
    term log-tcp {
        from {
            source-address 101.0.59.0/24;
            destination-address 101.0.59.0/24;
            protocol tcp;
        }
        then {
            allow;
            log;
        }
    }
}
```
This simple filter term produces log messages that show when the filter is triggered, and what the IP source and destination addresses were for the matching TCP frames.

**Example**

Filter log message output is shown below, displaying the filter number, port, source IP address, and destination IP address.

| Mar 2 12:38:16 | Juniper480_2 adc-mgmt[57812]: ADC adc-instance adc1 service-interface ms-1/0/0 Filter term term1 fired on DP 15 tcp 101.0.59.10:53150 101.0.59.101:80 [radware:adc-mgmt] |

**Using Per-Packet Load Balancing with Filters**

When using ADC filtering on a client-facing interface, connection table matching happens before term matching. This can cause interesting matching results if not carefully monitored.

When using **per-packet-load-balancing** filters, subsequent frames could match existing connection table entries rather than a **per-packet-load-balancing** filter.

**Example**

```
[edit extensions adc adc-instance demo1]
filters {
    term tcp-flag-urgent {
        from {
            protocol tcp;
            tcp-flags urg;
        }
        then {
            allow;
            per-packet-load-balancing;
        }
    }
    term tcp-other {
        from {
            protocol tcp;
        }
        then {
            load-balance {
                ...
            }
        }
    }
}
```

When a connection matches using the **tcp-other** term, it is added to the connection table. If a later frame in this connection has the **URG** TCP flag, it is load-balanced, as the connection table indicates, and not allowed according to the earlier term.
**Tunable Hash for Filter Redirection**

ADC software allows you to choose a number of options when using the hash parameter for filter redirection. Hashing can be based on source IP address, destination IP address, both source and destination IP address, or source IP address and source port. For example:

1. Configure hashing.
   a. Using the `by-http-header` parameter.

```
by-http-header <host | user-agent | user-define <string> length <length>);
```

b. Based on the source IP address.

```
[edit extensions adc adc-instance demo1]
filters {
  term term33 {
    from {
      protocol tcp;
      destination-port 80;
    }
    then {
      load-balance {
        group my-group;
        server-listening-port 3128;
        load-balancing-hash source-ip;
      }
    }
  }
}
```

Hashing on the 24-bit source IP address ensures that client requests access the same cache.

2. Set the load-balancing method for the real-server group to hash.
   The source IP address is passed to the real-server group for the load-balancing method.

```
[edit extensions adc adc-instance demo1]
groups {
  my-group {
    load-balance-method hash;
  }
}
```

**Filter-Based Security**

This section provides an example of configuring filters for providing the best security. It is recommended that you configure filters to deny all traffic except for those services that you specifically wish to allow. Consider the sample network shown in Figure 25 on page 138.
In this example, the network is made up of local clients on a collector module, a Web server, a mail server, a domain name server, and a connection to the Internet. All the local devices are on the same subnet.

In this example, the administrator wishes to install basic security filters to allow only the following traffic:
- External HTTP access to the local Web server
- External SMTP (mail) access to the local mail server
- Local clients browsing the World Wide Web
- Local clients using Telnet to access sites outside the intranet
- DNS traffic

All other traffic is denied and logged by the default filter.

**Note:** Since IP address and port information can be manipulated by external sources, filtering does not replace the necessity for a well-constructed network firewall.

**Configuring a Filter-Based Security Solution**

Before you begin, you must be connected to the module CLI as the administrator.

In this example, all filters are applied only to the port that connects to the Internet. If intranet restrictions are required, filters can be placed on ports connecting to local devices.
Also, filtering is not limited to the few protocols and TCP or UDP applications shown in this example. See [Well-Known Application Ports](#) for a list of well-known applications ports, and [ADC Filter terms](#) for a list of supported protocols.

1. Assign an IP address to each network device.

   For this example, the network devices have the IP addresses on the same IP subnet [Table 6 on page 139](#).

   Table 6: Web Cache Example: Real-Server IP Addresses

<table>
<thead>
<tr>
<th>Network Device</th>
<th>IP address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Subnet</td>
<td>205.177.15.0 - 205.177.15.255</td>
</tr>
<tr>
<td>Web Server</td>
<td>205.177.15.2</td>
</tr>
<tr>
<td>Mail Server</td>
<td>205.177.15.3</td>
</tr>
<tr>
<td>Domain Name Server</td>
<td>205.177.15.4</td>
</tr>
</tbody>
</table>

2. Create a filter that will allow external HTTP requests to reach the Web server.

   The filter must recognize and allow TCP traffic with the Web server's destination IP address and HTTP destination port.

   [edit extensions adc adc-instance demo1]
   filters {
     term allow-to-web {
       from {
         protocol tcp;
         destination-address 205.177.15.2/32;
         destination-port 80;
       }
       then {
         allow;
       }
     }
   }
3. Create a pair of filters to allow incoming and outgoing mail to and from the mail server.
   Filter 2 allows incoming mail to reach the mail server, and Filter 3 allows outgoing mail to reach
   the Internet.

   ```
   [edit extensions adc adc-instance demo1]
   filters {
     term allow-incoming-mail {
       from {
         protocol tcp;
         destination-address 205.177.15.3/32;
         destination-port 25;
       }
       then {
         allow;
       }
     }
     term allow-outgoing-mail {
       from {
         protocol tcp;
         source-address 205.177.15.2/32;
         source-port 25;
       }
       then {
         allow;
       }
     }
   }
   ```

4. Create a filter that will allow local clients to browse the Web.
   The filter must recognize and allow TCP traffic to reach the local client destination IP addresses if
   traffic originates from any HTTP source port.

   ```
   [edit extensions adc adc-instance demo1]
   filters {
     term allow-web-users {
       from {
         protocol tcp;
         destination-address 205.177.15.0/24;
         source-port 80;
       }
       then {
         allow;
       }
     }
   }
   ```
5. Create a filter that will allow local clients to Telnet anywhere outside the local intranet.
   The filter must recognize and allow TCP traffic to reach the local client destination IP addresses if
   originating from a Telnet source port.

   ```
   [edit extensions adc adc-instance demo1]
   filters {
     term allow-telnet-outside {
       from {
         protocol tcp;
         destination-address 205.177.15.0/24;
         source-port 23;
       }
       then {
         allow;
       }
     }
   }
   ```

6. Create a series of filters to allow Domain Name System (DNS) traffic.
   DNS traffic requires four filters; one pair is needed for UDP traffic (incoming and outgoing) and
   another pair for TCP traffic (incoming and outgoing).
   For UDP:

   ```
   [edit extensions adc adc-instance demo1]
   filters {
     term allow-dns-udp-in {
       from {
         protocol udp;
         destination-address 205.177.15.4/32;
         destination-port 53;
       }
       then {
         allow;
       }
     }
     term allow-dns-udp-out{
       from {
         protocol udp;
         source-address 205.177.15.4/32;
         source-port 53;
       }
       then {
         allow;
       }
     }
   }
   ```
Similarly, for TCP:

```plaintext
[edit extensions adc adc-instance demo1]
filters {
    term allow-dns-tcp-in {
        from {
            protocol tcp;
            destination-address 205.177.15.4/32;
            destination-port 53;
        }
        then {
            allow;
        }
    }
    term allow-dns-tcp-out{
        from {
            protocol tcp;
            source-address 205.177.15.4/32;
            source-port 53;
        }
        then {
            allow;
        }
    }
}

[edit extensions adc adc-instance demo1]
filters {
    term discard-and-log {
        from {
        }
        then {
            discard;
            log;
        }
    }
}
```

7. At the application module, create a default filter term to deny and log unwanted traffic. Make sure to put this term last within the filters section in order to give it the lowest order of precedence.

8. Assign the filters to the port that connects to the Internet.

**Note:** Changes to filters only affect new sessions. To make filter changes take effect immediately, deactivate the filter term configuration and reactivate it.
Matching TCP Flags

The ADC software supports packet filtering based on any of the TCP flags shown in Table 7 on page 143.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URG</td>
<td>Urgent</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledgement</td>
</tr>
<tr>
<td>PSH</td>
<td>Push</td>
</tr>
<tr>
<td>RST</td>
<td>Reset</td>
</tr>
<tr>
<td>SYN</td>
<td>Synchronize</td>
</tr>
<tr>
<td>FIN</td>
<td>Finish</td>
</tr>
</tbody>
</table>

Any filter can be set to match against a single TCP flag or more than one. If more than one flag is enabled, the flags are applied with a logical AND operator.

Example Matching multiple flags

tcp-flags "SYN & ACK";

In this example, all SYN-ACK flags are found using the filter.

Note: TCP flag filters must use per-packet-load-balancing. Exercise caution when using per-packet-load-balancing terms in conjunction with terms that use the connection table (the default behavior). For more information, see Using Per-Packet Load Balancing with Filters.

Configuring the TCP Flag Filter

Note: By default, all TCP filter options are disabled. TCP flags will not be inspected unless a TCP flag is selected.

Consider the network Figure 26 on page 144.
In this network, the Web servers inside the LAN must transfer mail to any SMTP-based mail server out on the Internet. At the same time, you want to prevent access to the LAN from the Internet, except for HTTP.

SMTP traffic uses well-known TCP port 25. The Web servers will originate TCP sessions to the SMTP server using TCP destination port 25, and the SMTP server will acknowledge each TCP session and data transfer using TCP source port 25.

Creating a filter with the ACK flag closes one potential security hole. Without the filter, the module would permit a TCP syn connection request to reach any listening TCP destination port on the Web servers inside the LAN, as long as it originated from TCP source port 25. The server would listen to the TCP syn, allocate buffer space for the connection, and reply to the connect request. In some syn attack scenarios, this could cause the server’s buffer space to fill, crashing the server or at least making it unavailable.

A filter with the ACK flag enabled prevents external devices from beginning a TCP connection (with a TCP syn) from TCP source port 25. The module drops any frames that have the ack flag turned off.

The following filters are required:

1. An allow filter for TCP traffic from LAN that allows the Web servers to pass SMTP requests to the Internet.

```plaintext
[edit extensions adc adc-instance demo1]
filters {
  term allow-smtp-from-web-servers {
    from {
      protocol tcp;
      source-address 203.122.186.0/24;
      destination-port 53;
    }
    then {
      allow;
    }
  }
}
```
2. A filter that allows SMTP traffic from the Internet to pass through the module only if the destination is one of the Web servers, and the frame is an acknowledgment (SYN-ACK) of a TCP session.

```
[edit extensions adc adc-instance demo1]
filters {
  term allow-smtp-reply-from-internet{
    from {
      protocol tcp;
      source-port 53;
      destination-address 203.122.186.0/24;
      tcp-flags "syn & ack";
    }
    then {
      allow;
    }
  }
}
```

3. A filter that allows SMTP traffic from the Internet to pass through the module only if the destination is one of the Web servers, and the frame is an acknowledgment (ACK-PSH) of a TCP session.

```
[edit extensions adc adc-instance demo1]
filters {
  term allow-smtp-to-web-servers{
    from {
      protocol tcp;
      source-port 53;
      destination-address 203.122.186.0/24;
      tcp-flags "ack & psh";
    }
    then {
      allow;
    }
  }
}
```
4. A filter that allows trusted HTTP traffic from the Internet to pass through the module to the Web servers.

```plaintext
[edit extensions adc adc-instance demo1]
filters {
  term allow-http-to-web-servers{
    from {
      protocol tcp;
      source-port 80;
      destination-address 203.122.186.0/24;
    }
    then {
      allow;
    }
  }
}
```

5. A filter that allows HTTP responses from the Web servers to pass through the module to the Internet.

```plaintext
[edit extensions adc adc-instance demo1]
filters {
  term allow-http-replies {
    from {
      protocol tcp;
      source-address 203.122.186.0/24;
      source-port 80;
      destination-port 80;
    }
    then {
      allow;
    }
  }
}
```

6. A default filter is required to deny all other traffic. Make sure this is the last filter in your list of filters so that it gets the lowest priority.

```plaintext
[edit extensions adc adc-instance demo1]
filters {
  term discard-and-log {
    from {
    }
    then {
      discard;
      log;
    }
  }
}
```

7. Make sure the client-side ports are defined in the client-facing interfaces for this ADC instance.
Deny Filter Based on Layer 7 Content

The ADC software allows you to secure your module from virus attacks or invalid data requests by configuring the module with filters containing a potential offending string. Examples of such strings include those embedded in an HTTP URL request, SOAPAction request bound to an HTTP header, or certain strings contained in the data portion of UDP traffic. The module examines the TCP or UDP content of the incoming client request for the matching string. If the matching virus identifying string is found, then the packet is dropped and a reset frame is sent to the offending client. System log messages and SNMP traps are generated warning operators of a possible attack.

A Layer 7 string deny filter works just like a basic deny filter, except that the deny action is delayed until the string content is examined to see if the packet should be denied.

Denying HTTP URL Requests

1. Before creating a deny filter with Layer 7 lookup, ensure that the module is configured for basic module functions.
   For information on how to configure your network, see Server Load Balancing.
2. Define the virus string or offending HTTP URL request to be blocked.

```
[edit extensions adc adc-instance demo1]
content-strings {
  string code-red {
    text-search {
      url-string ida;
    }
  }
  string code-blue1 {
    text-search {
      url-string %c1%9c;
    }
  }
  string code-blue2 {
    text-search {
      url-string %c0%af;
    }
  }
  string offending-url {
    text-search
      url-string playdog.com;
  }
}
```
3. Set the filter using the assigned strings.

```plaintext
[edit extensions adc adc-instance demo1]
filters {
  term protect-playdog {
    from {
    } then {
      content-term {
        match {
          content-strings [code-red code-blue1 code-blue2 offending-url];
          then {
            discard;
          }
        }
      }
    }
  }
}
```

**Denying HTTP Headers**

Layer 7 deny filters can be configured to match and deny on any HTTP headers.

**Note:** Traffic that matches the `from` clause of the term but does not match the `match` clause of the `content-term` part of the filter is accepted.

Examples of HTTP headers include:
- HTTPHDR:Host: www.playdog.com
- HTTPHDR:Host: www.yahoo.com:/image/hello.gif
- /default.asp (the URL by itself)
- HTTPHDR:User-Agent:Netscape*
- HTTPHDR:SoapAction=*
To configure a filter as described above

1. Define the HTTP header strings to be blocked.

```plaintext
[edit extensions adc adc-instance demo1]
content-strings {
    string playdog {
        text-search {
            http-header Host value www.playdog.com;
            url-string www.playdog.com;
        }
    }
    string soap-action {
        text-search {
            http-header SoapAction=*;
        }
    }
}
```

2. Assign the HTTP host header strings to the filter.

