Preface

The emergence of Web as a ubiquitous media for sharing content and services has led to the rapid growth of the Internet. At the same time, the number of users accessing Web-based content and services are growing exponentially. This has placed a heavy demand on Internet bandwidth and Web systems hosting content and application services. As a result, many Web sites are unable to manage this demand and offer their services in a timely manner.

Content Delivery Networks (CDNs) have emerged to overcome these limitations by offering infrastructure and mechanisms to deliver content and services in a scalable manner, and enhancing users' Web experience. Applications of CDNs can also be found in many communities, such as academic institutions, advertising media and Internet advertisement companies, data centers, Internet Service Providers (ISPs), online music retailers, mobile operators, consumer electronics manufacturers, and other carrier companies. Along with the proliferation, formation, and consolidation of the CDN landscape, new forms of Internet content and services are coming into picture while distribution and management of content is introducing new challenges in this domain. This raises new issues in the architecture, design and implementation of CDNs. The technological trends in this domain need to be explored in order to provide an exclusive research roadmap to the CDN community.

The book, entitled *"Content Delivery Networks"* offers the state-of-the-art CDN concepts, principles, characteristics, applications, platforms, design tips and hints, modeling, simulation, engineering approaches, and recent technological developments. The book builds on academic and industrial research and developments, and case studies that are being carried out at many different institutions around the world. In addition, the book identifies potential research directions and technologies that drive future innovations. We expect the book to serve as a valuable reference for larger audience such as systems architects, practitioners, product developers, researchers, and graduate level students.

Overview and scope of the book: This book will enable the readers to understand the basics, to identify the underlying technology, to summarize their knowledge on concepts, ideas, principles and various paradigms which span on broad CDNs areas. Therefore, aspects of CDNs in terms of basics, design process, practice, techniques, performances, platforms, applications, and experimental results have been presented in a proper order. Fundamental methods, initiatives, significant research results, as well as references for further study have also been provided. Comparison of different design and development approaches are described at the appropriate places so that new researchers as well as advanced practitioners can use the CDNs evaluation as a research roadmap. All the contributions have been reviewed, edited, processed, and placed in the appropriate order to maintain consistency so that any reader irrespective of their level of knowledge and technological skills in CDNs would get the most out of it. The book is organized into three parts, namely, Part I: CDN Fundamentals; Part II: CDN Modeling and Performance; and Part III: Advanced CDN Platforms and Applications. The organization ensures the smooth flow of material as successive chapters build on prior ones. In particular, the topics of the book are the following:

- CDN basics and state of the art
- A taxonomy for categorizing CDN technologies
- Dynamic, scalable and efficient content replication techniques
- Content distribution and management
- Integrated use of replication with caching and its performance
- Request redirection for dynamic content
- Economics-informed modeling of CDNs
- Pricing schemes and CDN business models
- Mathematical modeling for resource allocation and management
- CDN performance
- CDN internetworking scenarios, architecture, and methodology
- Media streaming
- Dynamic CDNs and QoS-based adaptive content delivery
- Mobile dynamic CDNs
- Applications: live and on-demand video services, content delivery for community networks

Part I of the book focuses on the basic ideas, techniques, and current practices in CDNs. In Chap. 1, Pathan et al. provide an introduction to CDNs and their origins, evolution, and the start-of-the-art. This chapter defines CDNs and related terminologies, identifies its uniqueness when compared to related distributed computing paradigms, provides insights for CDNs, and presents authors' visions for future technological evolutions in the CDN domain. As there exist a wide range of studies covering different aspects of CDNs such as content distribution, replication, caching, and Web server placement, in Chap. 2, Pathan and Buyya present a comprehensive taxonomy of CDNs with a broad coverage of applications, features, and implementation techniques. In Chap. 3, Chen highlights on the need of a dynamic, scalable, and efficient replication technique for CDNs. In this context, the author presents algorithms for dynamic and self-organized replica placements respecting client QoS and server capacity constraints. In Chap. 4, Cardellini et al. explore the issues of content delivery through CDNs, with a special focus on the delivery of dynamically generated and personalized content. To analyze and model simulations of various caching scheme, along with the integrated use of caching and replication, in Chap. 5, Stamos et al. present related design methodology and share implementation experiences, while cover various topics related to Web caching in a CDN simulation framework. In Chap. 6, Ranjan describes request redirection techniques in CDNs and presents a proof-of-the-concept implementation of a technique called WARD to assist the redirection of dynamic content.

Part II of this book focuses on the economic and mathematical modeling of CDNs and their performance. Building on game theory, in Chap. 7, Christin et al. present a cost-based model for agents participating in a CDN overlay and analyze incentives in link establishments in CDNs. In Chap. 8, Hosanagar discusses the economics of content delivery market and provides a model to capture content providers' value from CDN services and uses that to discuss pricing policies. In Chap. 9, Bektaş: and Ouveysi demonstrate how a variety of resource management and allocation problems in CDN domain can be formulated in terms of mathematical models. Sitaraman et al. present global overlay routing supported by Akamai along with its performance and availability benefits in Chap. 10.

Part III, the final part of the book, focuses on advanced CDN platforms and applications with wide appeal. In Chap. 11, Yoshida describes FCAN, a dynamic CDN network to alleviate flash crowds. Fortino et al. presents a CDN-based architecture supporting the collaborative playback service in Chap. 12, by describing the Hierarchical COoperative COntrol Protocol (HCOCOP) which enables the shared media streaming control in collaborative playback sessions. In Chap. 13, Czyrnek et al. address the key aspects of the multimedia CDN design based on the presentation of iTVP, a platform which is built for IP-based delivery of multimedia content on a country-wide scale to a large number of concurrent users. The information dissemination techniques in mobile CDNs, along with related challenges and current status are presented in Chap. 14 by Loulloudes et al. In Chap. 15, Plagemann et al. discuss the infrastructures for community networks based on CDNs. Finally, in the last chapter of the book, Pathan et al. present different models for internetworking between CDNs and identify the challenges in realizing them.

Acknowledgements: The book came into light due to the direct and indirect involvement of many researchers, academicians, developers, designers, and industry practitioners. Therefore, we acknowledge and thank the contributing authors, research institutions, and companies whose papers, reports, articles, notes, Web sites, study materials have been referred to in this book. Furthermore, many of the authors have acknowledged their respective funding agencies and co-researchers, who made significant influence in carrying out research. Finally, we offer our special appreciation to Springer and its publishing editor, Christoph Baumann, for helping us to bring this book out in record time.

Prior technical sources are acknowledged citing them at appropriate places in the book. In case of any error we would like to receive feedback so that it could be taken into consideration in the next edition.

