文，dns. umass. edu 用 gaia. cs. umass. edu 的 IP 地址进行响应。注意到在本例中，为了获得一台主机名的映射，共发送了 8 份 DNS 报文：4 份查询报文和 4 份回答报文！我们将很快明白利用 DNS 缓存减少这种查询流量的方法。

我们前面的例子假设了 TLD 服务器知道用于主机的权威 DNS 服务器的 IP 地址。一般而言，这种假设并不总是正确的。相反，TLD 服务器只是知道中间的某个 DNS 服务器，该中间 DNS 服务器依次才能知道用于该主机的权威 DNS 服务器。例如，再次假设马萨诸塞大学有一台用于本大学的 DNS 服务器，它称为 dns. umass. edu。同时假设该大学的每个系都有自己的 DNS 服务器，每个系的 DNS 服务器是本系所有主机的权威服务器。在这种情况下，当中间 DNS 服务器 dns. umass. edu 收到了对某主机的请求时，该主机名是以 cs. umass. edu 结尾，它向 dns. nyu. edu 返回 dns. cs. umass. edu 的 IP 地址，后者是所有以 cs. umass. edu 结尾的主机的权威服务器。本地 DNS 服务器 dns. nyu. edu 则向权威 DNS 服务器发送查询，该权威 DNS 服务器向本地 DNS 服务器返回所希望的映射，该本地服务器依次向请求主机返回该映射。在这个例子中，共发送了 10 份 DNS 报文！

图 2-18 所示的例子利用了**递归查询**（recursive query）和**迭代查询**（iterative query）。从 cse. nyu. edu 到 dns. nyu. edu 发出的查询是递归查询，因为该查询以自己的名义请求 dns. nyu. edu 来获得该映射。而后继的 3 个查询是迭代查询，因为所有的回答都是直接返回给 dns. nyu. edu。从理论上讲，任何 DNS 查询既可以是迭代的也能是递归的。例如，图 2-19 显示了一条 DNS 查询链，其中的所有查询都是递归的。实践中，查询通常遵循图 2-18 中的模式：从请求主机到本地 DNS 服务器的查询是递归的，其余的查询是迭代的。

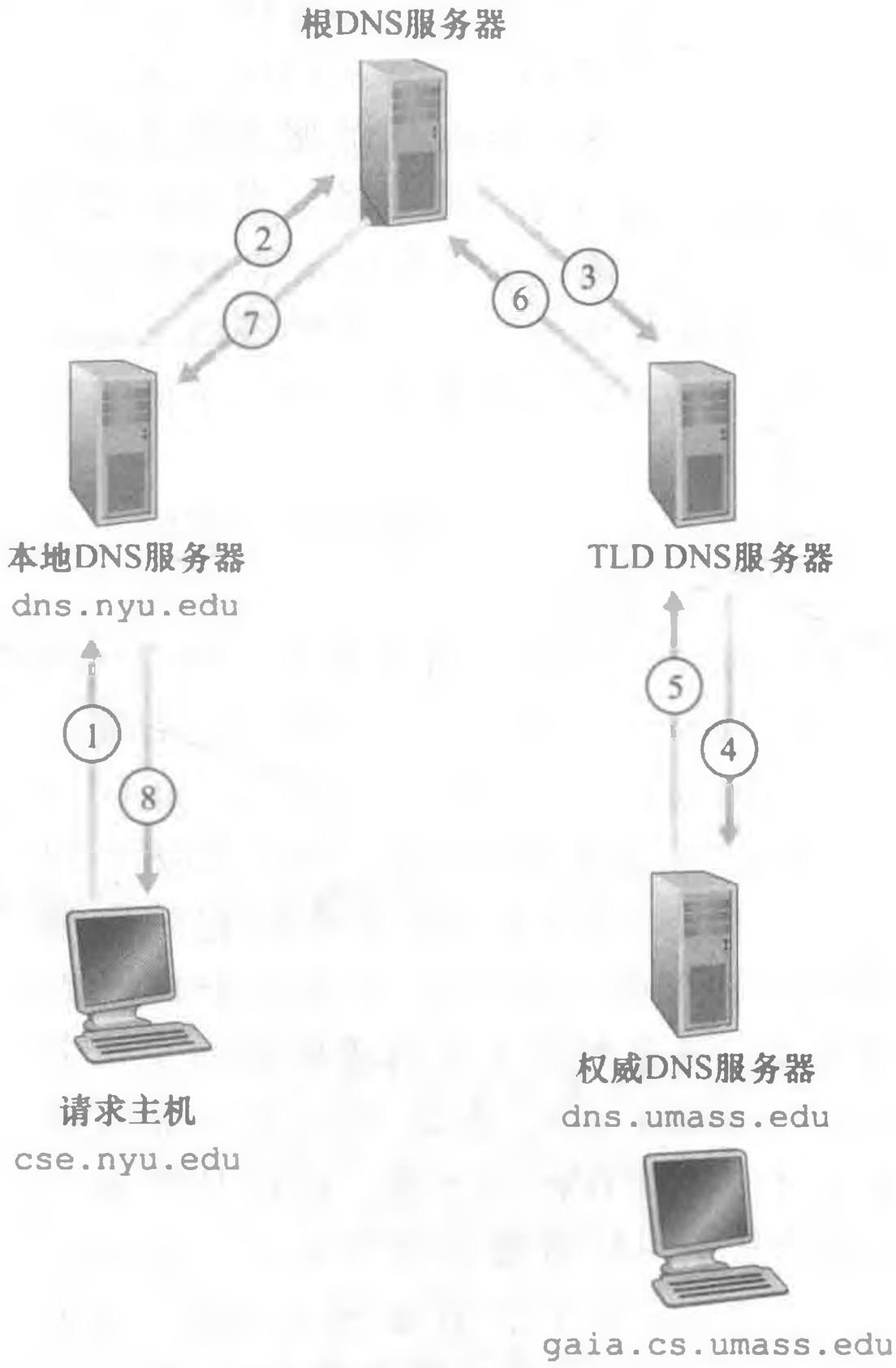

图 2-19　DNS 中的递归查询

2. DNS 缓存

至此我们的讨论一直忽略了 DNS 系统的一个非常重要特色：**DNS 缓存**（DNS caching）。实际上，为了改善时延性能并减少在因特网上到处传输的 DNS 报文数量，DNS 广泛使用了缓存技术。DNS 缓存的原理非常简单。在一个请求链中，当某 DNS 服务器接收一个 DNS 回答（例如，包含某主机名到 IP 地址的映射）时，它能将映射缓存在本地存储器中。例如，在图 2-18 中，每当本地 DNS 服务器 dns. nyu. edu 从某个 DNS 服务器接收到一个回答，它能够缓存包含在该回答中的任何信息。如果在 DNS 服务器中缓存了一台主机名/IP 地址对，另一个对相同主机名的查询到达该 DNS 服务器时，该 DNS 服务器就能够提供所要求的 IP 地址，即使它不是该主机名的权威服务器。由于主机和主机名与 IP 地址间的

