



Data Center Fundamentals

Understand Data Center network design and infrastructure architecture, including load balancing, SSL, and security

No Exchange
Media
Books



748122
C46-
482-S

\$6.99

FreeWheel, Ex. 1088
FreeWheel v. AlmondNet
IPR2024-00419

Mauricio Arregoces, CCIE® No. 3285
Maurizio Portolani



Data Center Fundamentals

Mauricio Arregoces, CCIE No. 3285
Maurizio Portolani

Cisco Press

Cisco Press
800 East 96th Street
Indianapolis, IN 46240 USA

Data Center Fundamentals

Mauricio Arregoces

Maurizio Portolani

Copyright © 2004 Cisco Systems, Inc.

Published by:

Cisco Press

800 East 96th Street

Indianapolis, IN 46240 USA

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the publisher, except for the inclusion of brief quotations in a review.

ISBN: 1-58705-023-4

Library of Congress Cataloging-in-Publication Number: 2001086631

Printed in the United States of America 1 2 3 4 5 6 7 8 9 0

First Printing December 2003

Trademark Acknowledgments

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Cisco Press or Cisco Systems, Inc., cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

This book is designed to provide information about Data Center technologies. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied.

The information is provided on an “as is” basis. The authors, Cisco Press, and Cisco Systems, Inc., shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the discs or programs that may accompany it.

The opinions expressed in this book belong to the author and are not necessarily those of Cisco Systems, Inc.

Feedback Information

At Cisco Press, our goal is to create in-depth technical books of the highest quality and value. Each book is crafted with care and precision, undergoing rigorous development that involves the unique expertise of members from the professional technical community.

Readers’ feedback is a natural continuation of this process. If you have any comments regarding how we could improve the quality of this book or otherwise alter it to better suit your needs, you can contact us through e-mail at feedback@ciscopress.com. Please make sure to include the book title and ISBN in your message.

We greatly appreciate your assistance.

Corporate and Government Sales

Cisco Press offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact:

U.S. Corporate and Government Sales 1-800-382-3419 corpsales@pearsontechgroup.com

For sales outside of the U.S. please contact: **International Sales** 1-317-581-3793 international@pearsontechgroup.com

TCP and UDP Connections to a Failed Server

No matter how robust the design of the application is, server failures are still possible for the following reasons:

- Power failure
- Hard disk failures
- NIC failures
- Cable failures
- Software bugs
- Memory leaks
- Disk data corruption
- Configuration errors

Depending on the server failure, the client detects the failure for an established connection as follows:

- **Server process fails**—The server sends a TCP FIN to the client, or the client sends a TCP segment and receives a TCP RST. In case of UDP, the server sends an ICMP port unreachable message.
- **Server machine fails or loses network connectivity**—The TCP connection times out on the client, unless the application has other mechanisms to detect the failure and close the connection.

When a client sends a request to an application for a new connection, there are two possible results, depending on which failure affects the server:

- **Server process failed**—The client sends a TCP SYN, and the server sends a TCP FIN. In case of UDP, the server sends an ICMP port unreachable message.
- **Server machine failed or lost network connectivity**—The client sends TCP SYNs for a number of times (these SYN packets are called TCP SYN retransmissions), depending on the operating system implementation of the TCP stack.

TCP Timeout

When a server is disconnected from the network or powered down, no data flows back to the client, and the connection can stay idle for hours. The TCP stack can keep a connection idle for a configurable amount of time. Because the default is typically two or three hours, however, it is up to the application to age out connections.

As an example, in the case of FTP on Linux, if you use **proftpd** in the configuration file **proftpd.conf**, you can set the parameter **TimeoutIdle** *TimeoutIdle* seconds (in which the default is 600 seconds) to age out connections that have been idle for the configured timeout.

If the application keeps the TCP connection idle for more than 2 hours, the TCP stack sends a *TCP keepalive* to verify that the peer entity (client or server) is still alive. The sender not receiving a response sends a number of keepalives (a typical number is 9) at an interval of approximately every 1 minute and eventually closes the TCP connection.

Although the recommendation is to not change the TCP keepalive defaults, on most OSs, you can change the value of the TCP keepalive. On UNIX-based OSs, this variable can take the name of `tcp_keepalive_interval`, `tcp_keepidle`, and `tcp_keepalive_time`. In Windows, this variable is called `KeepAliveTime`. The socket interface also allows the application to decide whether TCP should implement the keepalive mechanism.