```plaintext
[edit extensions adc adc-instance demo1]
filters {
    term discard-http-headers {
        from {
        }
        then {
            content-term {
                match {
                    content-strings [playdog soap-action];
                }
                then {
                    discard;
                }
            }
        }
    }
}
```

## Cache Server Load Balancing

Cache server load balancing improves network bandwidth and provides unique network solutions. Filters can be created to redirect traffic to cache and application servers, improving speed of access to repeated client access to common Web or application content and freeing valuable network bandwidth.

The following topics are addressed in this section:

- **Cache Server Load Balancing Overview** — Cache server load balancing helps reduce traffic congestion during peak loads by accessing locally cached information. This section also discusses how performance is improved by balancing cached requests across multiple servers.
• **Cache Redirection Environment**—This section provides a step-by-step procedure on how to intercept all Internet bound HTTP requests (on default TCP port 80) and redirect them to the cache servers.

• **RTSP Cache Redirection**—This section explains how to configure the ADC software to redirect data (multimedia presentations) to the cache servers and how to balance the load among the cache servers.

• **Excluding Noncacheable Sites**—This section describes how to filter out applications that prevent real-time session information from being redirected to cache servers.

• **Network Address Translation Options**—This section describes how cache redirection supports various types of Network Address Translation (NAT).

• **Content Intelligent Cache Redirection**—This section describes how to redirect cache requests based on different Layer 7 content.

### Cache Server Load Balancing Overview

Most of the information downloaded from the Internet is not unique, as clients will often access the Web page many times for additional information or to explore other links. Duplicate information also gets requested as the components that make up Internet data at a particular website (pictures, buttons, frames, text, and so on) are reloaded from page to page. When you consider this scenario in the context of many clients, it becomes apparent that redundant requests can consume a considerable amount of your available bandwidth to the Internet.

Application redirection can help reduce the traffic congestion during peak loads. When application redirection filters are properly configured for the ADC software-powered switch, outbound client requests for Internet data are intercepted and redirected to a group of application or cache servers on your network. The servers duplicate and store inbound Internet data that has been requested by your clients. If the servers recognize a client's outbound request as one that can be filled with cached information, the servers supply the information rather than send the request across the Internet.

In addition to increasing the efficiency of your network, accessing locally cached information can be much faster than requesting the same information across the Internet.

### Cache Redirection Environment

Consider the network shown in Figure 27 on page 151, where client HTTP requests begin to regularly overload the Internet router.

**Note:** By default, traffic that is load-balanced using filters is sent transparently to the servers; the destination address of the request is not changed, only the MAC address is changed. For more information about changing addresses when using filters to load-balance traffic, see Network Address Translation Options.
The network needs a solution that addresses the following key concerns:

- The solution must be readily scalable
- The administrator should not need to reconfigure all the clients' browsers to use proxy servers.
If you have more clients than device ports, then connect the clients to a Layer 2 ADC. See Figure 28 on page 152.

Adding an ADC module with optional Layer 4 software addresses these issues:

- Cache servers can be added or removed dynamically without interrupting services.
- Performance is improved by balancing the cached request load across multiple servers. More servers can be added at any time to increase processing power.
- The proxy is transparent to the client.
- Frames that are not associated with HTTP requests are normally passed to the router.

**Additional Application Redirection Options**

Application redirection can be used in combination with other Layer 4 options, such as load-balancing methods, health checks, real-server group backups, and more. See Implementing Server Load Balancing for details.

**Cache Redirection Configuration Example**

In this example, an ADC module is placed between the clients and the border gateway to the Internet. The ADC software is configured to intercept all Internet-bound HTTP requests (on default TCP port 80), and redirect them to the cache servers. The ADC software will distribute HTTP requests equally to the cache servers based on the destination IP address of the requests. If the cache servers do not have the requested information, then they behave like the client and forward the request to the Internet.
Also, filters are not limited to the few protocols and TCP or UDP applications shown in this example. See Well-Known Application Ports for a list of well-known applications ports, and ADC Filter terms for a list of supported protocols.

1. Assign an IP address to each cache server.
   
   Similar to SLB, the cache real servers are assigned an IP address and placed into a real-server group. The real servers must be in the same VLAN and must have an IP route to the ADC software that will perform the cache redirection. In addition, the path from the ADC software to the real servers must not contain a router. The router would stop HTTP requests from reaching the cache servers and, instead, direct them back out to the Internet.
   
   More complex network topologies can be used if configuring IP proxy addresses (see Excluding Noncacheable Sites).
   
   For this example, the three cache real servers have the IP addresses on the same IP subnet shown in Table 8 on page 153.

### Table 8: Cache Redirection Example: Real-Server IP Addresses

<table>
<thead>
<tr>
<th>Cache Server</th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server A</td>
<td>200.200.200.2</td>
</tr>
<tr>
<td>Server B</td>
<td>200.200.200.3</td>
</tr>
<tr>
<td>Server C</td>
<td>200.200.200.4</td>
</tr>
</tbody>
</table>

2. Install transparent cache software on all three cache servers.
   
   The router must have an IP interface on the same subnet as the three cache servers because, by default, the ADC software only remaps destination MAC addresses.

   ![Note Icon]
   
   **Note:** The IP interface and the real servers must be in the same subnet. This example assumes that all ports and IP interfaces use default VLAN 1, requiring no special VLAN configuration for the ports or IP interface.

3. Define each real server on the device. Make sure the servers are connected via a router interface that is defined as a server-facing interface for the adc-instance.
   
   For each cache real server, you must assign a real-server name and specify its actual IP address.

4. Define a real-server group.

   ```
   [edit extensions adc adc-instance demo1]
   real-servers {
     real1 {
       address 200.200.200.1;
     }
     real2 {
       address 200.200.200.2;
     }
     real3 {
       address 200.200.200.3;
     }
     real4 {
       address 200.200.200.4;
     }
   }
   ```

5. Place the cache real servers into one service group.
6. Create a filter that will intercept and redirect all client HTTP requests.

The filter must be able to intercept all TCP traffic for the HTTP destination port and must redirect it to the proper port on the real-server group.

```
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    real-servers [real1 real2 real3 real4];
  }
}
```

The server-listening-port parameter must be configured whenever TCP/UDP protocol traffic is load-balanced. The server-listening-port parameter defines the real-server TCP or UDP port to which redirected traffic is sent. The port defined by the server-listening-port parameter is used when performing Layer 4 health checks of TCP services.

Also, if NAT addresses are used on the ADC software, the server-listening-port parameter must be configured for all application redirection filters. Take care to use the proper port when using the server-listening-port parameter. If the transparent proxy operation resides on the host, the well-known port 80 (or HTTP) is probably required. If the transparent proxy occurs on the ADC software, make sure to use the service port required by the specific software package.

See Excluding Noncacheable Sites for more information on NAT addresses.

**Note:** When the protocol parameter is not TCP or UDP, then source-port and destination-port are ignored.

7. The client traffic is arriving at the router via one of its interfaces. Ensure that this interface is in the client-facing interfaces list.

**Note:** Changes to filters affect new sessions. To make filter changes take effect immediately, deactivate the filter term configuration and reactivate it.
RTSP Cache Redirection

ADC software supports cache redirection for the Real-Time Streaming Protocol (RTSP). RTSP cache redirection is similar to HTTP cache redirection in configuration and in concept. Multimedia presentations consume a lot of Internet bandwidth. The quality of these presentations depends upon the real-time delivery of the data. To ensure the high quality of multimedia presentations, several caching servers are needed to cache the multimedia data locally. This data is then made available quickly from the cache memory as required.

RTSP cache redirection redirects cached data transparently and balances the load among the cache servers. If there is no cache server, the request is directed to the origin server. Internet service providers use this feature to cache the multimedia data of a customer site locally. Since the requests for this data are directed to the local cache, they are served faster.

This section explains Layer 4 support for RTSP streaming cache redirection. For detailed information on two prominent commercial RTSP servers—Real Player and QuickTime—see Real-Time Streaming Protocol Server Load Balancing.

You can also configure the ADC software to redirect client request based on URL content. For information on Layer 7 RTSP streaming cache redirection, see RTSP Streaming Cache Redirection.

Figure 29: RTSP Cache Redirection

Follow this procedure to load-balance RTSP cache servers for the topology illustrated in Figure 29 on page 155:

1. Before configuring RTSP, do the following:
   - Connect each cache server to the device.
   - Configure the IP addresses on all devices connected to the device.
   - Configure the IP interfaces on the device.
2. Configure RTSP cache servers and the IP addresses.
3. Define a group to load balance the RTSP cache servers.

```
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    real-servers [real1 real2 real3 real4];
  }
}
```

4. Define the group load-balancing method for the RTSP cache servers.

   RTSP supports all the standard load balancing methods.

```
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    load-balance-method least-connections;
  }
}
```

5. Configure an RTSP redirection filter to cache data and balance the load among the cache servers.
6. Ensure that the client side ports are defined in the client-facing interfaces for this ADC instance.

7. Ensure that the servers are connected via server-facing interfaces for this ADC instance.

## Excluding Noncacheable Sites

Some sites provide content that is not well suited for redirection to cache servers. Such sites might provide browser-based games or applications that keep real-time session information or authenticate by client IP address.

To prevent such sites from being redirected to cache servers, create a filter that allows this specific traffic to pass normally through the ADC software. This filter must have a higher precedence (a lower filter number) than the application redirection filter.

For example, if you want to prevent a popular Web-based game site on subnet 200.10.10.* from being redirected, you could add the following to the previous example configuration:

```
[edit extensions adc adc-instance demo1]
filters {
  term non-cacheable-sites {
    from {
      protocol tcp;
      destination-address 200.10.10.0/24;
      destination-port 80;
    }
    then {
      accept;
    }
  }
}
```

## Network Address Translation Options

URL-based cache redirection supports three types of Network Address Translation (NAT): No NAT, Half NAT, and Full NAT.

- No NAT
In this NAT method, the traffic is redirected to the cache with the destination MAC address of the virtual server replaced by the MAC address of the cache. The destination IP address remains unchanged, and no modifications are made to the IP address or the MAC address of the source or origin server. This works well for transparent cache servers, which process traffic destined to their MAC address but use the IP address of some other device.

This is the default behavior.

- **Half NAT**

  In this most commonly used NAT method, the destination IP address is replaced by the IP address of the cache, and the destination MAC address is replaced by the MAC address of the cache. Both the IP address and the MAC address of the source remain unchanged.

  To use this behavior, set the `destination-nat` flag for the term.

  **Note:** This behavior is not supported for terms that match UDP traffic. For UDP traffic, Full NAT must be used.

- **Full NAT**

  In this NAT method, the source IP address and the source MAC address are replaced by the IP address and MAC address of the device. This method works well for proxy cache servers.

  To use this behavior, set both the `destination-nat` and `client-nat` flags for the term. The addresses used for `client-nat` are defined under `router-interfaces ms-interfaces`. For more information, see [NAT IP Address Configuration](#).

### Content Intelligent Cache Redirection

ADC software allows you to redirect cache requests based on different Layer 7 content, such as HTTP header information "Host:" header or "User-Agent," for browser-smart load-balancing.

The No Cache/Cache Control for cache redirection feature in ADC software allows you to offload the processing of non-cacheable content from cache servers by sending only appropriate requests to the cache server group. When a Cache-Control header is present in a HTTP 1.1 request, it indicates a client's special request with respect to caching, such as to guarantee up-to-date data from the origin server. If this feature (Cache-Control: no cache directive) is enabled, HTTP 1.1 GET requests are forwarded directly to the origin servers.

**Note:** The term **origin server** refers to the server originally specified in the request.

The HTTP 1.0 **Pragma: no-cache** header is equivalent to the HTTP 1.1 **Cache-Control** header. By enabling the **Pragma: no-cache** header, requests are forwarded to the origin server.

For cache redirection, at any given time one HTTP header is supported globally for the entire ADC. This section discusses the following types of cache redirection:

- **URL-Based Cache Redirection**
- **HTTP Header-Based Cache Redirection**
- **Browser-Based Cache Redirection**
- **URL Hashing for Cache Redirection**
- **RTSP Streaming Cache Redirection**
**URL-Based Cache Redirection**

URL parsing for cache redirection operates in a manner similar to URL-based server load balancing, except that in cache redirection, a virtual server on the ADC is the target of all IP/HTTP requests. For information on URL-based server load balancing, see Figure 30 on page 160.

By separating static and dynamic content requests via URL parsing, ADC software lets you send requests with specific URLs or URL strings to designated cache servers. The URL-based cache redirection option allows you to offload overhead processing from the cache servers by only sending appropriate requests to the cache server group.

**Note:** Both HTTP 1.0 and HTTP 1.1 requests are supported.

Each request is examined and handled as described below:

- If the request is a non-GET request such as HEAD, POST, PUT, or HTTP with cookies, it is not sent to the cache.
- If the request is an ASP or CGI request or a dynamically generated page, it is not sent to the cache.
- If the request contains a cookie, it can optionally bypass the cache.
  
Examples of matching string expressions are:

- `/product`
  
  Any URL that starts with `/product`, including any information in the `/product` directory
- `product`
  
  Any URL that has the string `product`

Some of the common noncacheable items that you can configure the ADC to add to, delete, or modify are:

- Dynamic content files:
  - Common gateway interface files (.cgi)
  - Cold fusion files (.cfm), ASP files (.asp)
  - BIN directory
  - CGI-BIN directory
  - SHTML (scripted HTML)
  - Microsoft HTML extension files (.htx)
  - Executable files (.exe)

- Dynamic URL parameters: +, !, %, =, &
Requests matching the URL are load-balanced among the multiple servers, depending on the load-balancing method specified for the real-server group (*least-connections* is the default).

**Configuring URL-Based Cache Redirection**

![Diagram of URL-Based Cache Redirection](image)

1. Before you can configure URL-based cache redirection, configure the ADC software for basic server load balancing (SLB).
   For information on how to configure your network for SLB, see [Server Load Balancing](#).
2. Configure the ADC software to support basic cache redirection.
   For information on cache redirection, see [Cache Server Load Balancing](#).
3. Configure the parameters and file extensions that bypass cache redirection.
   a. Add or remove strings that should not be cacheable.

```plaintext
[edit extensions adc adc-instance demo1 filters]
term cache {
   from {
   }
   then {
      load-balance {
         group cache-servers;
         exclude-by-content [<string-name> <string-name>];
      }
   }
}
```

b. Set the non-get-requests parameter for non-GETS (such as HEAD, POST, and PUT) to the origin server.
   - pass-through—The ADC software allows all non-GET requests to the origin server.
   - match-url—The ADC software compares all requests against the expression table to determine whether the request should be redirected to a cache server or the origin server.

c. Set redirection of requests that contain "cookie:" in the HTTP header.
   - pass-through—The ADC software redirects all requests that contain "cookie:" in the HTTP header to the origin server.
   - match-url—The ADC software compares the URL against the expression table to determine whether the request should be redirected to a cache server or the origin server. This is the default.

d. Set cache redirection of requests that contain "Cache-control:no cache" in the HTTP 1.1 header or "Pragma:no cache" in the HTTP 1.0 header to the origin server.
   - pass-through—The ADC software redirects all requests that contain Cache-control: no cache in the HTTP 1.1 header or Pragma:no cache in the HTTP 1.0 header to the origin server. This is the default.
   - match-url—The ADC software compares the URL against the expression table to determine whether the request should be redirected to a cache server or the origin server.

```plaintext
[edit extensions adc adc-instance demo1]
web-cache-redirection {
   non-get-requests <match-url | pass-through>;
   requests-with-cookie <match-url | pass-through>;
   no-cache-requests <match-url | pass-through>;
   select-by-content url-hash < length >;
}
```
4. Define the strings to be used for cache server load balancing.

```plaintext
[edit extensions adc adc-instance demo1]
content-match {
    string <name> {
        text-search {
            url-string <text>;
        }
    }
}
```

A default string **any** indicates that the particular server can handle all URL or cache requests. See the following examples.

### Example 1: String starting with the forward slash (/)
A string that starts with a forward slash ( / ), such as "/images," indicates that the server will process requests that start out with the "/images" string only.

For example, with the "/images" string, the server will handle these requests:

- /images/product/b.gif
- /images/company/a.gif
- /images/testing/c.jpg

The server will not handle these requests:

- /company/images/b.gif
- /product/images/c.gif
- /testing/images/a.gif

### Example 2: String without the forward slash (/)
A string that does not begin with a forward slash ( / ) indicates that the server will process any requests that contain the defined string. For example, with the "images" string, the server will process these requests:

- /images/product/b.gif
- /images/company/a.gif
- /images/testing/c.jpg
- /company/images/b.gif
- /product/images/c.gif
- /testing/images/a.gif

### Example 3: String with the forward slash (/) only
If a server is configured with the load-balance string ( / ) only, it will only handle requests to the root directory. For example, the server will handle any files in the ROOT directory:

- //index.htm
- /default.asp
- /index.shtm

5. Apply and save your configuration changes.
6. Identify the defined strings.

```plaintext
[edit extensions adc adc-instance demo1]
content-match {
    string gif-1 {
        text-search {
            url-string ".gif";
        }
    }
    string sales-1 {
        text-search {
            url-string "/sales";
        }
    }
    string xitami-1 {
        text-search {
            url-string "/xitami";
        }
    }
    string manual-1 {
        text-search {
            url-string "/manual";
        }
    }
    string jpg-1 {
        text-search {
            url-string ".jpg";
        }
    }
}
```

7. Configure the real servers to support cache redirection.

**Note:** If you do not add a defined string (or add the defined string **any**), the server will handle any request.

8. Add the defined strings to the real servers.

```plaintext
[edit extensions adc adc-instance demo1]
real-servers {
    real1 {
        content-strings [ gif-1 jpg-1 ];
    }
    real2 {
        content-strings [ sales-1 xitami-1 manual-1 ];
    }
}
```

The server can have multiple defined strings.

With these defined strings, the server can handle requests that begin with "/images" or "/sales" and any requests that contain ".gif."
9. Define a real-server group and add real servers to the group.

The following configuration combines three real servers into a group.

```
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    real-servers [ real1 real2 ];
  }
}
```

10. Configure a filter to support basic cache redirection.

The filter must be able to intercept all TCP traffic for the HTTP destination port and must redirect it to the proper port in the real-server group:

```
[edit extensions adc adc-instance demo1]
filters {
  term http-traffic {
    from {
      protocol tcp;
      destination-port 80;
    }
    then {
      load-balance {
        group group1;
        server-listening-port 80;
        select-by-content;
        exclude-by-content [movies];
      }
    }
  }
}
```

11. Select the appropriate NAT option.

The three NAT options are listed below. For more information about each option, see Network Address Translation Options.

- No NAT option—Default setting.
- Half NAT option—Use the destination-nat parameter.