We hope that this book will serve as a valuable text for students especially at graduate level and reference for researchers and practitioners working in the CDNs and its emerging consumer applications.

The University of Melbourne, Australia *Rajkumar Buyya*
The University of Melbourne, Australia *Mukaddim Pathan*
Aristotle University of Thessaloniki, Greece *Athena Vakali*

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:

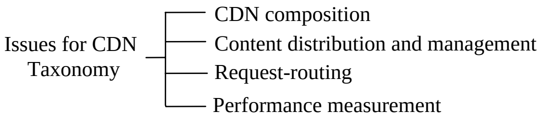
- A taxonomy of CDNs with a complete coverage of this field to provide a comprehensive account of applications, features, and implementation techniques. The main aim of the taxonomy, therefore, is to explore the functional and nonfunctional features of CDNs and to provide a basis for categorizing the 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.

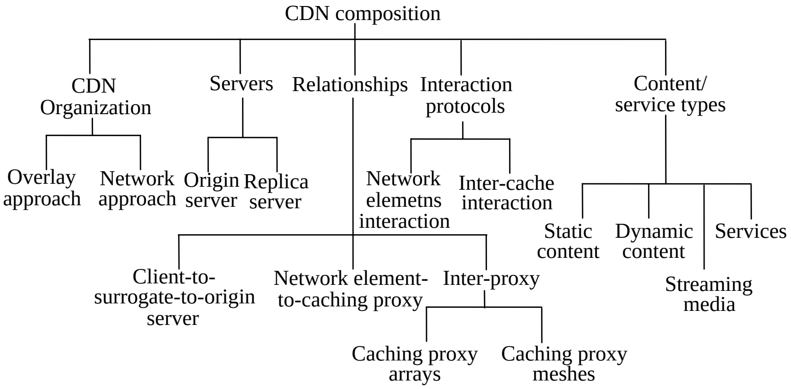


Fig. 2.2 CDN composition taxonomy

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

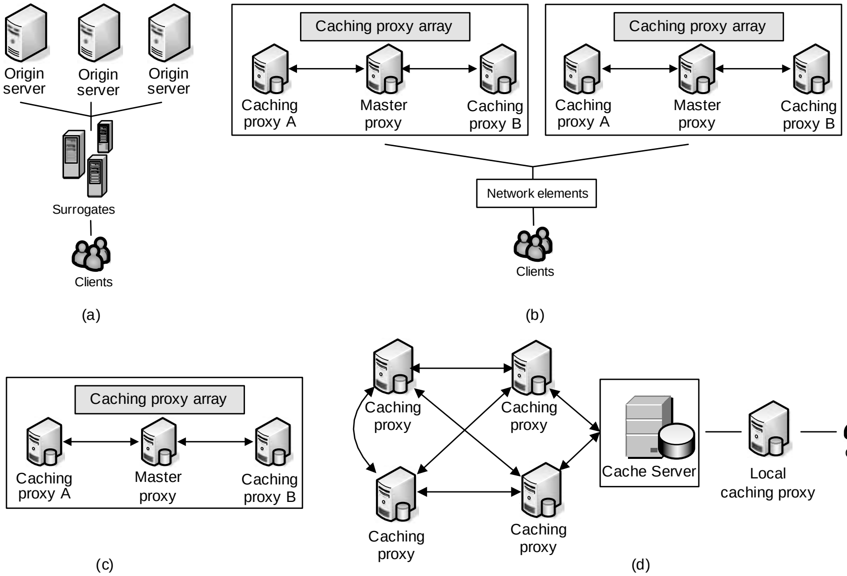


Fig. 2.3 Relationships: (a) Client-to-surrogate-to-origin server; (b) Network element-to-caching proxy; (c) Caching proxy arrays; (d) Caching proxy meshes

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

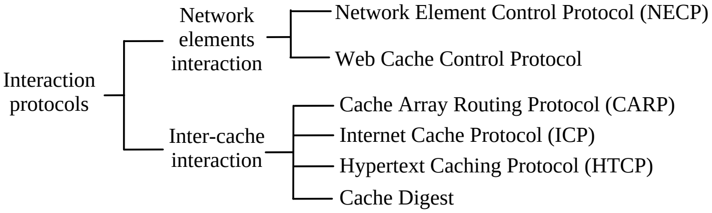

Interaction protocols
Network elements interaction
Network Element Control Protocol (NECP)
Web Cache Control Protocol
Inter-cache interaction
Cache Array Routing Protocol (CARP)
Internet Cache Protocol (ICP)
Hypertext Caching Protocol (HTCP)
Cache Digest

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.

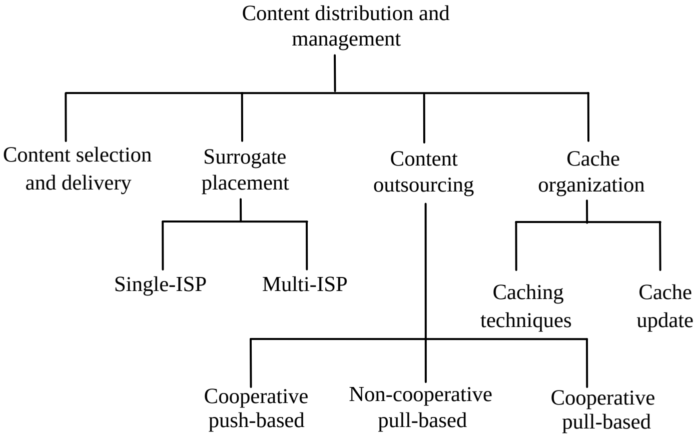


Fig. 2.5 Content distribution and management taxonomy

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

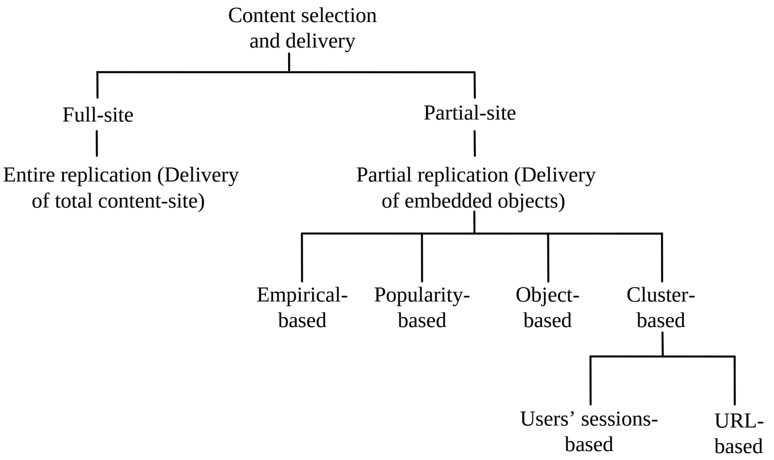

Content selection and delivery
Full-site
Partial-site
Entire replication (Delivery of total content-site)
Partial replication (Delivery of embedded objects)
Empirical-based
Popularity-based
Object-based
Cluster-based
Users' sessions-based
URL-based

