LECTURE NOTES IN ELECTRICAL ENGINEERING 9

Rajkumar Buyya
Mukaddim Pathan
Athena Vakali *Editors*

Content Delivery Networks

Springer

Lecture Notes Electrical Engineering

Volume 9

Rajkumar Buyya · Mukaddim Pathan · Athena Vakali (Eds.)

Content Delivery Networks

Springer

Editors

Rajkumar Buyya
University of Melbourne
Dept. Computer Science&
Software Engineering
111 Barry Street
Carlton VIC 3053
Australia
raj@csse.unimelb.edu.au

Mukaddim Pathan
University of Melbourne
Dept. Computer Science &
Software Engineering
111 Barry Street
Carlton VIC 3053
Australia
apathan@csse.unimelb.edu.au

Athena Vakali
Aristotle University of
Thessaloniki
Dept. Informatics
541 24 Thessaloniki
Greece
avakali@csd.auth.gr

ISBN: 978-3-540-77886-8 e-ISBN: 978-3-540-77887-5

Library of Congress Control Number: 2008930075

Cover design: eStudio Calamar S.L.

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Contents

Part I CDN Fundamentals

1 Content Delivery Networks: State of the Art, Insights, and Imperatives 3
Mukaddim Pathan, Rajkumar Buyya and Athena Vakali

2 A Taxonomy of CDNs 33
Mukaddim Pathan and Rajkumar Buyya

3 Dynamic, Scalable, and Efficient Content Replication Techniques . . . 79
Yan Chen

4 Content Delivery and Management 105
Claudia Canali, Valeria Cardellini, Michele Colajanni and Riccardo Lancellotti

5 Caching Techniques on CDN Simulated Frameworks 127
Konstantinos Stamos, George Pallis and Athena Vakali

6 Request Redirection for Dynamic Content 155
Supranamaya Ranjan

Part II CDN Modeling and Performance

7 Economics-Informed Design of CDNs 183
Nicolas Christin, John Chuang and Jens Grossklags

8 CDN Pricing 211
Kartik Hosanagar

9 Mathematical Models for Resource Management and Allocation in CDNs 225
Tolga Bektaş and Iradj Ouveysi

10 Performance and Availability Benefits of Global Overlay Routing . . . 251
Hariharan S. Rahul, Mangesh Kasbekar, Ramesh K. Sitaraman, and Arthur W. Berger

Part III Advanced CDN Platforms and Applications

11 Dynamic CDN Against Flash Crowds 275
Norihiko Yoshida

12 Collaborative Media Streaming Services Based on CDNs 297
Giancarlo Fortino, Carlo Mastroianni, and Wilma Russo

13 CDN for Live and On-Demand Video Services over IP 317
Mirosław Czyrnek, Ewa Kuśmierek, Cezary Mazurek, Maciej Stroiński, and Jan Węglarz

14 Information Dissemination in Mobile CDNs 343
Nicholas Loulloudes, George Pallis, and Marios D. Dikaiakos

15 Infrastructures for Community Networks 367
Thomas Plagemann, Roberto Canonico, Jordi Domingo-Pascual, Carmen Guerrero, and Andreas Mauthe

16 Internetworking of CDNs 389
Mukaddim Pathan, Rajkumar Buyya, and James Broberg

Index 415

Part I
CDN Fundamentals

Chapter 1
Content Delivery Networks: State of the Art, Insights, and Imperatives

Mukaddim Pathan, Rajkumar Buyya and Athena Vakali

1.1 Introduction

Over the last decades, users have witnessed the growth and maturity of the Internet which has caused enormous growth in network traffic, driven by the rapid acceptance of broadband access, the increases in systems complexity, and the content richness. The over-evolving nature of the Internet brings new challenges in managing and delivering content to users, since for example, popular Web services often suffer congestion and bottlenecks due to the large demands posed on their services. Such a sudden spike in Web content requests (e.g. the one occurred during the 9/11 incident in USA) is often termed as flash crowds [14] or SlashDot [11] effects. It may cause heavy workload on particular Web server(s), and as a result a "hotspot" [14] can be generated. Coping with such unexpected demand causes significant strain on a Web server and eventually the Web servers are totally overwhelmed with the sudden increase in traffic, and the Web site holding the content becomes temporarily unavailable.

A Content Delivery Network (CDN) [47, 51, 54, 61, 63] is a collaborative collection of network elements spanning the Internet, where content is replicated over several mirrored Web servers in order to perform transparent and effective delivery of content to the end users. Collaboration among distributed CDN components can occur over nodes in both homogeneous and heterogeneous environments. CDNs have evolved to overcome the inherent limitations of the Internet in terms of user perceived Quality of Service (QoS) when accessing Web content. They provide services that improve network performance by maximizing bandwidth, improving

Mukaddim Pathan
GRIDS Lab, Department of CSSE, The University of Melbourne, Australia,
e-mail: apathan@csse.unimelb.edu.au

Rajkumar Buyya
GRIDS Lab, Department of CSSE, The University of Melbourne, Australia,
e-mail: raj@csse.unimelb.edu.au

Athena Vakali
Department of Informatics, Aristotle University of Thessaloniki, Greece,
e-mail: avakali@csd.auth.gr

R. Buyya et al. (eds.), *Content Delivery Networks*,

accessibility, and maintaining correctness through content replication. The typical functionalities of a CDN include:

- *Request redirection and content delivery services,* to direct a request to the closest suitable CDN cache server using mechanisms to bypass congestion, thus overcoming flash crowds [14] or SlashDot [11] effects.
- *Content outsourcing and distribution services,* to replicate and/or cache content from the origin server to distributed Web servers.
- *Content negotiation services,* to meet specific needs of each individual user (or group of users).
- *Management services,* to manage the network components, to handle accounting, and to monitor and report on content usage.

The major application domains of CDNs are public content networking services, enterprise content networks, and edge services. As CDNs being a thriving research field, advances, solutions, and new capabilities are being introduced constantly. Therefore, in this chapter, we capture a "snapshot" of the state of the art at the time of writing this book. However, it can be expected that the core information and principles presented in this chapter will remain relevant and useful for the readers.

The remainder of this chapter is structured as follows: we start with providing an overview of CDNs. Next we describe the background highlighting the evolution of CDNs and identify uniqueness of CDNs from other related distributed computing paradigms. In Sect. 1.4 we provide insights for CDNs. The state of the art in CDN landscape is presented in Sect. 1.5. Our visions about future technological evolutions in CDNs domain follows next, along with a research roadmap in Sect. 1.7 by exploring future research directions. Finally, Sect. 1.8 concludes the chapter.

1.2 Overview

Figure 1.1 shows the model of a CDN where the replicated Web server clusters spanning the globe are located at the edge of the network to which end users are connected. A CDN distributes content to a set of Web servers, scattered over the globe, for delivering content to end users in a reliable and timely manner. The content is replicated either on-demand when users request for it, or it can be replicated beforehand, by pushing the content to the distributed Web servers. A user is served with the content from the nearby replicated Web server. Thus, the user ends up unknowingly communicating with a replicated CDN server close to it and retrieves files from that server.

1.2.1 Terminologies

In the context of CDNs, *content delivery* describes an action of servicing content based on end user requests. *Content* refers to any digital data resources and

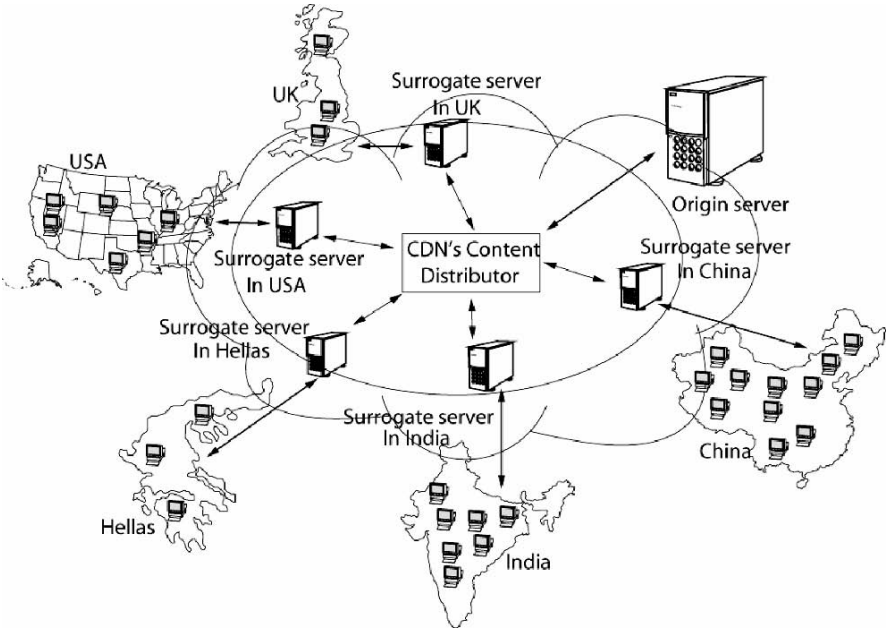

Fig. 1.1 Model of a CDN

it consists of two main parts: the *encoded media* and *metadata* [53]. The encoded media includes static, dynamic, and continuous media data (e.g. audio, video, documents, images and Web pages). Metadata is the content description that allows identification, discovery, and management of multimedia data, and facilitates its interpretation. Content can be pre-recorded or retrieved from live sources; it can be persistent or transient data within the system [53]. CDNs can be seen as a new virtual overlay to the Open Systems Interconnection (OSI) network reference model [32]. This layer provides overlay network services relying on application layer protocols such as Hyper Text Transfer Protocol (HTTP) or Real Time Streaming Protocol (RTSP) for transport [26].

The three main entities in a CDN system are the following: *content provider*, *CDN provider*, and *end users*. A *content provider* or *customer* is one who delegates the Uniform Resource Locator (URL) name space of the Web objects to be distributed. The *origin server* of the content provider holds those objects. A *CDN provider* is a proprietary organization or company that provides infrastructure facilities to content providers in order to deliver content in a timely and reliable manner. *End users* or *clients* are the entities who access content from the content provider's Web site.

CDN providers use *caching* and/or *replica servers* located in different geographical locations to replicate content. CDN cache servers are also called *edge servers* or *surrogates*. The edge servers of a CDN are called *Web cluster* as a whole. CDNs distribute content to the edge servers in such a way that all of them share the same

content and URL. Client requests are redirected to the nearby optimal edge server and it delivers requested content to the end users. Thus, transparency for users is achieved. Additionally, edge servers send accounting information for the delivered content to the accounting system of the CDN for traffic reporting and billing purposes.

1.2.2 CDN Components

Figure 1.2 shows the general architecture of a CDN system which involves four main components:

- The content-delivery component which consists of the origin server and a set of replica servers that deliver copies of content to the end users;
- The request-routing component which is responsible for directing client requests to appropriate edge servers and for interacting with the distribution component to keep an up-to-date view of the content stored in the CDN caches;
- The distribution component which moves content from the origin server to the CDN edge servers and ensures consistency of content in the caches; and
- The accounting component which maintains logs of client accesses and records the usage of the CDN servers. This information is used for traffic reporting

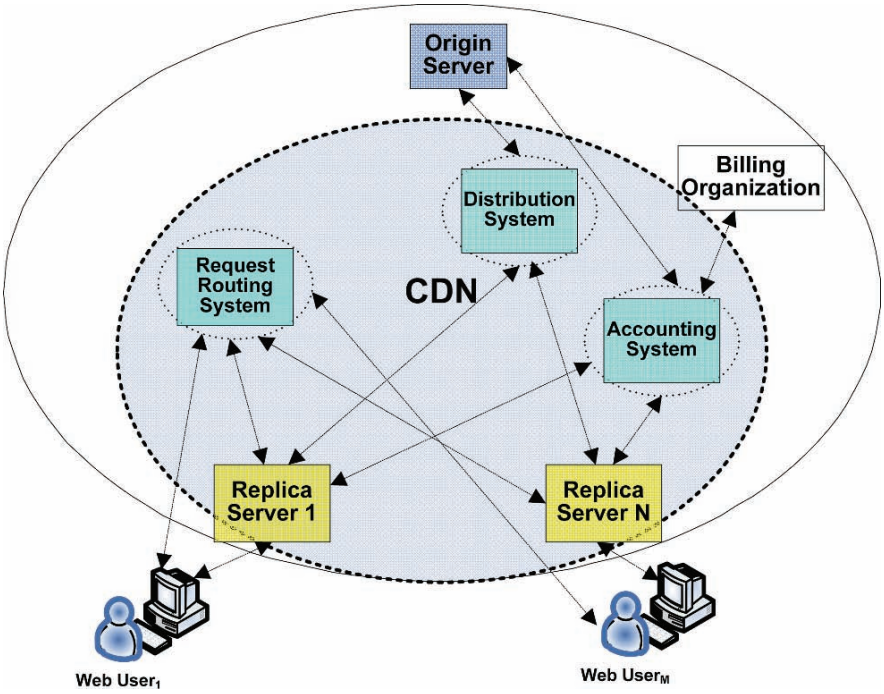

Fig. 1.2 Architectural components of a CDN

and usage-based billing by the content provider itself or by a third-party billing organization.

A CDN focuses on building its network infrastructure to provide the following services and functionalities: storage and management of content; distribution of content among edge servers; cache management; delivery of static, dynamic, and streaming content; backup and disaster recovery solutions; and monitoring, performance measurement, and reporting.

A content provider (i.e. customer) can sign up with a CDN provider for service and have its content placed on the cache servers. In practice, CDNs typically host third-party content including static content (e.g. static HTML pages, images, documents, software patches), streaming media (e.g. audio, real time video), User Generated Videos (UGV), and varying content services (e.g. directory service, e-commerce service, file transfer service). The sources of content include large enterprises, Web service providers, media companies, and news broadcasters. Typical customers of a CDN are media and Internet advertisement companies, data centers, Internet Service Providers (ISPs), online music retailers, mobile operators, consumer electronics manufacturers, and other carrier companies. Each of these customers wants to publish and deliver their content to the end users on the Internet in a reliable and timely manner. End users can interact with the CDN by specifying the content/service request through cell phone, smart phone/PDA, laptop and desktop. Figure 1.3 depicts the different content/services served by a CDN provider to end users.

CDN providers charge their customers according to the content delivered (i.e. traffic) to the end users by their edge servers. CDNs support an accounting mechanism that collects and tracks client usage information related to request-routing, distribution, and delivery [26]. This mechanism gathers information in real time

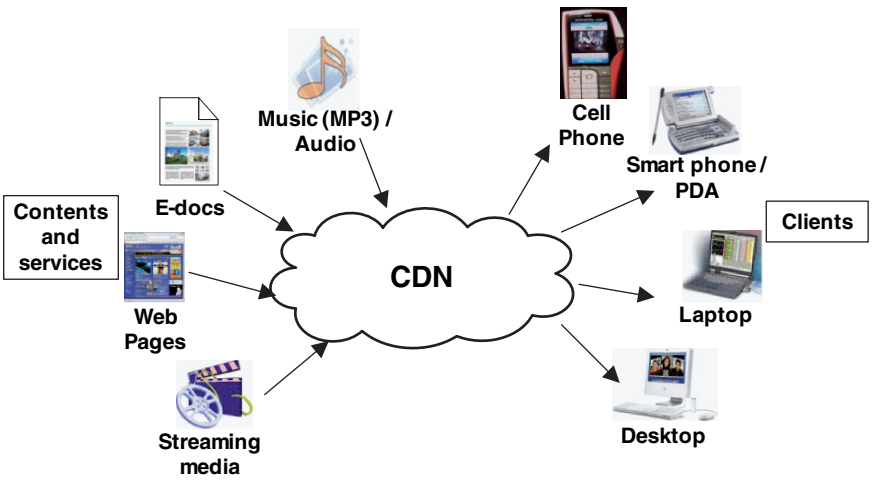

Fig. 1.3 Content/services provided by a CDN

and collects it for each CDN component. This information can be used in CDNs for accounting, billing, and maintenance purposes. The average cost of charging of CDN services is quite high [35], often out of reach for many small to medium enterprises (SME) or not-for-profit organizations. The most influencing factors [47] affecting the price of CDN services include:

- Bandwidth usage which is measured by the content provider to charge (per Mbps) customers typically on a monthly basis;
- Variation of traffic distribution which characterizes pricing under different situations of congestion and bursty traffic;
- Size of the content replicated over edge servers which is a critical criterion for posing charges (e.g. price per GB) on customer audiences;
- Number of edge servers which capture the ability of a CDN provider to offer content at charges that will not overcome the typical caching scenarios; and
- Reliability and stability of the whole system and security issues of outsourcing content delivery also inhibit a cost of sharing confidential data which varies over different content providers on the basis of the type of the protected content.