映射并不是永久的，DNS 服务器在一段时间后（通常设置为两天）将丢弃缓存的信息。

举一个例子，假定主机 apricot. nyu. edu 向 dns. nyu. edu 查询主机名 cnn. com 的 IP 地址。此后，假定过了几个小时，纽约大学的另外一台主机如 kiwi. nyu. edu 也向 dns. nyu. edu 查询相同的主机名。因为有了缓存，该本地 DNS 服务器可以立即返回 cnn. com 的 IP 地址，而不必查询任何其他 DNS 服务器。本地 DNS 服务器也能够缓存 TLD 服务器的 IP 地址，因而允许本地 DNS 绕过查询链中的根 DNS 服务器。事实上，因为缓存，除了少数 DNS 查询以外，根服务器被绕过了。

2.4.3 DNS 记录和报文

共同实现 DNS 分布式数据库的所有 DNS 服务器存储了**资源记录**（Resource Record，RR），RR 提供了主机名到 IP 地址的映射。每个 DNS 回答报文包含了一条或多条资源记录。在本小节以及后续小节中，我们概要地介绍 DNS 资源记录和报文，更详细的信息可以在［Albitz 1993］或有关 DNS 的 RFC 文档［RFC 1034；RFC 1035］中找到。

资源记录是一个包含了下列字段的 4 元组：

```
(Name, Value, Type, TTL)
```

TTL 是该记录的生存时间，它决定了资源记录应当从缓存中删除的时间。在下面给出的记录例子中，我们忽略掉 TTL 字段。Name 和 Value 的值取决于 Type：

- 如果 Type = A，则 Name 是主机名，Value 是该主机名对应的 IP 地址。因此，一条类型为 A 的资源记录提供了标准的主机名到 IP 地址的映射。例如（relay1. bar. foo. com，145. 37. 93. 126，A）就是一条类型 A 记录。
- 如果 Type = NS，则 Name 是个域（如 foo. com），而 Value 是个知道如何获得该域中主机 IP 地址的权威 DNS 服务器的主机名。这个记录用于沿着查询链来路由 DNS 查询。例如（foo. com，dns. foo. com，NS）就是一条类型为 NS 的记录。
- 如果 Type = CNAME，则 Value 是别名为 Name 的主机对应的规范主机名。该记录能够向查询的主机提供一个主机名对应的规范主机名，例如（foo. com，relay1. bar. foo. com，CNAME）就是一条 CNAME 类型的记录。
- 如果 Type = MX，则 Value 是个别名为 Name 的邮件服务器的规范主机名。举例来说，（foo. com，mail. bar. foo. com，MX）就是一条 MX 记录。MX 记录允许邮件服务器主机名具有简单的别名。值得注意的是，通过使用 MX 记录，一个公司的邮件服务器和其他服务器（如它的 Web 服务器）可以使用相同的别名。为了获得邮件服务器的规范主机名，DNS 客户应当请求一条 MX 记录；而为了获得其他服务器的规范主机名，DNS 客户应当请求 CNAME 记录。

如果一台 DNS 服务器是用于某特定主机名的权威 DNS 服务器，那么该 DNS 服务器会有一条包含用于该主机名的类型 A 记录（即使该 DNS 服务器不是其权威 DNS 服务器，它也可能在缓存中包含有一条类型 A 记录）。如果服务器不是用于某主机名的权威服务器，那么该服务器将包含一条类型 NS 记录，该记录对应于包含主机名的域；它还将包括一条类型 A 记录，该记录提供了在 NS 记录的 Value 字段中的 DNS 服务器的 IP 地址。举例来说，假设一台 edu TLD 服务器不是主机 gaia. cs. umass. edu 的权威 DNS 服务器，则该服务器将包含一条包括主机 cs. umass. edu 的域记录，如（umass. edu，dns. umass. edu，NS）；该 edu TLD 服务器还将包含一条类型 A 记录，如（dns. umass. edu，128. 119. 40. 111，A），

该记录将名字 dns. umass. edu 映射为一个 IP 地址。

1. DNS 报文

在本节前面，我们提到了 DNS 查询和回答报文。DNS 只有这两种报文，并且，查询和回答报文有着相同的格式，如图 2-20 所示。DNS 报文中各字段的语义如下：