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

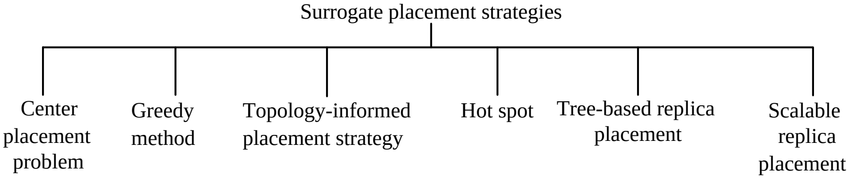

Surrogate placement strategies
Center placement problem
Greedy method
Topology-informed placement strategy
Hot spot
Tree-based replica placement
Scalable replica placement

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

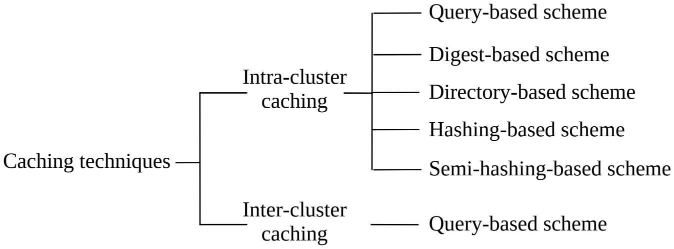


Fig. 2.8 Caching techniques taxonomy

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

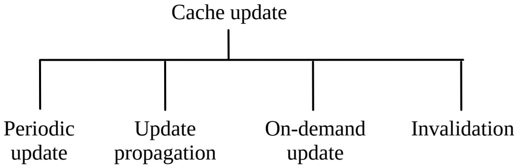


Fig. 2.9 Cache update taxonomy

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

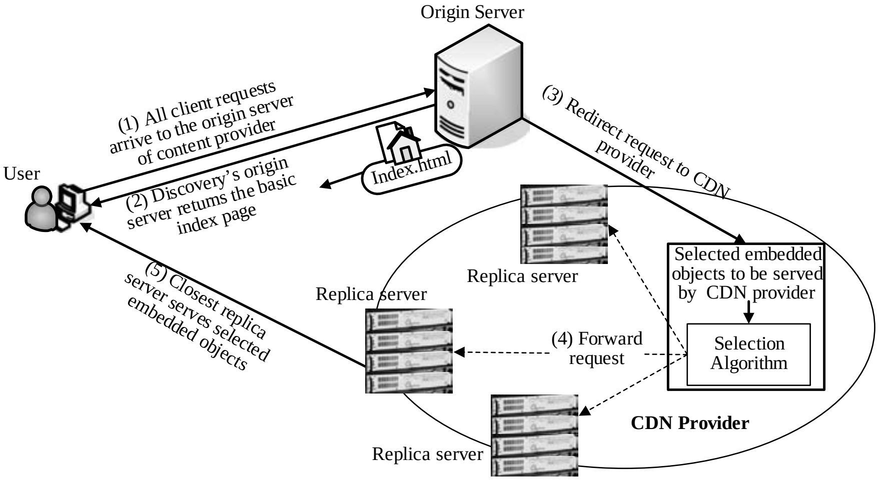

Origin Server
(1) All client requests arrive to the origin server of content provider
(2) Discovery's origin server returns the basic index page
Index.html
(3) Redirect request to CDN provider
User
(5) Closest replica server serves selected embedded objects
Replica server
Replica server
Replica server
Selected embedded objects to be served by CDN provider
Selection Algorithm
(4) Forward request
CDN Provider

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

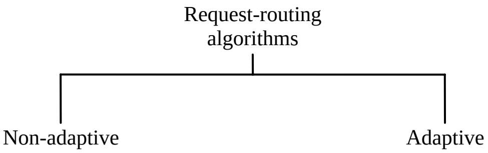


Fig. 2.11 Taxonomy of request-routing algorithms

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

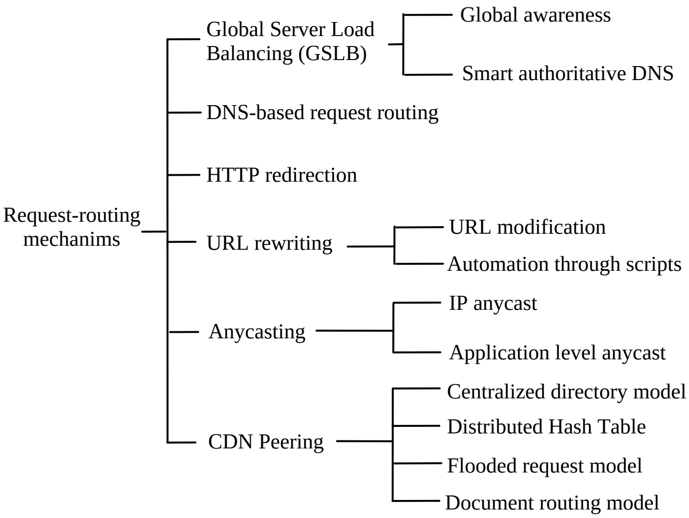


Fig. 2.12 Taxonomy of request-routing mechanisms

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:

- *Cache hit ratio:* It is defined as the ratio of the number of cached documents versus total documents requested. A high hit rate reflects that a CDN is using an effective cache technique to manage its caches.
- *Reserved bandwidth:* It is the measure of the bandwidth used by the origin server. It is measured in bytes and is retrieved from the origin server.
- *Latency:* It refers to the user perceived response time. Reduced latency indicates that less bandwidth is reserved by the origin server.
- *Surrogate server utilization:* It refers to the fraction of time during which the surrogate servers remain busy. This metric is used by the administrators to calculate CPU load, number of requests served and storage I/O usage.
- *Reliability:* Packet-loss measurements are used to determine the reliability of a CDN. High reliability indicates that a CDN incurs less packet loss and is always available to the clients.

Performance measurement can be accomplished based on internal performance measures as well as from the customer perspective. A CDN provider's own performance testing can be misleading, since it may perform well for a particular Web site and/or content, but poorly for others. To ensure reliable performance measurement, a CDN's performance can be measured by independent third-party such as Keynote Systems [3] or Giga Information Group [6]. The performance measurement taxonomy is shown in Fig. 2.13.