```
[edit extensions adc adc-instance demo1]
term <name> {
  then {
    load-balance {
      destination-nat;
    }
  }
}
```
— Full NAT option—Use both client-nat and destination-nat parameters.

12. Ensure that client traffic is coming via client-facing devices and that servers are connected via server-facing devices.

**Viewing Statistics for URL-Based Cache Redirection**

To show the number of hits to the cache server or origin server, use this command:

```plaintext
show extensions adc filters load-balance adc-instance <name>
```

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total URL based web cache redirection stats for adc-instance &lt;name&gt;:</td>
<td>0</td>
</tr>
<tr>
<td>Total cache server hits:</td>
<td>0</td>
</tr>
<tr>
<td>Total origin server hits:</td>
<td>0</td>
</tr>
<tr>
<td>Total straight to origin server hits:</td>
<td>0</td>
</tr>
<tr>
<td>Total none-GETs hits:</td>
<td>0</td>
</tr>
<tr>
<td>Total 'Cookie:' hits:</td>
<td>0</td>
</tr>
<tr>
<td>Total no-cache hits:</td>
<td>0</td>
</tr>
<tr>
<td>Total RTSP cache server hits:</td>
<td>0</td>
</tr>
<tr>
<td>Total RTSP origin server hits:</td>
<td>0</td>
</tr>
<tr>
<td>Total HTTP redirection hits:</td>
<td>0</td>
</tr>
</tbody>
</table>

**HTTP Header-Based Cache Redirection**

To configure the ADC for cache direction based on the Host header

1. Before you can configure header-based cache redirection, ensure that the ADC is configured for basic SLB (see **Server Load Balancing**) with the following tasks:
   - Assign an IP address to each of the real servers in the server group.
   - Define an IP interface on the ADC.
   - Define each real server.
   - Assign servers to real-server groups.
   - Define virtual servers and services.
2. Turn on Layer 7 lookup for the filter.

```plaintext
select-by-content;
```

3. Enable header load balancing for the Host header.

```plaintext
load-balancing-hash by-http-header host;
```
4. Define the hostnames.

```plaintext
[edit extensions adc adc-instance demo1]
content-strings {
  text-search {
    string com {
      url-string .com;
    }
  }
  text-search {
    string org {
      url-string .org;
    }
  }
  text-search {
    string net {
      url-string .net;
    }
  }
}
```

5. Configure the real servers to handle the appropriate load-balanced strings.

6. Add the defined strings to the real servers.

```plaintext
[edit extensions adc adc-instance demo1]
real-servers {
  real1 {
    content-strings [ com org net];
  }
}
```

**Note:** If you do not add a defined string, the server will handle any request.

**Browser-Based Cache Redirection**

Browser-based cache redirection uses the User-agent: header.

**To configure browser-based cache redirection**

1. Before you can configure header-based cache redirection, ensure that the device is already configured for basic server load balancing.
2. Enable header load balancing for the User-Agent header.

```plaintext
load-balancing-hash by-http-header user-agent;
```
3. Define the hostnames.

```plaintext
[edit extensions adc adc-instance demo1]
content-strings {
    text-search {
        string Mozilla {
            url-string Mozilla;
        }
    }
    text-search {
        string IE {
            url-string "Internet Explorer";
        }
    }
    text-search {
        string Netscape{
            url-string Netscape;
        }
    }
}
```

4. Add the defined strings to configure the real servers to handle the appropriate load-balanced strings.

```plaintext
[edit extensions adc adc-instance demo1]
real-servers {
    real1 {
        content-strings [ Mozilla IE];
    }
    real2 {
        content-strings Netscape;
    }
}
```

If you do not add a defined string, the server will handle any request.

**URL Hashing for Cache Redirection**

By default, hashing algorithms use the source IP address, destination IP address, or both (depending on the application area) to determine content location. For example, firewall load-balancing uses both source and destination IP addresses, while cache redirection uses only the destination IP address, and SLB uses only the source IP address.

Hashing is based on the URL, including the HTTP Host header (if present), up to a maximum of 255 bytes. You can optimize "cache hits" by using the hashing algorithm to redirect client requests going to the same page of an origin server to a specific cache server.

For example, the ADC software could use the string "radware.com/products/boo/" for hashing the following request:

```
GET http://products/Alteon/ HTTP/1.0
HOST:www.radware.com
```
To configure the ADC for cache redirection based on a hash key

1. Before you can configure header-based cache redirection, ensure that the device is configured for basic server load balancing (see Server Load Balancing).

2. Allow the hash to direct a cacheable URL request to a specific cache server.
   - By default, the host header field is used to calculate the hash key and URL hashing is disabled.
     - **Used**—Allows hashing based on the URL and the host header if it is present. Specify the length of the URL to hash into the cache server.
     - **Not used**—Does not allow hashing based on the URL. Instead, the host header field is used to calculate the hash key.
     
     If the host header field does not exist in the HTTP header, then the ADC uses the source IP address as the hash key.

---

Example 1: Hashing on the URL

In this example, URL hashing is enabled. If the Host field does not exist, the specified length of the URL is used to hash into the cache server, as shown in Figure 31 on page 169. If the Host field exists, the specified length of both the Host field and the URL is used to hash into the cache server. The same URL request goes to the same cache server, as shown below:

- Client 1 request http://www.radware.com/sales/index.htm is directed to cache server 1.
- Client 2 request http://www.radware.com/sales/index.htm is directed to cache server 1.
- Client 3 request http://www.radware.com/sales/index.htm is directed to cache server 1.
Example 2: Hashing on the Host header field only
In this example, URL hashing is disabled. If you use the Host header field to calculate the hash key, the same URL request goes to the same cache server:

- Client 1 request http://www.radware.com is directed to cache server 1.
- Client 2 request http://www.radware.com is directed to cache server 1.
- Client 3 request http://www.radware.com is directed to cache server 1.

Example 3: Hashing on the source IP address
In this example, URL hashing is disabled. Because the Host header field does not exist in the HTTP header, the source IP address is used as the hash key and requests from clients 1, 2, and 3 are directed to three different cache servers, as shown below.

- Client 1 request http://www.radware.com is directed to cache server 1.
- Client 2 request http://www.radware.com is directed to cache server 2.
- Client 3 request http://www.radware.com is directed to cache server 3.

RTSP Streaming Cache Redirection
RTSP load balancing with the URL hash load-balancing method can be used to load-balance cache servers that cache multimedia presentations. Since multimedia presentations consume a large amount of Internet bandwidth, and their correct presentation depends upon the real-time delivery of the data over the Internet, several caching servers cache the multimedia data.
As a result, the data is available quickly from the cache, when required. The Layer 7 load-balancing method of URL hashing directs all requests with the same URL to the same cache server, ensuring that no data is duplicated across the cache servers. All stream connections and control connections are switched to the same cache server to facilitate caching of entire presentations.

This section explains Layer 7 support for RTSP streaming cache redirection. For conceptual information on RTSP streaming cache redirection, see RTSP Cache Redirection. For detailed information on two prominent commercial RTSP servers—Real Player and QuickTime—see Real-Time Streaming Protocol Server Load Balancing.

In the scenario illustrated in Figure 32 on page 170, the cache servers are configured for forward proxy mode. The cache servers process the client request even though the destination IP address is not destined for the cache servers.

**Figure 32: Forward Proxy Mode**

To configure load-balancing RTSP cache servers as illustrated in Figure 32 on page 170

1. Before you start configuring the ADC software, do the following:
   - Connect each cache server to the device.
   - Configure the IP addresses on all devices connected to the device.
   - Configure the IP interfaces on the device.
2. Define the RTSP file extensions to load balance among the cache servers.

```
[edit extensions adc adc-instance demo1]
content-strings {
    string condor {
        text-search {
            url-string condor.rm;
        }
    }
    string tiger {
        text-search {
            url-string tiger.rm;
        }
    }
}
```

3. Assign the URL string to the cache servers.

```
[edit extensions adc adc-instance demo1]
real-servers {
    real1 {
        address 10.10.10.10;
        content-strings [ condor ];
    }
    real2 {
        address 20.20.20.20;
        content-strings [ condor ];
    }
    real3 {
        address 30.30.30.30;
        content-strings [ tiger ];
    }
    real4 {
        address 40.40.40.40;
        content-strings [ tiger ];
    }
}
```

4. Define a group to load balance the RTSP cache servers.

```
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        real-servers [real1 real2 real3 real4];
    }
}
```

5. Configure a redirection filter.

   a. select-by-content—Implies server selection is based on the content-strings as defined in the real-server configuration.
b. exclude-by-content—Defines strings that should bypass load balancing and be sent through to their original destination.

```plaintext
[edit extensions adc adc-instance demo1]
filters {
  term cache {
    from {
      protocol tcp;
      destination-port 554;
    }
    then {
      load-balance {
        group group1;
        server-listening port 554;
        select-by-content;
        exclude-by-content [movies];
      }
    }
  }
}
```

**Note:** If no strings are assigned to the server, the server will handle all requests.

Client requests for **condor.rm** or **tiger.rm** are retrieved from the local cache servers 1 or 2 and 3 or 4, respectively; however, a client request for **cheetah.mov** bypasses the local cache servers and is forwarded to the original server.

## HTTP Redirection

This section describes how to use filters to redirect HTTP requests to different gateways or servers. The following topics are discussed:

- HTTP Redirection Overview
- IP-Based HTTP Redirection
- TCP Service Port-Based HTTP Redirection
- MIME Type Header-Based Redirection
- URL-Based Redirection
- Source IP from HTTP Header and Host Header-Based Redirection
- HTTP to HTTPS Redirection

### HTTP Redirection Overview

Filters can be used to redirect HTTP requests to different gateways or servers. The following HTTP redirection types are supported:

- **IP redirection**—The ADC software redirects client requests to different service gateways based on the address range of the client device and requested URL. For details, see [IP-Based HTTP Redirection](#).

---

**Note:** If no strings are assigned to the server, the server will handle all requests.

Client requests for **condor.rm** or **tiger.rm** are retrieved from the local cache servers 1 or 2 and 3 or 4, respectively; however, a client request for **cheetah.mov** bypasses the local cache servers and is forwarded to the original server.
• **Port redirection**—The ADC software examines traffic entering on a TCP service port (such as HTTP port 80) and sends that traffic to a user-specified IP address on a different service port (such as 9090). For details, see TCP Service Port-Based HTTP Redirection.

• **MIME type redirection**—The ADC software examines the HTTP header or URL of an incoming request for a specific Multipurpose Internet Mail Extensions (MIME) type, and replaces the URL with another URL. For details, see MIME Type Header-Based Redirection.

• **URL redirection**—The ADC software examines a URL and redirects it to a preconfigured IP address or URL. For details, see URL-Based Redirection.

Note: The HTTP header redirection feature is not limited to the types of HTTP headers listed below.

**IP-Based HTTP Redirection**

In this example, the ADC software will redirect Web pages requested from a mobile phone, to a specific gateway based on the client's IP address. A mobile phone is set to access its home page via the default device gateway.

**Example Client phone configuration**

<table>
<thead>
<tr>
<th>Device Gateway IP address</th>
<th>10.168.107.101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home page</td>
<td><a href="http://wap.example.com">http://wap.example.com</a></td>
</tr>
<tr>
<td>WAP port</td>
<td>9001, CSD number as 18881234567</td>
</tr>
<tr>
<td>username</td>
<td>john</td>
</tr>
</tbody>
</table>

**Configuration Rules**

The following filter rules on the ADC software filter client requests from different WAP gateways:

- **Filter term wap-redirect-1**—If the client IP address is between 10.168.43.0 to 0.168.43.255 and the requested URL is http://wap.example.com, then redirect the client request to http://wap.yahoo.com.

- **Filter term wap-redirect-2**—If the client IP address is between 10.46.6.0.0 to 10.46.6.0.255 and the requested URL is http://wap.example.com then redirect the client request to http://wap.google.com.

- **Filter term wap-redirect-3**—If the client IP address is between 10.23.43.0 to 10.23.43.255 and the requested URL is http://wap.p-example.com, then redirect the client request to http://10.168.224.227/top.
Assuming that each client is in a different subnet, configure the ADC software with three filters to redirect client requests from each subnet, to the URLs specified above.

1. Configure the required content-match strings.

```plaintext
[edit extensions adc adc-instance demo1]
content-match {
    text-search {
        string wap-example {
            http-header Host value wap.example.com;
        }
    }
    text-search {
        string wap-yahoo {
            http-header Host value wap.yahoo.com ;
        }
    }
    text-search {
        string wap-google {
            http-header Host value wap.google.com;
        }
    }
    text-search {
        string wap-p-example {
            http-header Host value wap.p-example.com;
        }
    }
    text-search {
        string wap-host-ip {
            http-header Host value 10.168.224.227;
        }
    }
}
```

2. Configure filter term wap-redirect-1.

```plaintext
[edit extensions adc adc-instance demo1]
filters {
    term wap-redirect-1 {
        from {
            protocol tcp;
            source-address 10.168.43.0/24;
            destination-port 80;
        }
        then {
            http-redirect {
                from wap-example to wap-yahoo;
            }
        }
    }
}
```

```
[edit extensions adc adc-instance demo1]
filters {
    term wap-redirect-2 {
        from {
            protocol tcp;
            source-address 10.46.6.0/24;
            destination-port 80;
        }
        then {
            http-redirect {
                from wap-example to wap-yahoo;
            }
        }
    }
}
```


```
[edit extensions adc adc-instance demo1]
filters {
    term wap-redirect-3 {
        from {
            protocol tcp;
            source-address 10.43.43.0/24;
            destination-port 80;
        }
        then {
            http-redirect {
                from wap-p-example to wap-host-ip;
            }
        }
    }
}
```

**TCP Service Port-Based HTTP Redirection**

In this example, the ADC will redirect traffic entering the ADC on one TCP service port, and redirect it through another service port.

- **Filter term wap-redirect-4** — Configure a filter on the ADC to examine the URL request http://10.46.6.231:80/Connect1.jad on TCP service port 80, and redirect that URL to TCP service port 90.

- **Filter term wap-redirect-5** — Configure a filter on the ADC that intercepts all traffic entering on TCP service port 80, and send it to 10.168.120.129 on TCP service port 8080.
1. Configure the required content-match strings.

```
[edit extensions adc adc-instance demo1]
content-strings {
  text-search {
    string any-host {
      http-header Host value any;
    }
  }
  text-search {
    string port90 {
      http-header Host value any:80;
    }
  }
  text-search {
    string any-host {
      http-header Host value any;
    }
  }
  text-search {
    string port8080 {
      http-header Host value any:8080;
    }
  }
}
```


```
[edit extensions adc adc-instance demo1]
filters {
  term wap-redirect-4 {
    from {
      protocol tcp;
      source-address 10.46.6.231/32;
      destination-port 80;
    }
    then {
      http-redirect {
        from any-host to port90;
      }
    }
  }
}
```

```plaintext
[edit extensions adc adc-instance demo1]
filters {
  term wap-redirect-5 {
    from {
      protocol tcp;
      source-address 10.46.6.231/32;
      destination-port 80;
    }
    then {
      http-redirect {
        from any-host to port8080;
      }
    }
  }
}
```

**MIME Type Header-Based Redirection**

In this example, the ADC receives a URL request from a mobile client and examines the Multipurpose Internet Mail Extensions (MIME) type header in the URL. If the URL contains a pre-defined MIME type, text, or URL, the ADC will replace the URL. Use the string index numbers to configure a redirection map for the filter.

- **Filter term wap-redirect-6** — The mobile client executes a request for a URL `http://dev.example.com/java/ toggle.jad`. If the MIME type is `text/vnd.foo.j2me.app-descriptor`, or if the URL contains `jad` or `jar` as an extension, it will replace the URL with: `http://mobile.example.com/4g/w?url=dev.example.com/nava/toggle.jad`. 
1. Configure the content-string parameters for filter term wap-redirect-6.