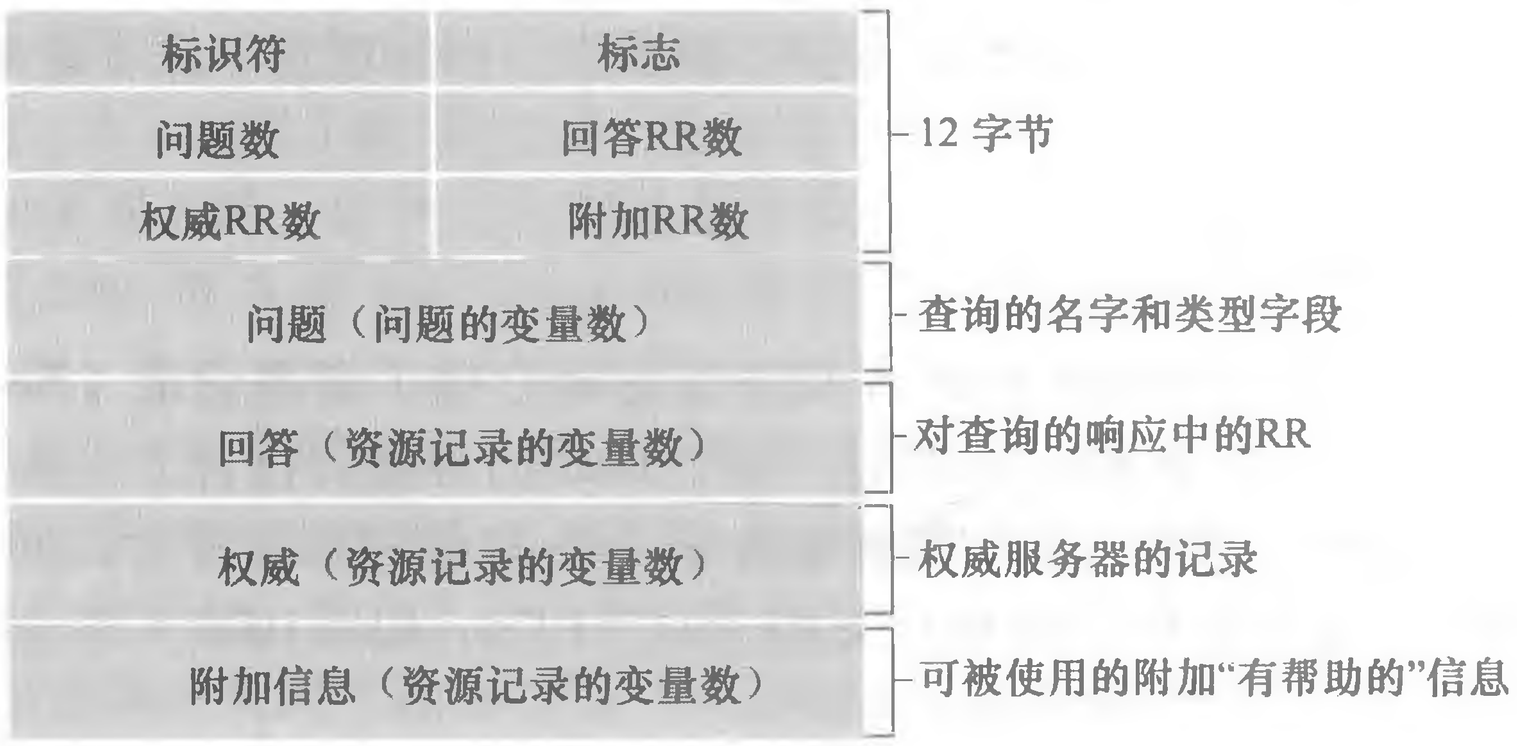

图 2-20　DNS 报文格式

- 前 12 个字节是首部区域，其中有几个字段。第一个字段（标识符）是一个 16 比特的数，用于标识该查询。这个标识符会被复制到对查询的回答报文中，以便让客户用它来匹配发送的请求和接收到的回答。标志字段中含有若干标志。1 比特的“查询/回答”标志位指出报文是查询报文（0）还是回答报文（1）。当某 DNS 服务器是所请求名字的权威 DNS 服务器时，1 比特的“权威的”标志位被置在回答报文中。如果客户（主机或者 DNS 服务器）在该 DNS 服务器没有某记录时希望它执行递归查询，将设置 1 比特的“希望递归”标志位。如果该 DNS 服务器支持递归查询，在它的回答报文中会对 1 比特的“递归可用”标志位置位。在该首部中，还有 4 个有关数量的字段，这些字段指出了在首部后的 4 类数据区域出现的数量。
- 问题区域包含着正在进行的查询信息。该区域包括：①名字字段，包含正在被查询的主机名字；②类型字段，指出有关该名字的正被询问的问题类型，例如主机地址是与一个名字相关联（类型 A）还是与某个名字的邮件服务器相关联（类型 MX）。
- 在来自 DNS 服务器的回答中，回答区域包含了对最初请求的名字的资源记录。前面讲过每个资源记录中有 Type（如 A、NS、CNAME 和 MX）字段、Value 字段和 TTL 字段。在回答报文的回答区域中可以包含多条 RR，因此一个主机名能够有多个 IP 地址（例如，就像本节前面讨论的冗余 Web 服务器）。
- 权威区域包含了其他权威服务器的记录。
- 附加区域包含了其他有帮助的记录。例如，对于一个 MX 请求的回答报文的回答区域包含了一条资源记录，该记录提供了邮件服务器的规范主机名。该附加区域包含一个类型 A 记录，该记录提供了用于该邮件服务器的规范主机名的 IP 地址。

你愿意从正在工作的主机直接向某些 DNS 服务器发送一个 DNS 查询报文吗？使用 **nslookup 程序**（nslookup program）能够容易地做到这一点，这对于多数 Windows 和 UNIX 平台，nslookup 程序是可用的。例如，从一台 Windows 主机打开命令提示符界面，直接键入“nslookup”即可调用该 nslookup 程序。在调用 nslookup 后，你能够向任何 DNS 服务器

(根、TLD 或权威) 发送 DNS 查询。在接收到来自 DNS 服务器的回答后，nslookup 将显示包括在该回答中的记录（以人可读的格式)。从你自己的主机运行 nslookup 还有一种方法，即访问允许你远程应用 nslookup 的许多 Web 站点之一（在一个搜索引擎中键入“nslookup”就能够得到这些站点中的一个)。本章最后的 DNS Wireshark 实验将使你更为详细地研究 DNS。

2. 在 DNS 数据库中插入记录

上面的讨论只是关注如何从 DNS 数据库中取数据。你可能想知道这些数据最初是怎么进入数据库中的。我们在一个特定的例子中看看这是如何完成的。假定你刚刚创建一个称为网络乌托邦（Network Utopia）的令人兴奋的新创业公司。你必定要做的第一件事是在注册登记机构注册域名 networkutopia. com。**注册登记机构**（registrar）是一个商业实体，它验证该域名的唯一性，将该域名输入 DNS 数据库（如下面所讨论的那样)，对提供的服务收取少量费用。1999 年前，唯一的注册登记机构是 Nework Solution，它独家经营对于 com、net 和 org 域名的注册。但是现在有许多注册登记机构竞争客户，因特网名字和地址分配机构（Internet Corporation for Assigned Names and Numbers，ICANN）向各种注册登记机构授权。在 http://www. internic. net 上可以找到授权的注册登记机构的列表。

当你向某些注册登记机构注册域名 networkutopia. com 时，需要向该机构提供你的基本和辅助权威 DNS 服务器的名字和 IP 地址。假定该名字和 IP 地址是 dns1. networkutopia. com 和 dns2. networkutopia. com 及 212. 212. 212. 1 和 212. 212. 212. 2。对这两个权威 DNS 服务器的每一个，该注册登记机构确保将一个类型 NS 和一个类型 A 的记录输入 TLD com 服务器。特别是对于用于 networkutopia. com 的基本权威服务器，该注册登记机构将下列两条资源记录插入该 DNS 系统中：

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

你还必须确保用于 Web 服务器 www. networkutopia. com 的类型 A 资源记录和用于邮件服务器 mail. networkutopia. com 的类型 MX 资源记录被输入你的权威 DNS 服务器中。（直到最近，每台 DNS 服务器中的内容都是静态配置的，例如来自系统管理员创建的配置文件。最近，在 DNS 协议中添加了一个更新（UPDATE）选项，允许通过 DNS 报文对数据库中的内容进行动态添加或者删除。[RFC 2136] 和 [RFC 3007] 定义了 DNS 动态更新。)

一旦完成所有这些步骤，人们将能够访问你的 Web 站点，并向你公司的雇员发送电子邮件。我们通过验证该说法的正确性来总结 DNS 的讨论。这种验证也有助于充实我们已经学到的 DNS 知识。假定在澳大利亚的 Alice 要观看 www. networkutopia. com 的 Web 页面。如前面所讨论，她的主机将首先向其本地 DNS 服务器发送请求。该本地服务器接着则联系一个 TLD com 服务器。（如果 TLD com 服务器的地址没有被缓存，该本地 DNS 服务器也将必须与根 DNS 服务器相联系。）该 TLD 服务器包含前面列出的类型 NS 和类型 A 资源记录，因为注册登记机构将这些资源记录插入所有的 TLD com 服务器。该 TLD com 服务器向 Alice 的本地 DNS 服务器发送一个回答，该回答包含了这两条资源记录。该本地 DNS 服务器则向 212. 212. 212. 1 发送一个 DNS 查询，请求对应于 www. networkutopia. com 的类型 A 记录。该记录提供了所希望的 Web 服务器的 IP 地址，如 212. 212. 71. 4，本地 DNS 服务器将该地址回传给 Alice 的主机。Alice 的浏览器此时能够向主机 212. 212. 71. 4 发起一个 TCP 连接，并在该连接上发送一个 HTTP 请求。当一个人在网上冲浪时，有比满足眼球更

多的事情在进行!