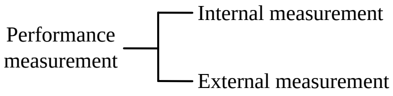


Fig. 2.13 Performance measurement taxonomy

2.2.4.1 Internal Measurement

CDN servers could be equipped with the ability to collect statistics in order to get an end-to-end measurement of its performance. In addition, deployment of probes (hardware and/or software) throughout the network and correlation of the information collected by probes with the cache and server logs can be used to measure the end-to-end performance.

2.2.4.2 External Measurement

In addition to internal performance measurement, external measurement of performance by an independent third-party informs the CDN customers about the verified and guaranteed performance. This process is efficient since the independent performance-measuring companies support benchmarking networks of strategically located measurement computers connected through major Internet backbones in several cities. These computers measure how a particular Web site performs from the end user's perspective, considering service performance metrics in critical areas [95].

2.2.4.3 Network Statistics Acquisition for Performance Measurement

For internal and external performance measurement, different network statistics acquisition techniques are deployed based on several parameters. Such techniques may involve network probing, traffic monitoring, and feedback from surrogates. Typical parameters in the network statistics acquisition process include geographical proximity, network proximity, latency, server load, and server performance as a whole. Figure 2.14 presents the mechanisms used by the CDNs to perform network statistics acquisition.

Network probing is a measurement technique where the possible requesting entities are probed in order to determine one or more metrics from each surrogate or a set of surrogates. Network probing can be used for P2P-based cooperative CDNs where the surrogate servers are not controlled by a single CDN provider. Example of such probing technique is an ICMP ECHO message that is sent periodically from a surrogate or a set of surrogates to a potential requesting entity. Active probing techniques are sometimes not suitable and limited for some reasons. It introduces additional network latency which may be significant for small Web requests. Moreover, performing several probes to an entity often triggers intrusion-detection alerts, resulting in abuse complaints [35]. Probing sometimes may lead to an inaccurate metric as ICMP traffic can be ignored or reprioritized due to concerns of Distributed Denial of Service (DDoS) attacks. A distributed anycasting system by Freedman et al. [35] has shown that ICMP probes and TCP probes to high random ports are often dropped by firewalls and flagged as unwanted port scans.

Traffic monitoring is a measurement technique where the traffic between the client and the surrogate is monitored to know the actual performance metrics. Once the client connects, the actual performance of the transfer is measured. This data is then fed back into the request-routing system. An example of such traffic monitoring is to watch the packet loss from a client to a surrogate or the user perceived response time (latency) by observing the TCP behavior. Latency is the simplest and mostly used distance metric, which can be estimated by monitoring the number of packets (i.e. traffic) traveled along the route between client and the surrogate. A metric estimation system such as IDMaps [35] measures and disseminates distance information on the global Internet in terms of latency and bandwidth. This system considers two types of distance information based on timeliness – load sensitive and “raw” (where distance information is obtained considering no load on the network). The estimation of these information is performed through traffic monitoring with an update frequency on the order of days, or if necessary, hours.

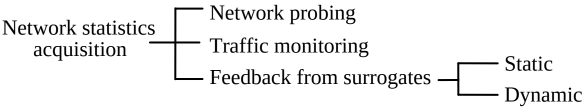


Fig. 2.14 Network statistics acquisition techniques

Feedback from surrogates can be obtained by periodically probing a surrogate by issuing application specific requests (e.g. HTTP) and taking related measures. Feedback information can also be obtained from agents that are deployed in the surrogates. These agents can communicate a variety of metrics about their nodes. Methods for obtaining feedback information can be static or dynamic. Static methods select a route to minimize the number of hops or to optimize other static parameters. Dynamic probing allows computing round-trip time or QoS parameters in "real time" [33].

Figure 2.15 shows the different metrics used by CDNs to measure the network and system performance. *Geographical proximity* is a measure of identifying a user's location within a certain region. It is often used to redirect all users within a certain region to the same Point of Presence (POP). The measurement of such network proximity is typically derived through probing of BGP routing tables. The end user perceived *latency* is a useful metric to select the suitable surrogate for that user. *Packet loss* information through a network path is a measurement metric that is used to select the path with lowest error rate. *Average bandwidth, startup time* and *frame rate* are the metrics used to select the best path for streaming media delivery. Server load state can be computed based on metrics such as CPU load, network interface load, active connection, and storage I/O load. This metric is used to select the server with the aggregated least load.

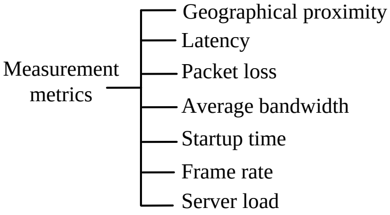


Fig. 2.15 Metrics used for measuring network and system performance

2.2.4.4 Performance Measurement through Simulation

Other than using internal and external performance measurement, researchers use simulation tools to measure a CDN's performance. Some researchers also experiment their CDN policies on real platforms such as PlanetLab [5]. The CDN simulators implemented in software are valuable tools for researchers to develop, test and diagnose a CDN's performance, since accessing real CDN traces and logs is not easy due to the proprietary nature of commercial CDNs. Such a simulation process is economical because of no involvement of dedicated hardware to carry out the experiments. Moreover, it is flexible because it is possible to simulate a link with any bandwidth and propagation delay and a router with any queue size and queue management technique. A simulated network environment is free of any uncontrollable factors such as unwanted external traffic, which the researchers may experience while running experiments in a real network. Hence, simulation results 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

2.3.4 Performance Measurement Taxonomy Mapping

Table 2.4 shows the mapping of different performance measurement techniques to representative CDNs.

Performance measurement of a CDN through some metric estimation measures its ability to serve the customers with the desired content and/or services. A CDN's performance should be evaluated in terms of cache hit ratio, bandwidth consumption, latency, surrogate server utilization, and reliability. In addition, other factors such as storage, communication overhead, and scalability can also be taken into account. The estimation of performance metrics gives an indication of system conditions and helps for efficient request-routing and load balancing in large systems. It is important to a content provider to conduct performance study of a CDN for selecting the most appropriate CDN provider. However, the proprietary nature of the CDN providers does not allow a content provider to conduct performance measurement on them.