```
[edit extensions adc adc-instance demo1]
content-strings {
  string jad {
    text-search {
      url-string jad;
    }
  }
  string jar {
    text-search {
      url-string jar;
    }
  }
  string mime {
    text-search {
      http-header Accept value text/vnd.foo.j2me.app-descriptor;
    }
  }
  string mobile1 {
    text-search {
      http-header Host value mobile.example.com;
      url-string "/4g/w?url=$HOST_URL";
      no-regular-expression;
    }
  }
  string toggle-jad {
    text-search {
      http-header Host value mobile.example.com;
      url-string "/4g/w?url=$HOST/nava/toggle.jad";
      no-regular-expression;
    }
  }
  string dev-example {
    text-search {
      http-header Host value mobile.example.com;
      url-string "/4g/w?url=dev.example.com/$URL";
      no-regular-expression;
    }
  }
}
```
2. Configure the filters parameters for filter term wap-redirect-6.

The filter intercepts string jad, jar, and mime and redirects them based on strings mobile1, toggle-jad, and dev-example. The $HOST_URL is replaced with the incoming request from HOST and URL string. The $HOST is replaced with the incoming request from HOST string. The $URL is replaced with the incoming request from the URL string.

```plaintext
[edit extensions adc adc-instance demo1]
filters {
  term wap-redirect-6 {
    from {
      protocol tcp;
      source-address 10.46.6.231/32;
      destination-port 80;
    }
    then {
      http-redirect {
        from jad to mobile1;
        from jar to toggle-jad;
        from mime dev-example;
      }
    }
  }
}
```

**URL-Based Redirection**

A request for a URL can be redirected to another URL as follows:

1. Configure filter term wap-redirect-7 to redirect the URL as described above.

```
[edit extensions adc adc-instance demo1]
content-strings {
  text-search {
    string wap-example {
      http-header Host value wap.example.com;
    }
  }
  text-search {
    string wap-host-ip {
      http-header Host value 10.168.224.227;
      url-string /top;
    }
  }
}
filters {
  term wap-redirect-7 {
    from {
      protocol tcp;
      destination-port 80;
    }
    then {
      http-redirect {
        from wap-example to wap-host-ip;
      }
    }
  }
}
```

By default, the filter protocol is **any**. Change it to **udp**.

**Source IP from HTTP Header and Host Header-Based Redirection**

In this example, a filter is configured as follows:

- **Filter term wap-redirect-8**—If X-Foo-ipaddress: 10.168.100.* and the request is to http://wap.example.com, then redirect the request to wap.yahoo.com.
1. Configure filter term wap-redirect-8 to redirect the URL as described above.

```plaintext
[edit extensions adc adc-instance demo1]
content-strings {
  text-search {
    string wap-example {
      http-header Host value wap.example.com;
    }
  }
  text-search {
    string wap-yahoo {
      http-header Host value wap.yahoo.com;
    }
  }
  text-search {
    string x-foo {
      http-header X-foo-ipaddress value 10.168.100.*;
    }
  }
}
filters {
  term wap-redirect-8 {
    from {
      protocol tcp;
      destination-port 80;
    }
    then {
      content-match {
        match {
          content-strings x-foo;
          then {
            http-redirect {
              from wap-example;
              to wap-yahoo;
            }
          }
        }
      }
    }
  }
}
```
HTTP to HTTPS Redirection

To redirect HTTP traffic to HTTPS connections, the following filters can be set:

- **Example 1 - Filter term wap-redirect-9**—Configure a filter that intercepts HTTP traffic to http://www.abc.com and redirects it to https://www.abc.com.

- **Example 2 - Filter term wap-redirect-10**—Configure a filter that intercepts HTTP traffic directed to 205.10.10.10 and redirects it to HTTPS.


```plaintext
[edit extensions adc adc-instance demo1]
content-strings {
  text-search {
    string any-host {
      http-header Host value any;
    }
  }
  text-search {
    string https {
      http-header Host value any:443;
    }
  }
  text-search {
    string abc {
      http-header Host value www.abc.com;
    }
  }
}
filters {
  term wap-redirect-9 {
    from {
      protocol tcp;
      destination-port 80;
    }
    then {
      http-redirect {
        from abc to https;
      }
    }
  }
  term wap-redirect-10 {
    from {
      protocol tcp;
      destination-address 255.10.10.10/32
      destination-port 80;
    }
    then {
      http-redirect {
        from any to https;
      }
    }
  }
}
```
Health checking allows you to verify content accessibility in large websites. As content grows and information is distributed across different server farms, flexible, customizable content health checks are critical to ensure end-to-end availability.

The following health-checking topics are described in this chapter.

- **Real-Server Health Check Configuration** — This section explains the router’s default health check parameters per real server, which checks the status of each service on each real server.
- **Health Check Source IP Address** — This section explains how to configure the source IP address for health checks.
- **TCP Health Checks** — TCP health checks help verify the TCP applications that cannot be scripted.
- **Ping Health Checks** — This section explains how ping health checks are used for UDP services.
- **Application-Based Health Checks**:
  - **HTTP Health Checks** — This section provides examples of HTTP-based health checks using hostnames.
  - **DNS Health Checks** — This section explains the functionality of the DNS health checks using UDP packets.
  - **TFTP Health Check** — This section explains how to health check a real server using the TFTP protocol.
  - **SNMP Health Check** — This section explains how to perform SNMP health checks to real servers running SNMP Agents.
  - **FTP Server Health Checks** — This section describes how the File Transfer Protocol (FTP) server is used to perform health checks and explains how to configure the ADC software to perform FTP health checks.
  - **POP3 Server Health Checks** — This section explains how to use a Post Office Protocol Version 3 (POP3) mail server to perform health checks between a client system and a mail server and how to configure the ADC software for POP3 health checks.
  - **SMTP Server Health Checks** — This section explains how to use a Simple Mail Transfer Protocol (SMTP) mail server to perform health checks between a client system and a mail server and how to configure the ADC software for SMTP health checks.
  - **IMAP Server Health Checks** — This section describes how the mail server Internet Message Access Protocol (IMAP) protocol is used to perform health checks between a client system and a mail server.
  - **NNTP Server Health Checks** — This section explains how to use the Network News Transfer Protocol (NNTP) server to perform health checks between a client system and a mail server and how to configure the ADC software for NNTP health checks.
  - **RADIUS Server Health Checks** — This section explains how the RADIUS protocol is used to authenticate dial-up users to Remote Access Servers (RASs).
  - **SSL Server Health Checks** — This section explains how the ADC software queries the health of the SSL servers by sending an SSL client "Hello" packet and then verifies the contents of the server's "Hello" response.
  - **WAP Gateway Health Checks** — This section discusses how the ADC software provides connectionless and connection-oriented WSP health check for WAP gateways.
  - **LDAP Health Checks** — This section describes how to configure the ADC software to perform Lightweight Directory Access Protocol (LDAP) health checks to determine whether or not the LDAP server is running.
  - **Windows Terminal Server Health Checks** — This section describes how to configure the ADC software to perform Windows Terminal Server protocol (RDP) health checks.
  - **RTSP Health Check** — This section describes how to configure the ADC software to perform Real-Time Streaming Protocol (RTSP) to test connectivity.
• **SIP Health Checks**—This section describes how to configure the ADC software to manage Internet telephony and multimedia connections.

• **Script-Based Health Checks**—This section describes how to configure the ADC software to send a series of health-check requests to real servers or real-server groups and monitor the responses. Health checks are supported for TCP and UDP protocols, using either binary or ASCII content.

• **Direct Server Return Health Checks**—This section explains how, when Direct Server Return is configured at the group level, the health check performed is sent to the virtual IP address and not to the actual server IPs.

• **Server-Based Group Health Check**—This section describes how to configure an expression to fine-tune health checking.

• **Buddy Server Health Checks**—This section describes how to configure buddy server health checking.

• **Failure Types**—This section explains the service failed and server failed states.

## Real-Server Health Check Configuration

The ADC software monitors the servers in the real-server group and the load-balanced applications running on them. If a router detects that a server or application has failed, it will not direct any new connection requests to that server. When a service fails, the ADC software can remove the individual service from the load-balancing algorithm without affecting other services provided by that server.

By default, the router checks the status of each service on each real server every five (5) seconds. Sometimes, the real server can be too busy processing connections to respond to health checks. If a service does not respond to four consecutive health checks, the router, by default, declares the service unavailable. You can modify both the health check interval and the number of retries.

```
[edit extensions adc adc-instance demo1]
real-servers {
  real1 {
    health-check {
      interval 5;
      failure-retries 6;
      recovery-retries 5;
    }
  }
}
```

• **Interval**—The amount of time, in seconds, between polls of the real server by the router.

• **Failure-retries**—The number of times the router will attempt its check on the real server before marking the server as unavailable.

• **Recovery-retries**—The number of times the router will attempt to recover the real-server connection.

**Note:** Health checks are performed sequentially when used in conjunction with a virtual server configured with multiple services and groups. As a result, the actual health-check interval could vary significantly from the value set for it using the `interval` parameter.
**Health Check Source IP Address**

The ADC software uses a configured IP address to send the checks to the real servers. The IP address is configured per instance, as in the following example:

```
[edit extensions adc adc-instance demo1]
health-check-source {
  unit <unit> {
    family inet {
      address <ip>;
    }
  }
}
```

The health check itself is defined at the `group` parameter. Select a health check based on the application running on the real server in question. If the real server is an LDAP server, for example, use the LDAP health check method. The supported health check methods are detailed later in this chapter.

It is important to make sure that the server can answer connections from the IP address configured. This source IP address must be “routable” back to the router.

Each server in the load-balancing instance has a sub-unit attached to it (see Virtual Routers and Unit). Before the ADC software sends a health check to a server, it checks the sub-unit attached to the server, then chooses the source IP address to use for this server health check according to the address configured under the same unit in the `health-check-source` configuration.

As a result, each sub-unit attached to a server must have a matching address in the `health-check-source` configuration. This way the ADC software can send health checks to servers using this sub-unit. When no health check address is defined for the unit, all servers with this unit are in a failed status.

*Family inet* is the only supported family under the `health-check-source` configuration.

**TCP Health Checks**

TCP health checks are useful in verifying user-specific TCP applications that cannot be scripted. Session routers monitor the health of servers and applications by sending Layer 4 connection requests (TCP SYN packets) for each load-balanced TCP service to each server in the server group on a regular basis. The rate at which these connection requests are sent is a user-configurable parameter. These connection requests identify both failed servers and failed services on a healthy server. When a connection request succeeds, the session router quickly closes the connection by sending a TCP FIN (finished) packet.

The TCP health check is sent to the real server according to the following configuration:

1. Ports of services in which this real-server group is associated.
2. When the `server-listening-port` parameter at one or more such services is set to 0, then the health check is sent to the server listening ports as configured on each real server.
Example Setting the health check type to TCP

```
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check {
      tcp;
    }
  }
}
```

Ping Health Checks

Ping health checks verify if the real server is alive. The Layer 3 echo-echo reply health check is used for UDP services or when ping health checks are configured.

Note: Ping health check is the default health check for a group.

Example Ping health check configuration

```
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check {
      ping;
    }
  }
}
```

Application-Based Health Checks

Application-based health checks include the following:

- HTTP Health Checks
- DNS Health Checks
- TFTP Health Check
- SNMP Health Check
- FTP Server Health Checks
- POP3 Server Health Checks
- SMTP Server Health Checks
- IMAP Server Health Checks
- NNTP Server Health Checks
• RADIUS Server Health Checks
• SSL Server Health Checks
• WAP Gateway Health Checks
  — Configuring WSP Health Checks
  — Configuring WTP Health Checks
  — Configuring WTLS Health Checks
• LDAP Health Checks
• Windows Terminal Server Health Checks
• RTSP Health Check

HTTP Health Checks

HTTP-based health checks can include the hostname for Host headers. The Host header and health-check URL are constructed from the components shown in Table 9 on page 187.

Table 9: Host Header and URL Health Check Components

<table>
<thead>
<tr>
<th>Item</th>
<th>Option</th>
<th>Configured Under</th>
<th>Max. Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual server hostname</td>
<td>hostname</td>
<td>http-virtual-service &lt;name&gt;</td>
<td>9 characters</td>
</tr>
<tr>
<td>Domain name</td>
<td>domain-name</td>
<td>virtual-server &lt;name&gt;</td>
<td>35 characters</td>
</tr>
<tr>
<td>Server group health check field</td>
<td>http</td>
<td>group &lt;name&gt;</td>
<td>34 characters</td>
</tr>
</tbody>
</table>

If the Host header is required, an HTTP/1.1 GET will result. Otherwise, an HTTP/1.0 GET will result. HTTP health check is successful if you get a return code of 200.

Example 1:
hostname= everest
domain-name= example.com
http= index.html
Health check is performed using:
GET /index.html HTTP/1.1
Host: everest.example.com

Note: If content is not specified, the health check is performed using the / character.

Example 2:
hostname= (none)
domain-name= raleighduram.cityguru.com
http= /page/gen/?_template=alteon
Health check is performed using:
GET /page/gen/?_template=alteon HTTP/1.1
Host: raleighduram.cityguru.com
Example 3:
hostname= (none)
domain-name= jansus
http= index.html
Health check is performed using:
GET /index.html HTTP/1.1 Host: jansus

Example 4:
hostname= (none)
domain-name= (none)
http= index.html
Health check is performed using:
GET /index.html HTTP/1.0 (since no HTTP HOST: header is required)

Example 5:
hostname= (none)
domain-name= (none)
http= //everest/index.html
Health check is performed using:
GET /index.html HTTP/1.1
Host: everest

To configure HTTP health checks
1. Go to the real-server group for which you want to set up a health-checking method.
2. Configure the health check content.
   In this example, group1 and the http health check are used.

```bash
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check {
      http {
        url <url>;  
      }
    }
  }
}
```
3. Go to the virtual-server for which you want to set up this health-checking method.
4. Configure the hostname and domain name.

```
[edit extensions adc adc-instance demo1]
virtual-server <name>{
    domain-name <domain-name>;
    http-virtual-service http1 {
        hostname <hostname>;
    }
}
```

**DNS Health Checks**

The ADC software supports both TCP and UDP-based DNS health checking. This health check is performed by sending a DNS query over either protocol and watching for the server reply. The domain name to be queried can be modified using the configuration.

![Toolbox](image)

**To configure DNS health checks**

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   - In this example, group1 and both dnstcp and dns health checks are used.
   - The following is an example of the TCP DNS health check:

```
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        health-check {
            dnstcp {
                dnstcp host <hostname>;
            }
        }
    }
}
```

- The following is an example of the UDP DNS health check:

```
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        health-check {
            dns {
                host <hostname>;
            }
        }
    }
}
```
TFTP Health Check

The ADC software supports the Trivial File Transfer Protocol (TFTP) health check that uses the TFTP protocol to request a file from the server. At regular intervals, the ADC transmits TFTP read requests (RRQ) to all the servers in the group. The health check is successful if the server successfully responds to the RRQ. The health check fails if the ADC receives an error packet from the real server.

To configure TFTP health checks

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   In this example, group1 and the tftp health check are used.
3. Specify the filename that the ADC requests from the real servers.
   Make sure the file is less than 512 bytes, so that you do not incur additional traffic between the server and the router. Depending on the implementation of the TFTP daemon on the real servers you are health checking, you may have to specify the full pathname of the file (/tftpboot/<filename>) on some systems, while on others a filename is sufficient. By default, the ADC checks the /tftpboot folder.
   If full pathname is specified, add quotation marks; for example, "/tftpboot/test".

```bash
[edit extensions adc adc-instance demo1]
groups {  
group1 {  
    health-check {  
      tftp {  
        <filename>;
      }
    }
  }
}
```

SNMP Health Check

The ADC software supports SNMP health checks by sending an SNMP GET request to the real server running an SNMP-based agent. SNMP health checks can be used on any real servers, provided they have an SNMP agent. The SNMP health check is performed by polling a single variable within the MIB. For each SNMP health check, you configure the Object Identifier (OID) and community string to be queried. These values are obtained by using an MIB browser and an MIB compiler to find the OID of the desired variable.

The ADC software also allows you to configure the real-server weights to dynamically readjust, based on the SNMP health check response. To adjust the server weights based on the SNMP health check response, use the `adjust-server-weight` parameter at the group health check configuration. The router will then use the value sent in the SNMP health check response packet to dynamically adjust the real-server weight. If the value in the response packet is greater than 63, then 63 is used as the weight.
Example SNMP health check for a real-server group configuration

1. Select the real-server group you want to edit, specify the object identifier (OID), and set the community string on the device that notifies the SNMP agent on the real server to accept the SNMP packet from the device.

   In this example, group1 and the snmp health check are used.
   The OID is obtained from the MIB file of the device. For example, you can enter the OID for checking the status of a physical port on the router port that is connected to the group of IDS servers.
   This community string must match the community string specified on the real server.

2. Configure either a success value or failure value for the health check, if required. Add either the success-indicator <value> or failure-indicator <value> commands to the end of the health-check line.

   By default, when using SNMP, any response is marked as a success.