1.3 Background and Related Systems

Content providers view the Web as a vehicle to bring rich content to their users since decreases in services quality, along with high access delays (mainly caused by long download times) leaves users in frustration. Companies earn significant financial incentives from Web-based e-business and they are concerned to improve the service quality experienced by the users while accessing their Web sites. As such, the past few years have seen an evolution of technologies that aim to improve content delivery and service provisioning over the Web. When used together, the infrastructures supporting these technologies form a new type of network, which is often referred to as "content network" [26].

1.3.1 The Evolution of CDNs

Several content networks attempt to address the performance problem by using different mechanisms to improve QoS:

- An initial approach is to modify the traditional Web architecture by improving the Web server hardware adding a high-speed processor, more memory and disk space, or maybe even a multi-processor system. This approach is not flexible, since small enhancements are not possible and at some point, the complete server system might have to be replaced [31].
- Caching proxy deployment by an ISP can be beneficial for the narrow bandwidth users accessing the Internet, since to improve performance and reduce bandwidth

utilization, caching proxies are deployed close to the users. Caching proxies may also be equipped with technologies to detect a server failure and maximize efficient use of caching proxy resources. Users often configure their browsers to send their Web request through these caches rather than sending directly to origin servers. When this configuration is properly done, the user's entire browsing session goes through a specific caching proxy. Thus, the caches contain most popular content viewed by all the users of the caching proxies.

- A provider may also deploy different levels of local, regional, international caches at geographically distributed locations. Such arrangement is referred to as *hierarchical caching*. This may provide additional performance improvements and bandwidth savings [17]. The establishment of server farms is a more scalable solution which has been in widespread use for several years. A server farm is comprised multiple Web servers, each of them sharing the burden of answering requests for the same Web site [31]. It also makes use of a Layer 4-7 switch (intelligent switching based on information such as URL requested, content type, and username, which can be found in layers 4-7 of the OSI stack of the request packet), Web switch or content switch that examines content requests and dispatches them among the group of servers. A server farm can also be constructed with surrogates instead of a switch [24]. This approach is more flexible and shows better scalability. Moreover, it provides the inherent benefit of fault tolerance. Deployment and growth of server farms progresses with the upgrade of network links that connects the Web sites to the Internet.
- Although server farms and hierarchical caching through caching proxies are useful techniques to address the Web performance problem, they have limitations. In the first case, since servers are deployed near the origin server, they do little to improve the network performance due to network congestion. Caching proxies may be beneficial in this case. But they cache objects based on client demands. This may force the content providers with a popular content source to invest in large server farms, load balancing, and high bandwidth connections to keep up with the demand. To address these limitations, another type of content network has been deployed in late 1990s. This is termed as *Content Distribution Network* or *Content Delivery Network*, which is a system of computers networked together across the Internet to cooperate transparently for delivering content to end users.

With the introduction of CDN, content providers started putting their Web sites on a CDN. Soon they realized its usefulness through receiving increased reliability and scalability without the need to maintain expensive infrastructure. Hence, several initiatives kicked off for developing infrastructure for CDNs. As a consequence, Akamai Technologies [1, 27] evolved out of an MIT research effort aimed at solving the flash crowd problem and scientists developed a set of breakthrough algorithms for intelligently routing and replicating content over a large network of distributed servers spanning the globe. Within a couple of years, several companies became specialists in providing fast and reliable delivery of content, and CDNs became a huge market for generating large revenues. The flash crowd events [14, 34] like the 9/11 incident in USA [10], resulted in serious caching problems for some sites. This influenced the providers to invest more in CDN infrastructure development, since

CDNs provide desired level of protection to Web sites against flash crowds. First generation CDNs mostly focused on static or Dynamic Web documents [36, 61]. On the other hand, for second generation of CDNs the focus has shifted to Video-on-Demand (VoD), news on-demand, audio and video streaming with high user interactivity. The CDNs of this generation may also be dedicated to deliver content to mobile devices. However, most of the research efforts on this type of CDNs are still in research phase and have not yet exclusively reached the market. We anticipate that the third generation CDNs would be community-based CDNs, i.e. it would be mainly driven by the common "people" or the average end users. More information on such community-based CDNs can be found in Chap. 15 of this book. Figure 1.4 shows the evolutions of CDNs over time with a prediction of their evolution in the upcoming years.

With the booming of the CDN business, several standardization activities also emerged since vendors started organizing themselves. The Internet Engineering Task Force (IETF) as an official body has taken several initiatives through releasing Request For Comments (RFCs) [15, 16, 24, 26] in relation to many research initiatives in this domain. Other than IETF, several other organizations such as Broadband Services Forum (BSF) [3], ICAP forum [6], Internet Streaming Media Alliance [7] have taken initiatives to develop standards for delivering broadband content, streaming rich media content – video, audio, and associated data – over the Internet. In the same breath, by 2002, large-scale ISPs started building their own CDN functionality, providing customized services.

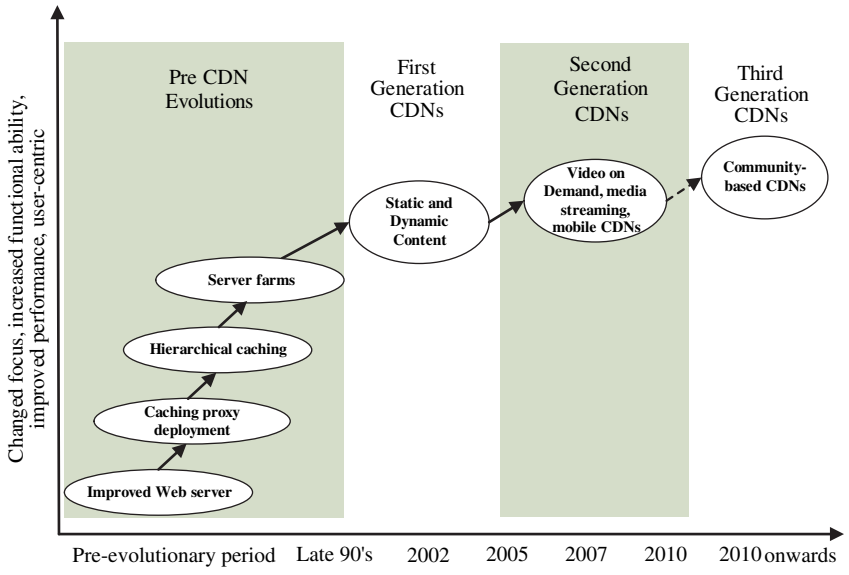

Fig. 1.4 CDN evolutions

manage heavy traffic (i.e. flash crowds) caused by the demand for highly popular files. This is in contrast to CDNs where the main goal lies in respecting client's performance requirements rather than efficiently sharing file/content among peers. Moreover, CDNs differ from the P2P networks because the number of nodes joining and leaving the network per unit time is negligible in CDNs, whereas the rate is important in P2P networks.

1.4 Insights for CDNs

From the above discussion it is clear that a CDN is essentially aimed at content providers or customers who want to ensure QoS to the end users when accessing their Web content. The analysis of present day CDNs reveals that, at the minimum, a CDN focuses on the following business goals: scalability, security, reliability, responsiveness and performance.

1.4.1 Scalability

Scalability refers to the ability of the system to expand in order to handle new and large amounts of data, users, and transactions without any significant decline in performance. To expand to a global scale, CDN providers need to invest time and costs in provisioning additional network connections and infrastructures. It includes provisioning resources dynamically to address flash crowds and varying traffic. A CDN should act as a shock absorber for traffic by automatically providing capacity-on-demand to meet the requirements of flash crowds. This capability allows a CDN to avoid costly over-provisioning of resources and to provide high performance to every user.

Within the structure of present day CDN business model, content providers pay the CDN providers to maximize the impact of their content. However, current trends reveal that the type of applications that will be supported by CDNs in future, will transform the current business model [53]. In future, the content providers as well as the end users will also pay to receive high quality content. In this context, scalability will be an issue to deliver high quality content, maintaining low operational costs.

1.4.2 Security

One of the major concerns of a CDN is to provide potential security solutions for confidential and high-value content [19]. Security is the protection of content against unauthorized access and modification. Without proper security control, a CDN platform is subject to cyber fraud, Distributed Denial-of-Service (DDoS) attacks,

viruses, and other unwanted intrusions that can cripple business. A CDN aims at meeting the stringent requirements of physical, network, software, data, and procedural security. Once the security requirements are met, a CDN can eliminate the need for costly hardware and dedicated component to protect content and transactions. In accordance to the security issues, a CDN provider combat against any other potential risk concerns including DDoS attacks or other malicious activity that may interrupt business.

1.4.3 Reliability, Responsiveness, and Performance

Reliability refers to when a service is available and what are the bounds on service outages that may be expected. A CDN provider can improve client access to specialized content through delivering it from multiple locations. For this a fault-tolerant network with appropriate load balancing mechanisms is to be implemented [42]. Responsiveness implies, while in the face of possible outages, how soon a service would start performing the normal course of operation. Performance of a CDN is typically characterized by the response time (i.e. latency) perceived by end users. Slow response time is the single greatest contributor to customers' abandoning Web sites and processes. The reliability and performance of a CDN is affected by the distributed content location and routing mechanism, as well as by data replication and caching strategies. Hence, a CDN employs caching and streaming to enhance performance especially for delivery of media content [57]. A CDN hosting a Web site also focuses on providing fast and reliable service since it reinforces the message that the company is reliable and customer-focused.

1.5 Existing CDNs: State of the Art

In this section, we provide the state of art in current CDN landscape. We also describe the different services and technologies of existing CDNs. First, we provide a brief description on commercial CDNs (e.g. Akamai, EdgeStream, Limelight Networks, and Mirror Image) which exist in the content distribution space. Then we present a snapshot on academic CDNs (e.g. CoDeeN, Coral, and Globule) which gives a picture of what the CDN technologies are at this moment.

1.5.1 Commercial CDNs

Most or all of the operational CDNs are developed by commercial companies which are subject to consolidation over time due to acquisition and/or mergers. Hence, in the section, we focus on studying only those commercial CDNs that have been

Table 1.2 Summary of the existing commercial CDNs

CDN Name	Service Type	Coverage	Products/Solutions (If any)
Akamai www.akamai.com	Provides CDN service, including streaming	Covers 85% of the market. 25,000 servers in 900 networks in 69 countries. It handles 20% of total Internet traffic today	Edge Platform for handling static as well as dynamic content, Edge Control for managing applications, and Network Operations Control Center (NOCC)
EdgeStream www.edgestream.com	Provides disrupted video streaming applications over the public Internet	Provides video streaming over consumer cable or ADSL modem connections around the globe, even over paths that have 20 router hops between server and end user	EdgeStream video on-demand and IPTV Streaming software for video streaming
Limelight Networks www.limelightnetworks.com	Provides distributed on-demand and live delivery of video, music, games and download	Edge servers located in 72 locations around the world	Limelight ContentEdge for distributed content delivery via HTTP, Limelight MediaEdge Streaming for distributed video and music delivery via streaming, and Limelight Custom CDN for custom distributed delivery solutions
Mirror Image www.mirror-image.com	Provides content delivery, streaming media, Web computing and reporting services	Edge servers located in 22 countries	Global Content Caching, Extensible Rules Engine (XRE), Video On-Demand, and Live Webcasting

in stable operation for a significant period of time. Table 1.2 shows a list of four commercial CDNs and presents a brief summary of each of them. An updated listing of most of the existing commercial CDNs can be found in the research directories of Davison [25] and Pathan [49].

The proprietary nature of commercial CDNs makes it difficult to reveal detailed information about the technical and business strategies used by them. However, in the presented state-of-the-art survey of commercial CDNs, we provide information to significant details. In this context, it is worth mentioning that many CDN-specific information such as fees charged by CDNs, existing customers of CDNs are ignored since they are highly likely to change quickly over time. Therefore, the information provided in this section is expected to be stable and up-to-date. However, for readers' understanding on how a CDN charges its customers (i.e. CDN pricing strategies); we refer to Chap. 8 of the book, which outlines the pricing policies used for CDN services.

1.5.1.1 Akamai

Akamai technologies [1, 27] was founded in 1998 at Massachusetts, USA. It evolved out of an MIT research effort aimed at solving the flash crowd problem. Akamai is the market leader in providing content delivery services. It owns more than 25,000 servers over 900 networks in 69 countries [1]. Akamai's approach is based on the observation that serving Web content from a single location can present serious problems for site scalability, reliability and performance. Hence, a system is devised to serve requests from a variable number of cache servers at the network edge. Akamai servers deliver static (e.g. HTML pages, embedded images, executables, and PDF documents), dynamic content (e.g. animations, scripts, and DHTML), and streaming audio and video.

Akamai's infrastructure handles flash crowds by allocating more servers to sites experiencing high load, while serving all clients from nearby servers. The system directs client requests to the nearest available server likely to have the requested content. Akamai provides automatic network control through the mapping technique (i.e. the direction of request to content servers), which uses a dynamic, fault-tolerant DNS system. The mapping system resolves a hostname based on the service requested, user location, and network status. It also uses DNS for network load-balancing. Akamai name servers resolve hostnames to IP addresses by mapping requests to a server. Akamai agents communicate with certain border routers as peers; the mapping system uses BGP (Border Gateway Protocol) [56] information to determine network topology. The mapping system in Akamai combines the network topology information with live network statistics – such as traceroute data [39] – to provide a detailed, dynamic view of network structure, and quality measures for different mappings.

Akamai's DNS-based load balancing system continuously monitors the state of services and their servers and networks. To monitor the entire system's health end-to-end, Akamai uses agents that simulate the end user behavior by downloading Web objects and measuring their failure rates and download times. Akamai uses this information to monitor overall system performance and to automatically detect and suspend problematic data centers or servers. Each of the content servers frequently reports its load to a monitoring application, which aggregates and publishes

load reports to the local DNS server. That DNS server then determines which IP addresses (two or more) to return when resolving DNS names. If a certain server's load exceeds a certain threshold, the DNS server simultaneously assigns some of the server's allocated content to additional servers. If the server's load exceeds another threshold, the server's IP address is no longer available to clients. The server can thus shed a fraction of its load when it experiences moderate to high load. The monitoring system in Akamai also transmits data center load to the top-level DNS resolver to direct traffic away from overloaded data centers. In addition to load balancing, Akamai's monitoring system provides centralized reporting on content service for each customer and content server. This information is useful for network operational and diagnostic purposes.