From Table 2.4, we can see that performance measurement of a CDN is done through internal measurement technologies as well as from the customer perspective. It is evident that, most of the CDNs use internal measurement based on network probing, traffic monitoring or the like. Akamai uses proactive traffic monitoring and network probing for measuring performance. In the academic domain, CoDeeN has the local monitoring ability that examines a service's primary resources, such as free file descriptors/sockets, CPU cycles, and DNS resolver service; Coral has the ability

Table 2.4 Performance measurement taxonomy mapping

CDN Name	Performance Measurement
Akamai	Internal measurement ● Network probing ● Traffic monitoring (proactive) External measurement ● Performed by a third party (Giga Information group)
EdgeStream	Internal measurement ● Traffic monitoring through Real Time Performance Monitoring Service (RPMS)
Limelight Networks	N/A
Mirror Image	Internal measurement ● Network probing ● Traffic monitoring and reporting
CoDeeN	Internal measurement ● Local traffic and system monitoring
Coral	Internal measurement ● Traffic monitoring ● Liveness checking of a proxy via UDP RPC
Globule	Internal measurement ● Traffic monitoring ● Monitoring of server availability by the redirectors

to perform a proxy's liveness check (via UDP remote procedure call (RPC)) prior to replying to a DNS query; whereas, Globule has monitoring ability implemented in its redirector servers which checks for the availability of other servers.

External performance measurement of CDN providers is not common because most of the operating CDNs are commercial enterprises, which are not run transparently, and there are commercial advantages to keep the performance metrics and methodologies undisclosed. Despite this, some CDNs such as Akamai allow a third-party to perform external measurements.

2.4 Discussion

As stated at the beginning of this chapter, a full-fledged CDN development requires addressing additional issues (other than the four core issues considered for the taxonomy) such as fault tolerance, security, and ability for Web application hosting. In this section, we present a brief discussion on them and assist the readers to comprehend respective fields by providing referral to relevant research materials.

CDNs being a complex fabric of distributed network elements, failures can occur at many places. Following a concentrated architecture such as local clustering may improve fault-tolerance. However, it creates a single-point of failure, when the ISP connectivity to the cluster is lost. This problem can be solved through deploying Web clusters in distributed locations (mirroring) or using multiple ISPs to provide connectivity (multihoming). While clustering, mirroring, or multihoming addresses the CDN robustness issue to some extent, they introduce additional problems. Clustering suffers from scalability, while mirroring requires each mirror to carry entire load, and multihoming requires each connection to carry the entire traffic. Commercial CDNs follow their own proprietary approaches to provide fault-tolerance and scalability. As for instance, Akamai developed a distributed monitoring service that ensures that server or network failures are handled immediately without affecting the end users. Other than this, there are numerous solutions available in literature, some of which are widely used in real systems. Interested readers are referred to [46, 77, 85] to find descriptions on the explicit fault-tolerance solutions in wide-area systems such as CDNs.

Ensuring security in CDNs pose extra challenges in system development. There are security concerns at different levels of a CDN such as network, routers, DNS or Web clusters. One common security threat is the DDoS attack. The DDoS attack can be aimed at (a) consumption of scarce resources such as network bandwidth or CPU; (b) destruction or modification of configuration information; and (c) physical destruction or modifications of network components [82]. Security threats also include attacks which exploit software vulnerabilities (intrusion attacks) and protocol inconsistencies (protocol attacks). There exist many prior works addressing various wide-area security problems. Extensive coverage and documentation of security related solutions are available in the literature [40, 50, 51, 53, 56, 57, 82].

Nowadays, commercial CDNs such as Akamai provide usage-based content and application delivery solutions to the end users. Akamai Edge Computing

Infrastructure (ECI) [1], Active Cache [19], and ACDN [80] replicate the application code to the edge servers without replicating the application data itself. Rather, the data is kept in a centralized server. It enables the Web tier applications to extend to the CDN platform so that end user requests for application object would execute at the replica server rather than at the origin. However, this approach suffers from increased wide-area latency due to excessive data access and bottleneck due to the concentration on a centralized server. To overcome these limitations, Sivasubramanian et al. [87] propose an approach for replicating Web applications on-demand. This approach employs partial replication to replicate data units only to the servers who access them often. In another work, application specific edge service architecture [39] is presented where the application itself is responsible for its replication with the compromise of a weaker consistency model. For more information on hosting wide-area applications readers are referred to [88].

2.5 Summary and Conclusions

In this chapter, we have analyzed and categorized CDNs according to the functional and non-functional attributes. We have developed a comprehensive taxonomy for CDNs based on four issues: CDN composition, content distribution and management, request-routing, and performance measurement. We further built up taxonomies for each of these paradigms to classify the common trends, solutions, and techniques in content networking. Additionally, we identify three issues, namely, fault tolerance, security, and ability for Web application hosting as to introduce challenges in CDN development. Hereby, we provided pointers to related research work in this context. Our taxonomy provides a basis for comparison of existing CDNs. In doing so, we assist the readers to gain insights into the technology, services, strategies, and practices that are currently followed in this field. We have also performed a mapping of the taxonomy to representative commercial and academic CDN systems. Such a mapping provides a basis to realize an in-depth understanding of the state-of-the-art technologies in content distribution space, and to validate the applicability and accuracy of the taxonomy.

Recently, the CDN industry is getting consolidated as a result of acquisitions and/or mergers. During the preparation of this chapter, we have experienced significant changes in the content distribution landscape due to this consolidation. Consequently, content distribution, caching, and replication techniques are gaining more attention in order to meet up the new technical and infrastructure requirements for the next generation CDNs. This may lead to new issues in the design, architecture, and development of CDNs. Present trends in content networking domain indicate that better understanding and interpretation of the essential concepts in this area is necessary. Therefore, we hope that the comprehensive comparison framework based on our taxonomy, presented in this chapter, will not only serve as a tool to understand this complex area, but also will help to map the future research efforts in content networking.

Acknowledgements We would like to acknowledge the efforts of all the developers of the commercial and academic CDNs surveyed in this paper. We thank the anonymous reviewers for their insightful comments and suggestions that have improved the presentation and correctness of this chapter. We also thank our colleagues at the University of Melbourne – James Broberg, Marcos Assunção, and Charity Lourdes for sharing thoughts and for making incisive comments and suggestions on this chapter. We would like to express our gratitude to Athena Vakali (Aristotle University of Thessaloniki, Greece), George Pallis (The University of Cyprus, Cyprus), Carlo Mastroianni (ICAR-CNR, Italy), Giancarlo Fortino (Università della Calabria, Italy), Christian Vecchiola (University of Genova, Italy), and Vivek Pai (Princeton University, USA) for their visionary comments on various parts of the taxonomy. We are also thankful to Fahim Husain (Akamai Technologies, Inc., USA), William Good (Mirror Image Internet, Inc., USA), and Lisa Amini (IBM T. J. Watson Research Center, USA) for providing valuable research papers, technical reports, white papers, and data sheet while preparing the manuscript.