```
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check {
      snmp {
        oid 1.3.6.1.2.1.2.2.1.8.257;
        community real1;
        success-indicator 0;
      }
    }
  }
}
```

FTP Server Health Checks

The Internet File Transfer Protocol (FTP) provides facilities for transferring files to and from remote computer systems. Usually you are transferring a file that requires authority to log in and access files on the remote system. This protocol is documented in RFC 1123.

In normal Internet operation, the FTP server listens on well-known port number 21 for control connection requests. The client sends a control message indicating the port number on which the client is prepared to accept an incoming data connection request.

When a transfer is being set up, it is always initiated by the client. However, either the client or the server may be the sender of data. Along with transferring user requested files, the data transfer mechanism is also used for transferring directory listings from server to client.

**Note:** To configure the ADC for FTP health checks, the FTP server must accept anonymous user login.

**Configuring the ADC for FTP Health Checks**

Create any filename from an FTP server under the FTP server directory, for example, .txt, .exe, or .bin.
To configure FTP health checks

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   In this example, group1 and the ftp health check are used.
3. Specify the file-name that the ADC requests from the real servers.

```
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check {
      ftp {
        file-name <filename>;
      }
    }
  }
}
```

**POP3 Server Health Checks**

Post Office Protocol Version 3 (POP3) is intended to permit a workstation to dynamically access a mail drop on a server host. The POP3 protocol allows a workstation to retrieve mail that the server is holding for it. This protocol is documented in RFC 1939.

When the user on a client host wants to enter a message into the transport system, it establishes an SMTP connection to its relay host and sends all mail to it.

Initially, the server host starts the POP3 service by listening on TCP port 110. When a client host wants to make use of the service, it establishes a TCP connection with the server host.

**Configuring the ADC for POP3 Health Checks**

To support health checking on the UNIX POP3 server, the network administrator must configure a username and password value in the group configuration.

To configure POP3 health checks

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   In this example, group1 and the pop3 health check are used.
3. Configure the user-name for the POP3 connection.

```bash
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        health-check {
            pop3 {
                user-name {
                    <username>;
                }
            }
        }
    }
}
```

**SMTP Server Health Checks**

The Simple Mail Transfer Protocol (SMTP) transfers e-mail messages between servers reliably and efficiently. This protocol traditionally operates over TCP, port 25, and is documented in RFC 821. Most e-mail systems that send mail over the Internet use SMTP to send messages from one server to another; the messages can then be retrieved with an e-mail client using either POP or IMAP.

**Configuring the ADC for SMTP Health Checks**

To support SMTP health checking, the network administrator must configure a username and password value in the group configuration.

![Image](image.png)

**To configure SMTP health checks**

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   
   In this example, group1 and the smtp health check are used.
3. Configure the user-name parameter.

```bash
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        health-check {
            smtp {
                user-name <username>;
            }
        }
    }
}
```

**IMAP Server Health Checks**

Internet Message Access Protocol (IMAP) is a mail server protocol used between a client system and a mail server that allows a user to retrieve and manipulate mail messages. IMAP is not used for mail transfers between mail servers. IMAP servers listen to TCP port 143.
Configuring the ADC for IMAP Health Check

To support IMAP health checking, the network administrator must configure a username and password value in the group configuration.

To configure IMAP health checks
1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   In this example, group1 and the imap health check are used.
3. Configure the user-name and password for the IMAP connection.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check {
      imap {
        user-name <username>;
        password <pw>;
      }
    }
  }
}
```

NNTP Server Health Checks

Net News Transfer Protocol (NNTP) is a TCP protocol based upon text strings sent bidirectionally over 7-bit ASCII TCP channels, and listens to port 119. It is used to transfer articles between servers as well as to read and post articles. NNTP specifies a protocol for the distribution, inquiry, retrieval, and posting of news articles using a reliable stream-based transmission of news among the ARPA-Internet community. NNTP is designed so that news articles are stored in a central database, allowing a subscriber to select only those items he wishes to read.

NNTP is documented in RFC 977. Articles are transmitted in the form specified by RFC 1036.

Configuring the ADC for NNTP Health Checks

To configure NNTP health checks
1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   In this example, group1 and the nntp health check are used.
3. Configure the NNTP newsgroup name for the NNTP connection.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        health-check {
            nntp {
                newsgroup <newsgroup name>;
            }
        }
    }
}
```

**RADIUS Server Health Checks**

The ADC software allows you to use the Remote Authentication Dial-In User Service (RADIUS) protocol to health check the RADIUS accounting and authentication services on RADIUS servers. RADIUS is stateless and uses UDP as its transport protocol. Before you start configuring RADIUS health checks, ensure that you configure the network-attached storage IP parameter on the RADIUS server.

This parameter is the source IP address used for the health check. RADIUS accounting packets are sent over UDP to port 1812 or 1645 on the server.

```plaintext
[edit extensions adc adc-instance demo1]
health-check-source {
    unit <unit> {
        family inet {
            address <ip> ;
        }
    }
}
```

**Configuring RADIUS Authentication Health Checks**

To configure RADIUS authentication health checks

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   In this example, group1 and the radius health check are used.
3. Configure the check as authentication.
4. Configure the secret parameter.
5. Configure the **user-name** and **password** parameters for the RADIUS request.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check {
      radius {
        authentication;
        user-name <username>;
        password <password>;
        secret <secret>;
      }
    }
  }
}
```

**Configuring RADIUS Accounting Health Checks**

RADIUS accounting packets are sent over UDP to port 1813 or 1646 on the server.

![Tool](image)

To configure the health check RADIUS accounting service

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   - In this example, group1 and the radius health check are used.
3. Configure the check as accounting.
4. Configure the **user-name** and **password** for the RADIUS request.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check {
      radius {
        accounting;
        user-name <username>;
        password <password>;
        secret <secret>;
      }
    }
  }
}
```

**Note:** The ADC software allows you to couple the RADIUS accounting health check with the WAP health checks. If you set it up this way and one of the WAP health checks fails, then the accounting service is disabled.
**Configuring Combined RADIUS Health Checks**

Instead of configuring separate RADIUS authentication and accounting health checks, the ADC software lets you create a combined RADIUS health check.

This is done using both the authentication and accounting health check types. When a server group uses this health check type, the health check task queries the service port. If the service port is determined to be representing a RADIUS authentication service, then a RADIUS authentication health check is performed. If a RADIUS accounting service is detected, a RADIUS accounting health check is performed. If the service cannot be determined, a TCP health check is performed.

In using this health check type, a single group can be used to health check both types of services. If one service should fail, the other will go into a blocking state.

**To configure a combined RADIUS health check**

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   
   In this example, group1 and the radius health check are used.
3. Configure the check as authentication and accounting.
4. Configure the `user-name` and `password` for the RADIUS request.

```
[edit extensions adc adc-instance demo1]
groups { 
  group1 { 
    health-check { 
      radius { 
        authentication; 
        accounting; 
        user-name <username>; 
        password <password>; 
        secret <secret>; 
      } 
    } 
  } 
}
```

**Note:** RADIUS health checks also support non-standard RFC ports.

**SSL Server Health Checks**

The SSL-Hello health check option on the group configuration allows the ADC to query the health of the Secure Sockets Layer (SSL) servers by sending an SSL client "Hello" packet and then verifying the contents of the server's "Hello" response. The SSL health check is performed using the server listening port configured, under the virtual service configuration, or using the virtual service port when the server listening port is not configured.

The following is a summary of the SSL enhanced health check process:

- The ADC sends an SSL "Hello" packet to the SSL server.
- If it is up and running, the SSL server responds with the "Server Hello" message.
- The ADC verifies fields in the response and marks the service "Up" if the fields are OK.
During the handshake, you and the server exchange security certificates, negotiate an encryption and compression method, and establish a session ID for each session.

**To configure SSL-Hello health checks**

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   - In this example, group1 and the ssl-hello health check are used.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        health-check {
            ssl-hello;
        }
    }
}
```

**Note:** By default, SSL Version 3 is used. When SSL Version 2 is required, set the `version2` parameter.

**WAP Gateway Health Checks**

The Wireless Application Protocol, or WAP, is a specification for wireless devices that use TCP and HTTP as part of a standards-based implementation. The translation from HTTP/HTML to WAP/WML (Wireless Markup Language) is implemented by servers known as WAP gateways on the land-based part of the network.

The Wireless Session Protocol (WSP) is used within the WAP suite to manage sessions between wireless devices and WAP content servers or WAP gateways. The ADC software provides a content-based health check mechanism where customized WSP packets are sent to the WAP gateways, and the ADC verifies the expected response, in a manner similar to scriptable health checks.

WSP content health checks can be configured in two modes: connectionless and connection-oriented. Connectionless WSP runs on the UDP/IP protocol, ports 9200/9202. Connection-oriented traffic runs on ports 9201 and 9203. The ADC software can be used to load-balance the gateways in both modes of operation.

The ADC software allows you to configure three WAP gateway health check types (WSP, WTP, and WTLS) for all four WAP services (default ports 9200, 9201, 9202, and 9203) deployed on WAP gateway/servers. See **Table 10 on page 199**.
### Health Checking

**Note:** In the ADC software, all four WAP services are grouped together. If a health check to one of the services fails, then all four WAP services (9200, 9201, 9202, or 9203) are disabled.

#### What Is WTLS?

Wireless Transport Layer Security, or WTLS, is the security layer of the WAP, providing privacy, data integrity, and authentication for WAP services. WTLS, designed specifically for the wireless environment, is needed because the client and the server must be authenticated in order for wireless transactions to remain secure and because the connection must be encrypted. For example, a user making a transaction with a bank over a wireless device must know that the connection is secure and private and not subject to a security breach during transfer. WTLS is needed because mobile networks do not provide complete end-to-end security.

#### How WAP Health Check Works

The content of the WSP/UDP packet that is sent to the gateway is configured as a hexadecimal string, encapsulated in a UDP packet, and shipped to the server. Therefore, this byte string should include all applicable WSP headers.

The content that the ADC expects to receive from the gateway is also specified in the form of a hexadecimal byte string. The ADC matches each byte of this string with the received content.

If there is a mismatch of even a single byte on the received content, the gateway fails the health check. You can also configure an offset for the received WSP packet: a byte index to the WSP response content from where the byte match can be performed. The offset value (WSP or WTP) is for the receiving content only, and is the number of bytes from the beginning of the UDP data area, at which the comparison begins to match with the expected receive content.

**Note:** A maximum of 255 bytes of input are allowed on the ADC command line. You can remove spaces between the numbers to save space on the command line. For example, type `010203040506` instead of `01 02 03 04 05 06`.

<table>
<thead>
<tr>
<th>Health Check Type</th>
<th>Type of Traffic Mode</th>
<th>WAP Service</th>
<th>To configure, see</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSP</td>
<td>Connectionless WSP</td>
<td>9200</td>
<td>Configuring WSP Health Checks</td>
</tr>
<tr>
<td>WTP(^i)</td>
<td>Connection-oriented WTP + WSP</td>
<td>9201</td>
<td>Configuring WTP Health Checks</td>
</tr>
<tr>
<td>WTLS(^ii)</td>
<td>Encrypted connectionless WTLS + WSP</td>
<td>9202</td>
<td>Configuring WTLS Health Checks</td>
</tr>
<tr>
<td>WTLS</td>
<td>Encrypted connection-oriented WTLS + WTP + WSP</td>
<td>9203</td>
<td>Configuring WTLS Health Checks</td>
</tr>
</tbody>
</table>

\(^i\) – Wireless Transaction Protocol  
\(^ii\) – Wireless Transport Layer Security
**Coupling with RADIUS Accounting Health Check**
The ADC software allows you to couple the WAP health checks with the RADIUS accounting health check. If you enable the `couple-wap-radius-health` parameter and the RADIUS accounting health check fails, then all of the WAP services are brought down. Similarly, if one of the WAP health checks fail, then all the WAP services and the RADIUS accounting service fails.

**Configuring WSP Health Checks**
Configure the health check for connectionless and unencrypted WAP traffic.

![Tips](image)

**To configure WSP health checks**
1. Go to the real-server group for which you want to set up a health-checking method.
2. Configure the health check content.
   - In this example, group1 and the wap <wsp> health check are used.
3. Set the `send` value to provide the content to be sent to the WSP gateway.
4. Set the `receive` value to provide the content the ADC software should expect from the WSP gateway.
5. Set the `offset` value.
   - The offset value is the number of bytes from the beginning of the UDP data area, at which the comparison begins to match with the expected receive content. For 9200 service, the UDP data area is the WSP response content.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check {
      wap <wsp> {
        send <hex string>
        receive <hex-string>
        offset <offset>
      }
    }
  }
}
```

**Configuring WTP Health Checks**
Configure the health check for connection-oriented, unencrypted WAP traffic.

![Tips](image)

**To configure WTP health checks**
1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   - In this example, group1 and the wap <wtp> health check are used.
3. Set the `connect-content` parameter.

   Use the `connect-content` parameter to enter the content for the first ADC-generated WSP session packet. This command allows you to customize the headers in the `connect-content` message.

4. Set the `send` value to provide the content to be sent to the WTP gateway.

5. Set the `receive` value to provide the content the ADC software should expect from the WTP gateway.

6. Set the `offset` value.

   The offset value is the number of bytes from the beginning of the UDP data area, at which the comparison begins to match with the expected receive content. For 9200 service, the UDP data area is the WSP response content.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check {
      wap <wtp> {
        connect-content <string>;
        send <hex string>;
        receive <hex-string>;
        offset <offset>;
      }
    }
  }
}
```

### Configuring WTLS Health Checks

Wireless Transport Layer Security (WTLS) is used to health check encrypted WAP traffic. The encrypted WAP traffic can be connectionless (WTLS+WSP) or connection-oriented (WTLS+WTP+WSP). The connectionless encrypted WTLS traffic uses default port 9202, and the connection-oriented encrypted WTLS traffic uses port 9203. The ADC sends a new WTLS Client Hello to the WAP gateway, and checks to see if it receives a valid WTLS Server Hello back from the WAP gateway.

The contents for the WTLS health check are not configurable and are generated by the ADC.

To configure WTLS health checks

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.

   In this example, group1 and the wap <wtls> health check are used.
LDAP Health Checks

Lightweight Directory Access Protocol (LDAP) health checks enable the ADC to determine whether the LDAP server is alive or not. LDAP versions 2 and 3 are described in RFC 1777 and RFC 2251.

The LDAP health check process consists of three LDAP messages over one TCP connection:

- **Bind request**—The ADC first creates a TCP connection to the LDAP server on port 339, which is the default port. After the connection is established, the ADC initiates an LDAP protocol session by sending an anonymous bind request to the server.

- **Bind response**—On receiving the bind request, the server sends a bind response to the ADC. If the resulting code indicates that the server is alive, the ADC marks the server as up. Otherwise, the ADC marks the server as down even if the server did this because the server did not respond within the timeout window.

- **Unbind request**—If the server is alive, the ADC sends a request to unbind the server. This request does not require a response. It is necessary to send an unbind request as the LDAP server may crash if too many protocol sessions are active.

If the server is up, the ADC closes the TCP connection after sending the unbind request. If the server is down, the connection is torn down after the bind response, if one arrives. The connection will also be torn down if it crosses the timeout limit, irrespective of the server's condition.

**Configuring the ADC for LDAP Health Checks**

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   - In this example, group1 and the ldap health check are used.
3. Configure the LDAP health check to verify the domain name in the `dn-string` field.
   - For example, to verify the domain name "ldap.example.com", set the `dn-string` to include "cn=Admin, dc=ldap, dc=example, dc=com", and set the password = test.
4. Set the version of LDAP. The default is version 3.
Windows Terminal Server Health Checks

Application health checking can be performed on Windows Terminal Servers similar to LDAP health checking. The WTS protocol is a binary protocol, so scripted health checks cannot be used in this instance. Therefore, this health check only entails checking server availability on TCP port 3389.

To enable WTS health checking on a real-server group
1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   In this example, group1 and the wts health check are used.
3. Configure the user-name and password for the check (optional).

```
[edit extensions adc adc-instance demol]
groups {
    group1 {
        health-check {
            ldap {
                dn-string "cn=Admin,dc=ldap,dc=example,dc=com";
                password test;
                version2;
            }
        }
    }
}
```

RTSP Health Check

The ADC software supports health checking with the Real-Time Streaming Protocol (RTSP). This health check works by sending RTSP packets to the server group to test connectivity.