Akamai delivers static and dynamic content over HTTP and HTTPS. Akamai content servers apply lifetime and other features (e.g. ability to serve secure content over HTTPS protocol, support alternate content, transfer encodings, and handle cookies) to the static content based on its type. Based on these attributes the edge server ensures the consistency of the content. On the other hand, Akamai handle dynamic content on the edge servers with the use of Edge Side Includes (ESI) [4] technology. The use of ESI enables the content providers to break their dynamic content into fragments with independent cacheability properties. These fragments can be maintained as separate objects in Akamai's edge servers and are dynamically assembled to a dynamic Web page in response to the end user requests.

Akamai supports Microsoft Windows Media, Real, and Apple's QuickTime format for delivering streaming services (live and on-demand media). A live stream is captured and encoded by the content provider and sent to the entry point server of a set of Akamai edge servers, which in turn serve content to the end users. In order to avoid all single points of failure, backups are maintained for the entry point server. Moreover, the entry point server sends data on multiple redundant paths to the edge servers through using information dispersal techniques.

More information on Akamai and its overlay routing, including its performance, availability benefits, different uses in live streaming, application, and IP acceleration can be found in Chap. 10 of this book.

1.5.1.2 EdgeStream

EdgeStream [23] was founded in 2000 at California, USA. It is a provider of video streaming applications over the public Internet. It provides video on-demand and IPTV streaming software to enable transportation of high bit rate video over Internet. It uses HTTP streaming for content delivery. EdgeStream supports different compression formats for delivering content. It has developed Continuous Route Optimization Software (CROS), Internet Congestion Tunnel Through (ICTT) and Real Time Performance Monitoring Service (RPMS) technologies, which together assist to address the latency, packet loss, and congestion bottlenecks. Embedded applications in Consumer Electronics Devices, wireless handheld devices, IP set top boxes, and advanced digital TV's can use the EdgeStream software for video streaming.

EdgeStream platform is made of client and server software modules. The server software consists of Content Management and Online Reporting (CMOR) Server Software Module, EdgeStream Control Server Software Module, EdgeStream Database System Module, and EdgeStream Streaming Server Module. All server modules may be combined to run on a single server, or run separately. CMOR module manages accounts, content, and all other servers in the system. It also generates Web-based real time reports for viewing statistics and transactions from a SQL database. The control module provides necessary authority to obtain the content location information along with streaming delivery management and logging functions. The database module maintains logs for accounting and billing purpose. It uses the Microsoft SQL 2000 Standard or Enterprise server software. The streaming server module is designed for load balancing and for running on standard low cost server platforms. When running on a dual processor server, streaming capacity can excess 500 Mbps with terabyte storage capacity.

EdgeStream client software provides a plug-in interface to the Windows Media and Real players. It can also be used to measure the Internet connection quality on a second by second basis. The PC client software is available for standard Windows platform and it is a 600 KB download. The Firmware client is a 300 KB (or smaller) download and can be either embedded in Windows XP or used with Windows CE.

1.5.1.3 Limelight Networks

Limelight Networks [30] was founded in 2001 at Tempe, Arizona, USA. Its content delivery services include HTTP/Web distribution of digital media files such as video, music, games, software and social media. It delivers content to media companies, including businesses operating in television, music, radio, newspaper, magazine, movie, video game, and software industries.

Content providers upload content either directly to the Limelight CDN's servers or to their own servers, which are connected directly to Limelight's network. Upon request from an end user, Limelight distributes that content to one or more Web server clusters which feed the specially configured servers at each content delivery location around the world. The content is then delivered directly to the end users either through ISPs or over the public Internet if appropriate. Like other commercial CDNs, it uses DNS redirection to reroute client requests to local clusters of machines, having built detailed maps of the Internet through a combination of BGP feeds and their own measurements, such as traceroutes from numerous vantage points.

Limelight Networks support Adobe Flash, MP3 audio, Microsoft Windows Media, Real, and Apple's QuickTime format for delivering on-demand streaming services. Limelight Networks proprietary software include Limelight ContentEdge for distributed content delivery via HTTP, Limelight MediaEdge Streaming for distributed video and music delivery via streaming, Limelight StorageEdge for storing customer's content library within Limelight's CDN architecture, and Limelight Custom CDN for custom distributed delivery solutions. Content providers using

Limelight's streaming services use Limelight User Exchange (LUX), which is a Web-based management and reporting console for tracking the end users' activity with real time reporting. All these software together assist in managing the content delivery system.

1.5.1.4 Mirror Image

Mirror Image [40] was founded in 1999 at Massachusetts, USA. It is a provider of online content, application, streaming media, Web computing, reporting, and transaction delivery services to the end users. It follows a Concentrated "Superstore" architecture, where content is placed in large Web server clusters in central locations close to densely populated user regions. Mirror Image exploits a global Content Access Point (CAP) infrastructure on top of the Internet to provide content providers, service providers, and enterprises with a platform for content delivery.

When a user request for content comes from a Mirror Image provisioned Web site, it is automatically routed to a global load balancer on the CAP network. The load balance uses DNS routing to determine the CAP location with fastest response time. Upon reception of the request at the selected CAP location, the caches and then the core databases are checked for the requested content. If the content is found, it is delivered to the user. On cache miss, the CAP network automatically returns a redirection status code "302" to the origin server's URL. Then the requested content is delivered to the user from the origin server and the CAP network retrieves (or pull) the content from the origin server and stores it for future subsequent requests.

Mirror Image provides content delivery, streaming media, and Web computing solutions, including Global Content Caching solution to offload traffic spikes while serving static content; Digital Asset Download solution to manage the storage and download of digital content; Video On-Demand solution for streaming delivery of digital content; Extensible Rules Engine (XRE) to give customers control over the delivery process; and Webcasting solution to allow users to send "one-to-many" messages for training, marketing, and distance learning outlets.

1.5.2 Academic CDNs

Unlike commercial CDNs, the use of P2P technologies is mostly common in academic CDNs. Thus, the content delivery follows a decentralized approach and request load is spread across all the participating hosts, and thus the system can handle node failures and sudden load surges. Academic CDNs built using P2P techniques are effective for static content only and therefore, are unable to handle dynamically generated content due to the uncachable nature of dynamic content. In this section, we present three representative academic CDNs, namely CoDeeN, Coral, and Globule. Table 1.3 provides a brief summary of these academic CDNs. Two other academic CDNs – FCAN (adaptive CDN for alleviating flash crowds) and COMODIN

Table 1.3 Summary of the existing academic CDNs

CDN Name	Description	Service Type	Implementation and Testing	Availability
CoDeeN www.codeen.cs. princeton. edu	CoDeeN is an academic testbed CDN built on top of PlanetLab	Provides caching of content and redirection of HTTP requests	Implemented in C/C++ and tested on Linux(2.4/2.6) and MacOS(10.2/10.3)	N/A
Coral www.coralcdn.org	Coral is a free P2P CDN. It is hosted on PlanetLab	Provides content replication in proportion to the content's popularity	Implemented in C++ and tested on Linux, OpenBSD, FreeBSD, and Mac OS X	No official release yet. Coral is a Free software, licensed under GPLv2 (GNU General Public License)
Globule www.globule.org	Globule is an open source collaborative CDN	Provides replication of content, monitoring of servers and redirecting client requests to available replicas	Implemented using PHP scripting, C/C++ and tested on Unix/Linux and Windows	Globule is open source, subject to a BSD-style license and the Apache software license for the packaged Apache HTTP server

(streaming CDN for collaborative media streaming services), are presented respectively in Chap. 11 and Chap. 12 of this book.

1.5.2.1 CoDeeN

CoDeeN [46, 64] is a P2P-based proxy server system developed at Princeton University, USA. It is an HTTP-based CDN, which gives *participating* users better performance to *most* Web sites. CoDeeN provides caching of Web content and redirection of HTTP requests. It is built on top of PlanetLab [9], consisting of a network of high performance proxy servers. CoDeeN nodes are deployed as "open" proxies in order to allow access from outside the hosting organization. Each CoDeeN node is capable of acting as a forward proxy, a reverse proxy, and a redirector. CoDeeN operates in the following way: (1) users set their internet caches to a nearby high bandwidth proxy that participates in the CoDeeN system; (2) the CoDeeN node acts as a forward proxy and tries to satisfy the request locally. On cache miss, the redirector logic built in the CoDeeN node determines where the request should be sent. For most requests the redirector take into account request locality, system load, reliability, and proximity information to forward the requests to other CoDeeN nodes, which act as a reverse proxy for the forwarded requests. Requests which are still not satisfied at this stage are sent to the origin server.

CoDeeN has the local monitoring ability that examines the service's primary resources, such as free file descriptors/sockets, CPU cycles, and DNS resolver service. It gathers information about the CoDeeN instance's state and its host environment. This information assists in assessing resource connection as well as external service availability. To monitor the health and status of the peers, each CoDeeN node employs two mechanisms – a lightweight UDP-based heartbeat and a "heavier" HTTP/TCP-level "fetch" helper [64]. In the first case, each proxy sends a heartbeat message once per second to one of its peers, which then responds (heartbeat acknowledgement or ACK) with piggybacked load information including peer's average load, system time, file descriptor availability, proxy and node uptimes, average hourly traffic, and DNS timing/failure statistics. By coupling the history of ACKs with their piggybacked local status information, each CoDeeN instance independently assesses the health of other nodes. In the later case, each CoDeeN node is employed with a toll to specify what fails when it can not retrieve a page within the allotted time. A history of failed fetches for each peer is maintained, which in combination with UDP-level heartbeats assists in determining if a node is viable for request redirection.

A number of projects are related to CoDeeN – CoBlitz (a scalable Web-based distribution system for large files), CoDeploy (an efficient synchronization tool for PlanetLab slices), CoDNS (a fast and reliable name lookup service), CoTop (a command line activity monitoring tool for PlanetLab), CoMon (a Web-based slice monitor that monitors most PlanetLab nodes), and CoTest (a login debugging tool).

A significant application service running on top of CoDeeN is CoBlitz [48]. It is a file transfer service which distributes large files without requiring any modifications

to standard Web servers and clients, since all the necessary support is located on CoDeeN itself. One of the motivations for building CoBlitz on top of CoDeeN is to ensure long duration caching so that client requested content can be served quickly even after demand for it drops. CoBlitz is publicly accessible which allows the clients to prepend the original URL with "`http://coblitz.codeen.org:3125`" and fetch it like any other URL. A customized DNS server maps the name coblitz.codeen.org to a nearby PlanetLab node. To deploy CoBlitz, the HTTP CDN, CoDeeN is made amenable to handling large files. This approach includes modifying large file handling to efficiently support them on CoDeeN and modifying its request-routing to enable more swarm-like behavior under heavy load. In CoBlitz, a large file is considered as a set of small files (chunks) that can be spread across the CDN. CoBlitz works if the chunks are fully cached, partially cached, or not at all cached, fetching any missing chunks from the origin as needed. Thus, while transferring large files over CoBlitz, no assumptions are made about the existence of the file on the peers.

1.5.2.2 Coral

Coral [28] is a free P2P content distribution network. It was developed by the New York University's Secure Computer Systems group during their visit to Stanford University, USA. It is designed to mirror Web content and its goal is to give most *users* better performance to *participating* Web sites. It uses the bandwidth of volunteers to avoid flash crowd and to reduce the load on Web sites and other Web content providers in general. CoralCDN is deployed on PlanetLab, instead of third party volunteer systems. To use CoralCDN, a content publisher, end-host client, or someone posting a link to a high-traffic portal has to append "`.nyud.net:8090`" to the hostname in a URL. Clients are redirected to the nearby Coral Web caches transparently through DNS redirection. Coral Web caches cooperate to transfer data from nearby peers whenever possible, minimizing both the load on the origin Web server and the latency perceived by the user. CoralCDN is built on top of the Coral key-value indexing layer. It allows nodes to access nearby cached objects without redundantly querying more distant nodes. It also prevents the creation of hotspots in the indexing infrastructure, even under degenerate loads.

CoralCDN is comprised of three main parts: a network of cooperative HTTP proxies for handling client requests; a network of DNS nameservers for ".nyud.net" that map clients to nearby CoralCDN HTTP proxy; and an underlying indexing infrastructure and clustering machinery on which the first two applications rely.

Coral uses an indexing abstraction called Distributed Sloppy Hash Table (DSHT), which is a variant of Distributed Hash Tables (DHTs) for building key value indexes. DSHTs are designed for applications storing soft-state key-value pairs, where multiple values may be stored under the same key. A DSHT caches key-value pairs at nodes whose identifiers are increasingly close to the key being referenced, as an inverse function of load. It has a "sloppy" storage technique that leverages cross-layer interaction between the routing and storage layers.

The CoralHTTP proxy satisfies HTTP requests for Coralized URLs. To minimize the load on the origin servers, a CoralHTTP proxy fetch Web pages from other proxies whenever possible. Each proxy keeps a local cache to fulfill requests immediately. If a CoralHTTP proxy discovers the requested content in one or more other proxies, it establishes parallel TCP connections to them (multiple other proxies) and issues an HTTP request to the first proxy to which it successfully connects. Once the neighboring proxy begins to send valid content, all other established TCP connections are closed. When a client requests content from a non-resident URL, CoralHTTP proxy first attempts to locate a cached copy. If the Coral indexing layer does not provide any referral or any of its referrals return the requested content, CoralHTTP proxy fetches the content directly from the origin server. In the face of a flash crowd, all CoralHTTP proxies naturally form a kind of "multicast tree" for retrieving the Web page, instead of making simultaneous requests to the origin server and data flows from the proxy that initially fetch the content from the origin server to those arriving later. Such behavior in CoralCDN is provided by combining optimistic references and cut-through routing.

The CoralDNS server maps the IP addresses to the hostnames of Coralized URLs and returns it to CoralHTTP proxies. Coral's architecture is based on clusters of well-connected machines. Clusters are exposed in the interface to higher-level software, and in fact form a crucial part of the DNS redirection mechanism. In order to improve locality, when a DNS resolver contacts a nearby CoralDNS server instance, the CoralDNS server returns the proxies within an appropriate cluster and ensures that future DNS requests form this client does not leave the cluster. A CoralDNS server only returns the CoralHTTP proxy addresses which is has recently verified first hand in order to check a proxy's liveness status synchronously prior to replying to a DNS query.

1.5.2.3 Globule

Globule [52] is an open-source collaborative CDN developed at the Vrije Universiteit in Amsterdam, the Netherlands. It aims to allow Web content providers to organize together and operate their own world-wide hosting platform. In particular, it is an overlay network composed of end user nodes that operate in a P2P fashion across a wide-area network, where participating members offer resources such as storage capacity, bandwidth, and processing power. It provides replication of content, monitoring of servers and redirection of client requests to available replicas.

In Globule, a site is defined as a collection of documents that belong to one specific user (the site's owner) and a server is a process running on a machine connected to a network, which executes an instance of the Globule software. Each site is composed of the origin, backup, replica, and redirector servers. The origin server(s) has the authority to contain all documents of the site and has the responsibility to distribute content among other involved servers. The backup servers maintain a full up-to-date copy of the hosted site. Other than backup servers, a number of replica servers can be used to host a site. While backup servers just maintain a copy, replica

servers aim to maximize performance based on the request load and QoS requirements. A replica server for a site is typically operated by a different user than the origin and a replica server typically contain a partial copy of the hosted site. One can view the replica server as a caching proxy which fetches the content from the origin server on a local cache miss. A redirector server is responsible for redirecting client requests to the optimal replica server for serving a given request. Redirectors in Globule can use either HTTP or DNS-based redirection. A redirector also implements a policy for client redirection. The default policy redirects clients to the closest replica in terms of estimated latency. Redirectors also monitor the availability of other servers to ensure effective redirection of requests. Depending on the role a server can perform as origin, replica, backup and/or redirector servers.

Globule takes inter-node latency as the proximity measure. This metric is used to optimally place replicas to the clients, and to redirect the clients to an appropriate replica server. Globule is implemented as a third-party module for the Apache HTTP Server that allows any given server to replicate its documents to other Globule servers. To replicate content, content providers only need to compile an extra module into their Apache server and edit a simple configuration file.