关注安全性

DNS 脆弱性

我们已经看到 DNS 是因特网基础设施的一个至关重要的组件，对于包括 Web、电子邮件等的许多重要的服务，没有它都不能正常工作。因此，我们自然要问，DNS 怎么能够被攻击呢？DNS 是一个易受攻击的目标吗？它是将会被淘汰的服务吗？大多数因特网应用会随同它一起无法工作吗？

想到的第一种针对 DNS 服务的攻击是分布式拒绝服务（DDoS）带宽洪泛攻击（参见 1.6 节）。例如，某攻击者能够试图向每个 DNS 根服务器发送大量的分组，使得大多数合法 DNS 请求得不到回答。这种对 DNS 根服务器的 DDoS 大规模攻击实际发生在 2002 年 10 月 21 日。在这次攻击中，该攻击者利用一个僵尸网络向 13 个 DNS 根服务器中的每个都发送了大批的 ICMP ping 报文负载。（5.6 节中讨论了 ICMP 报文。此时，知道 ICMP 分组是特殊类型的 IP 数据报就可以了。）幸运的是，这种大规模攻击所带来的损害很小，对用户的因特网体验几乎没有或根本没有影响。攻击者的确成功地将大量的分组指向了根服务器，但许多 DNS 根服务器受到了分组过滤器的保护，配置的分组过滤器阻挡了所有指向根服务器的 ICMP ping 报文。这些被保护的服务器因此未受伤害并且与平常一样发挥着作用。此外，大多数本地 DNS 服务器缓存了顶级域名服务器的 IP 地址，使得这些请求过程通常绕过了 DNS 根服务器。

对 DNS 的潜在更为有效的 DDoS 攻击将是向顶级域名服务器（例如向所有处理.com 域的顶级域名服务器）发送大量的 DNS 请求。过滤指向 DNS 服务器的 DNS 请求将更为困难，并且顶级域名服务器不像根服务器那样容易绕过。但是这种攻击的严重性通过本地 DNS 服务器中的缓存技术可将部分地被缓解。

DNS 能够潜在地以其他方式被攻击。在中间人攻击中，攻击者截获来自主机的请求并返回伪造的回答。在 DNS 毒害攻击中，攻击者向一台 DNS 服务器发送伪造的回答，诱使服务器在它的缓存中接收伪造的记录。这些攻击中的任一种，都能够将毫无疑虑的 Web 用户重定向到攻击者的 Web 站点。然而，这些攻击难以实现，因为它们要求截获分组或扼制住服务器［Skoudis 2006］。

总而言之，DNS 自身已经显示了对抗攻击的令人惊讶的健壮性。至今为止，还没有一个攻击已经成功地妨碍了 DNS 服务。

2.5　P2P 文件分发

在目前为止本章中描述的应用（包括 Web、电子邮件和 DNS）都采用了客户 - 服务器体系结构，极大地依赖于总是打开的基础设施服务器。2.1.1 节讲过，使用 P2P 体系结构，对总是打开的基础设施服务器有最小的（或者没有）依赖。与之相反，成对间歇连接的主机（称为对等方）彼此直接通信。这些对等方并不为服务提供商所拥有，而是受用户控制的桌面计算机和膝上计算机。

在本节中我们将研究一个非常自然的 P2P 应用，即从单一服务器向大量主机（称为对

等方）分发一个大文件。该文件也许是一个新版的 Linux 操作系统，对于现有操作系统或应用程序的一个软件补丁，一个 MP3 音乐文件，或一个 MPEG 视频文件。在客户 - 服务器文件分发中，该服务器必须向每个对等方发送该文件的一个副本，即服务器承受了极大的负担，并且消耗了大量的服务器带宽。在 P2P 文件分发中，每个对等方能够向任何其他对等方重新分发它已经收到的该文件的任何部分，从而在分发过程中协助该服务器。到 2016 年止，最为流行的 P2P 文件分发协议是 BitTorrent。该应用程序最初由 Bram Cohen 所研发，现在有许多不同的独立且符合 BitTorrent 协议的 BitTorrent 客户，就像有许多符合 HTTP 协议的 Web 浏览器客户一样。在下面的小节中，我们首先考察在文件分发环境中的 P2P 体系结构的自扩展性。然后我们更为详细地描述 BitTorrent，突出它的最为重要的特性和特色。

1. P2P 体系结构的扩展性

为了将客户 - 服务器体系结构与 P2P 体系结构进行比较，阐述 P2P 的内在自扩展性，我们现在考虑一个用于两种体系结构类型的简单定量模型，将一个文件分发给一个固定对等方集合。如图 2-21 所示，服务器和对等方使用接入链路与因特网相连。其中 u_s 表示服务器接入链路的上载速率，u_i 表示第 i 对等方接入链路的上载速率，d_i 表示了第 i 对等方接入链路的下载速率。还用 F 表示被分发的文件长度（以比特计），N 表示要获得的该文件副本的对等方的数量。**分发时间**（distribution time）是所有 N 个对等方得到该文件的副本所需要的时间。在下面分析分发时间的过程中，我们对客户 - 服务器和 P2P 体系结构做了简化（并且通常是准确的［Akella 2003］）的假设，即因特网核心具有足够的带宽，这意味着所有瓶颈都在网络接入链路。我们还假设服务器和客户没有参与任何其他网络应用，因此它们的所有上传和下载访问带宽能被全部用于分发该文件。

图 2-21 文件分发问题的示例图

我们首先来确定对于客户 - 服务器体系结构的分发时间，我们将其表示为 D_{cs}。在客户 - 服务器体系结构中，没有对等方参与来帮助分发文件。我们做下列观察：

- 服务器必须向 N 个对等方的每个传输该文件的一个副本。因此该服务器必须传输 NF 比特。因为该服务器的上载速率是 u_s，分发该文件的时间必定是至少为 NF/u_s。
- 令 d_{min} 表示具有最小下载速率的对等方的下载速率，即 $d_{min} = \min\{d_1, d_p, \cdots, d_N\}$。具有最小下载速率的对等方不可能在少于 F/d_{min} 秒时间内获得该文件的所有 F 比特。因此最小分发时间至少为 F/d_{min}。

将这两个观察放在一起，我们得到

$$D_{cs} \geqslant \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}$$

该式提供了对于客户 - 服务器体系结构的最小分发时间的下界。在课后习题中将请你给出服务器能够调度它的传输以便实际取得该下界的方法。因此我们取上面提供的这个下界作为实际发送时间，即

$$D_{cs} = \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\} \tag{2-1}$$

我们从式（2-1）看到，对足够大的 N，客户 - 服务器分发时间由 NF/u_s 确定。所以，该分发时间随着对等方 N 的数量线性地增加。因此举例来说，如果从某星期到下星期对等方的数量从 1000 增加了 1000 倍，到了 100 万，将该文件分发到所有对等方所需要的时间就要增加 1000 倍。