Configuring the ADC for RTSP Health Checks

To configure DHCP health checks
1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content.
   In this example, group1 and the rtsp health check are used.
3. Configure the file-name for the check (optional).

```plaintext
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        health-check {
            rtsp [file-name <filename>];
        }
    }
}
```

# SIP Health Checks

The Session Initiation Protocol (SIP) is an application-level control (signaling) protocol for Internet multimedia conferencing, telephony, event notification, and instant messaging. The protocol initiates call setup, routing, authentication, and other feature messages to endpoints within an IP domain. SIP is used to locate users (where the caller and called parties are), determine user capability (what type of protocol TCP, UDP, and other capabilities the user can support), user availability, call setup (how to create the call), and call handling (how to keep the call up and how to bring down the call).

The SIP health check process consists of an SIP request sent to the server to determine whether the SIP server is alive. The SIP health check is successful if the ADC software gets a return code of 200 OK.

There are two types of SIP health checks that are sent to the servers, according to user configuration:

- **SIP Ping**—An SIP Ping request is sent to the SIP server.
- **SIP Option**—An SIP options request is sent to the SIP server.

## Configuring the SIP Health Checks

**To configure the ADC software to verify if the SIP server is alive**

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the SIP health check to verify the user-name and domain-name.
   a. SIP ping health check (default).

```plaintext
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        health-check {
            sip {
                user-name <username>;
                domain-name <domain>;
            }
        }
    }
}
```
b. SIP options health check.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    health-check {
      sip {
        options;
        user-name <username>;
        domain-name <domain>;
      }
    }
  }
}
```

**Script-Based Health Checks**

Health check scripts dynamically verify application and content availability by executing a sequence of tests based on `send` and `expect` commands.

**Configuring Script-Based Health Checks**

You can configure the ADC software to send a series of health check requests to real servers or real-server groups and monitor the responses. Both ASCII and binary-based scripts, for TCP and UDP protocols, can be used to verify application and content availability.

The benefits of using script-based health checks are:

- Ability to send multiple commands.
- Checks for any return ASCII string or binary pattern.
- Tests availability of different applications.
- Tests availability of multiple domains or websites.

The ADC software supports the following capacity for a single ADC:

- 6K bytes per script
- 64 scripts per load-balancing instance

The commands are grouped together as a list so you can change their order. Each script command is made up of one or more tcp-command or udp-command containers. Commands exist to open a connection to a specific TCP or UDP port, send a request to the server, and expect an ASCII string or binary pattern. The string or pattern configured with an `expect` (or in the case of binary, `binary-expect`) command is searched for in each response packet. If it is not seen anywhere in any response packet before the real-server health-check interval expires, the server does not pass the `expect` (or `binary-expect`) step and fails the health check. A script can contain any number of these commands, up to the allowable number of characters that a script supports.

**Notes:**

>>> There is no need to use double slashes when configuring a script that uses special characters with single slashes. For example, the script entry `GET /index.html HTTP/1.1\r\n\n\nHOST:www.hostname.com\r\n\r\n` does not require the use of `\r` or `\n` to ensure proper functioning of the script.

>>> Only one protocol can be configured per script.
Script Formats

Health check script formats use different commands based on whether the content to be sent is ASCII-based or binary-based.
Each script should start with the command open <protocol port number>,<protocol-name>. The next line can be either a send or expect (for ASCII-based), or bsend or bexpect (binary-based).

ASCII-Based Health Check
The general format for TCP-based health-check scripts is as follows:

```
[edit extensions adc adc-instance demo1]
custom-health-check {
  script script1 {
    tcp-commands ASCII {
      command <name> open <port>;
      command <name> send request <text>;
      command <name> expect response <text>;
      command <name> send request <text>;
      command <name> expect response <text>;
      command <name> send request <text>;
      command <name> expect response <text>;
    }
  }
}
```

Binary-Based UDP Health Check
The general format for UDP binary-based health check scripts is shown below. Specify the binary content in hexadecimal format.

```
[edit extensions adc adc-instance demo1]
custom-health-check {
  script script2 {
    udp-commands Binary-UDP {
      command <name> open <port>;
      command <name> binary-request request 1;
      command <name> binary-expect response 1 {
        offset <count>;
        depth <number of bytes from offset to count>;
      }
    }
  }
}
```

Note: UDP-based health check scripts can use either ASCII strings or binary patterns.
**Binary-Based TCP Health Check**

The `binary-send` and `binary-expect` commands are used to specify binary content. The `offset` and `depth` commands are used to specify where in the packet to start looking for the binary content. For example, if your script is configured to look for an HTTP 200 (ok) response, this typically appears starting from the seventh byte in the packet, so an offset value of 7 can be specified.

```
[edit extensions adc adc-instance demo1]
custom-health-check {
  script script3 {
    tcp-commands 200OK {
      command 10 open 80;
      command 20 binary-send "<binary content for request 1>";
      command 30 binary-expect "<binary content for response 1>" {
        offset 7;
        depth 10;
        wait 100;
      }
    }
  }
}
```

**Notes:**

>>> If you are doing HTTP 1.1 pipelining, you need to individually open each response in the script.

>>> For HTTP-based health checks, the first word is the method. The method is usually the `get` command. However, HTTP also supports several other commands, including `put` and `head`. The second word indicates the content desired, or request-URI, and the third word represents the version of the protocol used by the client.

If you supplied HTTP/1.1 for the protocol version, you must also add in the following line: `Host: www.hostname.com`

**Example**

```
command <name> send GET /index.html HTTP/1.1\r\n Host: www.hostname.com\r\n\r\n
```

This is known as a host header. It is important to include because most websites now require it for proper processing. Host headers are optional in HTTP/1.0 but are required with HTTP/1.1+

>>> In order to tell the application server you have finished entering header information, a blank line of input is needed after all headers. At this point, the URL is processed and the results returned to you.

>>> If you make an error, enter a `rem` to remove the last typed script line entered. To remove more than one line, enter a `rem` for each line that must be removed.
The ADC provides the "\" prompt, which equals one Enter keystroke. When using the send command, note what happens when you type the send command with the command string. When you type send, press Enter and allow the ADC to format the command string (that is, \ versus \\).

**Scripting Commands**

Listed below are the currently available commands for building a script-based health check:

- **open**—Specifies which destination real-server UDP port to use; for example, OPEN 9201. After entering the destination port, you is prompted to specify a protocol; choose udp.
- **send**—Specifies the send content in raw hexadecimal format.
- **binary-send** (for binary content only)—Used to specify binary content (in hexadecimal format) for the request packet.
- **expect**—Specify the expected content in raw hexadecimal format.
- **binary-expect** (for binary content only)—Used to specify the binary content (in hex format) to be expected from the server response packet.
- **offset** (for binary content only)—Specifies the offset from the beginning of the binary data area to start matching the content specified in the binary-expect command. The offset command is supported for both UDP and TCP-based health checks. Specify the offset command after a binary-expect command if an offset is desired. If this command is not present, an offset of zero is assumed.
- **depth** (for binary content only)—Specifies the number of bytes in the IP packet that should be examined. If no offset value is specified, depth is specified from the beginning of the packet. When depth is not specified, it is the length of the content. This means that the content is expected exactly at the offset specified (or 0 when the offset is not specified).
- **wait**—Specifies a wait interval before the expected response is returned. The wait window begins when the send string is sent from the ADC. If the expected response is received within the window, the wait step passes. Otherwise, the health check fails. The wait window is in units of milliseconds. When the wait value is not specified the script waits according to the real-server configured interval.
- Wildcard character (*)—Used to trigger a match as long as a response is received from the server. The wildcard character is allowed with the binary-expect command, as in 20 binary-expect *. Any offset or depth commands that follow a wildcard character is ignored.

**Scripting Guidelines**

- Use generic result codes that are standard and defined by the RFC, as applicable. This helps ensure that if the server software changes, the servers will not start failing unexpectedly.
- Avoid tasks that may take a long time to perform or the health check will fail. For example, avoid tasks that exceed the interval for load balancing.

**Adding the Script to a Group**

Once all of the configurations are done creating the script, the script must be set up and configured for use by the group.
To configure a script health check for a group

1. Go to the real-server group for which you want to set up a health checking method.
2. Configure the health check content. In this example, group1 and the script health check are used.
3. Configure the script-name for the check.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        health-check {
            script <script-name>;
        }
    }
}
```

**Script Configuration Examples**

**Example 1: A basic ASCII TCP-based health check**

Configure the ADC software to check a series of Web pages (HTML or dynamic CGI scripts) before it declares a real server is available to receive requests.
You must use quotes (") to indicate the beginning and end of each command string.

```plaintext
[edit extensions adc adc-instance demo1]
custom-health-check {
    script script1 {
        tcp-commands URL1 {
            command 10 open 80;
            command 20 Send "GET /index.html HTTP/1.1\r\n\nHOST:www.hostname1.com\r\n\n\n";
            command 30 Expect "HTTP/1.1 200";
        }
        tcp-commands URL2 {
            command 50 open 80;
            command 60 Send "GET /index.html HTTP/1.1\r\n\nHOST:www.hostname2.com\r\n\n\n";
            command 70 Expect "HTTP/1.1 200";
        }
        tcp-commands SSL1 {
            command 90 open 443;
            ...
        }
    }
}
```

**Note:** When you are entering the send string as an argument to the `send` command, you must type two "\"s before an "n" or "r."

**Example 2: GSLB URL health check**

Using the customized health check scripts feature, you can set up health check statements to check all the content-strings associated with all of the real servers.

Virtual Server 1 is using the following real servers:
- Real Server 1 and Real Server 2: "images"
- Real Server 3 and Real Server 4: "html"
- Real Server 5 and Real Server 6: "cgi" and "bin"
- Real Server 7: "any"

Virtual Server 2 is using the following real servers:
- Real Server 1 and Real Server 2: "images"
- Real Server 3 and Real Server 4: "html"
- Real Server 5 and Real Server 6: "cgi" and "bin"
- Real Server 7: "any"
A sample script is shown below:

```ada
[edit extensions adc adc-instance demol]
custom-health-check {
  script script2 {
    tcp-commands URL1 {
      command 10 open 80;
      command 20 send "GET /images/default.asp HTTP/1.1\r\n\n";
      command 30 expect "HTTP/1.1 200";
    }
    tcp-commands URL2 {
      command 50 open 80;
      command 60 send "GET /install/default.html HTTP/1.1\r\n\n";
      command 70 expect "HTTP/1.1 200";
    }
    tcp-commands URL3 {
      command 90 open 80;
      command 100 send "GET /script.cgi HTTP/1.1\r\n\n";
      command 110 expect "HTTP/1.1 200";
    }
  }
}
```

Script-based health checking is intelligent in that it will only send the appropriate requests to the relevant servers. In the example above, the first GET statement will only be sent to Real Server 1 and Real Server 2. Going through the health-check statements serially will ensure that all content is available by at least one real server on the remote site.

Configure the remote real-server IP address (the virtual-server IP address of the remote site) to accept "any" URL requests. The purpose of the first GET is to check if Real Server 1 or Real Server 2 is up—that is, to check if the remote site has at least one server for "images" content. Either Real Server 1 or Real Server 2 will respond to the first GET health check.

If all the real-server IP addresses are down, Real Server 7 (the virtual-server IP address of the remote site) will respond with an HTTP Redirect (respond code 302) to the health check. Thus, the health check will fail as the expected response code is 200, ensuring that the HTTP Redirect messages will not cause a loop.

---

**Example 3: A UDP-based health check using binary content**

Health-check scripts can be designed to be sent over the UDP protocol with a few minor differences from a TCP-based health-check script.
The following is an example of a UDP-based script that uses binary content to health check the UDP port on a real server.

```
[edit extensions adc adc-instance demo1]
custom-health-check {
  script script3 {
    udp-commands Binary-UDP {
      command 10 open 53, udp;
      command 20 binary-send "53 53 01 00 00 01 00 00 00 00 00 00 03 77 77
        77 04 74 65 73 74 03 63 6f 6d 00 00 01 00 01";
      command 30 binary-expect "00 01 00 01" {
        offset 1;
        depth 32;
        wait 1024;
      }
    }
  }
}
```

**Example 4: A TCP-based health check using binary content**

Health-check scripts can also be sent over the TCP protocol using binary content. The following is an example of a TCP-based script that uses binary content to send an HTTP GET request, and expect an HTTP 200 response.

```
[edit extensions adc adc-instance demo1]
custom-health-check {
  scripts script4 {
    tcp-commands Binary-TCP {
      command 10 open 80, tcp;
      command 20 binary-send "474554202F746573742E68746D20"
        "485454502F312E300D0A0D0A";
      command 30 binary-expect "203230" "3020" {
        offset 7;
        depth 10;
        wait 100;
      }
    }
  }
}
```

**Verifying Script-Based Health Checks**

Use the `show` command at either the real-server or the group level to display information relating to the health check.

```
user@host> show extensions adc real-server
```

Or

```
user@host> show extensions adc group
```
Direct Server Return Health Checks

Direct Server Return health checks are used to verify the existence of a server-provided service where the server replies directly back to the client without responding through the virtual-server IP address. In this configuration, the server is configured with a real-server IP address and virtual-server IP address. The virtual-server IP address is configured to be the same address as your virtual-server IP address. When Direct Server Return health checks are used, the specified health check is sent originating from the configured health check address. It is destined for the virtual-server IP address with the MAC address that was acquired from the real-server IP Address Resolution Protocol (ARP) entry. Direct Server Return is configured at the group level. If a group is configured with "direct-server-return" the health check performed is sent to the virtual IP and not to the actual server IPs.

The ADC software lets you to perform health checks for Direct Server Return configurations (for more information, see Direct Server Return). The router is able to verify that the server correctly responds to requests made to the virtual-server IP address, as required in Direct Server Return configurations. To perform this function, the real-server IP address is replaced with the virtual-server IP address in the health check packets that are forwarded to the real servers for health checking. With this feature enabled, the health check will fail if the real server is not properly configured with the virtual-server IP address.

Configuring the Router for Direct Server Return

Example Direct Server Return VIP configuration

```plaintext
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    direct-server-return;
  }
}
```

There is no special configuration to do in order to make the health check work for direct-server-return. When you set the group to work as direct-server-return, the health check behavior is adjusted as well.

Server-Based Group Health Check

The ADC software allows you to configure an expression to fine tune the selected health check for a real-server group.

For example, you have configured a real-server group with four real servers. Two of the real servers are handling the contents of the website and the other two real servers are handling audio files. If the two content servers fail due to traffic distribution, then you want the two audio servers to fail automatically. However, you want the audio servers to be up if at least one of the content servers is up.

The server-based group health check feature allows you to create a boolean expression to health check the real-server group based on the state of the virtual services. This feature supports two boolean operators: AND and OR. The two boolean operators are used to manipulate TRUE/FALSE values as follows:

- **OR operator (|)**—A boolean operator that returns a value of TRUE if either (or both) of its operands is TRUE. This is called an inclusive OR operator.
AND operator (&) — A boolean operator that returns a value of TRUE if both of its operands are TRUE.

Using parenthesis with the boolean operators, you can create a boolean expression to state the health of the server group. The following two boolean expressions show two examples with real servers 1, 2, 3, and 4 in two different groups.

**Examples**

**A**  
(1|2)&(3|4)  
Real servers real1, real2, real3, and real4 are configured in group 1 and assigned to virtual service \(x\) in virtual server 1. The boolean expression is used to calculate the status of a virtual service using group 1 based on the status of the real servers.  
Virtual service \(x\) of virtual server 1 is marked UP if real servers real1 or real2 and real servers real3 or real4 are health checked successfully.

```bash
[edit extensions adc adc-instance demo1]
groups {
  group1 {
    real-servers real1;
    server-based-group-health (real1 | real2)&(real3 | real4);
  }
}
```

**B**  
(1&2)|(2&3)|(1&3)  
Real servers real1, real2, and real3 are configured in group 2 and assigned to virtual service \(x\) in virtual server 1. The boolean expression is used to calculate the status of the virtual service using group 2 based on the status of the real servers.  
Virtual service \(x\) of virtual server 1 is marked UP only if at least two of the real servers are health checked successfully.

```bash
[edit extensions adc adc-instance demo1]
groups {
  group2 {
    real-servers real1;
    server-based-group-health
      (real1&real2)|(real2&real3)|(real1&real3);
  }
}
```

**Buddy Server Health Checks**

Buddy server health checking gives the administrator the ability to tie the health of a real server to another real server. This real server can be in the same real-server group but can be in a separate group. In this configuration, a real server will only be considered healthy if the buddy it is associated with is healthy also.

**Note:** It is not mandatory for a buddy server group to be part of any virtual service.
**Figure 33 on page 215** is an example network topology.