1.6 Visionary Thoughts for Practitioners

We envision the following technological evolutions to be realized in the coming years in CDN industry related research.

1.6.1 A Unified Content Network

To make content transformations and processing and infrastructure service accessible by the user, vendors have implemented Content Service Networks (CSN) [38], which act as another network infrastructure layer built upon CDNs and provide next generation of CDN services. CSN appears to be a variation of the conventional CDN. Network resources provided by a CSN is used as a "service" distribution channel for value added service providers in order to make their applications as an infrastructure service. This logical separation between content and services under the "Content Delivery/Distribution" and "Content Services" domain, is undesirable considering the on-going trend in content networking. Hence, a unified content network, which supports the coordinated composition and delivery of content and services, is highly desirable.

1.6.2 Dynamic Content

Dynamic content refers to the content which is generated on-demand using Web applications based on user requests. Such content generation is customized depending

Chapter 2
A Taxonomy of CDNs

Mukaddim Pathan and Rajkumar Buyya

2.1 Introduction

Content Delivery Networks (CDNs) [79, 97] have received considerable research attention in the recent past. A few studies have investigated CDNs to categorize and analyze them, and to explore the uniqueness, weaknesses, opportunities, and future directions in this field. Peng presents an overview of CDNs [75]. His work describes the critical issues involved in designing and implementing an effective CDN, and surveys the approaches proposed in literature to address these problems. Vakali et al. [95] present a survey of CDN architecture and popular CDN service providers. The survey is focused on understanding the CDN framework and its usefulness. They identify the characteristics and current practices in the content networking domain, and present an evolutionary pathway for CDNs, in order to exploit the current content networking trends. Dilley et al. [29] provide an insight into the overall system architecture of the leading CDN, Akamai [1]. They provide an overview of the existing content delivery approaches and describe Akamai's network infrastructure and its operations in detail. They also point out the technical challenges that are to be faced while constructing a global CDN like Akamai. Saroiu et al. [84] examine content delivery from the point of view of four content delivery systems: Hypertext Transfer Protocol (HTTP) Web traffic, the Akamai CDN, Gnutella [8, 25], and KaZaa [62, 66] peer-to-peer file sharing systems. They also present significant implications for large organizations, service providers, network infrastructure providers, and general content delivery providers. Kung et al. [60] describe a taxonomy for content networks and introduce a new class of content networks that perform "semantic aggregation and content-sensitive placement" of content. They classify content networks based on their attributes in two dimensions: content aggregation and content placement. Sivasubramanian et al. [89] identify the issues

Mukaddim Pathan
GRIDS Lab, Department of CSSE, The University of Melbourne, Australia,
e-mail: apathan@csse.unimelb.edu.au

Rajkumar Buyya
GRIDS Lab, Department of CSSE, The University of Melbourne, Australia,
e-mail: raj@csse.unimelb.edu.au

R. Buyya et al. (eds.), *Content Delivery Networks*,

for building a Web replica hosting system. Since caching infrastructure is a major building block of a CDN (e.g. Akamai) and content delivery is initiated from the origin server, they consider CDNs as replica hosting systems. In this context, they propose an architectural framework, review related research work, and categorize them. A survey of peer-to-peer (P2P) content distribution technologies [11] studies current P2P systems and categorize them by identifying their non-functional properties such as security, anonymity, fairness, increased scalability, and performance, as well as resource management, and organization capabilities. Through this study the authors make useful insights for the influence of the system design on these properties. Cardellini et al. [20] study the state of the art of Web system architectures that consists of multiple server nodes distributed on a local area. They provide a taxonomy of these architectures, and analyze routing mechanisms and dispatching algorithms for them. They also present future research directions in this context.

2.1.1 Motivations and Scope

As mentioned above, there exist a wide range of work covering different aspects of CDNs such as content distribution, replication, caching, and Web server placement. However, none of them attempts to perform a complete categorization of CDNs by considering the functional and non-functional aspects. The first aspects include technology usage and operations of a CDN, whereas the latter focus on CDN characteristics such as organization, management, and performance issues. Our approach of considering both functional and non-functional aspects of CDNs assists in examining the way in which the characteristics of a CDN are reflected in and affected by the architectural design decision followed by the given CDN. Therefore, our aim is to develop a comprehensive taxonomy of CDNs that identifies and categorizes numerous solutions and techniques related to various design dynamics.

The taxonomy presented in this chapter is built around the core issues for building a CDN system. In particular, we identify the following key issues/aspects that pose challenges in the development of a CDN:

- *What is required for a harmonious CDN composition?* It includes decisions based on different CDN organization, node interactions, relationships, and content/service types.
- *How to perform effective content distribution and management?* It includes the right choice of content selection, surrogate placement, content outsourcing, and cache organization methodologies.
- *How to route client requests to appropriate CDN node?* It refers to the usage of dynamic, scalable, and efficient routing techniques.
- *How to measure a CDN's performance?* It refers to the ability to predict, monitor, and ensure the end-to-end performance of a CDN.

A full-fledged CDN system design seeks to address additional issues to make the system robust, fault tolerant (with the ability to detect wide-area failures), secure,

and capable of wide-area application hosting. In this context, the issues to be addressed are:

- *How to handle wide-area failures in a CDN?* It involves the use of proper tools and systems for failure detection.
- *How to ensure security in a wide-area CDN system?* It refers to the solutions to counter distributed security threats.
- *How to achieve wide-area application hosting?* It seeks to develop proper techniques to enable CDNs to perform application hosting.

Each of the above issues aspects is an independent research area itself and many solutions and techniques can be found in literature and in practice. While realizing proper solution for the additional issues is obvious for a CDN development, the taxonomy presented in this chapter concentrates only on the first four core issues. However, we present the ideas in the context of the additional issues and also provide pointers to some recent related research work. Thus, the readers can get sufficient materials to comprehend respective issues to dive directly into their interested topic.

2.1.2 Contributions and Organization

The key contributions of this chapter are twofold:

- A taxonomy of CDNs with a complete coverage of this field to provide a comprehensive account of applications, features, and implementation techniques. The main aim of the taxonomy, therefore, is to explore the functional and non-functional features of CDNs and to provide a basis for categorizing the related solutions and techniques in this area.
- Map the taxonomy to a few representative CDNs to demonstrate its applicability to categorize and analyze the present-day CDNs. Such a mapping helps to perform "gap" analysis in this domain. It also assists to interpret the related essential concepts of this area and validates the accuracy of the taxonomy.

The remainder of this chapter is structured as follows: we start by presenting the taxonomy of CDNs in Sect. 2.2. In the next section, we map the taxonomy to the representative CDN systems, along with the insights perceived from this mapping. Thus, we prove the validity and applicability of the taxonomy. We discuss the additional issues in CDN development in Sect. 2.4 by highlighting research work in failure handling, security, and application hosting. Finally, we summarize and conclude the chapter in Sect. 2.5.

2.2 Taxonomy

In this section, we present a taxonomy of CDNs based on four key issues as shown in Fig. 2.1. They are – *CDN composition*, *content distribution and management*, *request-routing*, and *performance measurement*.

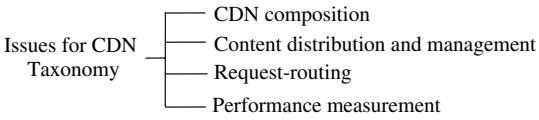

Fig. 2.1 Issues for CDN taxonomy

The first issue covers several aspects of CDNs related to organization and formation. This classifies the CDNs with respect to their structural attributes. The next issue pertains to the content distribution mechanisms in the CDNs. It describes the content distribution and management approaches of CDNs in terms of surrogate placement, content selection and delivery, content outsourcing, and organization of caches/replicas. Request-routing techniques in the existing CDNs are described as the next issue. Finally, the last issue deals with the performance measurement methodologies of CDNs.

2.2.1 CDN Composition

The analysis of the structural attributes of a CDN reveals that CDN infrastructural components are closely related to each other. Moreover, the structure of a CDN varies depending on the content/services it provides to its users. Within the structure of a CDN, a set of surrogates is used to build the content-delivery component, some combinations of relationships and mechanisms are used for redirecting client requests to a surrogate and interaction protocols are used for communications between CDN elements.

Figure 2.2 shows a taxonomy based on the various structural characteristics of CDNs. These characteristics are central to the composition of a CDN and they address the organization, types of servers used, relationships, and interactions among CDN components, as well as the different content and services provided.

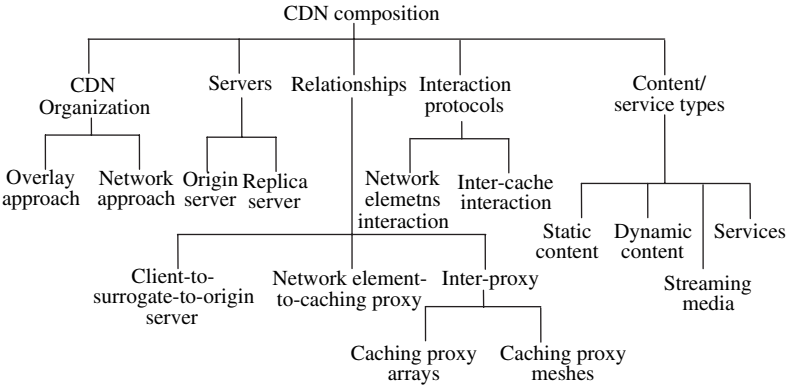

Fig. 2.2 CDN composition taxonomy

2.2.1.1 CDN Organization

There are two general approaches for building CDNs: *overlay* and *network* approach [61]. In the overlay approach, application-specific servers and caches at several places in the network handle the distribution of specific content types (e.g. Web content, streaming media, and real time video). Other than providing the basic network connectivity and guaranteed QoS for specific request/traffic, the core network components such as routers and switches play no active role in content delivery. Most of the commercial CDN providers such as Akamai and Limelight Networks follow the overlay approach for CDN organization. These CDN providers replicate content to cache servers worldwide. When content requests are received from the end users, they are redirected to the nearest CDN server, thus improving Web site response time. As the CDN providers need not to control the underlying network elements, the management is simplified in an overlay approach and it opens opportunities for new services.

In the network approach, the network components including routers and switches are equipped with code for identifying specific application types and for forwarding the requests based on predefined policies. Examples of this approach include devices that redirect content requests to local caches or switch traffic (coming to data centers) to specific servers, optimized to serve specific content types. Some CDNs (e.g. Akamai, Mirror Image) use both network and overlay approaches for CDN organization. In such case, a network element (e.g. switch) can act at the front end of a server farm and redirects the content request to a nearby application-specific surrogate server.

2.2.1.2 Servers

The servers used by a CDN are of two types – *origin* and *replica* servers. The server where the definitive version of the content resides is called origin server. It is updated by the content provider. On the other hand, a replica server stores a copy of the content but may act as an authoritative reference for client responses. The origin server communicates with the distributed replica servers to update the content stored in it. A replica server in a CDN may serve as a media server, Web server or as a cache server. A media server serves any digital and encoded content. It consists of media server software. Based on client requests, a media server responds to the query with the specific video or audio clip. A Web server contains the links to the streaming media as well as other Web-based content that a CDN wants to handle. A cache server makes copies (i.e. caches) of content at the edge of the network in order to bypass the need of accessing origin server to satisfy every content request.

2.2.1.3 Relationships

The complex distributed architecture of a CDN exhibits different relationships between its constituent components. The graphical representations of these

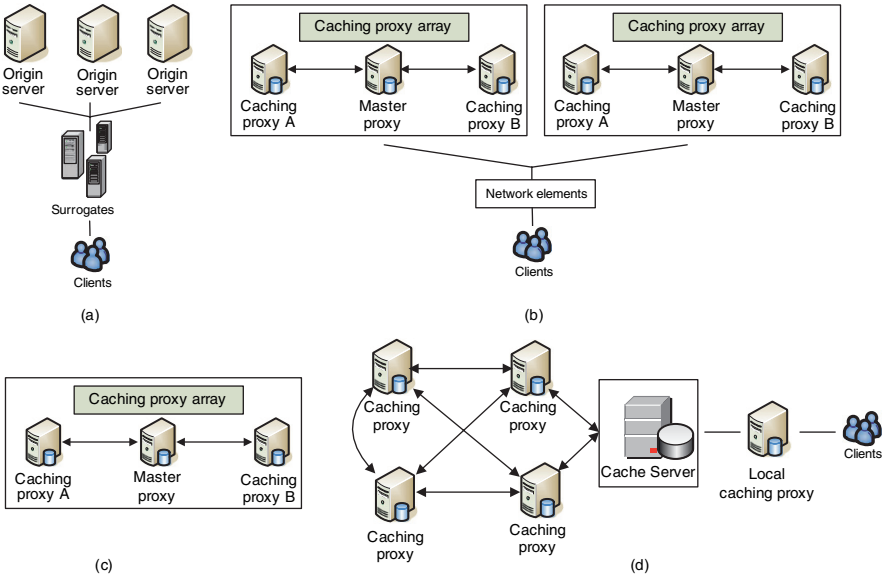

Fig. 2.3 Relationships: (a) Client-to-surrogate-to-origin server; (b) Network element-to-caching proxy; (c) Caching proxy arrays; (d) Caching proxy meshes

relationships are shown in Fig. 2.3. These relationships involve components such as clients, surrogates, origin server, proxy caches, and other network elements. These components communicate to replicate and cache content within a CDN. Replication involves creating and maintaining duplicate copy of a given content on different computer systems. It typically involves "pushing" content from the origin server to the replica servers [17]. On the other hand, caching involves storing cacheable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests [26, 27, 99].

In a CDN environment, the basic relationship for content delivery is among the client, surrogates and origin servers. A client may communicate with surrogate server(s) for requests intended for one or more origin servers. Where a surrogate is not used, the client communicates directly with the origin server. The communication between a user and surrogate takes place in a transparent manner, as if the communication is with the intended origin server. The surrogate serves client requests from its local cache or acts as a gateway to the origin server. The relationship among client, surrogates, and the origin server is shown in Fig. 2.3(a).

As discussed earlier, CDNs can be formed using a network approach, where logic is deployed in the network elements (e.g. router, switch) to forward traffic to caching servers/proxies that are capable of serving client requests. The relationship in this case is among the client, network element, and caching servers/proxies (or proxy arrays), which is shown in Fig. 2.3(b). Other than these relationships, caching proxies within a CDN may communicate with each other. A proxy cache is an application-layer network service for caching Web objects. Proxy caches can be

simultaneously accessed and shared by many users. A key distinction between the CDN proxy caches and ISP-operated caches is that the former serve content only for certain content provider, namely CDN customers, while the latter cache content from all Web sites [41].

Based on inter-proxy communication [26], caching proxies can be arranged in such a way that proxy arrays (Fig. 2.3(c)) and proxy meshes (Fig. 2.3(d)) are formed. A caching proxy array is a tightly-coupled arrangement of caching proxies. In a caching proxy array, an authoritative proxy acts as a master to communicate with other caching proxies. A user agent can have relationship with such an array of proxies. A caching proxy mesh is a loosely-coupled arrangement of caching proxies. Unlike the caching proxy arrays, proxy meshes are created when the caching proxies have one-to-one relationship with other proxies. Within a caching proxy mesh, communication can happen at the same level between peers, and with one or more parents [26]. A cache server acts as a gateway to such a proxy mesh and forwards client requests coming from client's local proxy.