我们现在来对 P2P 体系结构进行简单的分析，其中每个对等方能够帮助服务器分发该文件。特别是，当一个对等方接收到某些文件数据，它能够使用自己的上载能力重新将数据分发给其他对等方。计算 P2P 体系结构的分发时间在某种程度上比计算客户 - 服务器体系结构的更为复杂，因为分发时间取决于每个对等方如何向其他对等方分发该文件的各个部分。无论如何，能够得到对该最小分发时间的一个简单表示式［Kumar 2006］。至此，我们先做下列观察：

- 在分发的开始，只有服务器具有文件。为了使社区的这些对等方得到该文件，该服务器必须经其接入链路至少发送该文件的每个比特一次。因此，最小分发时间至少是 F/u_s。（与客户 - 服务器方案不同，由服务器发送过一次的比特可能不必由该服务器再次发送，因为对等方在它们之间可以重新分发这些比特。）
- 与客户 - 服务器体系结构相同，具有最低下载速率的对等方不能够以小于 F/d_{min} 秒的分发时间获得所有 F 比特。因此最小分发时间至少为 F/d_{min}。
- 最后，观察到系统整体的总上载能力等于服务器的上载速率加上每个单独的对等方的上载速率，即 $u_{total} = u_s + u_1 + \cdots + u_N$。系统必须向这 N 个对等方的每个交付（上载）F 比特，因此总共交付 NF 比特。这不能以快于 u_{total} 的速率完成。因此，最小的分发时间也至少是 $NF/(u_s + u_1 + \cdots + u_N)$。

将这三个观察放在一起，我们获得了对 P2P 的最小分发时间，表示为 D_{P2P}。

$$D_{P2P} \geqslant \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \tag{2-2}$$

式（2-2）提供了对于 P2P 体系结构的最小分发时间的下界。这说明，如果我们认为一旦每个对等方接收到一个比特就能够重分发一个比特的话，则存在一个重新分发方案能实际取得这种下界［Kumar 2006］。（我们将在课后习题中证明该结果的一种特情形。）实

际上，被分发的是文件块而不是一个个比特。式（2-2）能够作为实际最小分发时间的很好近似。因此，我们取由式（2-2）提供的下界作为实际的最小分发时间，即

$$D_{\mathrm{P2P}} = \max\left\{\frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \tag{2-3}$$

图 2-22 比较了客户 - 服务器和 P2P 体系结构的最小分发时间，其中假定所有的对等方具有相同的上载速率 u。在图 2-22 中，我们已经设置了 $F/u = 1$ 小时，$u_s = 10u$，$d_{\min} \geqslant u_s$。因此，在一个小时中一个对等方能够传输整个文件，该服务器的传输速率是对等方上载速率的 10 倍，并且（为了简化起见）对等方的下载速率被设置得足够大，使之不会产生影响。我们从图 2-22 中看到，对于客户 - 服务器体系结构，随着对等方数量的增加，分发时间呈线性增长并且没有界。然而，对于 P2P 体系结构，最小分发时间不仅总是小于客户 - 服务器体系结构的分发时间，并且对于任意的对等方数量 N，总是小于 1 小时。因此，具有 P2P 体系结构的应用程序能够是自扩展的。这种扩展性的直接成因是：对等方除了是比特的消费者外还是它们的重新分发者。

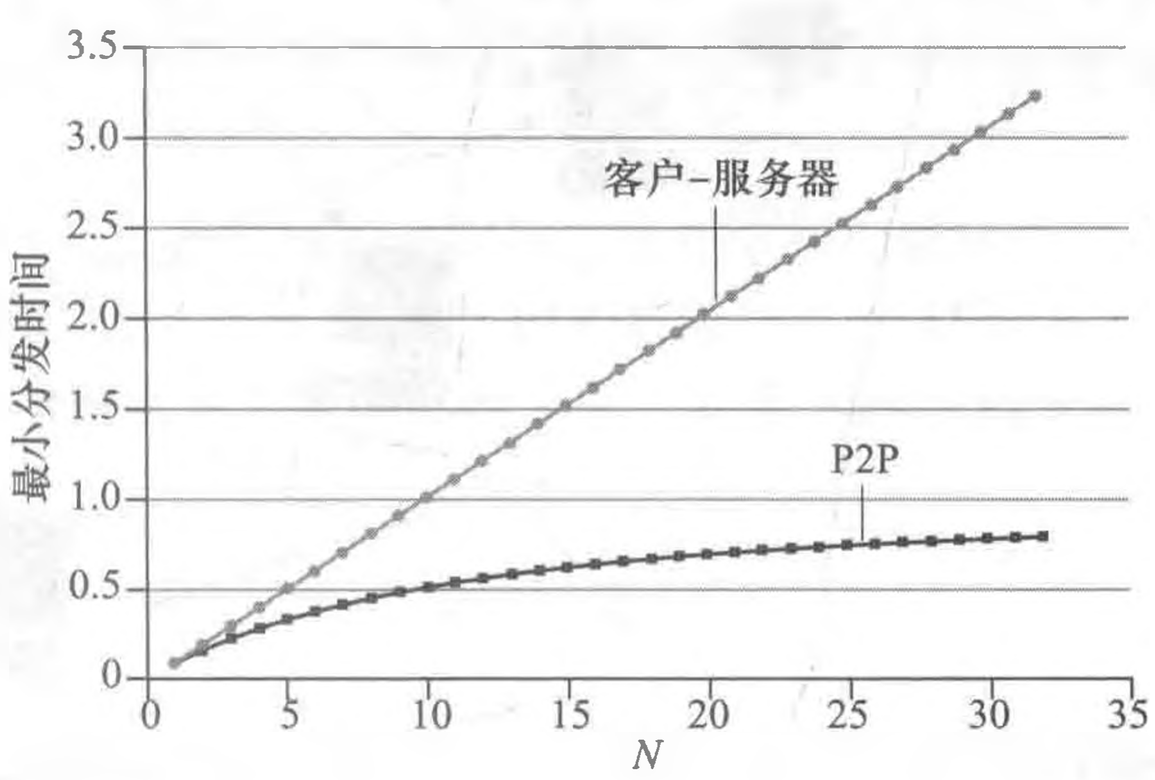

图 2-22 P2P 和客户 - 服务器体系结构的分发时间

2. BitTorrent

BitTorrent 是一种用于文件分发的流行 P2P 协议［Chao 2011］。用 BitTorrent 的术语来讲，参与一个特定文件分发的所有对等方的集合被称为一个洪流（torrent）。在一个洪流中的对等方彼此下载等长度的文件块（chunk），典型的块长度为 256KB。当一个对等方首次加入一个洪流时，它没有块。随着时间的流逝，它累积了越来越多的块。当它下载块时，也为其他对等方上载了多个块。一旦某对等方获得了整个文件，它也许（自私地）离开洪流，或（大公无私地）留在该洪流中并继续向其他对等方上载块。同时，任何对等方可能在任何时候仅具有块的子集就离开该洪流，并在以后重新加入该洪流中。