**To define a real server as a buddy server for another real server**

1. Configure real servers for use as buddy servers.
   
   In the example, buddy servers buddy1 and buddy2 are configured.

   ```
   [edit extensions adc adc-instance demo1]
   real-servers {
     buddy1 {
       address 100.100.100.1;
     }
     buddy2 {
       address 100.100.100.2;
     }
   }
   ```

2. Create a group and associate the buddy servers to that group.
   
   In the example, buddy servers buddy1 and buddy2 are associated with server group group1.
3. Configure real servers and associate buddy servers to them.

In the example, real server real1 is configured and buddy servers buddy1 is associated to it, and real server real2 is configured and buddy server buddy2 is associated to it.

4. Configure real-server groups and assign real servers to it.

In the example, server group group2 is configured and real servers real1 and real2 are associated to it.
5. Configure a virtual server, associate a DNS service to the virtual server, and associate a server group to the DNS service.

   In the example, virtual server virt1 has DNS service dns1 associated to it, which in turn has server group group1 associated to it.

   ```
   [edit extensions adc adc-instance demo1]
   virtual-server virt1 {
       address 120.10.10.10;
       dns-virtual-service dns1 {
           port 53;
           group group1;
       }
   }
   ```

### Failure Types

The ADC software includes the following failure types:

- **Service Failure**
- **Server Failure**

### Service Failure

If a certain number of connection requests for a particular service fail, the ADC software places the service into the *service failed* state. While in this state, no new connection requests are sent to the server for this service. However, if graceful real-server failure is enabled, state information about existing sessions is maintained and traffic associated with existing sessions continues to be sent to the server. Connection requests to, and traffic associated with, other load-balanced services continue to be processed by the server.

#### Example

A real server is configured to support HTTP and FTP within two real-server groups. If a session ADC detects an HTTP service failure on the real server, it removes that real-server group from the load-balancing algorithm for HTTP, but keeps the real server in the mix for FTP. Removing only the failed service from load balancing allows users access to all healthy servers supporting a given service.

When a service on a server is in the *service failed* state, the ADC software sends Layer 4 connection requests for the failed service to the server. When the session ADC has successfully established a connection to the failed service, the service is restored to the load-balancing algorithm.

### Server Failure

If all load-balanced services supported on a server fail to respond to ADC connection requests within the specified number of attempts, then the server is placed in the *server failed* state. While in this state, no new connection requests are sent to the server. However, if graceful real-server failure is enabled, state information about existing sessions is maintained and traffic associated with existing sessions continues to be sent to the server.

All load-balanced services on a server must fail before the ADC places the server in the *server failed* state.
The server is brought back into service as soon as the first service is proven to be healthy. Additional services are brought online as they are subsequently proven to be healthy.

**Preventing a Flood of Server Connections**

The ADC performs a slow start on the real server that returns after the health check fails to prevent a flood of connections. This results for groups configured with least-connections as the load-balancing metric. When the real server comes up, it contains zero connections and all the new connections are directed to it. This heavy connection flow brings down the real server.

To prevent this flood, the ADC software temporarily changes the group load-balancing method to round-robin for some time and reverts back to the least-connections method assuming the next real server that comes up is ready to accept new connections.

To check the slow-start mode of a real server in a group

- Set the `slow-start` parameter under the group you want to edit.

```plaintext
[edit extensions adc adc-instance demo1]
groups {
    group1 {
        slow-start <time in seconds>;
    }
}
```
Chapter 6 – High Availability

The ADC software uses the Junos OS-based high-availability mechanism called the Redundant Multiservice PIC (RMS) for high availability.

RMS Introduction

A Redundant Multiservice PIC (RMS) is a virtual Multiservices PIC that actually contains two Multiservices PICs, a primary and a secondary PIC. While working, the primary Multiservices PIC is the active PIC, receiving traffic that is directed to the RMS, while the secondary Multiservices PIC is its backup. If the primary Multiservices PIC fails, the secondary Multiservices PIC becomes active and handles all the traffic sent to the RMS. When the failed primary PIC is restored, it comes up as the backup.

Example Configuring an RMS in Junos OS

[edit interfaces]
rms0 {
  redundancy-options {
    primary ms-1/0/0;
    secondary ms-1/1/0;
    hot-standby;
  }
}

The following terms are important to understanding the setup and use of RMS:

- **Primary/secondary multiservice-pic** — The roles of the multiservice-pic as defined in the configuration. This attribute is static, as long as no change was done in configuration.

- **Active/backup Multiservices PIC** — The current roles of the Multiservices PICs. The active Multiservices PIC is the Multiservices PIC that is currently handling traffic. The backup Multiservices PIC is a hot-standby for the primary. These roles are dynamic. The roles are determined by Junos OS according to the current Multiservices PIC in addition to the configuration.

  By default, the primary Multiservices PIC will become active at router initialization.

- **Failover** — When the active Multiservices PIC experiences a problem, or when you command the router to switch the active and backup Multiservices PICs, a failover event occurs. As a result of a failover event, the Multiservices PIC that was in the backup role becomes active and starts handling traffic. If the previously active Multiservices PIC becomes available again, it takes over the backup role.

For more details on RMS, please see your Junos OS documentation.
**RMS Support**

The ADC software is configured to run on Multiservices-DPCs. An RMS is treated just like any other Multiservices-DPC NPU. The main difference is that in RMS there are two NPUs that run the ADC software package. While one of the NPU is handling traffic, the other NPU is a backup in hot-standby mode. The backup NPU runs the ADC software package and it is ready to handle traffic when needed.

You should configure the ADC software to run fully on physical (non-RMS) Multiservices-DPCs or to run fully on RMS interfaces.
To configure the ADC software to run on an RMS

1. Configure two Multiservices-DPC NPUs to run the ADC software. These two NPUs should have the same configuration.

```
[edit]
chassis {
  fpc 1 {
    pic 0 {
      adaptive-services {
        service-package {
          extension-provider {
            control-cores 1;
            data-cores 7;
            object-cache-size 512;
            wired-process-mem-size 256;
            data-flow-affinity {
              hash-key layer-3;
            }
            package adc-ctrl;
            package adc-data;
          }
        }
      }
    }
    fpc 1 {
      pic 1 {
        adaptive-services {
          service-package {
            extension-provider {
              control-cores 1;
              data-cores 7;
              object-cache-size 512;
              wired-process-mem-size 256;
              data-flow-affinity {
                hash-key layer-3;
              }
              package adc-ctrl;
              package adc-data;
            }
          }
        }
      }
    }
  }
}
```
2. Configure rms0 to have ms-1/0/0 as a primary NPU and ms-1/1/0 as a secondary NPU.

```plaintext
[edit interfaces]
rms0 {  
  redundancy-options {  
    primary ms-1/0/0;  
    secondary ms-1/1/0;  
    hot-standby;  
  }  
}
```

3. Configure the `slb-instance` to work on the RMS interface.

```plaintext
[edit extensions adc]
adc-instance demo {  
  router-interfaces {  
    ms-interfaces {  
      rms0;  
    }  
  }  
}
```

**Notes:**

- >> When the ADC software is configured to run on an RMS interface, both Multiservices-DPC NPUs in the RMS are always running the ADC software. At any given time, only one of the multiservices interfaces within the RMS is actually load-balancing traffic.

- >> When the ADC software is configured to run on RMS interfaces and on regular (physical) Multiservices-DPC NPUs, that are not inside an RMS, only the data that is sent to the RMS will have the backup fallback. The RMS is a 1:1 Multiservices-DPC NPU backup option, and a Multiservices-DPC NPU can back up only one other Multiservices-DPC NPU.

## Connection Synchronization

The Junos OS makes traffic move transparently from the active Multiservices PIC to the backup Multiservices PIC in case of failover. However, the traffic shift is not enough for transparent failover in some load-balancing cases. In stateful load balancing, for example, some information is kept on the Multiservices PIC for each session. When the failover happens, the ADC software must make sure this information also exists in the new active Multiservices PIC that used to be the backup Multiservices PIC.

The process of moving all relevant information from the active Multiservices PIC to the backup Multiservices PIC is called connection synchronization. The connection synchronization is a background process that continuously sends data from the active Multiservices PIC to the backup Multiservices PIC.

The data that is being synchronized is in the connection table and the persistency table. The connection table contains the online information on the current open connections that are handled by the ADC software. The persistency table contains long-lasting information about server-selection persistency for existing and connections.
By default, if you configure the ADC software to work with RMS, all relevant data is synchronized. You can enable the synchronization data transfer from the active Multiservices PIC to the backup Multiservices PIC by configuration. This can be done for troubleshooting or for any other reason using the configuration.

The connection synchronization can be disabled per virtual-service or per filter (which will enable the synchronization of all data related to this virtual service/filter only). The connection synchronization can also be disabled for an entire adc-instance for debugging purposes.

Example Enabling the connection synchronization data transfer for a specific service

```
[edit extensions adc adc-instance demo]
virtual-server v1 {
    http-virtual-service http1 {
        sync-connections;
    }
}
```

Example Enabling the connection synchronization data transfer for a specific filter

```
[edit extensions adc adc-instance demo filters]
term term1 {
    then {
        load-balance {
            sync-connections;
        }
    }
}
```

Example Disabling the connection synchronization data transfer for an entire adc-instance

```
[edit extensions adc]
adc-instance demo {
    no-connections-sync;
}
```

Note: Since connection synchronization involves transferring data related to the connection table, if you configured the per-packet-load-balancing parameter under virtual-service, the synchronization will have no effect on this service. Data on such services does not have any data to mirror.
Chapter 7 – Content String Handling

This chapter describes how to create and manage the content-string content used for configuring content-intelligent load-balancing and redirection features described elsewhere in this manual. Content string handling applies to the DNS, RTSP, HTTP services, and to filters. For more information, see Domain Name System Server Load Balancing, Real-Time Streaming Protocol Server Load Balancing, HTTP Server Load Balancing, and Filtering.

The following topics are addressed in this chapter:

- **Avoidance HTTP String Matching for Real Servers**
- **Regular Expression Matching**
- **Using Variables in Strings for HTTP**
- **Content Precedence Lookup**
- **String Case Sensitivity**
- **Configurable HTTP Methods**

**Note:** All content-intelligent routing features cannot be used in conjunction with fast load balancing.

### Avoidance HTTP String Matching for Real Servers

URL-based server load balancing and application redirection can match or exclude up to 128 content strings. Examples of content strings are as follows:

- "/product," matches URLs that start with /product.
- "product," matches URLs that have the string "product" anywhere in the URL.

You can assign one or more content strings to each real server. When more than one URL string is assigned to a real server, requests matching any string are redirected to that real server. There is also a special string known as "any" that matches all content.

For HTTP traffic, the ADC software also supports avoidance string matching. Using this option, you can define a server to accept any requests regardless of the URL, except requests with a specific string.

**Note:** Once avoidance string matching is enabled, clients cannot access the URL strings that are added to that real server. This means you cannot configure a dedicated server to receive a certain string and, at the same time, have it exclude other URL strings. The avoidance feature is enabled per server, not per string.

For example, the following strings are assigned to a real server:

- **string 1 = cgi**
- **string 2 = NOT cgi/form_A**
- **string 3 = NOT cgi/form_B**

As a result, all cgi scripts are matched except form_A and form_B.
Configuring Avoidance URL String Matching

This configuration example shows you how to configure a server to handle any requests except requests that contain the string "test" or requests that start with "/images" or "/product".

To configure avoidance URL string matching

1. Before you can configure URL string matching, ensure that the router is configured for basic server load balancing.
   For information on how to configure your network for server load balancing, see Server Load Balancing.
2. Add the load-balancing strings (for example test, /images, and /product) to the real server.

```
[edit extensions adc adc-instance demo1]
content-match {
  string test {
    text-search {
      url-string test;
    }
  }
  string images {
    text-search {
      url-string /images;
    }
  }
  string product {
    text-search {
      url-string /images;
    }
  }
}
```

3. Assign the URL strings to the real server and set the avoidance string matching option.

```
[edit extensions adc adc-instance demo1]
real-servers {
  server1 {
    address 1.1.1.1;
    content-strings [test images product];
    avoid-http-strings;
  }
}
```

If you configured a string "any" and enabled the exclusion option, the server will not handle any requests. This has the same effect as disabling the server.
Regular Expression Matching

Regular expressions are used to describe patterns for string matching. They enable you to match the exact string, such as URLs, hostnames, or IP addresses. It is a powerful and effective way to express complex rules for content-string matching. Both content-string HTTP server load balancing and cache redirection can use regular expressions as a resource. Configuring regular expressions can enhance content-based routing in the following areas:

- HTTP header matching
- URL matching

Standard Regular Expression Characters

Table 11 on page 227 shows a list of standard regular expression special characters that are supported in the ADC software.

<table>
<thead>
<tr>
<th>Construction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Matches any string of zero or more characters</td>
</tr>
<tr>
<td>.</td>
<td>Matches any single character</td>
</tr>
<tr>
<td>+</td>
<td>Matches one or more occurrences of the pattern it follows</td>
</tr>
<tr>
<td>?</td>
<td>Matches zero or one occurrences of its followed pattern</td>
</tr>
<tr>
<td>$</td>
<td>Matches the end of a line</td>
</tr>
<tr>
<td>\</td>
<td>Escape the following special character</td>
</tr>
<tr>
<td>[abc]</td>
<td>Matches any of the single character inside the bracket</td>
</tr>
<tr>
<td>[^abc]</td>
<td>Matches any single character except those inside the bracket</td>
</tr>
<tr>
<td>^</td>
<td>Matches the pattern exactly only if it appears at the beginning of a line</td>
</tr>
</tbody>
</table>

Use the following rules to describe patterns for string matching:

- Supports one layer of parentheses.
- Supports only single "$" (match at end of line) which must appear at the end of the string. For example, "abc$*def" is not supported.
- Size of your input string must be 40 characters or less.
- Size of the regular expression structure after compilation cannot exceed 43 bytes for load-balancing strings and 23 bytes for cache redirection. The size of the regular expression after compilation varies, based on regular expression characters used in your input string.
- Use "/" at the beginning of the regular expression. Otherwise a regular expression will have "*" prefixed to it automatically. For example, "html/*\htm" appears as "*html/*/\htm".
- Incorrectly or ambiguously formatted regular expressions are rejected instantly. For example:
  - where a "+" or "?" follows a special character like the "*"
  - A single "+" or "?" sign
  - Unbalanced brackets and parenthesis

Configuring Regular Expressions

The regular expression feature is applicable to both path strings used for URL-based server load-balancing, and expression strings used for URL-based application redirection.
Configure regular expressions at the CLI prompt:

```bash
[edit extensions adc adc-instance demo1]
content-match {
    string <name> {
        text-search {
            url-string <url>;
        }
    }
}
```

As a result, both HTTP server load balancing and application redirection can use regular expression as the resource.

**Note:** The more complex the structure of the string, the longer it will take for the server to load-balance the incoming packets.

### Using Variables in Strings for HTTP

The following variables are supported as part of the text-search value matching:

- **$HOST_URL**—The incoming request from the HOST and URL string.
- **$HOST**—The incoming request from the HOST string.
- **$URL**—The incoming request from the URL string.

These variables can be used as part of the value of any parameter in text-search strings. Variables use the dollar sign ($), and cannot be used in conjunction with regular expressions. When variables are used, the *no-regular-expression* parameter should be set for the string.
Content Precedence Lookup

Content precedence lookup in the ADC software allows you to give precedence to one content match parameter over another and selectively decide which parameter should be analyzed first.

Content precedence lookup allows you to combine up to two content-string load-balancing mechanisms. You can specify which types of content to examine, the order in which they are examined, and a logical operator (and/or) for their evaluation. The following content types can be specified:

- URL server load balancing
- HTTP host
- Cookie
- Browsers (user agent)
- URL hash
- Header hash

Using the above content types with the **and** and **or** operators, the ADC software is configured to refine HTTP-based server load balancing multiple times on a single client HTTP request in order to bind it to an appropriate server. Typically, when you combine two content types with an operator (and/or), URL hash and header hash are used in combination with host, cookie, or browser content types. For example, the following load-balancing types can be configured using content precedence lookup:

- Virtual host and/or URL-based load balancing
Cookie persistence and URL-based load balancing
Cookie load-balancing and/or URL-based load balancing
Cookie persistence and HTTP server load balancing together in the same service
Multiple HTTP server load balancing process types on the same service

**Note:** Cookie persistence can also be combined with the content-string content types. For more information on cookie persistence, see **Persistency**.