SYN Retransmission

When a client tries to open a connection to a failed server, the TCP stack reacts by retransmitting the SYN for a number of times, depending on the OS. Windows 2000 and XP perform two retransmissions; other OSs perform more. The timeout depends on the number of retransmissions and the interval between them.

The following list shows the behavior of three OSs:

- **Windows**—Uses `TcpMaxConnectRetransmissions`, which is 3 SYNs in Windows NT and 2 SYNs in Windows 2000. This translates into a timeout of 22 seconds in Windows 2000 (with the first retransmission happening in 3 sec) and 45 seconds in Windows NT.
- **Solaris**—Uses `tcp_ip_abort_cinterval`, which is the timeout when TCP performs an active open and, by default, is four minutes. The total number of retransmissions depends on other parameters, such as the retransmission interval.
- **Linux**—Uses `tcp_syn_retries` with a default of 10.

TIP

Understanding the SYN retransmission mechanism is key to using the in-band health-monitoring feature of load-balancing devices. For more details, see Chapter 17.

Status of HTTP Applications

The previous section dealt with the impact of a failed server on TCP and UDP traffic. Another scenario could be an application whose process is alive and capable of accepting new connections, yet is not functioning properly. This section analyzes the behavior of a web server and how the server informs the client about possible errors.

Image refers to data in the form of images. Images typically require special software or hardware to view the information. There are only two subtypes defined in the RFC 2046 specification: JPEG and GIF. Other common image types, some of which are registered and some are not, are BMP (bitmap), PNG (portable network graphic images), PCX (Paintbrush Image), and so on.

Audio refers to data in the form of audio. Audio data typically requires special devices to listen to the data. Audio originally had a single subtype, “basic.” Basic audio is a single audio channel using 8 bits at a sample rate of 8000 Hz. Currently, the list of subtypes includes more complex audio format such as G729 and MPEG.

Video refers to data in the form of video, which is considered moving images. MPEG, JPEG, Digital Video (DV), and QuickTime are some of the registered subtypes.

Application refers to some other data type that is understood by an application. The data could be a raw stream of binary information. Two subtypes were defined by the specification: octet-stream and PostScript. Octet-stream deals with arbitrary binary data, and PostScript refers to the PostScript program. Other application types are PDF, visio, ms-powerpoint and msword.

Multipart refers to multiple parts of discrete data types for a single data entity. RFC 2046 defined four multipart subtypes, and a few others have been added:

- **Mixed**—Generic mix of different media types.
- **Alternative**—Same data represented in multiple formats.
- **Parallel**—Multiple parts intended to be viewed simultaneously.
- **Digest**—Multiple parts in which each part has a default type message (subtype RFC 822).

Message refers to an encapsulated message that might contain other objects. The subtypes include the following:

- **RFC 822**—Encapsulated content in an RFC 822 message
- **partial**—A partial RFC 822 message for fragmented transmissions of entities that are too large to transmit as a single unit
- **external-body**—Specifies large entities by reference to an external data source instead of including the actual body of data

HTTP Overview

The HTTP specification and history are rather rich, but before discussing such topics, it is important to describe the impact of HTTP on the Internet and intranets.

TCP is the dominant protocol on the Internet and intranets. HTTP makes up the majority of Internet traffic and is a significant portion of the traffic on private networks. HTTP is also

the protocol behind the WWW, and its adoption has extended to private networks, where companies are building their client/server infrastructures based on web technology. You can read a technical description of the use and applications of HTTP in Chapter 2, “Server Architecture Overview,” which presents the technical details behind the servers that support HTTP, and Chapter 3, which provides details on application environments that use HTTP.

HTTP was first available in Version 0.9 (HTTP/0.9) as a simple protocol for raw data transfer across the Internet. The first documented version in RFC 1945 for HTTP/1.0, an informational RFC, improved the protocol by allowing messages to be in the format of MIME-like messages, containing meta information about the data transferred and modifiers on the request/response semantics. HTTP/1.0, however, did not address the effects of hierarchical proxies, caching, the need for persistent connections, and virtual hosts. Some incomplete implementations of HTTP/1.0 forced the use of HTTP versions to control the communication capabilities of the hosts attempting to communicate.

The original definition, based on RFC 1945, for HTTP is

“...an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands). A feature of HTTP is the typing of data representation, allowing systems to be built independently of the data being transferred.”

A new version of HTTP, HTTP/1.1, was released in RFC 2068, which was further revised by RFC 2616, a standards-track RFC. HTTP/1.1 specifies more stringent requirements than HTTP/1.0 to ensure a consistent implementation of its features. A more pragmatic definition of HTTP, introduced by RFC 2068 and maintained in RFC 2616, describes it as “an application-level protocol for distributed, collaborative, hypermedia information systems.”