References

1. Akamai Technologies, 2007. www.akamai.com
2. CDNSim, A Content Distribution Network Simulator, 2007. http://oswinds.csd.auth.gr/∼cdnsim/
3. Keynote Systems—Web and Mobile Service Performance Testing Corporation, 2007. http://www.keynote.com/
4. Network simulators, 2007. http://www-nrg.ee.lbl.gov/kfall/netsims.html
5. PlanetLab Consortium, 2007. http://www.planet-lab.org/
6. The GigaWeb Corporation, 2007. http://www.gigaWeb.com/
7. The network simulator – ns-2, 2007. http://www.isi.edu/nsnam/ns/
8. Aberer, K. and Hauswirth, M. An overview on peer-to-peer information systems. In *Proc. of the Workshop on Distributed Data and Structures (WDAS)*, France, 2002.
9. Aggarwal, A. and Rabinovich, M. Performance of dynamic replication schemes for an Internet hosting service. Technical Report, HA6177000-981030-01-TM, AT&T Research Labs, Florham Park, NJ, USA, 1998.
10. Andrews, M., Shepherd, B., Srinivasan, A., Winkler, P., and Zane, F. Clustering and server selection using passive monitoring. In *Proc. of IEEE INFOCOM*, NY, USA, 2002.
11. Androutsellis-Theotokis, S. and Spinellis, D. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4), ACM Press, NY, USA, pp. 335–371, 2004.
12. Ardaiz, O., Freitag, F., and Navarro, L. Improving the service time of Web clients using server redirection. *ACM SIGMETRICS Performance Evaluation Review*, 29(2), ACM Press, NY, USA, pp. 39–44, 2001.
13. Bakiras, S. and Loukopoulos, T. Combining replica placement and caching techniques in content distribution networks. *Computer Communications*, 28(9), pp. 1062–1073, 2005.
14. Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Looking up data in P2P systems. *Communications of the ACM*, 46(2), ACM Press, NY, USA, pp. 43–48, 2003.
15. Barbir, A., Cain, B., Nair, R., and Spatscheck, O. Known content network request-routing mechanisms. Internet Engineering Task Force RFC 3568, 2003. www.ietf.org/rfc/rfc3568.txt
16. Bartal, Y. Probabilistic approximation of metric space and its algorithmic applications. In *Proc. of 37th Annual IEEE Symposium on Foundations of Computer Science*, 1996.
17. Brussee, R., Eertink, H., Huijsen, W., Hulsebosch, B., Rougoor, M., Teeuw, W., Wibbels, M., and Zandbelt, H. Content distribution network state of the art," Telematica Instituut, 2001.
18. Byers, J., Considine, J., and Mitzenmacher, J. Simple load balancing for distributed hash tables. In *Proc. of 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, pp. 31–35, 2003.
19. Cao, P., Zhang, J., and Beach, K. Active cache: Caching dynamic contents on the Web. In *Proc. of the Middleware Conference*, pp. 373–388, 1998.

20. Cardellini, V., Casalicchio, E., Colajanni, M., and Yu, P. S. The state of the art in locally distributed Web-server systems. *ACM Computing Surveys*, 34(2), ACM Press, NY, USA, pp. 263–311, 2002.
21. Chen, Y., Katz, R. H., and Kubiatowicz, J. D. Dynamic replica placement for scalable content delivery. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS 02)*, LNCS 2429, Springer-Verlag, pp. 306–318, 2002.
22. Chen, C. M., Ling, Y., Pang, M., Chen, W., Cai, S., Suwa, Y., Altintas, O. Scalable request-routing with next-neighbor load sharing in multi-server environments. In *Proc. of the 19th International Conference on Advanced Information Networking and Applications*, IEEE Computer Society, Washington, DC, USA, pp. 441–446, 2005.
23. Chen, Y., Qiu, L., Chen, W., Nguyen, L., and Katz, R. H. Efficient and adaptive Web replication using content clustering. *IEEE Journal on Selected Areas in Communications*, 21(6), pp. 979–994, 2003.
24. Cieslak, M., Foster, D., Tiwana, G., and Wilson, R. Web cache coordination protocol version 2. http://www.Web-cache.com/Writings/Internet-Drafts/draft-wilson-wrec-wccp-v2-00.txt
25. Clip2 Distributed Search Solutions, The Gnutella Protocol Specification v0.4. www.content-networking.com/papers/gnutella-protocol-04.pdf
26. Cooper, I., Melve, I., and Tomlinson, G. Internet Web replication and caching taxonomy. Internet Engineering Task Force RFC 3040, 2001. www.ietf.org/rfc/rfc3040.txt
27. Davison, B. D. Web caching and content delivery resources. http://www.Web-caching.com, 2007.
28. Delgadillo, K. Cisco DistributedDirector, Cisco Systems, Inc., 1997.
29. Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., and Weihl, B. Globally distributed content delivery. *IEEE Internet Computing*, pp. 50–58, 2002.
30. Douglis, F. and Kaashoek, M. F. Scalable Internet services. *IEEE Internet Computing*, 5(4), pp. 36–37, 2001.
31. Emtage, A. and Deutsch, P. Archie: an electronic directory service for the Internet. In *Proc. of the Winter Usenix Conference*, San Francisco, CA, USA, pp. 93–110, January 1992.
32. Fei, Z., Bhattacharjee, S., Zugura, E. W., and Ammar, M. H. A novel server selection technique for improving the response time of a replicated service. In *Proc. of IEEE INFOCOM*, San Francisco, California, USA, pp. 783–791, 1998.
33. Francis, P., Jamin, S., Jin, C., Jin, Y., Raz, D., Shavitt, Y., and Zhang, L. IDMaps: a global Internet host distance estimation service. *IEEE/ACM Transactions on Networking (TON)*, 9(5), ACM Press, NY, USA, pp. 525–540, 2001.
34. Freedman, M. J., Freudenthal, E., and Mazières, D. Democratizing content publication with Coral. In *Proc. of 1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, San Francisco, CA, USA, 2004.
35. Freedman, M. J., Lakshminarayanan, K., and Mazières, K. OASIS: anycast for any service. In *Proc. of 3rd Symposium of Networked Systems Design and Implementation (NSDI'06)*, Boston, MA, USA, 2006.
36. Fujita, N., Ishikawa, Y., Iwata, A., and Izmailov, R. Coarse-grain replica management strategies for dynamic replication of Web contents. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 45(1), pp. 19–34, 2004.
37. Gadde, S., Chase, J., and Rabinovich, M. Web caching and content distribution: a view from the interior. *Computer Communications*, 24(2), pp. 222–231, 2001.
38. Gadde, S., Rabinovich, M., and Chase, J. Reduce, reuse, recycle: an approach to building large Internet caches. In *Proc. of 6th Workshop on Hot Topics in Operating Systems*, pp. 93–98, 1997.
39. Gao, L., Dahlin, M., Nayate, A., Zheng, J., and Iyengar, A. Application specific data replication for edge services. In *Proc. of the Twelfth International World-Wide Web Conference*, Hungary, pp. 449–460, 2003.
40. Garg, A. and Reddy, A. L. N. Mitigating denial of service attacks using qos regulation. In *Proc. of International Workshop on Quality of Service (IWQoS)*, 2002.
41. Gayek, P., Nesbitt, R., Pearthree, H., Shaikh, A., and Snitzer, B. A Web content serving utility. *IBM Systems Journal*, 43(1), pp. 43–63, 2004.