2.2.1.4 Interaction Protocols

Based on the communication relationships described earlier, we can identify the interaction protocols that are used for interaction among CDN components. Such interactions can be broadly classified into two types: *interaction between network elements* and *interaction between caches*. Figure 2.4 shows different protocols that are used in a CDN for interaction among CDN elements. Examples of protocols for network element interaction are *Network Element Control Protocol (NECP)* and *Web Cache Control Protocol*. On the other hand, *Cache Array Routing Protocol (CARP)*, *Internet Cache Protocol (ICP)*, *Hypertext Caching protocol (HTCP)*, and *Cache Digest* are the examples of inter-cache interaction protocols.

The Network Element Control Protocol (NECP) [24] is a lightweight protocol for signaling between servers and the network elements that forward traffic to them. The network elements consist of a range of devices, including content-aware switches and load-balancing routers. NECP allows network elements to perform load balancing across a farm of servers and redirection to interception proxies. However, it does not dictate any specific load balancing policy. Rather, this protocol provides methods for network elements to learn about server capabilities, availability and hints as

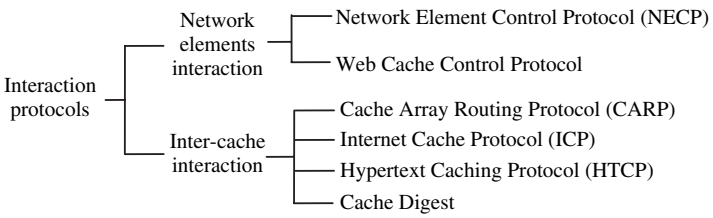

Fig. 2.4 Interaction protocols

to which flows can and cannot be served. Hence, network elements gather necessary information to make load balancing decisions. Thus, it avoids the use of proprietary and mutually incompatible protocols for this purpose. NECP is intended for use in a wide variety of server applications, including for origin servers, proxies, and interception proxies. It uses the Transport Control Protocol (TCP). When a server is initialized, it establishes a TCP connection to the network elements using a well-known port number. Messages can then be sent bi-directionally between the server and network elements. Most messages consist of a request followed by a reply or acknowledgement. Receiving a positive acknowledgement implies the recording of some state in a peer. This state can be assumed to remain in that peer until it expires or the peer crashes. In other words, this protocol uses a "hard state" model. Application level KEEPALIVE messages are used to detect a crashed peer in such communications. When a node detects that its peer has been crashed, it assumes that all the states in that peer need to be reinstalled after the peer is revived.

The Web Cache Control Protocol (WCCP) [24] specifies interaction between one or more routers and one or more Web-caches. It runs between a router functioning as a redirecting network element and interception proxies. The purpose of such interaction is to establish and maintain the transparent redirection of selected types of traffic flow through a group of routers. The selected traffic is redirected to a group of Web-caches in order to increase resource utilization and to minimize response time. WCCP allows one or more proxies to register with a single router to receive redirected traffic. This traffic includes user requests to view pages and graphics on World Wide Web (WWW) servers, whether internal or external to the network, and the replies to those requests. This protocol allows one of the proxies, the designated proxy, to dictate to the router how redirected traffic is distributed across the caching proxy array. WCCP provides the means to negotiate the specific method used to distribute load among Web caches. It also provides methods to transport traffic between router and cache.

The Cache Array Routing Protocol (CARP) [96] is a distributed caching protocol based on a known list of loosely coupled proxy servers and a hash function for dividing URL space among those proxies. An HTTP client implementing CARP can route requests to any member of the Proxy Array. The proxy array membership table is defined as a plain ASCII text file retrieved from an Array Configuration URL. The hash function and the routing algorithm of CARP take a member proxy defined in the proxy array membership table, and make an on-the-fly determination about the proxy array member which should be the proper container for a cached version of a resource pointed to by a URL. Since requests are sorted through the proxies, duplication of cache content is eliminated and global cache hit rates are improved. Downstream agents can then access a cached resource by forwarding the proxied HTTP request for the resource to the appropriate proxy array member.

The Internet Cache Protocol (ICP) [101] is a lightweight message format used for inter-cache communication. Caches exchange ICP queries and replies to gather information to use in selecting the most appropriate location in order to retrieve an object. Other than functioning as an object location protocol, ICP messages can also be used for cache selection. ICP is a widely deployed protocol. Although, Web

caches use HTTP for the transfer of object data, most of the caching proxy implementations support it in some form. It is used in a caching proxy mesh to locate specific Web objects in neighboring caches. One cache sends an ICP query to its neighbors and the neighbors respond with an ICP reply indicating a "HIT" or a "MISS". Failure to receive a reply from the neighbors within a short period of time implies that the network path is either congested or broken. Usually, ICP is implemented on top of User Datagram Protocol (UDP) in order to provide important features to Web caching applications. Since UDP is an unreliable and connectionless network transport protocol, an estimate of network congestion and availability may be calculated by ICP loss. This sort of loss measurement together with the round-trip-time provides a way to load balancing among caches.

The Hyper Text Caching Protocol (HTCP) [98] is a protocol for discovering HTTP caches, cached data, managing sets of HTTP caches and monitoring cache activity. HTCP is compatible with HTTP 1.0. This is in contrast with ICP, which was designed for HTTP 0.9. HTCP also expands the domain of cache management to include monitoring a remote cache's additions and deletions, requesting immediate deletions, and sending hints about Web objects such as the third party locations of cacheable objects or the measured uncacheability or unavailability of Web objects. HTCP messages may be sent over UDP or TCP. HTCP agents must not be isolated from network failure and delays. An HTCP agent should be prepared to act in useful ways in the absence of response or in case of lost or damaged responses.

Cache Digest [42] is an exchange protocol and data format. It provides a solution to the problems of response time and congestion associated with other inter-cache communication protocols such as ICP and HTCP. They support peering between cache servers without a request-response exchange taking place. Instead, other servers who peer with it fetch a summary of the content of the server (i.e. the Digest). When using Cache Digest it is possible to accurately determine whether a particular server caches a given URL. It is currently performed via HTTP. A peer answering a request for its digest will specify an expiry time for that digest by using the HTTP Expires header. The requesting cache thus knows when it should request a fresh copy of that peer's digest. In addition to HTTP, Cache Digest could be exchanged via FTP. Although the main use of Cache Digest is to share summaries of which URLs are cached by a given server, it can be extended to cover other data sources. Cache Digest can be a very powerful mechanism to eliminate redundancy and making better use of Internet server and bandwidth resources.

2.2.1.5 Content/Service Types

CDN providers host third-party content for fast delivery of any digital content, including – *static content*, *dynamic content*, *streaming media* (e.g. audio, real time video), and different *content services* (e.g. directory service, e-commerce service, and file transfer service). The sources of content are large enterprises, Web

service providers, media companies, and news broadcasters. Variation in content and services delivered requires a CDN to adopt application-specific characteristics, architectures, and technologies. Due to this reason, some of the CDNs are dedicated for delivering particular content and/or services. Here, we analyze the characteristics of the content/service types to reveal their heterogeneous nature.

Static content refers to content for which the frequency of change is low. It does not change depending on user requests. It includes static HTML pages, embedded images, executables, PDF documents, software patches, audio and/or video files. All CDN providers support this type of content delivery. This type of content can be cached easily and their freshness can be maintained using traditional caching technologies.

Dynamic content refers to the content that is personalized for the user or created on-demand by the execution of some application process. It changes frequently depending on user requests. It includes animations, scripts, and DHTML. Due to the frequently changing nature of the dynamic content, usually it is considered as uncachable.

Streaming media can be live or on-demand. Live media delivery is used for live events such as sports, concerts, channel, and/or news broadcast. In this case, content is delivered "instantly" from the encoder to the media server, and then onto the media client. In case of on-demand delivery, the content is encoded and then is stored as streaming media files in the media servers. The content is available upon requests from the media clients. On-demand media content can include audio and/or video on-demand, movie files and music clips. Streaming servers are adopted with specialized protocols for delivery of content across the IP network.

A CDN can offer its network resources to be used as a service distribution channel and thus allows the value-added services providers to make their application as an Internet infrastructure service. When the edge servers host the software of value-added services for content delivery, they may behave like transcoding proxy servers, remote callout servers, or surrogate servers [64]. These servers also demonstrate capability for processing and special hosting of the value-added Internet infrastructure services. Services provided by CDNs can be directory, Web storage, file transfer, and e-commerce services. Directory services are provided by the CDN for accessing the database servers. Users query for certain data is directed to the database servers and the results of frequent queries are cached at the edge servers of the CDN. Web storage service provided by the CDN is meant for storing content at the edge servers and is essentially based on the same techniques used for static content delivery. File transfer services facilitate the worldwide distribution of software, virus definitions, movies on-demand, and highly detailed medical images. All these contents are static by nature. Web services technologies are adopted by a CDN for their maintenance and delivery. E-commerce is highly popular for business transactions through the Web. Shopping carts for e-commerce services can be stored and maintained at the edge servers of the CDN and online transactions (e.g. third-party verification, credit card transactions) can be performed at the edge of CDNs. To facilitate this service, CDN edge servers should be enabled with dynamic content caching for e-commerce sites.

2.2.2 Content Distribution and Management

Content distribution and management is strategically vital in a CDN for efficient content delivery and for overall performance. Content distribution includes – *content selection and delivery* based on the type and frequency of specific user requests; *placement of surrogates* to some strategic positions so that the edge servers are close to the clients; and *content outsourcing* to decide which outsourcing methodology to follow. Content management is largely dependent on the techniques for *cache organization* (i.e. caching techniques, cache maintenance, and cache update). The content distribution and management taxonomy is shown in Fig. 2.5.

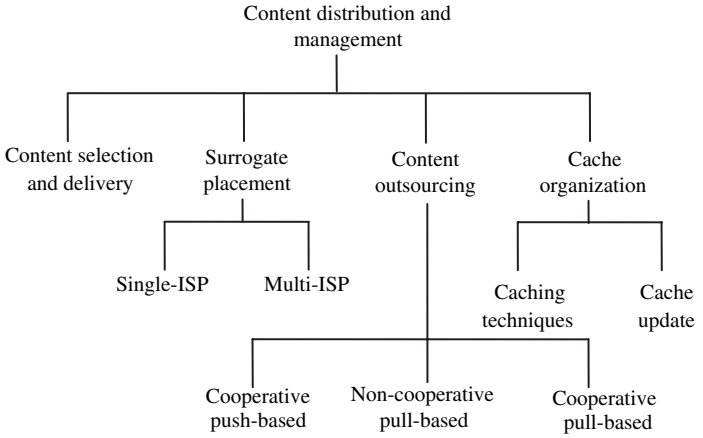

Fig. 2.5 Content distribution and management taxonomy

2.2.2.1 Content Selection and Delivery

The efficiency of content delivery lies in the right selection of content to be delivered to the end users. An appropriate content selection approach can assist in the reduction of client download time and server load. Figure 2.6 shows the taxonomy of content selection and delivery techniques. Content can be delivered to the customers in *full* or *partial*.

Full-site content selection and delivery is a simplistic approach where the surrogate servers perform "entire replication" in order to deliver the total content site to the end users. With this approach, a content provider configures its DNS in such a way that all client requests for its Web site are resolved by a CDN server, which then delivers all of the content. The main advantage of this approach is its simplicity. However, such a solution is not feasible considering the on-going increase in the size of Web objects. Although the price of storage hardware is decreasing, sufficient storage space on the edge servers is never guaranteed to store all the content

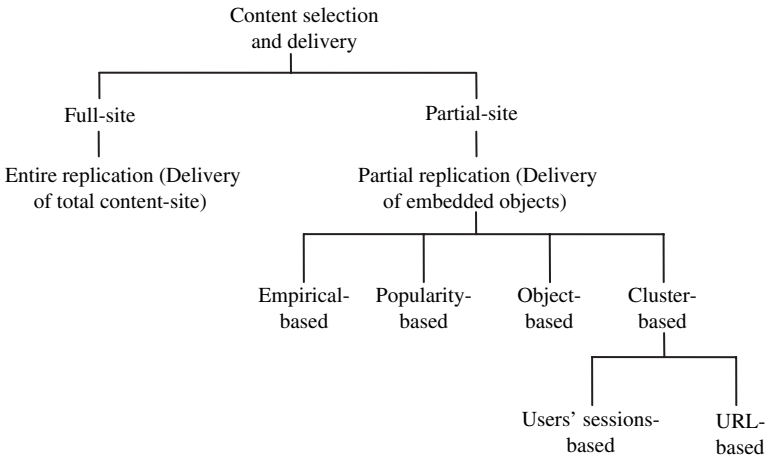

Fig. 2.6 Taxonomy of content selection and delivery

from content providers. Moreover, since the Web content is not static, the problem of updating such a huge collection of Web objects is unmanageable.

On the other hand, in partial-site content selection and delivery, surrogate servers perform "partial replication" to deliver only embedded objects – such as Web page images – from the corresponding CDN. With partial-site content delivery, a content provider modifies its content so that links to specific objects have host names in a domain for which the CDN provider is authoritative. Thus, the base HTML page is retrieved from the origin server, while embedded objects are retrieved from CDN cache servers. A partial-site approach is better than the full-site approach in the sense that the former reduces load on the origin server and on the site's content generation infrastructure. Moreover, due to the infrequent change of embedded content, a partial-site approach exhibits better performance.

Content selection is dependent on the suitable management strategy used for replicating Web content. Based on the approach to select embedded objects to perform replication, partial-site approach can be further divided into – *empirical*, *popularity*, *object*, and *cluster*-based replication. In a empirical-based [23] approach, the Web site administrator empirically selects the content to be replicated to the edge servers. Heuristics are used in making such an empirical decision. The main drawback of this approach lies in the uncertainty in choosing the right heuristics. In a popularity-based approach, the most popular objects are replicated to the surrogates. This approach is time consuming and reliable objects request statistics is not guaranteed due to the popularity of each object varies considerably. Moreover, such statistics are often not available for newly introduced content. In an object-based approach, content is replicated to the surrogate servers in units of objects. This approach is greedy because each object is replicated to the surrogate server (under storage constraints) that gives the maximum performance gain [23, 102]. Although such a greedy approach achieve the best performance, it suffers from high

complexity to implement on real applications. In a cluster-based approach, Web content is grouped based on either correlation or access frequency and is replicated in units of content clusters. The clustering procedure is performed either by fixing the number of clusters or by fixing the maximum cluster diameter, since neither the number nor the diameter of the clusters can ever be known. The content clustering can be either users' sessions-based or URL-based. In a user's session-based [36] approach, Web log files are used to cluster a set of users' navigation sessions, which show similar characteristics. This approach is beneficial because it helps to determine both the groups of users with similar browsing patterns and the groups of pages having related content. In a URL-based approach, clustering of Web content is done based on Web site topology [23, 36]. The most popular objects are identified from a Web site and are replicated in units of clusters where the correlation distance between every pair of URLs is based on a certain correlation metric. Experimental results show that content replication based on such clustering approaches reduce client download time and the load on servers. However, these schemes suffer from the complexity involved to deploy them.

2.2.2.2 Surrogate Placement

Since location of surrogate servers is closely related to the content delivery process, extra emphasis is put on the issue of choosing the best location for each surrogate. The goal of optimal surrogate placement is to reduce user perceived latency for accessing content and to minimize the overall network bandwidth consumption for transferring replicated content from servers to clients. The optimization of both of these metrics results in reduced infrastructure and communication cost for the CDN provider. Therefore, optimal placement of surrogate servers enables a CDN to provide high quality services and low CDN prices [88].