我们现在更为仔细地观察 BitTorrent 运行的过程。因为 BitTorrent 是一个相当复杂的协议，所以我们将仅描述它最重要的机制，而对某些细节视而不见；这将使得我们能够通过树木看森林。每个洪流具有一个基础设施节点，称为追踪器（tracker）。当一个对等方加入某洪流时，它向追踪器注册自己，并周期性地通知追踪器它仍在该洪流中。以这种方式，追踪器跟踪参与在洪流中的对等方。一个给定的洪流可能在任何时刻具有数以百计或数以千计的对等方。

如图 2-23 所示，当一个新的对等方 Alice 加入该洪流时，追踪器随机地从参与对等方的集合中选择对等方的一个子集（为了具体起见，设有 50 个对等方），并将这 50 个对等方的 IP 地址发送给 Alice。Alice 持有对等方的这张列表，试图与该列表上的所有对等方创建并行的 TCP 连接。我们称所有这样与 Alice 成功地创建一个 TCP 连接的对等方为“邻近对等方”（在图 2-23 中，Alice 显示了仅有三个邻近对等方。通常，她应当有更多的对等方）。随着时间的流逝，这些对等方中的某些可能离开，其他对等方（最初 50 个以外的）

可能试图与 Alice 创建 TCP 连接。因此一个对等方的邻近对等方将随时间而波动。

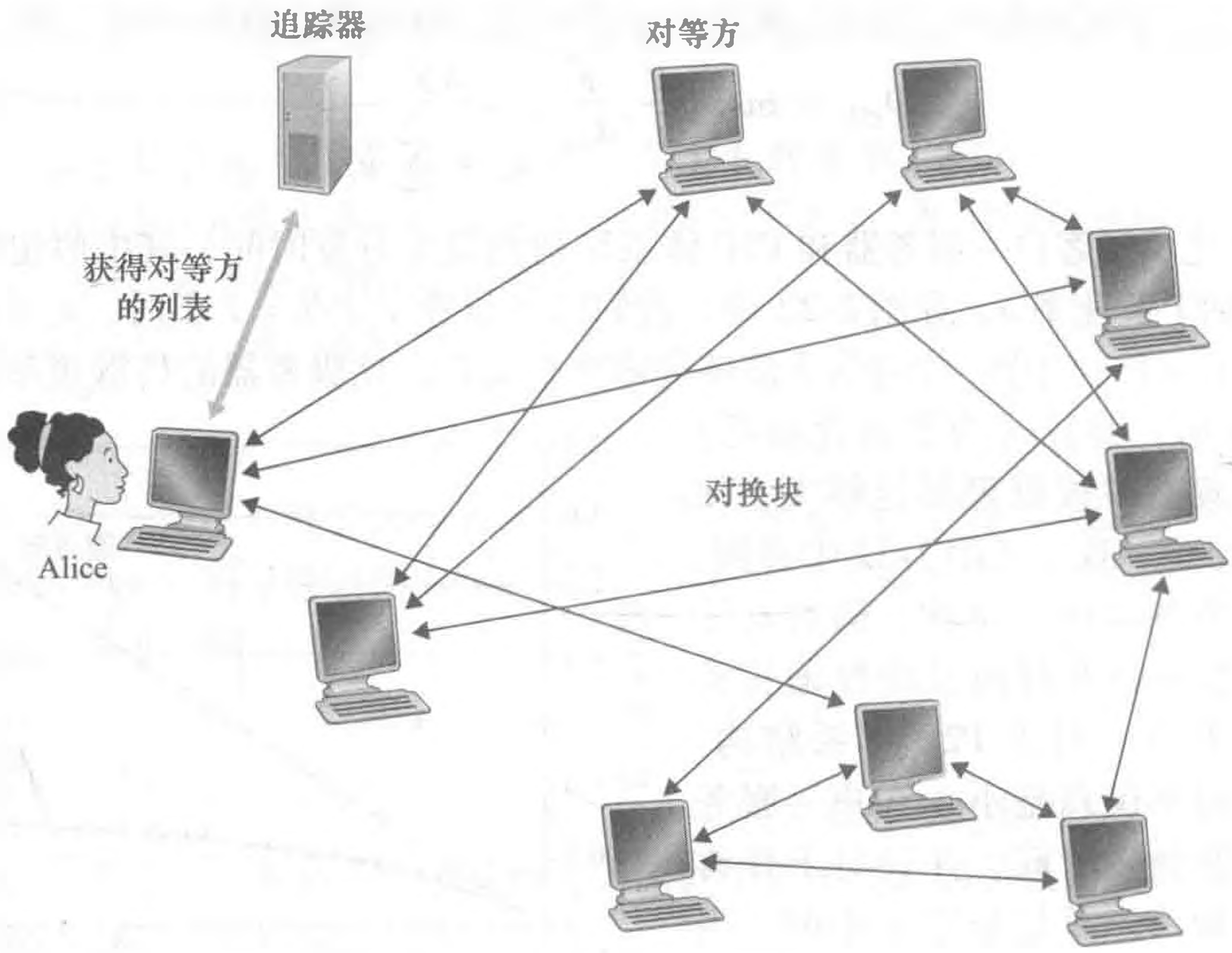

图 2-23　用 BitTorrent 分发文件

在任何给定的时间，每个对等方将具有来自该文件的块的子集，并且不同的对等方具有不同的子集。Alice 周期性地（经 TCP 连接）询问每个邻近对等方它们所具有的块列表。如果 Alice 具有 L 个不同的邻居，她将获得 L 个块列表。有了这个信息，Alice 将对她当前还没有的块发出请求（仍通过 TCP 连接）。

因此在任何给定的时刻，Alice 将具有块的子集并知道它的邻居具有哪些块。利用这些信息，Alice 将做出两个重要决定。第一，她应当从她的邻居请求哪些块呢？第二，她应当向哪些向她请求块的邻居发送块？在决定请求哪些块的过程中，Alice 使用一种称为**最稀缺优先**（rarest first）的技术。这种技术的思路是，针对她没有的块在她的邻居中决定最稀缺的块（最稀缺的块就是那些在她的邻居中副本数量最少的块），并首先请求那些最稀缺的块。这样，最稀缺块得到更为迅速的重新分发，其目标是（大致地）均衡每个块在洪流中的副本数量。

为了决定她响应哪个请求，BitTorrent 使用了一种机灵的对换算法。其基本想法是，Alice 根据当前能够以最高速率向她提供数据的邻居，给出其优先权。特别是，Alice 对于她的每个邻居都持续地测量接收到比特的速率，并确定以最高速率流入的 4 个邻居。每过 10 秒，她重新计算该速率并可能修改这 4 个对等方的集合。用 BitTorrent 术语来说，这 4 个对等方被称为**疏通**（unchoked）。重要的是，每过 30 秒，她也要随机地选择另外一个邻居并向其发送块。我们将这个被随机选择的对等方称为 Bob。因为 Alice 正在向 Bob 发送数据，她可能成为 Bob 前 4 位上载者之一，这样的话 Bob 将开始向 Alice 发送数据。如果 Bob 向 Alice 发送数据的速率足够高，Bob 接下来也能成为 Alice 的前 4 位上载者。换言之，每过 30 秒 Alice 将随机地选择一名新的对换伴侣并开始与那位伴侣进行对换。如果这两名对等方都满足此对换，它们将对方放入其前 4 位列表中并继续与对方进行对换，直到该对等方之一发现了一个更好的伴侣为止。这种效果是对等方能够以趋向于找到彼此的协调的