42. Hamilton, M., Rousskov, A., and Wessels, D. Cache digest specification – version 5. 1998. http://www.squid-cache.org/CacheDigest/cache-digest-v5.txt
43. Harren, M., Hellerstein, J. M., Huebsch, R., Loo, B. T., Shenker, S., and Stoica, I. Complex queries in DHT-based peer-to-peer networks. In *Proc. of 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.
44. Hofmann, M. and Beaumont, L. R. *Content Networking: Architecture, Protocols, and Practice*. Morgan Kaufmann Publishers, San Francisco, CA, USA, pp. 129–134, 2005.
45. Huffaker, B., Fomenkov, M., Plummer, D. J., Moore, D., and Claffy, K. Distance metrics in the Internet. In *Proc. of IEEE International Telecommunications Symposium*, IEEE CS Press, Los Alamitos, CA, USA, 2002.
46. Jalote, P. *Fault Tolerance in Distributed Systems*. Prentice Hall, Englewood Cliffs, NJ, USA, 1994.
47. Jamin, S., Jin, C., Jin, Y., Raz, D., Shavitt, Y., and Zhang, L. On the placement of Internet instrumentation. In *Proc. of IEEE INFOCOM*, Tel-Aviv, Israel, pp. 295–304, 2000.
48. Jamin, S., Jin, C., Kure, A. R., Raz, D., and Shavitt, Y. Constrained mirror placement on the Internet. In *Proc. of IEEE INFOCOM*, Anchorage, Alaska, USA, 2001.
49. Johnson, K. L., Carr, J. F., Day, M. S., and Kaashoek, M. F. The measured performance of content distribution networks. *Computer Communications*, 24(2), pp. 202–206, 2001.
50. Jung, J., Krishnamurthy, B. and Rabinovich, M. Flash crowds and denial of service attacks: Characterization and implications for CDNs and Web sites. In *Proc. of the International World Wide Web Conference*, Hawaii, USA, pp. 252–262, 2002.
51. Jung, J., Paxson, V., Berger, A. W., and Balakrishnan, H. Fast portscan detection using sequential hypothesis testing. In *Proc. of IEEE Symposium on Security and Privacy*, Oakland, 2004.
52. Kahle, B. and Medlar, A. An information system for corporate users: wide area information servers. ConneXions—The Interoperability Report, 5(11), November 1991.
53. Kandula, S., Katabi, D., Jacob, M., and Berger, A. W. Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *Proc. of Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, 2005.
54. Kangasharju, J., Roberts, J., and Ross, K. W. Object replication strategies in content distribution networks. *Computer Communications*, 25(4), pp. 367–383, 2002.
55. Karger, D., Sherman, A., Berkheimer, A., Bogstad, B., Dhanidina, R., Iwamoto, K., Kim, B., Matkins, L., and Yerushalmi, Y. Web caching with consistent hashing. *Computer Networks*, 31(11–16), pp. 1203–1213, 1999.
56. Kargl, F., Maier, J., and Weber, M. Protecting Web servers from distributed denial of service attacks, In *Proc. of the International World Wide Web Conference*, pages 514–524, Hong Kong, 2001.
57. Kim, Y., Lau, W. C., Chuah, M. C., and Chao, H. J. Packetscore: Statistics based overload control against distributed denial-of-service attacks. In *Proc. of INFOCOM*, Hong Kong, 2004.
58. Krishnamurthy, B., Willis, C., and Zhang, Y. On the use and performance of content distribution network. In *Proc. of 1st International Internet Measurement Workshop*, ACM Press, pp. 169–182, 2001.
59. Krishnan, P., Raz, D., and Shavitt, Y. The cache location problem. *IEEE/ACM Transaction on Networking*, 8(5), 2000.
60. Kung, H. T. and Wu, C. H. Content networks: taxonomy and new approaches. *The Internet as a Large-Scale Complex System*, (Kihong Park and Walter Willinger eds.), Oxford University Press, 2002.
61. Lazar, I. and Terrill, W. Exploring content delivery networking. *IT Professional*, 3(4), pp. 47–49, 2001.
62. Lee, J. An End-User Perspective on File-Sharing Systems. *Communications of the ACM*, 46(2), ACM Press, NY, USA, pp. 49–53, 2003.
63. Li, B., Golin, M. J., Italiano, G. F., Xin, D., and Sohraby, K. On the optimal placement of Web proxies in the Internet. In *Proc. of IEEE INFOCOM*, NY, USA, pp. 1282–1290, 1999.