Figure 2.7 shows different surrogate server placement strategies. Theoretical approaches such as *minimum k-center problem* and *k-Hierarchically well-Separated Trees (k-HST)* model the server placement problem as the *center placement problem* which is defined as follows: for the placement of a given number of centers, minimize the maximum distance between a node and the nearest center. The k-HST [16, 47] algorithm solves the server placement problem according to graph theory. In this approach, the network is represented as a graph $G(V,E)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of links. The algorithm consists of two

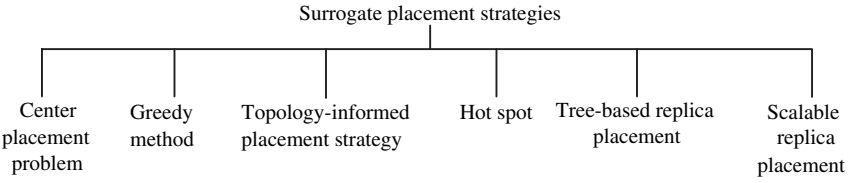

Fig. 2.7 Surrogate placement strategies

phases. In the first phase, a node is arbitrarily selected from the complete graph (parent partition) and all the nodes which are within a random radius from this node form a new partition (child partition). The radius of the child partition is a factor of k smaller than the diameter of the parent partition. This process continues until each of the nodes is in a partition of its own. Thus the graph is recursively partitioned and a tree of partitions is obtained with the root node being the entire network and the leaf nodes being individual nodes in the network. In the second phase, a virtual node is assigned to each of the partitions at each level. Each virtual node in a parent partition becomes the parent of the virtual nodes in the child partitions and together the virtual nodes form a tree. Afterwards, a greedy strategy is applied to find the number of centers needed for the resulted k-HST tree when the maximum center-node distance is bounded by D. The minimum k-center problem [47] can be described as follows: (1) Given a graph $G(V,E)$ with all its edges arranged in non-decreasing order of edge cost $c : c(e_1) \leq c(e_2) \leq \ldots. \leq c(e_m)$, construct a set of square graphs $G^2{}_1, G^2{}_2, \ldots., G^2{}_m$. Each square graph of G, denoted by G^2 is the graph containing nodes V and edges (u,v) wherever there is a path between u and v in G. (2) Compute the maximal independent set M_i for each $G^2{}_i$. An independent set of G^2 is a set of nodes in G that are at least three hops apart in G and a maximal independent set M is defined as an independent set V' such that all nodes in $V - V'$ are at most one hop away from nodes in V'. (3) Find smallest i such that $M_i \leq K$, which is defined as j. (4) Finally, M_j is the set of K center.

Due to the computational complexity of these algorithms, some heuristics such as *Greedy replica placement* and *Topology-informed placement strategy* have been developed. These suboptimal algorithms take into account the existing information from CDN, such as workload patterns and the network topology. They provide sufficient solutions with lower computation cost. The greedy algorithm [59] chooses M servers among N potential sites. In the first iteration, the cost associated with each site is computed. It is assumed that access from all clients converges to the site under consideration. Hence, the lowest-cost site is chosen. In the second iteration, the greedy algorithm searches for a second site (yielding the next lowest cost) in conjunction with the site already chosen. The iteration continues until M servers have been chosen. The greedy algorithm works well even with imperfect input data. But it requires the knowledge of the clients locations in the network and all pair wise inter-node distances. In topology-informed placement strategy [48], servers are placed on candidate hosts in descending order of outdegrees (i.e. the number of other nodes connected to a node). Here the assumption is that nodes with more outdegrees can reach more nodes with smaller latency. This approach uses Autonomous Systems (AS) topologies where each node represents a single AS and node link corresponds to Border Gateway Protocol (BGP) peering. In an improved topology-informed placement strategy [81], router-level Internet topology is used instead of AS-level topology. In this approach, each LAN associated with a router is a potential site to place a server, rather than each AS being a site.

Other server placement algorithms like *Hot Spot* [78] and *Tree-based* [63] replica placement are also used in this context. The *hotspot* algorithm places replicas near the clients generating greatest load. It sorts the N potential sites according to the

amount of traffic generated surrounding them and places replicas at the top M sites that generate maximum traffic. The *tree-based* replica placement algorithm is based on the assumption that the underlying topologies are trees. This algorithm models the replica placement problem as a dynamic programming problem. In this approach, a tree T is divided into several small trees T_i and placement of t proxies is achieved by placing t_i' proxies in the best way in each small tree T_i, where $t = \sum_i t_i'$. Another example is *Scan* [21], which is a scalable replica management framework that generates replicas on demand and organizes them into an application-level multicast tree. This approach minimizes the number of replicas while meeting clients' latency constraints and servers' capacity constraints. More information on Scan can be found in Chap. 3 of this book.

For surrogate server placement, the CDN administrators also determine the optimal number of surrogate servers using *single-ISP* and *multi-ISP* approach [95]. In the Single-ISP approach, a CDN provider typically deploys at least 40 surrogate servers around the network edge to support content delivery [30]. The policy in a single-ISP approach is to put one or two surrogates in each major city within the ISP coverage. The ISP equips the surrogates with large caches. An ISP with global network can thus have extensive geographical coverage without relying on other ISPs. The drawback of this approach is that the surrogates may be placed at a distant place from the clients of the CDN provider. In Multi-ISP approach, the CDN provider places numerous surrogate servers at as many global ISP Points of Presence (POPs) as possible. It overcomes the problems with single-ISP approach and surrogates are placed close to the users and thus content is delivered reliably and timely from the requesting client's ISP. Large CDN providers such as Akamai have more than 25000 servers [1, 29]. Other than the cost and complexity of setup, the main disadvantage of the multi-ISP approach is that each surrogate server receives fewer (or no) content requests which may result in idle resources and poor CDN performance [71]. Estimation of performance of these two approaches shows that single-ISP approach works better for sites with low-to-medium traffic volumes, while the multi-ISP approach is better for high-traffic sites [30].

2.2.2.3 Content Outsourcing

Given a set of properly placed surrogate servers in a CDN infrastructure and a chosen content for delivery, choosing an efficient content outsourcing practice is crucial. Content outsourcing is performed using *cooperative push-based*, *non-cooperative pull-based*, or *cooperative pull-based* approaches.

Cooperative push-based approach depends on the pre-fetching of content to the surrogates. Content is pushed to the surrogate servers from the origin, and surrogate servers cooperate to reduce replication and update cost. In this scheme, the CDN maintains a mapping between content and surrogate servers, and each request is directed to the closest surrogate server or otherwise the request is directed to the origin server. Under this approach, greedy-global heuristic algorithm is suitable for making replication decision among cooperating surrogate servers [54]. Still it is

considered as a theoretical approach since it has not been used by any commercial CDN provider [23, 36].

In non-cooperative pull-based approach, client requests are directed to their closest surrogate servers. If there is a cache miss, surrogate servers pull content from the origin server. Most popular CDN providers (e.g. Akamai, Mirror Image) use this approach. The drawback of this approach is that an optimal server is not always chosen to serve content request [49]. Many CDNs use this approach since the cooperative push-based approach is still at the experimental stage [71].

The cooperative pull-based approach differs from the non-cooperative approach in the sense that surrogate servers cooperate with each other to get the requested content in case of a cache miss. In the cooperative pull-based approach client requests are directed to the closest surrogate through DNS redirection. Using a distributed index, the surrogate servers find nearby copies of requested content and store it in the cache. The cooperative pull-based approach is reactive wherein a data object is cached only when the client requests it. An academic CDN Coral [34], using a distributed index, follows the cooperative pull-based approach where the proxies cooperate each other in case of case miss.

In the context of content outsourcing, it is crucial to determine in which surrogate servers the outsourced content should be replicated. Several works can be found in literature demonstrating the effectiveness of different replication strategies for outsourced content. Kangasharju et al. [54] have used four heuristics, namely *random*, *popularity*, *greedy-single*, and *greedy-global*, for replication of outsourced content. Tse [94] has presented a set of greedy approaches where the placement is occurred by balancing the loads and sizes of the surrogate servers. Pallis et al. [72] have presented a self-tuning, parameterless algorithm called *lat-cdn* for optimally placing outsourced content in CDN's surrogate servers. This algorithm uses object's latency to make replication decision. An object's latency is defined as the delay between a request for a Web object and receiving the object in its entirety. An improvement of the lat-cdn algorithm is *il2p* [70], which places the outsourced objects to surrogate servers with respect to the latency and load of the objects.

2.2.2.4 Cache Organization and Management

Content management is essential for CDN performance, which is mainly dependent on the cache organization approach followed by the CDN. Cache organization is in turn composed of the caching techniques used and the frequency of cache update to ensure the freshness, availability, and reliability of content. Other than these two, the cache organization may also involve the integrated use of caching and replication on a CDN's infrastructure. Such integration may be useful for a CDN for effective content management. Potential performance improvement is also possible in terms of perceived latency, hit ratio, and byte hit ratio if replication and caching are used together in a CDN [91]. Moreover, the combination of caching with replication assists to fortify against flash crowd events. In this context, Stamos et al. [90] have presented a generic non-parametric heuristic method that integrates Web caching with

content replication. They have developed a placement similarity approach, called SRC, for evaluating the level of integration. Another integrated approach called Hybrid, which combines static replication and Web caching using an analytic model of LRU is presented by Bakiras et al. [13]. Hybrid gradually fills the surrogate servers caches with static content in each iteration, as long as it contributes to the optimization of response times. More information on the integrated use of caching and replication can be found in Chap. 4 and Chap. 5 of this book.

Content caching in CDNs can be *intra-cluster* or *inter-cluster* basis. A taxonomy of caching techniques is shown in Fig. 2.8. *Query-based*, *digest-based*, *directory-based*, or *hashing-based* scheme can be used for intra-cluster caching of content. In a query-based [101] scheme, on a cache miss a CDN server broadcasts a query to other cooperating CDN servers. The problems with this scheme are the significant query traffic and the delay because a CDN server has to wait for the last "miss" reply from all the cooperating surrogates before concluding that none of its peers has the requested content. Because of these drawbacks, the query-based scheme suffers from implementation overhead. The digest-based [83] approach overcomes the problem of flooding queries in query-based scheme. In the digest-based scheme, each of the CDN servers maintains a digest of content held by the other cooperating surrogates. The cooperating surrogates are informed about any sort of update of the content by the updating CDN server. On checking the content digest, a CDN server can take the decision to route a content request to a particular surrogate. The main drawback is that it suffers from update traffic overhead, because of the frequent exchange of the update traffic to make sure that the cooperating surrogates have correct information about each other. The directory-based [38] scheme is a centralized version of the digest-based scheme. In directory-based scheme, a centralized server keeps content information of all the cooperating surrogates inside a cluster. Each CDN server only notifies the directory server when local updates occur and queries the directory server whenever there is a local cache miss. This scheme experiences potential bottleneck and single point of failure since the directory server receives update and query traffic from all cooperating surrogates. In a hashing-based [55, 96] scheme, the cooperating CDN servers maintain the same hashing function. A designated CDN server holds a content based on content's URL, IP addresses of the CDN servers, and the hashing function. All requests for that particular content are directed to that designated server. Hashing-based scheme is more

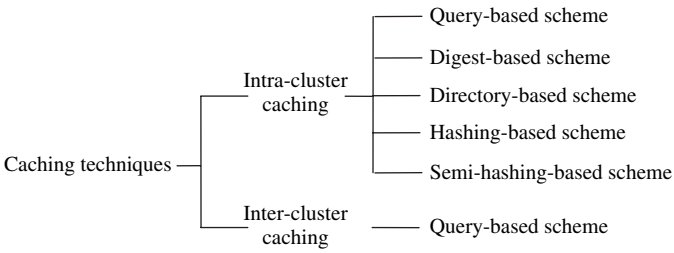

Fig. 2.8 Caching techniques taxonomy

efficient than other schemes since it has the smallest implementation overhead and highest content sharing efficiency. However, it does not scale well with local requests and multimedia content delivery since the local client requests are directed to and served by other designated CDN servers. To overcome this problem, a semi-hashing-based scheme [24, 67] can be followed. Under the semi-hashing-based scheme, a local CDN server allocates a certain portion of its disk space to cache the most popular content for its local users and the remaining portion to cooperate with other CDN servers via a hashing function. Like pure hashing, semi-hashing has small implementation overhead and high content sharing efficiency. In addition, it has been found to significantly increase the local hit rate of the CDN.

A hashing-based scheme is not appropriate for inter-cluster cooperative caching, because representative CDN servers of different clusters are normally distributed geographically. The digest-based or directory-based scheme is also not suitable for inter-cluster caching since the representative CDN servers have to maintain a huge content digest and/or directory including the content information of CDN servers in other clusters. Hence, a query-based scheme can be used for inter-cluster caching [68]. In this approach, when a cluster fails to serve a content request, it queries other neighboring cluster(s). If the content can be obtained from this neighbor, it replies with a "hit" message or if not, it forwards the request to other neighboring clusters. All the CDN servers inside a cluster use hashing based scheme for serving content request and the representative CDN server of a cluster only queries the designated server of that cluster to serve a content request. Hence, this scheme uses the hashing-based scheme for intra-cluster content routing and the query-based scheme for inter-cluster content routing. This approach improves performance since it limits flooding of query traffic and overcomes the problem of delays when retrieving content from remote servers through the use of a Timeout and Time-to-Live (TTL) value with each query message.

Cached objects in the surrogate servers of a CDN have associated expiration times after which they are considered stale. Ensuring the freshness of content is necessary to serve the clients with up to date information. If there are delays involved in propagating the content, a CDN provider should be aware that the content may be inconsistent and/or expired. To manage the consistency and freshness of content at replicas, CDNs deploy different cache update techniques. The taxonomy of cache update mechanisms is shown in Fig. 2.9.

The most common cache update method is the *periodic update*. To ensure content consistency and freshness, the content provider configures its origin Web servers to provide instructions to caches about what content is cacheable, how long different content is to be considered fresh, when to check back with the origin server

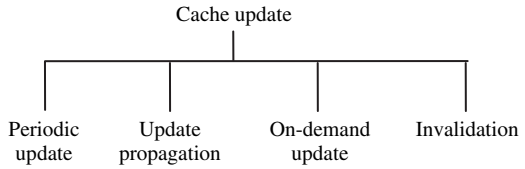

Fig. 2.9 Cache update taxonomy

for updated content, and so forth [41]. With this approach, caches are updated in a regular fashion. But this approach suffers from significant levels of unnecessary traffic generated from update traffic at each interval. The *update propagation* is triggered with a change in content. It performs active content pushing to the CDN cache servers. In this mechanism, an updated version of a document is delivered to all caches whenever a change is made to the document at the origin server. For frequently changing content, this approach generates excess update traffic. *On-demand update* is a cache update mechanism where the latest copy of a document is propagated to the surrogate cache server based on prior request for that content. This approach follows an assume nothing structure and content is not updated unless it is requested. The disadvantage of this approach is the back-and-forth traffic between the cache and origin server in order to ensure that the delivered content is the latest. Another cache update approach is *invalidation*, in which an invalidation message is sent to all surrogate caches when a document is changed at the origin server. The surrogate caches are blocked from accessing the documents when it is being changed. Each cache needs to fetch an updated version of the document individually later. The drawback of this approach is that it does not make full use of the distribution network for content delivery and belated fetching of content by the caches may lead to inefficiency of managing consistency among cached contents.

Generally, CDNs give the content provider control over freshness of content and ensure that all CDN sites are consistent. However, content providers themselves can build their own policies or use some heuristics to deploy organization specific caching policies. In the first case, content providers specify their caching policies in a format unique to the CDN provider, which propagates the rule sets to its caches. These rules specify instructions to the caches on how to maintain the freshness of content through ensuring consistency. In the latter case, a content provider can apply some heuristics rather than developing complex caching policies. With this approach, some of the caching servers adaptively learn over time about the frequency of change of content at the origin server and tune their behavior accordingly.