速率上载。随机选择邻居也允许新的对等方得到块，因此它们能够具有对换的东西。除了这 5 个对等方（“前”4 个对等方和一个试探的对等方）的所有其他相邻对等方均被“阻塞”，即它们不能从 Alice 接收到任何块。BitTorrent 有一些有趣的机制没有在这里讨论，包括片（小块）、流水线、随机优先选择、残局模型和反怠慢［Cohen 2003］。

刚刚描述的关于交换的激励机制常被称为“一报还一报”（tit-for-tat）［Cohen 2003］。已证实这种激励方案能被回避［Liogkas 2006；Locher 2006；Piatek 2007］。无论如何，BitTorrent“生态系统”取得了广泛成功，数以百万计的并发对等方在数十万条洪流中积极地共享文件。如果 BitTorrent 被设计为不采用一报还一报（或一种变种），然而在别的方面却完全相同的协议，BitTorrent 现在将很可能不复存在了，因为大多数用户将成为搭便车者了［Sarouiu 2002］。

我们简要地提一下另一种 P2P 应用——分布式散列表（DHT）来结束我们的讨论。分布式散列表是一种简单的数据库，其数据库记录分布在一个 P2P 系统的多个对等方上。DHT 得到了广泛实现（如在 BitTorrent 中），并成为大量研究的主题。在配套网站的 Video Note 中对 DHT 进行了概述。

2.6 视频流和内容分发网

事先录制的流式视频现在是北美住宅 ISP 中的流量主体。特别是，Netflix 和 YouTube 在 2015 年分别消耗了住宅 ISP 流量的 37% 和 16%［Sandvine 2015］。在本节中，我们将对如何在今天的因特网中实现流行的视频流服务进行概述。我们将看到它们的实现方式是使用应用层协议和以像高速缓存那样方式运行的服务器。在专注于多媒体网络的第 9 章中，我们将进一步研究因特网视频以及其他因特网多媒体服务。

2.6.1 因特网视频

在流式存储视频应用中，基础的媒体是预先录制的视频，例如电影、电视节目、录制好的体育事件或录制好的用户生成的视频（如通常在 YouTube 上可见的那些）。这些预先录制好的视频放置在服务器上，用户按需向这些服务器发送请求来观看视频。许多因特网公司现在提供流式视频，这些公司包括 Netflix、YouTube（谷歌）、亚马逊和优酷。

但在开始讨论视频流之前，我们先迅速感受一下视频媒体自身。视频是一系列的图像，通常以一种恒定的速率（如每秒 24 或 30 张图像）来展现。一幅未压缩、数字编码的图像由像素阵列组成，其中每个像素是由一些比特编码来表示亮度和颜色。视频的一个重要特征是它能够被压缩，因而可用比特率来权衡视频质量。今天现成的压缩算法能够将一个视频压缩成所希望的任何比特率。当然，比特率越高，图像质量越好，用户的总体视觉感受越好。

从网络的观点看，也许视频最为突出的特征是它的高比特率。压缩的因特网视频的比特率范围通常从用于低质量视频的 100kbps，到用于流式高分辨率电影的超过 3Mbps，再到用于 4K 流式展望的超过 10Mbps。这能够转换为巨大的流量和存储，特别是对高端视频。例如，单一 2Mbps 视频在 67 分钟期间将耗费 1GB 的存储和流量。到目前为止，对流式视频的最为重要的性能度量是平均端到端吞吐量。为了提供连续不断的布局，网络必须为流式应用提供平均吞吐量，这个流式应用至少与压缩视频的比特率一样大。

我们也能使用压缩生成相同视频的多个版本，每个版本有不同的质量等级。例如，我们能够使用压缩生成相同视频的 3 个版本，比特率分别为 300kbps、1Mbps 和 3Mbps。用户

则能够根据他们当前可用带宽来决定观看哪个版本。具有高速因特网连接的用户也许选择3Mbps版本，使用智能手机通过3G观看视频的用户可能选择300kbps版本。

2.6.2　HTTP流和DASH

在HTTP流中，视频只是存储在HTTP服务器中作为一个普通的文件，每个文件有一个特定的URL。当用户要看该视频时，客户与服务器创建一个TCP连接并发送对该URL的HTTP GET请求。服务器则以底层网络协议和流量条件允许的尽可能快的速率，在一个HTTP响应报文中发送该视频文件。在客户一侧，字节被收集在客户应用缓存中。一旦该缓存中的字节数量超过预先设定的门限，客户应用程序就开始播放，特别是，流式视频应用程序周期性地从客户应用程序缓存中抓取帧，对这些帧解压缩并且在用户屏幕上展现。因此，流式视频应用接收到视频就进行播放，同时缓存该视频后面部分的帧。

如前一小节所述，尽管HTTP流在实践中已经得到广泛部署（例如，自YouTube发展初期开始），但它具有严重缺陷，即所有客户接收到相同编码的视频，尽管对不同的客户或者对于相同客户的不同时间而言，客户可用的带宽大小有很大不同。这导致了一种新型基于HTTP的流的研发，它常常被称为**经HTTP的动态适应性流**（Dynamic Adaptive Streaming over HTTP，DASH）。在DASH中，视频编码为几个不同的版本，其中每个版本具有不同的比特率，对应于不同的质量水平。客户动态地请求来自不同版本且长度为几秒的视频段数据块。当可用带宽量较高时，客户自然地选择来自高速率版本的块；当可用带宽量较低时，客户自然地选择来自低速率版本的块。客户用HTTP GET请求报文一次选择一个不同的块［Akhshabi 2011］。

DASH允许客户使用不同的以太网接入速率流式播放具有不同编码速率的视频。使用低速3G连接的客户能够接收一个低比特率（和低质量）的版本，使用光纤连接的客户能够接收高质量的版本。如果端到端带宽在会话过程中改变的话，DASH允许客户适应可用带宽。这种特色对于移动用户特别重要，当移动用户相对于基站移动时，通常他们能感受到其可用带宽的波动。

使用DASH后，每个视频版本存储在HTTP服务器中，每个版本都有一个不同的URL。HTTP服务器也有一个**告示文件**（manifest file），为每个版本提供了一个URL及其比特率。客户首先请求该告示文件并且得知各种各样的版本。然后客户通过在HTTP GET请求报文中对每块指定一个URL和一个字节范围，一次选择一块。在下载块的同时，客户也测量接收带宽并运行一个*速率决定算法*来选择下次请求的块。自然地，如果客户缓存的视频很多，并且测量的接收带宽较高，它将选择一个高速率的版本。同样，如果客户缓存的视频很少，并且测量的接收带宽较低，它将选择一个低速率的版本。因此DASH允许客户自由地在不同的质量等级之间切换。