HTTP has been used in the WWW since 1990 (as part of the WWW global information initiative). HTTP has become the protocol of communication between web browsers and the servers. The use of HTTP has gone beyond the Internet, and it is heavily used on intranets and extranets as well. HTTP is so prevalent as a web communication protocol that most software vendors are developing their applications to utilize web technology, as pointed out in Chapter 1, “Overview of Data Centers.”

HTTP Operation

HTTP is a request/response protocol that uses TCP (port 80) as the transport layer.

NOTE

TCP port 80 for HTTP is the default port in HTTP server implementations, but it is changeable to fit specific needs.

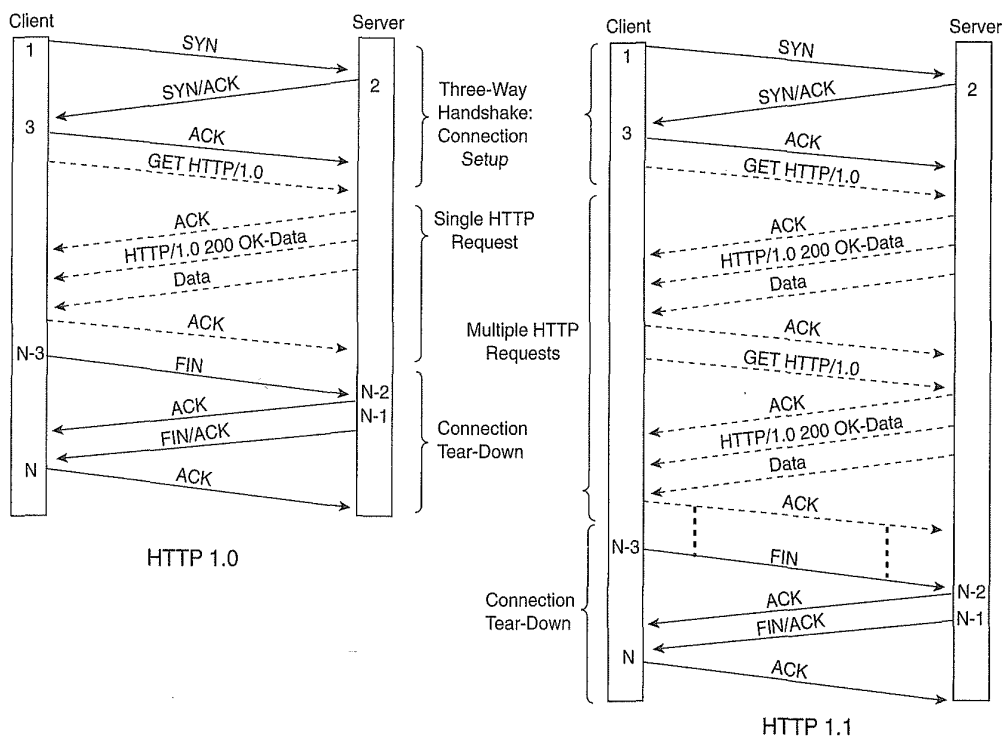
The response message includes the general header fields and the response header fields. The response header fields are “server” and “accept-ranges,” which are intermixed with the general header fields.

HTTP/1.0 and HTTP/1.1 might have different behavior during HTTP connections. HTTP/1.1 introduced two concepts: persistent connections and pipelining.