For example, content precedence lookup can be used in the following scenarios:

- If the client request is sent without a cookie and if no HTTP server load balancing is configured, then the router binds the request to the real server using normal server load-balancing.
- If the client request is sent without a cookie, but HTTP server load balancing is configured on the router, then the request is bound to real server based on HTTP server load balancing.
- If the client request is sent with a cookie, and a real server associated to the cookie is found in the local session table, then the request will stay bound to that real server.

### Using the or / and Operators

Figure 34 on page 230 shows a network with real servers 1 and 3 configured for URL server load balancing, and real servers 2 and 3 configured for HTTP host server load balancing.

**Figure 34: Using or/and Operators**

If content precedence lookup is configured with the **or** and **and** operators, the request from the client is as follows:

- **HTTP host or URL content-based server selection**
  
  The HTTP host header takes precedence because it is specified first. If there is no host header information and because **or** is the operator, the URL string is examined next.
  
  - If a request from a client contains no host header but has a URL string (such as "/gold"), the request is load-balanced between Server 1 or Server 3.
If a request from a client contains a host header, then the request is load-balanced between Server 2 and Server 3. The URL string is ignored because the HTTP host was specified and matched first.

- **HTTP host and URL content-based server selection**
  
  The HTTP host header takes precedence because it is specified first. Because **and** is the operator, both a host header and URL string are required. If either is not available, the request is dropped.

  - If a request from a client contains a URL string (such as "/gold") but not a host header, it is not served by any real server.
  
  - If a request from a client contains a URL string (such as "/gold") and host header, it is served only by real server 3.

### Assigning Multiple Strings

**Figure 35 on page 232** shows an example of a company providing content for two large customers: Customers A and B. Customer A uses www.a.com as their domain name, and Customer B uses www.b.com.

The company has a limited number of public IP addresses and wishes to assign them on a very conservative basis. As a result, the company implements virtual hosting by advertising a single virtual-server IP address that includes both customers' websites. Additionally, the hosting company assigns only one service (HTTP port 80) to support the virtual server.

The virtual hosting company wishes to maintain the flexibility to allow different types of content to be placed on different servers. To make most efficient use of their server resources, they separate their servers into two groups, using their fastest servers to process dynamic content (such as .cgi files) and their slower servers to process all static content (such as .jpg files).
To configure content precedence lookup for the example, the hosting company groups all real servers into one real-server group even though different servers provide services for different customers and different types of content. In this case, the servers are set up for the purposes shown in Table 12 on page 232.

Table 12: Real-Server Content

<table>
<thead>
<tr>
<th>Server</th>
<th>Customer</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server 1</td>
<td>Customer A</td>
<td>Static .jpg files</td>
</tr>
<tr>
<td>Server 2</td>
<td>Customer A</td>
<td>Static .jpg files</td>
</tr>
<tr>
<td>Server 3</td>
<td>Customer A</td>
<td>Dynamic .cgi files</td>
</tr>
<tr>
<td>Server 4</td>
<td>Customer B</td>
<td>Static .jpg files</td>
</tr>
<tr>
<td>Server 5</td>
<td>Customer B</td>
<td>Dynamic .cgi files</td>
</tr>
</tbody>
</table>

When a client request is received with www.a.com in the host header and .jpg in the URL, the request is load-balanced between Server 1 and Server 2.

To accomplish this configuration, you must assign multiple strings (a host header string and a URL string) for each real server.
String Case Sensitivity

By default, the ADC software supports case-insensitive matching when performing a lookup of string content.

If the following content strings are configured for a real server:
1. default.asp
2. search.asp

Any incoming request containing "GET /Default.asp" would bind to string 1.

String case sensitivity is sometimes required in order to distinguish between requests such as "GET /default.asp," which is considered legal, and requests such as "GET /Default.asp" or "GET /DEFAULT.ASP," which is blocked.

```plaintext
[edit extensions adc adc-instance demo1]
content-match {
  string <string-name> {
    text-search {
      url-string default.asp;
    }
  }
  case-sensitive;
}
```

**Note:** Once case-sensitive string matching is set, all text-search strings are searched using case sensitivity. Case-sensitive matching is set for the adc-instance, not per string.

Configurable HTTP Methods

Requests that use HTTP methods defined in RFC 2616 are evaluated for string matching. These methods are:
- GET
- POST
- HEAD
- CONNECT
- DELETE
- OPTIONS
- PUT
- TRACE
You can set additional HTTP methods to be inspected for string matching using the following configuration:

```
[edit extensions adc adc-instance demo1]
http-strings {
    <string name> {
        methods [DCOPY];
    }
}
```

The list of supported HTTP methods is updated regularly in ADC software as the HTTP protocol evolves.

**Pattern-Based Content-Match**

In addition to setting content-match based on string matching, you can define content-match based on patterns. This can be used for content-based filtering for applications that are not textual. For example, you can configure filters that scan the first IP packet and discard it if one or all of the configured patterns are found. If no match is found, the packets is allowed through.

Pattern matching is constructed in much the same way as any other filter configured to examine Layer 7 content.

**Pattern Criteria**

The ADC software can be configured to examine an IP packet from either the beginning, from a specific offset value (starting point) within the IP packet, and/or from a specified depth (number of characters) into the IP packet. It then performs a matching operation.

**Text-Pattern**

To configure a text-pattern

```
[edit extensions adc adc-instance demo1]
content-match {
    string <string-name> {
        text-pattern {
            string <ascii string, can use regex>;  
            offset <0-1500>;  
            depth <0-1500>;  
        }
    }
}
```

Parameter description:

- **string**—A fixed or regular expression string pattern in ASCII characters.
- **offset**—The byte count from the start of the IP header, from which a search or compare operation is performed. By default the search starts at the beginning (offset 0).
• **depth**—The number of bytes in the IP packet that should be examined from either the beginning of the packet or from the offset value.
  
  If no depth is specified in ASCII matches, the exact pattern is matched from the offset value to the end of the pattern.

### Binary-Pattern

To configure a binary-pattern

```plaintext
[edit extensions adc adc-instance demo1]
content-match {
  string <string-name> {
    binary-pattern {
      value <binary> <and-higher | and-lower>;
      offset <0-1500>;
      depth <0-1500>;
    }
  }
}
```

**Parameter description:**

- **value**—A binary-pattern is a string of hexadecimal characters. For example, to specify the binary pattern 1111 1100 0010 1101, enter FC2D. The value must be an even number of digits.
  
  Optionally, you can specify `and-higher` or `and-lower`:
  
  - **and-higher**—Values that are higher than the specified value are considered a match.
  - **and-lower**—Values that are lower than the specified value are considered a match.

- **offset**—The byte count from the start of the IP header, from which a search or compare operation is performed. By default the search starts at the beginning (offset 0).
  
  For example, if an offset of 12 is specified, the ADC software starts examining the hexadecimal representation of a binary string, from the 13th byte. In the IP packet, the 13th byte starts at the Source IP Address portion of the IP payload.

- **depth**—The number of bytes in the IP packet that are examined from either the beginning of the packet or from the offset value.
  
  For example, if an offset of 12 and a depth of 8 is specified, the search begins at the 13th byte in the IP packet and will match 8 bytes. An offset of 12 and depth of 8 encompasses the Source IP Address and Destination IP Address fields in the IP payload.

  If no depth is specified in ASCII matches, the exact pattern is matched from the offset value to the end of the pattern. A depth must be specified for binary matches that are larger than the pattern length, in bytes.
Appendix A – ADC Software Load Command Set

This appendix provides a full command set for loading the ADC software onto a device.

Command Set for Loading the ADC Software Onto a Device

In order to make it easier for you to load the ADC software onto a device, the following full command set describes in detail the steps found in the Loading the ADC Software onto a Device section.

To load the ADC software using the full command set

1. Use FTP to transfer the package to the device.
2. Allow the package to run on the device.

```bash
user@host> configure
user@host# set system extensions providers radware license-type strategic deployment-scope commercial
```

3. Commit and quit.

```bash
user@host# commit and-quit
```

4. Add the new software to the device.

```bash
user@host> request system software add <package>
```
5. Install the ADC software on the Multiservices-DPC NPU. Repeat as needed for all relevant NPUs.

   The following example is for ms-1/0/0:

   ```
   user@host# set chassis fpc 1 pic 0 adaptive-services service-package
   extension-provider control-cores 1
   user@host# set chassis fpc 1 pic 0 adaptive-services service-package
   extension-provider data-cores 7
   user@host# set chassis fpc 1 pic 0 adaptive-services service-package
   extension-provider object-cache-size 512
   user@host# set chassis fpc 1 pic 0 adaptive-services service-package
   extension-provider wired-process-mem-size 256
   user@host# set chassis fpc 1 pic 0 adaptive-services service-package
   extension-provider data-flow-affinity hash-key layer-3
   user@host# set chassis fpc 1 pic 0 adaptive-services service-package
   extension-provider package adc-ctrl
   user@host# set chassis fpc 1 pic 0 adaptive-services service-package
   extension-provider package adc-data
   ```

6. Configure the commit and op scripts used by the ADC software.

   ```
   user@host# set system scripts commit allow-transients
   user@host# set system scripts commit file radware-conf.slax
   user@host# set system scripts op file radware-op.slax
   ```

7. Configure the device to use ECMP hash on Layer 3 with IPs only.

   ```
   user@host# set forwarding-options hash-key family inet layer-3 destination-address
   user@host# set forwarding-options hash-key family inet layer-3 source-address
   user@host# set forwarding-options enhanced-hash-key services-loadbalancing family inet layer-3-services source-address
   ```

8. Configure the device to capture logs.

   ```
   user@host# set system syslog file device-log any any
   user@host# set system syslog file adc-log any any
   user@host# set system syslog file adc-log match adc
   ```

9. Commit the changes and quit configuration mode.

   ```
   user@host# commit and-quit
   ```

**Note:** After committing the configuration as listed above, the Multiservices-DPC NPUs become active and the ms-x/y/0 interfaces appear in the configuration.
Index

A
ADC software
  load command set 237
allow (filtering) 132
and
  using operators 230
application health checking 187
application ports 44
application redirection 134
  client IP address authentication 157
  example with NAT 154
  games and real-time applications 157
  noncacheable sites 157
  proxies 152, 154
  See cache redirection 152
topologies 153
Web-cache redirection example 150, 158
assigning multiple strings 231

B
backup servers 50, 51
bandwidth (SLB method) 47

C
cache redirection
  browser-based 166
  example 152
  HTTP header-based 165
  layer 7 traffic 158
  RTSP 155, 170
  See application redirection 150
  servers 150, 153
  URL hashing 167
  URL-based 159
CGI-bin scripts 50
command set
  loading the ADC software onto a device 237
configuring
  cache redirection 153
cookie
  different types 104
  expiration timer 108
  header 103
  names 104
  passive mode 106
  permanent 103
  rewrite 106
topologies 153
Web-cache redirection example 150, 158
assigning multiple strings 231

D
delayed binding 59
deny (filtering) 132
direct real server access 57
disabling real servers 45
DNS
  server load balancing 71
  DNS load balancing 73, 74
Domain Name System (DNS)
  filtering 138, 141
  load balancing, layer 7 74
  round robin 36
dport (filtering option) 154

e
expiration timer, insert cookie 108

F
failed server protection, SLB 36
failure types
  server 217
  service 217
fault tolerance
  Server Load Balancing 39
filtering
  configuration example 138
default filter 135, 142
  host header-based redirection 180
  HTTP redirection 172
  HTTP redirection overview 172
  HTTP to HTTPS redirection 182
  MIME redirection 177
  numbering 135
  order of precedence 134
  content intelligent
    cache redirection 158
    server load balancing 113
  content precedence lookup 229
  content-string handling 225
  cookie
    different types 104
    expiration timer 108
    header 103
    names 104
    passive mode 106
    permanent 103
    rewrite 106
topologies 153
Web-cache redirection example 150, 158
assigning multiple strings 231
security example 137
source IP redirection 180
TCP service port based HTTP redirection 175
URL-based redirection 179
firewalls 138
Follow 155
FTP
  Server Load Balancing
    configuring 66, 67, 72
    server load balancing 65
H
hash (SLB method) 46
hashing on any HTTP header 125
health checks 154, 193
  configuration using scripts 209
  format 206
  hostname for HTTP content 187, 188
  HTTPS/SSL 197
  IMAP server 193
  RADIUS server 195
  real server parameters 187
  real servers 45
  script-based 205, 212
  SNMP 190
  verifying scripts 212
  wireless session protocol 198
  WTLS 201
High 219
high availability 219
  connection synchronization 222
  RMS support 220
hostname, for HTTP health checks 187
HTTP
  application health checks 187
  redirection 172
  redirection overview 172
  using variables in strings 228
HTTP header
  hashing 125
HTTP redirection
  IP-based 173
  TCP service port-based 175
  URL-based 179
HTTP to HTTPS
  redirection 182
HTTP URL request 147
HTTPS/SSL health checks 197
I
IMAP server health checks 193
insert cookie mode
  expiration timer 108
Internet Service Provider (ISP), SLB example 38
IP address
  proxies 152, 154
  real server groups 164
  virtual servers 39, 43
L
Layer 4
  cache redirection 153
  server load balancing 21, 41
Layer 7
  cache redirection 158
  server load balancing 113
  string matching 225
least connections (SLB method) 47
limitations
  NAT IP 64
load balancing
  DNS 74
  FTP traffic 65
  layer 7 traffic 113
  types of 65
  WAP traffic 91
load command set 237
load-balancing methods
  real server groups 46
loading the adc software
  command set 237
log
  filtering option 132
  log (filtering option) 138
M
matching TCP flags 143
maximum connections 49, 50
maximum connections limit 49, 50
MIME
  redirection 177
multimedia servers 78
multiple services, configuring 46
N
NAT
  IP limitation 64
Network Address Translation (NAT) 154
NFS server 38
noncacheable sites 157
O
operators
  using or/and 230
or
  using operators 230
overflow servers 50, 51
overview
  HTTP redirection 172
P

persistence
  Client IP-based 102
  cookie-based 102
  multi-response cookie search 112
ports
  for services 44
proxies 152
proxy servers 151

Q

QuickTime Streaming Server 78

R

RADIUS
  health checks 195
RADIUS snooping 92, 95
RADIUS/WAP persistence 95
real server groups
  configuration example 164
real servers 39
  backup/overflow servers 50
  connection timeouts 49
  disable/enable 45
  health checks 187
  maximum connections 49
  weights 48
redirect
  HTTP to HTTPS 182
  MIME type 177
  source IP and host header based 180
  URL-based 179
regular expression
  configuring 227
  standard characters 227
regular expression matching 227
response time (SLB method) 47
rewrite cookie mode 106
RMS support 220
roundrobin
  SLB Real Server metric 47
round-robin (SLB method) 47
routers
  using redirection to reduce Internet congestion 149
  web-cache redirection example 150
RTSP
  cache redirection 155
  server load balancing 77

S

scalability, service 36
script-based health checks 205, 212
searching for cookie 112

security
  filter-based 137
  filtering 132, 137
  firewalls 138
  from viruses 132
  layer 7 deny filter 147
server
  failure types 217
server failure 217
Server Load Balancing
  backup servers 50, 51
  configuration example 38
  DNS 71, 74
  failed server protection 36
  fault tolerance 39
  health checks 187
  maximum connections 49
  methods 46
  overflow servers 51
  overview 37
  real server group 164
  real servers 39
  topology considerations 39
  virtual IP address (VIP) 39
  WAP 91
  weights 48
server load balancing
  DNS 71
  FTP 65
  RTSP 77
  SIP 96
  TFTP 67
server pool 36
service
  failure types 217
service failure 217
service ports 44
shared services 36
SIP
  server load balancing 96
SNMP
  SNMP content health check 190
  SNMP health check 190
  sport (filtering option) 154
  standard regular expression characters 227
  streaming media 78
strings
  assigning multiple 231
syslog
  messages 132

T

TCP 141, 142
  health checking using 45
  HTTP redirection 175
TCP flags 143
telnet 138
tftp
server load balancing 67
timeout for real server connections 49
transparent proxies 152, 154
types
  server failure 217
  service failure 217

U
UDP 141
  server status using 45
URL
  redirecton 179
URL, in HTTP redirection 179

V
variables
  using in strings for HTTP 228
virtual IP address (VIP) 39
virtual servers
  IP address 43
virus attacks, preventing 147
VLANs
  VLAN 1 (default) 153

W
WAP
  WTLS health check 201
WAP Gateway 91
WAP load balancing
  RADIUS snooping 92, 95
  RADIUS/WAP persistence 95
Web hosting 38
weights 48
Well-Known Application Ports, page 171 139, 153
Wireless Application Protocol 91
World Wide Web, client security for browsing 138
WSP health checks 198
WTLS health check 201