2.2.3 Request-Routing

A *request-routing system* is responsible for routing client requests to an appropriate surrogate server for the delivery of content. It consists of a collection of network elements to support request-routing for a single CDN. It directs client requests to the replica server "closest" to the client. However, the closest server may not be the best surrogate server for servicing the client request [22]. Hence, a request-routing system uses a set of metrics such as network proximity, client perceived latency, distance, and replica server load in an attempt to direct users to the closest surrogate that can best serve the request. The content selection and delivery techniques (i.e. full-site and partial-site) used by a CDN have a direct impact on the design of its request-routing system. If the full-site approach is used by a CDN, the request-routing system assists to direct the client requests to the surrogate servers as they hold all the outsourced content. On the other hand, if the partial-site approach is

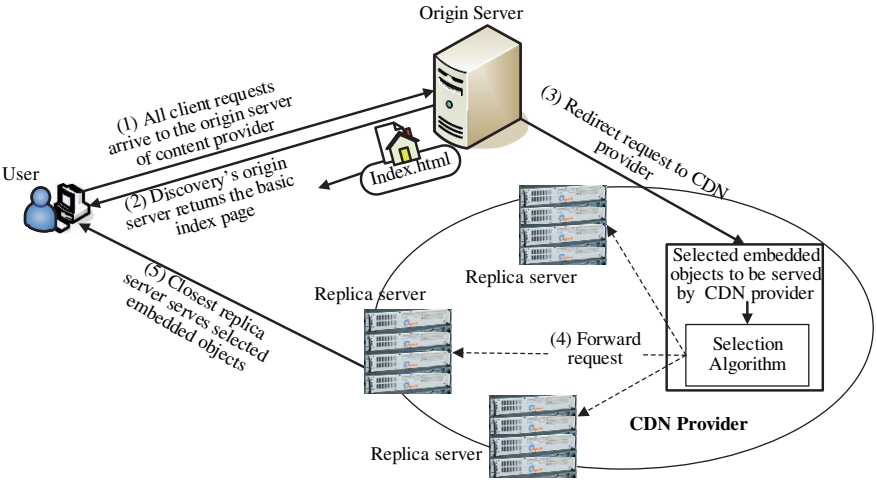

Fig. 2.10 Request-routing in a CDN environment

used, the request-routing system is designed in such a way that on receiving the client request, the origin server delivers the basic content while surrogate servers deliver the embedded objects. The request-routing system in a CDN has two parts: deployment of a request-routing algorithm and use of a request-routing mechanism [89]. A request-routing algorithm is invoked on receiving a client request. It specifies how to select an edge server in response to the given client request. On the other hand, a request-routing mechanism is a way to inform the client about the selection. Such a mechanism at first invokes a request-routing algorithm and then informs the client about the selection result it obtains.

Figure 2.10 provides a high-level view of the request-routing in a typical CDN environment. The interaction flows are: (1) the client requests content from the content provider by specifying its URL in the Web browser. Client's request is directed to its origin server; (2) when origin server receives a request, it makes a decision to provide only the basic content (e.g. index page of the Web site) that can be served from its origin server; (3) to serve the high bandwidth demanding and frequently asked content (e.g. embedded objects – fresh content, navigation bar, and banner advertisements), content provider's origin server redirects client's request to the CDN provider; (4) using the proprietary selection algorithm, the CDN provider selects the replica server which is "closest" to the client, in order to serve the requested embedded objects; (5) selected replica server gets the embedded objects from the origin server, serves the client requests and caches it for subsequent request servicing.

2.2.3.1 Request-Routing Algorithms

The algorithms invoked by the request-routing mechanisms can be *adaptive* or *non-adaptive* (Fig. 2.11). Adaptive algorithms consider the current system condition to

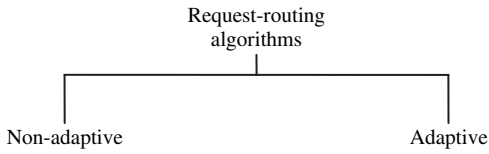

Fig. 2.11 Taxonomy of request-routing algorithms

select a cache server for content delivery. The current condition of the system is obtained by estimating some metrics like load on the replica servers or the congestion of selected network links. Non-adaptive request-routing algorithms use some heuristics for selecting a cache server rather than considering the current system condition. A non-adaptive algorithm is easy to implement, while the former is more complex. Complexity of adaptive algorithms arises from their ability to change behavior to cope with an enduring situation. A non-adaptive algorithm works efficiently when the assumptions made by the heuristics are met. On the other hand, an adaptive algorithm demonstrates high system robustness [100] in the face of events like flash crowds.

An example of the most common and simple non-adaptive request-routing algorithm is round-robin, which distributes all requests to the CDN cache servers and attempts to balance load among them [93]. It is assumed that all the cache servers have similar processing capability and that any of them can serve any client request. Such simple algorithms are efficient for clusters, where all the replica servers are located at the same place [69]. But the round-robin request-routing algorithm does not perform well for wide area distributed systems where the cache servers are located at distant places. In this case it does not consider the distance of the replica servers. Hence, client requests may be directed to more distant replica servers, which cause poor performance perceived by the users. Moreover, the aim of load balancing is not fully achieved since processing different requests can involve significantly different computational costs.

In another non-adaptive request-routing algorithm, all replica servers are ranked according to the predicted load on them. Such prediction is done based on the number of requests each of the servers has served so far. This algorithm takes client-server distance into account and client requests are directed to the replica servers in such a way that load is balanced among them. The assumption here is that the replica server load and the client-server distance are the most influencing factors for the efficiency of request processing [89]. Though it has been observed by Aggarwal et al. [9] that deploying this algorithm can perform well for request-routing, the client perceived performance may still be poor.

Several other interesting non-adaptive request-routing algorithms are implemented in the Cisco DistributedDirector [28]. One of these algorithms considers the percentage of client requests that each replica server receives. A server receiving more requests is assumed to be more powerful. Hence, client requests are directed to the more powerful servers to achieve better resource utilization. Another algorithm defines preference of one server over another in order to delegate the former to serve client requests. The DistributedDirector also supports random request distribution to replica servers. Furthermore, some other non-adaptive algorithms can be found which considers the client's geographic location to redirect requests to the nearby replica.

However, this algorithm suffers from the fact that client requests may be assigned to overloaded replica servers, which may degrade client perceived performance.

Karger et al. [55] have proposed a request-routing algorithm to adapt to hotspots. It calculates a hashing function h from a large space of identifiers, based on the URL of the content. This hashing function is used to route client requests efficiently to a logical ring consisting of cache servers with IDs from the same space. It is assumed that the cache server having the smallest ID larger than h is responsible for holding the referenced data. Hence, client requests are directed to it. Variations of this algorithm have been used in the context of intra-cluster caching [67, 68] and P2P file sharing systems [14].

Globule [76] uses an adaptive request-routing algorithm that selects the replica server closest to the clients in terms of network proximity [93]. The metric estimation in Globule is based on path length which is updated periodically. The metric estimation service used in globule is passive, which does not introduce any additional traffic to the network. However, Huffaker et al. [45] show that the distance metric estimation procedure is not very accurate.

Andrews et al. [10] and Ardiaz et al. [12] have proposed adaptive request-routing algorithms based on client-server latency. In this approach, either client access logs or passive server-side latency measurements are taken into account, and the algorithms decide to which replica server the client requests are to be sent. Hence, they redirect a client request to a replica which has recently reported the minimal latency to the client. These algorithms are efficient since they consider latency measurements. However, they require the maintenance of central database of measurements, which limits the scalability of systems on which these algorithms are deployed [89].

Cisco DistributedDirector [28] has implemented an adaptive request-routing algorithm. The request-routing algorithm deployed in this system takes into account a weighted combination of three metrics, namely – inter-AS distance, intra-AS distance, and end-to-end latency. Although this algorithm is flexible since it makes use of three metrics, the deployment of an agent in each replica server for metric measurement makes it complex and costly. Moreover, the active latency measurement techniques used by this algorithm introduce additional traffic to the Internet. Furthermore, the isolation of DistributedDirector component from the replica server makes it unable to probe the servers to obtain their load information.

Akamai [1, 29] uses a complex request-routing algorithm which is adaptive to flash crowds. It takes into consideration a number of metrics such as replica server load, the reliability of loads between the client and each of the replica servers, and the bandwidth that is currently available to a replica server. This algorithm is proprietary to Akamai and the technology details have not been revealed.

2.2.3.2 Request-Routing Mechanisms

Request-routing mechanisms inform the client about the selection of replica server generated by the request-routing algorithms. Request-routing mechanisms can be classified according to several criteria. In this section we classify them according

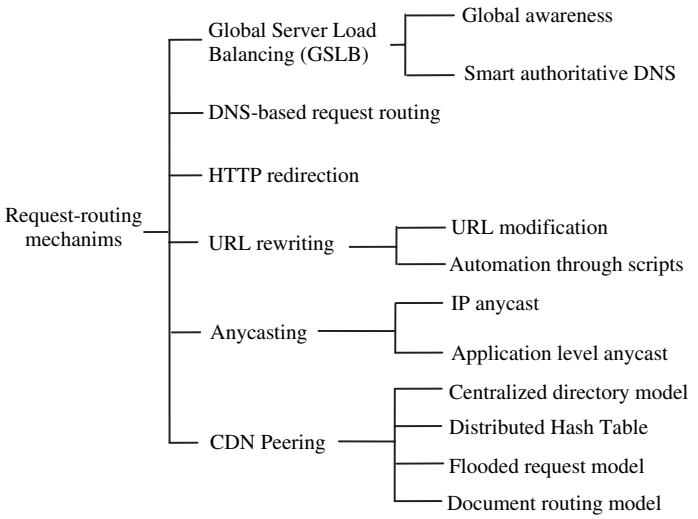

Fig. 2.12 Taxonomy of request-routing mechanisms

to request processing. As shown in Fig. 2.12, they can be classified as: *Global Server Load Balancing (GSLB)*, *DNS-based request-routing*, *HTTP redirection*, *URL rewriting*, *anycasting*, and *CDN peering*.

In GSLB [44] approach, service nodes, which serve content to the end users, consisting of a GSLB-enabled Web switch and a number of real Web servers are distributed in several locations around the world. Two new capabilities of the service nodes allow them to support global server load balancing. The first is *global awareness* and the second is *smart authoritative DNS* [44]. In local server load balancing, each service node is aware of the health and performance information of the Web servers directly attached to it. In GSLB, one service node is aware of the information in other service nodes and includes their virtual IP address in its list of servers. Hence, the Web switches making up each service node are globally aware and each knows the addresses of all the other service nodes. They also exchange performance information among the Web switches in GSLB configuration. To make use of such global awareness, the GSLB switches act as a smart authoritative DNS for certain domains. The advantage of GSLB is that since the service nodes are aware of each other, each GSLB switch can select the best surrogate server for any request. Thus, this approach facilitates choosing servers not only from the pool of locally connected real servers, but also the remote service nodes. Another significant advantage of GSLB is that the network administrator can add GSLB capability to the network without adding any additional networking devices. A disadvantage of GSLB is the manual configuration of the service nodes to enable them with GSLB capability.

In DNS-based request-routing approach, the content distribution services rely on the modified DNS servers to perform the mapping between a surrogate server's symbolic name and its numerical IP address. It is used for full-site content selection and

delivery. In DNS-based request-routing, a domain name has multiple IP addresses associated to it. When an end user's content request comes, the DNS server of the service provider returns the IP addresses of servers holding the replica of the requested object. The client's DNS resolver chooses a server among these. To decide, the resolver may issue probes to the servers and choose based on response times to these probes. It may also collect historical information from the clients based on previous access to these servers. Both full and partial-site CDN providers use DNS redirection. The performance and effectiveness of DNS-based request-routing has been examined in a number of recent studies [15, 41, 65, 86]. The advantage of this approach is the transparency as the services are referred to by means of their DNS names, and not their IP addresses. DNS-based approach is extremely popular because of its simplicity and independence from any actual replicated service. Since it is incorporated to the name resolution service it can be used by any Internet application [89]. In addition, the ubiquity of DNS as a directory service provides advantages during request-routing. The disadvantage of DNS-based request-routing is that, it increases network latency because of the increase in DNS lookup times. CDN administrators typically resolve this problem by splitting CDN DNS into two levels (low-level DNS and high-level DNS) for load distribution [58]. Another limitation is that DNS provides the IP address of the client's Local DNS (LDNS), rather than the client's IP address. Clients are assumed to be near to the LDNS. When DNS-based server selection is used to choose a nearby server, the decision is based on the name server's identity, not the client's. Thus, when clients and name servers are not proximal, the DNS-based approach may lead to poor decisions. Most significantly, DNS cannot be relied upon to control all incoming requests due to caching of DNS data at both the ISP and client level. Indeed, it can have control over as little as 5% of requests in many instances [20]. Furthermore, since clients do not access the actual domain names that serve their requests, it leads to the absence of any alternate server to fulfill client requests in case of failure of the target surrogate server. Thus, in order to remain responsive to changing network or server conditions, DNS-based schemes must avoid client-side caching or decisions.

HTTP redirection propagates information about replica server sets in HTTP headers. HTTP protocols allow a Web server to respond to a client request with a special message that tells the client to re-submit its request to another server. HTTP redirection can be used for both full-site and partial-site content selection and delivery. This mechanism can be used to build a special Web server, which accepts client requests, chooses replica servers for them and redirects clients to those servers. It requires changes to both Web servers and clients to process extra headers. The main advantage of this approach is flexibility and simplicity. Another advantage is that replication can be managed at fine granularity, since individual Web pages are considered as a granule [75]. The most significant disadvantage of HTTP redirection is the lack of transparency. Moreover, the overhead perceived through this approach is significant since it introduces extra message round-trip into request processing as well as over HTTP.

Though most CDN systems use a DNS based routing scheme, some systems use the URL rewriting or Navigation hyperlink. It is mainly used for partial-site

content selection and delivery where embedded objects are sent as a response to client requests. In this approach, the origin server redirects the clients to different surrogate servers by rewriting the dynamically generated pages' URL links. For example, with a Web page containing an HTML file and some embedded objects, the Web server would modify references to embedded objects so that the client could fetch them from the best surrogate server. To automate this process, CDNs provide special scripts that transparently parse Web page content and replace embedded URLs [58]. URL rewriting can be pro-active or reactive. In the pro-active URL rewriting, the URLs for embedded objects of the main HTML page are formulated before the content is loaded in the origin server. The reactive approach involves rewriting the embedded URLs of an HTML page when the client request reaches the origin server. The main advantage of URL rewriting is that the clients are not bound to a single surrogate server, because the rewritten URLs contain DNS names that point to a group of surrogate servers. Moreover, finer level of granularity can be achieved through this approach since embedded objects can be considered as granule. The disadvantages through this approach are the delay for URL-parsing and the possible bottleneck introduced by an in-path element. Another disadvantage is that content with modified reference to the nearby surrogate server rather than to the origin server is non-cacheable.

The anycasting approach can be divided into *IP anycasting* and *Application-level anycasting*. IP anycasting, proposed by Partridge et al. [73], assumes that the same IP address is assigned to a set of hosts and each IP router holds a path in its routing table to the host that is closest to this router. Thus, different IP routers have paths to different hosts with the same IP address. IP anycasting can be suitable for request-routing and service location. It targets network-wide replication of the servers over potentially heterogeneous platforms. A disadvantage of IP anycasting is that some parts of the IP address space is allocated for anycast address. Fei et al. [32] proposed an application level anycasting mechanism where the service consists of a set of *anycast resolvers*, which perform the *anycast domain names* to IP address mapping. Clients interact with the anycast resolvers by generating an anycast query. The resolver processes the query and replies with an anycast response. A metric database, associated with each anycast resolver contains performance data about replica servers. The performance is estimated based on the load and the request processing capability of the servers. The overhead of the performance measurement is kept at a manageable level. The performance data can be used in the selection of a server from a group, based on user-specified performance criteria. An advantage of application level anycasting is that better flexibility can be achieved through this approach. One disadvantage of this approach is that deploying the anycasting mechanism for request-routing requires changes to the servers as well as to the clients. Hence, it may lead to increased cost considering possibly large number of servers and clients.