Persistent Connections and Pipelining

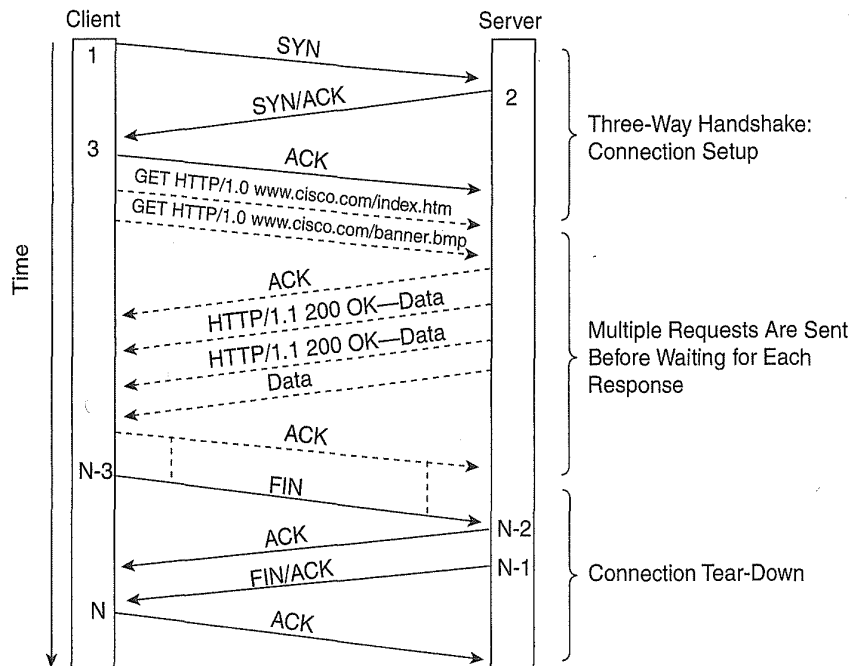
Persistent HTTP connections allow a TCP connection to remain open for more than one HTTP request. The behavior prior to persistent connections allowed only a single HTTP request, as depicted in Figure 8-6. In contrast, an HTTP/1.1 persistent connection allows multiple HTTP requests and their responses as part of the same TCP connection. This behavior reduces unnecessary overhead on the network and server because a single TCP connection serves many HTTP requests instead of each TCP connection serving one TCP request. Figure 8-9 presents the contrast between an HTTP/1.0 and an HTTP/1.1 persistent connection.

Figure 8-9 Persistent Connections



As depicted in Figure 8-9, an HTTP/1.1 connection requires a single TCP connection for multiple HTTP requests, whereas HTTP/1.0 requires a TCP connection per HTTP request. Note that the HTTP requests in the HTTP/1.1 connection are sequential. *Pipelining* allows multiple HTTP requests to be generated without waiting for each response, which implies multiple HTTP requests can be processed concurrently instead of sequentially. Figure 8-10 presents an HTTP/1.1 connection using pipelining.

Figure 8-10 Connection Pipelining



As shown in Figure 8-10, multiple HTTP requests are sent over single TCP connection, and the subsequent requests do not have to wait for the previous ones to complete.

Persistent connections offer a few advantages over nonpersistent connections:

- Less traffic is processed by the network (routers, switches, server load balancers, caches, firewalls, proxies, etc.), the clients, and the servers, which decreases CPU utilization and memory consumption.
- Multiple HTTP requests and responses can be supported on a single TCP connection.
- Clients can make multiple requests without waiting for each response using pipelining.
- Network traffic is reduced by reducing the TCP connection overhead. See the section “HTTP Performance” for more information.

- End-to-end delay on pipelined requests is reduced by avoiding further TCP handshakes, their associated round-trip delays, and the sequential nature of HTTP requests along with their associated round-trip delays.

The persistent-connection behavior is the default in HTTP/1.1. HTTP/1.0 also supports persistent connections, but it is not the default behavior. Persistent connections are controlled by the Connection header, which can have various values: Keepalive, Open, and Close.

Pipelined connections inherit the behavior and characteristics of persistent connections. Additionally, servers are expected to respond to the multiple requests in the same order they were received.

NOTE

To date, there are few pipeline implementations, and most load balancers are not currently “pipelining”-aware, which would cause the connection to break. Persistent connections, on the other hand, are fairly common, and load balancers are capable of handling them. Refer to Chapter 16, “Load-Balancing Modes and Predictors,” for more information.

HTTP Performance

This section discusses HTTP performance from two viewpoints:

- By presenting the advantages of using HTTP/1.1 over HTTP/1.0
- By using HTTP compression between the client and the server

Performance of HTTP/1.1 Versus HTTP/1.0

The obvious advantages of using HTTP/1.1 over HTTP/1.0 are based on persistent connections and pipelining.

Persistent connections provide a direct savings on the number of packets and bytes that the intermediate devices and the client and server have to process. These savings translate to a lower byte and packet count when downloading the same page with HTTP/1.1 versus HTTP/1.0 and lower CPU utilization on the client and server resulting from a lower number of TCP connections that must be established.

The W3C presented the results of a research project on the effects of persistent connections, pipelining, and link compression on the client and server implementations. The following is a summary of the report:

- The client used was libwww robot.
- The servers used were W3C Jigsaw and Apache.
- Servers supported HTTP/1.0 and HTTP/1.1 with persistent connections, HTTP/1.1 with pipelined requests, and HTTP/1.1 with pipelined requests and deflate data compression.