64. Ma, W. Y., Shen, B., and Brassil, J. T. Content services network: architecture and protocols. In *Proc. of 6th International Workshop on Web Caching and Content Distribution (IWCW6)*, 2001.
65. Mao, Z. M., Cranor, C. D., Douglis, F., Rabinovich, M., Spatscheck, O., and Wang, J. A precise and efficient evaluation of the proximity between Web clients and their Local DNS servers. In *Proc. of the USENIX 2002 Annual Technical Conference*, Monterey, CA, USA, pp. 229–242, 2002.
66. Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., and Xu, Z. Peer-to-peer computing. Technical Report, HP Laboratories, Palo Alto, CA, HPL-2002-57, 2002. www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf
67. Ni, J. and Tsang, D. H. K. Large scale cooperative caching and application-level multicast in multimedia content delivery networks. *IEEE Communications*, 43(5), pp. 98–105, 2005.
68. Ni, J., Tsang, D. H. K., Yeung, I. S. H., and Hei, X. Hierarchical content routing in large-scale multimedia content delivery network. In *Proc. of IEEE International Conference on Communications (ICC)*, pp. 854–859, 2003.
69. Pai, V. S., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W., Nahum, E. Locality-aware request distribution in cluster-based network servers. *ACM SIGPLAN Notices*, 33(11), ACM Press, NY, USA, pp. 205–216, 1998.
70. Pallis, G., Stamos, K., Vakali, A., Sidiropoulos, A., Katsaros, D., and Manolopoulos, Y. Replication-based on objects load under a content distribution network. In *Proc. of the 2nd International Workshop on Challenges in Web Information Retrieval and Integration (WIRI)*, Altanta, Georgia, USA, 2006.
71. Pallis, G. and Vakali, A. Insight and perspectives for content delivery networks. *Communications of the ACM*, 49(1), ACM Press, NY, USA, pp. 101–106, 2006.
72. Pallis, G., Vakali, A., Stamos, K., Sidiropoulos, A., Katsaros, D., and Manolopoulos, Y. A latency-based object placement approach in content distribution networks. In *Proc. of the 3rd Latin American Web Congress (La-Web 2005)*, IEEE Press, Buenos Aires, Argentina, pp. 140–147, 2005.
73. Partridge, C., Mendez, T., and Milliken, W. Host anycasting service. Internet Engineering Task Force RFC 1546, 1993. www.ietf.org/rfc/rfc1546.txt
74. Pathan, M., Broberg, J., Bubendorfer, K., Kim, K. H., and Buyya, R. An Architecture for Virtual Organization (VO)-Based Effective Peering of Content Delivery Networks, UPGRADE-CN'07. In *Proc. of the 16th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, Monterey, California, USA, 2007.
75. Peng, G. CDN: Content distribution network. Technical Report TR-125, Experimental Computer Systems Lab, Department of Computer Science, State University of New York, Stony Brook, NY, 2003. http://citeseer.ist.psu.edu/peng03cdn.html
76. Pierre, G. and van Steen, M. Globule: a collaborative content delivery network. *IEEE Communications*, 44(8), 2006.
77. Pradhan, D. *Fault-Tolerant Computer System Design*. Prentice Hall, Englewood Cliffs, NJ, USA, 1996.
78. Qiu, L., Padmanabhan, V. N., and Voelker, G. M. On the placement of Web server replicas. In *Proc. of IEEE INFOCOM*, Anchorage, Alaska, USA, pp. 1587–1596, 2001.
79. Rabinovich, M. and Spatscheck, O. *Web Caching and Replication*, Addison Wesley, USA, 2002.
80. Rabinovich, M., Xiao, Z., and Agarwal, A. Computing on the edge: A platform for replicating internet applications. In *Proc. of the Eighth International Workshop on Web Content Caching and Distribution*, Hawthorne, NY, USA, 2003.
81. Radoslavov, P., Govindan, R., and Estrin, D. Topology-informed Internet replica placement. In *Proc. of Sixth International Workshop on Web Caching and Content Distribution*, Boston, Massachusetts, 2001.
82. Ranjan, S., Swaminathan, R., Uysal, M., and Knightly, E. DDoS-Resilient scheduling to counter application layer attacks under Imperfect Detection. In *Proc. of INFOCOM*, pp. 1–13, 2006

83. Rousskov, A. and Wessels, D. Cache digests. *Computer Networks and ISDN Systems*, 30(22), pp. 2155–2168, November 1998.
84. Saroiu, S., Gummadi, K. P., Dunn, R. J., Gribble, S. D., and Levy, H. M. An analysis of Internet content delivery systems. *ACM SIGOPS Operating Systems Review*, 36, pp. 315–328, 2002.
85. Schneider, F. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial, 1 *ACM Computing Surveys*, 22(4), pp.299–320, 1990.
86. Shaikh, A., Tewari, R., and Agrawal, M. On the effectiveness of DNS-based server selection." In *Proceedings of IEEE INFOCOM*, Anchorage, AK, USA, pp. 1801–1810, April 2001.
87. Sivasubramanian, S., Pierre, G., and van Steen, M. Replicating Web applications on-demand. In *Proc. of IEEE International Conference on Services Computing (SCC)*, pp. 227–236, China, 2004.
88. Sivasubramanian, S., Pierre, G., van Steen, M., and Alonso, G. Analysis of caching and replication strategies for Web applications. *IEEE Internet Computing*, 11(1), pp. 60–66, 2007.
89. Sivasubramanian, S., Szymaniak, M., Pierre, G., and Van Steen, M. Replication of Web hosting systems. *ACM Computing Surveys*, 36(3), ACM Press, NY, USA, 2004.
90. Stamos, K., Pallis, G., Thomos, C., and Vakali, A. A similarity-based approach for integrated Web caching and content replication in CDNs. In *Proc. of 10th International Databased Engineering and Applications Symposium (IDEAS 2006)*, IEEE Press, New Delhi, India, 2006.
91. Stamos, K., Pallis, G., and Vakali, A. Integrating caching techniques on a content distribution network. In *Proc. of 10th East-European Conference on Advances in Databases and Information Systems (ADBIS 2006)*, Springer-Verlag, Thessaloniki, Greece, 2006.
92. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. Chord: a scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Transactions on Networking (TON)*, 11(1), ACM Press, NY, USA, pp. 17–32, 2003.
93. Szymaniak, M., Pierre, G., and van Steen, M. Netairt: a DNS-based redirection system for apache. In *Proc. of International Conference WWW/Internet*, Algrave, Portugal, 2003.
94. Tse, S. S. H. Approximate algorithms for document placement in distributed Web servers. *IEEE Transactions on Parallel and Distributed Systems*, 16(6), pp. 489–496, 2005.
95. Vakali, A. and Pallis, G. Content delivery networks: status and trends. *IEEE Internet Computing*, 7(6), IEEE Computer Society, pp. 68–74, 2003.
96. Valloppillil, V. and Ross, K. W. Cache array routing protocol v1.0. Internet Draft, 1998.
97. Verma, D. C. *Content Distribution Networks: An Engineering Approach*, John Wiley & Sons, Inc., New York, 2002.
98. Vixie, P. and Wessels, D. Hyper text caching protocol (HTCP/0.0). Internet Engineering Task Force RFC 2756, 2000. www.ietf.org/rfc/rfc2756.txt
99. Wang, J. A survey of Web caching schemes for the Internet. *SIGCOMM Computer Communication Review*, 29(5), ACM Press, NY, USA, pp. 36–46, 1999.
100. Wang, L., Pai, V. S., and Peterson, L. The effectiveness of request redirection on CDN robustness. In *Proc. of 5th Symposium on Operating Systems Design and Implementation*, Boston, MA, USA, pp. 345–360, 2002.
101. Wessels, D. and Claffy, K. Internet cache protocol (ICP) version 2. Internet Engineering Task Force RFC 2186, 1997. www.ietf.org/rfc/rfc2186.txt
102. Wu, B. and Kshemkalyani, A. D. Objective-optimal algorithms for long-term Web Prefetching. *IEEE Transactions on Computers*, 55(1), pp. 2–17, 2006.