Peer-to-peer content networks are formed by symmetrical connections between host computers. Peered CDNs deliver content on each other's behalf. Thus, a CDN could expand its reach to a larger client population by using partnered CDN servers and their nearby forward proxies. A content provider usually has contracts with

only one CDN and each CDN contacts other peer CDNs on the content provider's behalf [74]. Peering CDNs are more fault-tolerant as the necessary information retrieval network can be developed on the peering members themselves instead of relying on a dedicated infrastructure like traditional CDNs. To locate content in CDN peering, a *centralized directory model*, *Distributed Hash Table (DHT)*, *flooded request model*, or *document routing model* can be used [44, 66].

In a centralized directory model, peers contact a centralized directory where all the peers publish content that they want to share with others. When the directory receives a request it responses with the information of the peer that holds the requested content. When more than one peer matches the request, the best peer is selected based on metrics such as network proximity, highest bandwidth, least congestion and highest capacity. On receiving the response from the directory, the requesting peer contacts the peer that it has been referred to for content retrieval. The drawback of this approach is that, the centralized directory is subject to a single point of failure. Moreover, the scalability of a system based on a centralized directory is limited to the capacity of the directory. Archi [31], WAIS [52] are the examples of centralized directory systems for retrieving FTP files located on various systems. In systems using DHTs, peers are indexed through hashing keys within a distributed system. Then a peer holding the desired content can be found through applying queries and lookup functions [43]. Example of a protocol using DHT is Chord [92]. The advantage of this approach is the ability to perform load balancing by offloading excess loads to the less-loaded peers [18]. In the flooded request model, a request from a peer is broadcast to the peers directly connected to it. These peers in turn forward the messages to other peers directly connected to them. This process continues until the request is answered or some broadcast limit is reached. The drawback of this approach is that it generates unnecessary network traffic and hence, it requires enormous bandwidth. Thus, it suffers from scalability problem and it limits the size of the network [44]. Gnutella [8, 25] is the example of a system using the flooded request model. In document routing model an authoritative peer is asked for referral to get the requested content. Each peer in the model is helpful, though they partially complete the referral information [44]. In this approach, each peer is responsible for a range of file IDs. When a peer wants to get some file, it sends a request a request containing the file ID. The request is forwarded to the peer whose ID is most similar to the file ID. Once the file is located, it is transferred to the requesting peer. The main advantage of this approach is that it can complete a comprehensive search within a bounded *O(log n)* number of steps. Moreover, it shows good performance and is scalable enough to grow significantly large.

2.2.4 Performance Measurement

Performance measurement of a CDN is done to measure its ability to serve the customers with the desired content and/or service. Typically five key metrics are used by the content providers to evaluate the performance of a CDN [30, 37, 58]. Those are:

are reproducible and easy to analyze. A wide range of network simulators [4, 7] are available which can be used to simulate a CDN to measure its performance. Moreover, there are also some specific CDN simulation systems [2, 7, 23, 54, 100] that allow a (closely) realistic approach for the research community and CDN developers to measure performance and experiment their policies. However, the results obtained from a simulation may be misleading if a CDN simulation system does not take into account several critical factors such as the bottlenecks that are likely to occur in a network, the number of traversed nodes etc., considering the TCP/IP network infrastructure.

2.3 Mapping of the Taxonomy to Representative CDNs

In this section, we provide the categorization and mapping of our taxonomy to a few representative CDNs that have been surveyed in Chap. 1 of this book. We also present the perceived insights and a critical evaluation of the existing systems while classifying them. Our analysis of the CDNs based on the taxonomy also examines the validity and applicability of the taxonomy.

2.3.1 CDN Composition Taxonomy Mapping

Table 2.1 shows the annotation of the representative CDNs based on the CDN composition taxonomy. As shown in the table, the majority of the existing CDNs use overlay approach for CDN organization, while some use network approach or both. The use of both overlay and network approaches is common among commercial CDNs such as Akamai and Mirror Image. When a CDN provider uses a combination of these two approaches for CDN formation, a network element can be used to redirect HTTP requests to a nearby application-specific surrogate server.

Academic CDNs are built using P2P techniques, following an overlay approach. However, each of them differs in the way the overlay is built and deployed. For example, CoDeeN overlay consists of deployed "open" proxies, whereas Coral overlay (consisting of cooperative HTTP proxies and a network of DNS servers) is built relying on an underlying indexing infrastructure, and Globule overlay is composed of the end user nodes.

In an overlay approach, the following relationships are common – client-to-surrogate-to-origin server and network element-to-caching proxy. Inter-proxy relationship is also common among the CDNs, which supports inter-cache interaction. When using network approach, CDNs rely on the interaction of network elements for providing services through deploying request-routing logic to the network elements based on predefined policies. The overlay approach is preferred over the network approach because of the scope for new services integration and simplified

Table 2.1 CDN composition taxonomy mapping

CDN Name and Type		CDN Organization	Servers	Relationships	Interaction Protocols	Content/Service Types
Commercial CDNs	Akamai	Network and overlay approach	Origin and replica servers	Client-to-surrogate-to-origin server, Network element-to-caching proxy, Inter-proxy	Network elements interaction, inter-cache interaction	Static content, dynamic content, streaming media, and services (network monitoring, geographic targeting)
	Edge Stream	Network approach	N/A	N/A	Network elements interaction	Video streaming, video hosting services
	Limelight Networks	Overlay approach	Origin and replica servers	Client-to-surrogate-to-origin server, Network element-to-caching proxy	Network elements interaction	Static content, streaming media
	Mirror Image	Network and Overlay approach	Origin and replica servers	Client-to-surrogate-to-origin server, Network element-to-caching proxy	Network elements interaction	Static content, streaming media, Web computing and reporting services
Academic CDNs	CoDeeN	Overlay approach with "open" proxies	Origin and replica/proxy (forward, reverse, redirector) servers	Client-to-surrogate-to-origin server, Network element-to-caching proxy, inter-proxy	Network elements interaction, inter-cache interaction	*Participating* users receive better performance to *most* sites; only provides static content

Table 2.1 (continued)

CDN Name and Type	CDN Organization	Servers	Relationships	Interaction Protocols	Content/Service Types
Coral	Overlay approach with an underlying indexing infrastructure	Origin and replica/ (cooperative) proxy cache servers	Client-to-surrogate-to-origin server, Network element-to-caching proxy, inter-proxy	Network elements interaction, inter-cache interaction	Most *users* receive better performance to *participating* sites; only provides static content
Globule	Overlay approach with end user nodes	Origin, replica, backup and/or redirector servers	Client-to-surrogate-to-origin server, Network element-to-caching proxy, inter-node	Network elements interaction, inter-cache interaction	A Web site's performance and availability is improved; provides static content and monitoring services

management of underlying network infrastructure. Offering a new service in overlay approach is as simple as distributing new code to CDN servers [61].

CDNs use origin and replica servers to perform content delivery. Most of the replica servers are used as Web servers for serving Web content. Some CDNs such as Akamai, EdgeStream, Limelight Networks, and Mirror Image use their replica servers as media servers for delivering streaming media and video hosting services. Replica servers can also be used for providing services like caching, large file transfer, reporting, and DNS services. In the academic CDN domain, proxy/replica servers can be configured for different purposes. For example, each CoDeeN node is capable of acting as a forward, a reverse, and a redirection proxy; Coral proxies are cooperative; and Globule node can play the role of an origin, replica, backup, and/or replica server.

From Table 2.1, it can also be seen that most of the CDNs are dedicated to provide particular content, since variation of services and content requires the CDNs to adopt application-specific characteristics, architectures and technologies. Most of them provide static content, while only some of them provide streaming media, broadcasting, and other services. While the main business goal of commercial CDNs is to gain profit through content and/or service delivery, the goal of academic CDNs differs from each other. As for instance, CoDeeN provides static content with the goal of providing *participating* users better performance to *most* Web sites; Coral aims to provide most *users* better performance to *participating* Web sites; and Globule targets to improve a Web site's performance, availability and resistance (to a certain extent) to flash crowds and the Slashdot effects.

2.3.2 Content Distribution and Management Taxonomy Mapping

The mapping of the content distribution and management taxonomy to the representative CDNs is shown in Table 2.2.

Most of the CDNs support partial-site content delivery, while both full and partial-site content delivery is also possible. CDN providers prefer to support partial-site content delivery because it reduces load on the origin server and on the site's content generation infrastructure. Moreover, due to the infrequent change of embedded content, partial-site approach performs better than the full-site content delivery approach. Only few CDNs – Akamai, Mirror Image and Coral to be specific, are found to support clustering of contents. The content distribution infrastructure of other CDNs does not reveal any information whether other CDNs use any scheme for content clustering. Akamai and Coral cluster content based on users' sessions. This approach is beneficial because it helps to determine both the groups of users with similar browsing patterns and the groups of pages having related content. The only CDN to use the URL-based content clustering is Mirror Image. But URL-based approach is not popular because it suffers from the complexity involved to deploy them.

From the table it is clear that most of the representative CDNs with extensive geographical coverage follow the multi-ISP approach to place numerous number of surrogate servers at many global ISP POPs. Commercial CDNs such as Akamai, Limelight Networks, Mirror Image, and academic CDNs such as Coral [34] and CoDeeN use multi-ISP approach. The single-ISP approach suffers from the distant placement of the surrogates with respect to the locality of the end users. However, the setup cost, administrative overhead, and complexity associated with deploying and managing of the system in multi-ISP approach is higher. An exception to this can be found for sites with high traffic volumes. Multi-ISP approach performs better in this context since single-ISP approach is suitable only for sites with low-to-medium traffic volumes [95].

Content outsourcing of the commercial CDNs mostly use non-cooperative pull-based approach because of its simplicity enabled by the use of DNS redirection or URL rewriting. Cooperative push-based approach is still theoretical and none of the existing CDNs supports it. Cooperative pull-based approach involves complex technologies (e.g. DHT) as compared to the non-cooperative approach and it is used by the academic CDNs following P2P architecture [71]. Moreover, it imposes a large communication overhead (in terms of number of messages exchanged) when the number of clients is large. It also does not offer high fidelity when the content changes rapidly or when the coherency requirements are stringent.

From Table 2.2 it is also evident that representative commercial and academic CDNs with large geographic coverage, use inter-cluster (and a combination of inter and intra-cluster) caching. CDNs mainly use on-demand update as their cache update policy. Only Coral uses invalidation for updating caches since it delivers static content which changes very infrequently. Globule follows an adaptive cache update policy to dynamically choose between different cache consistency enforcement techniques. Of all the cache update policies, periodic update has the greatest reach since the caches are updated in a regular fashion. Thus, it has the potential to be most effective in ensuring cache content consistency. Update propagation and invalidation are not generally applicable as steady-state control mechanisms, and they can cause control traffic to consume bandwidth and processor resources that could otherwise be used for serving content [41]. Content providers themselves may administer to deploy specific caching mechanisms or heuristics for cache update. Distributing particular caching mechanism is simpler to administer but it has limited effects. On the other hand, cache heuristics are a good CDN feature for content providers who do not want to develop own caching mechanisms. However, heuristics will not deliver the same results as well-planned policy controls [41].

2.3.3 Request-Routing Taxonomy Mapping

Table 2.3 maps the request-routing taxonomy to the representative CDNs. It can be observed from the table that DNS-based mechanisms are very popular for request-routing. The main reason of this popularity is its simplicity and the ubiquity of

Table 2.2 Content distribution and management taxonomy mapping

CDN Name	Content Selection and Delivery	Surrogate Placement	Content Outsourcing	Cache Organization
Akamai	Content selection • Full and partial-site delivery Content Clustering • Users' sessions based	Multi-ISP approach; Hotspot placement by allocating more servers to sites experiencing high load	Non-cooperative pull-based	Caching technique • Intra and inter-cluster caching Cache update • Update propagation • On-demand
Edge Stream	Content selection • Partial-site delivery Content Clustering N/A	Single-ISP approach	Non-cooperative pull-based	Caching technique • Inter-cluster caching Cache update N/A
Limelight Networks	Content selection • Partial-site delivery Content Clustering N/A	Multi-ISP approach	Non-cooperative pull-based	Caching technique • Intra-cluster caching Cache update • On-demand
Mirror Image	Content selection • Partial-site delivery Content Clustering • URL based	Multi-ISP approach; Center placement following a concentrated "Superstore" architecture	Non-cooperative pull-based	Caching technique • Intra-cluster caching Cache update • On-demand

Table 2.2 (continued)

CDN Name	Content Selection and Delivery	Surrogate Placement	Content Outsourcing	Cache Organization
CoDeeN	Content selection • Partial-site delivery Content Clustering N/A	Multi-ISP approach; Topology-informed replica placement	Cooperative pull-based	Caching technique • Intra and inter-cluster caching Cache update • On-demand
Coral	Content selection • Full and partial-site delivery Content Clustering • Users' sessions based	Multi-ISP approach; Tree-based replica placement	Cooperative pull-based	Caching technique • Intra and inter-cluster caching Cache update • Cache invalidation
Globule	Content selection • Full and partial-site delivery Content Clustering N/A	Single-ISP approach; Best replica placement strategy is dynamically selected through regular evaluation of different strategies	Cooperative pull-based	Caching technique • Intra and inter-cluster caching Cache update • Adaptive cache update

DNS as a directory service. DNS-based mechanisms mainly consist of using a specialized DNS server in the name resolution process. Among other request-routing mechanisms, HTTP redirection is also highly used in the CDNs because of the finer level of granularity on the cost of introducing an explicit binding between a client and a replica server. Flexibility and simplicity are other reasons of using HTTP redirection for request-routing in CDNs. Some CDNs such as Mirror Image uses GSLB for request-routing. It is advantageous since less effort is required to add GSLB capability to the network without adding any additional network devices. Among the academic CDNs, Coral exploits overlay routing techniques, where indexing abstraction for request-routing is done using DSHT. Thus, it makes use of P2P mechanism for request redirection. As we mentioned earlier, the request-routing system of a CDN is composed of a request-routing algorithm and a request-routing mechanism. The request-routing algorithms used by the CDNs are proprietary in nature. The technology details of most of them have not been revealed. Our analysis of the existing CDNs indicates that Akamai and Globule use adaptive request-routing algorithm for their request-routing system. Akamai's adaptive (to flash crowds) request-routing takes into account server load and various network metrics; whereas Globule measures only the number of AS that a request needs to pass through. In case of CoDeeN, the request-routing algorithm takes into account request locality, system load, reliability, and proximity information. On the other hand, Coral's request-routing algorithm improves locality by exploiting on-the-fly network measurement and storing topology hints in order to increase the possibility for the clients to discover nearby DNS servers.

Table 2.3 Request-routing taxonomy mapping

CDN Name	Request-routing Technique
Akamai	• Adaptive request-routing algorithms which takes into account server load and various network metrics • Combination of DNS-based request-routing and URL rewriting
EdgeStream	HTTP redirection
Limelight Networks	DNS-based request-routing
Mirror Image	Global Server Load Balancing (GSLB) • Global awareness • Smart authoritative DNS
CoDeeN	• Request-routing algorithm takes into account request locality, system load, reliability, and proximity information. • HTTP redirection.
Coral	• Request-routing algorithms with improved locality by exploiting on-the-fly network measurement and storing topology hints • DNS-based request-routing
Globule	• Adaptive request-routing algorithms considering AS-based proximity • Single-tier DNS-based request-routing