2.6.3　内容分发网

今天，许多因特网视频公司日复一日地向数以百万计的用户按需分发每秒数兆比特的流。例如，YouTube的视频库藏有几亿个，每天向全世界的用户分发几亿条流。向位于全世界的所有用户流式传输所有流量同时提供连续播放和高交互性显然是一项有挑战性的任务。

对于一个因特网视频公司，或许提供流式视频服务最为直接的方法是建立单一的大规模数据中心，在数据中心中存储其所有视频，并直接从该数据中心向世界范围的客户传输流式视频。但是这种方法存在三个问题。首先，如果客户远离数据中心，服务器到客户的

分组将跨越许多通信链路并很可能通过许多 ISP，其中某些 ISP 可能位于不同的大洲。如果这些链路之一提供的吞吐量小于视频消耗速率，端到端吞吐量也将小于该消耗速率，给用户带来恼人的停滞时延。（第 1 章讲过，一条流的端到端吞吐量由瓶颈链路的吞吐量所决定。）出现这种事件的可能性随着端到端路径中链路数量的增加而增加。第二个缺陷是流行的视频很可能经过相同的通信链路发送许多次。这不仅浪费了网络带宽，因特网视频公司自己也将为向因特网反复发送相同的字节而向其 ISP 运营商（连接到数据中心）支付费用。这种解决方案的第三个问题是单个数据中心代表一个单点故障，如果数据中心或其通向因特网的链路崩溃，它将不能够分发任何视频流了。

为了应对向分布于全世界的用户分发巨量视频数据的挑战，几乎所有主要的视频流公司都利用**内容分发网**（Content Distribution Network，CDN）。CDN 管理分布在多个地理位置上的服务器，在它的服务器中存储视频（和其他类型的 Web 内容，包括文档、图片和音频）的副本，并且所有试图将每个用户请求定向到一个将提供最好的用户体验的 CDN 位置。CDN 可以是**专用 CDN**（private CDN），即它由内容提供商自己所拥有；例如，谷歌的 CDN 分发 YouTube 视频和其他类型的内容。另一种 CDN 可以是**第三方 CDN**（third-party CDN），它代表多个内容提供商分发内容；Akamai、Limelight 和 Level-3 都运行第三方 CDN。现代 CDN 的一个可读性强的展望见［Leighton 2009；Nygren 2010］。

学 习 案 例

谷歌的网络基础设施

为了支持谷歌的巨量云服务阵列，包括搜索、Gmail、日程表、YouTube 视频、地图、文档和社交网络，谷歌已经部署了一个广泛的专用网和 CDN 基础设施。谷歌的 CDN 基础设施具有三个等级的服务器集群：

- 14 个“百万数据中心”，其中 8 个位于北美，4 个位于欧洲，2 个位于亚洲［Google Locations 2016］，每个数据中心具有 10 万台量级的服务器。这些“百万数据中心”负责服务于动态的（并且经常是个性化的）内容，包括搜索结果和 Gmail 报文。
- 在 IXP 中大约 50 个集群分布于全球，其中每个集群由 100～500 台服务器组成［Adhikari 2011a］。这些集群负责服务于静态内容，包括 YouTube 视频［Adhikari 2011a］。
- 数以百计的“深入（enter-deep）”集群位于一个接入 ISP 中。这里一个集群通常由位于一个机架上的数十台服务器组成。这些“深入服务器”执行 TCP 分岔（参见 3.7 节）并服务于静态内容［Chen 2011］，包括体现搜索结果的 Web 网页的静态部分。

所有这些数据中心和集群位置与谷歌自己的专用网连接在一起。当某用户进行搜索请求时，该请求常常先经过本地 ISP 发送到邻近的“深入服务器”缓存中，从这里检索静态内容；同时将该静态内容提供给客户，邻近的缓存也经谷歌的专用网将请求转发给“大型数据中心”，从这里检索个性化的搜索结果。对于某 YouTube 视频，该视频本身可能来自一个“邀请坐客服务器”缓存，而围绕该视频的 Web 网页部分可能来自邻近的“深入服务器”缓存，围绕该视频的广告来自数据中心。总的来说，除了本地 ISP，谷歌云服务在很大程度上是由独立于公共因特网的网络基础设施提供的。

CDN 通常采用两种不同的服务器安置原则［Huang 2008］：

- **深入**。第一个原则由 Akamai 首创，该原则是通过在遍及全球的接入 ISP 中部署服务器集群来深入到 ISP 的接入网中。（在 1.3 节中描述了接入网。）Akamai 在大约 1700 个位置采用这种方法部署集群。其目标是靠近端用户，通过减少端用户和 CDN 集群之间（内容从这里收到）链路和路由器的数量，从而改善了用户感受的时延和吞吐量。因为这种高度分布式设计，维护和管理集群的任务成为挑战。
- **邀请做客**。第二个设计原则由 Limelight 和许多其他 CDN 公司所采用，该原则是通过在少量（例如 10 个）关键位置建造大集群来邀请到 ISP 做客。不是将集群放在接入 ISP 中，这些 CDN 通常将它们的集群放置在因特网交换点（IXP）（参见 1.3 节）。与深入设计原则相比，邀请做客设计通常产生较低的维护和管理开销，可能以对端用户的较高时延和较低吞吐量为代价。

一旦 CDN 的集群准备就绪，它就可以跨集群复制内容。CDN 可能不希望将每个视频的副本放置在每个集群中，因为某些视频很少观看或仅在某些国家中流行。事实上，许多 CDN 没有将视频推入它们的集群，而是使用一种简单的拉策略：如果客户向一个未存储该视频的集群请求某视频，则该集群检索该视频（从某中心仓库或者从另一个集群），向客户流式传输视频时的同时在本地存储一个副本。类似于因特网缓存（参见 2.2.5 节），当某集群存储器变满时，它删除不经常请求的视频。

1. CDN 操作

在讨论过这两种部署 CDN 的重要方法后，我们现在深入看看 CDN 操作的细节。当用户主机中的一个浏览器指令检索一个特定的视频（由 URL 标识）时，CDN 必须截获该请求，以便能够：①确定此时适合用于该客户的 CDN 服务器集群；②将客户的请求重定向到该集群的某台服务器。我们很快将讨论 CDN 是如何能够确定一个适当的集群的。但是我们首先考察截获和重定向请求所依赖的机制。

大多数 CDN 利用 DNS 来截获和重定向请求；这种使用 DNS 的一个有趣讨论见［Vixie 2009］。我们考虑用一个简单的例子来说明通常是怎样涉及 DNS 的。假定一个内容提供商 NetCinema，雇佣了第三方 CDN 公司 KingCDN 来向其客户分发视频。在 NetCinema 的 Web 网页上，它的每个视频都被指派了一个 URL，该 URL 包括了字符串“video”以及该视频本身的独特标识符；例如，转换器 7 可以指派为 http://video.netcinema.com/6Y7B23V。接下来出现如图 2-24 所示的 6 个步骤：

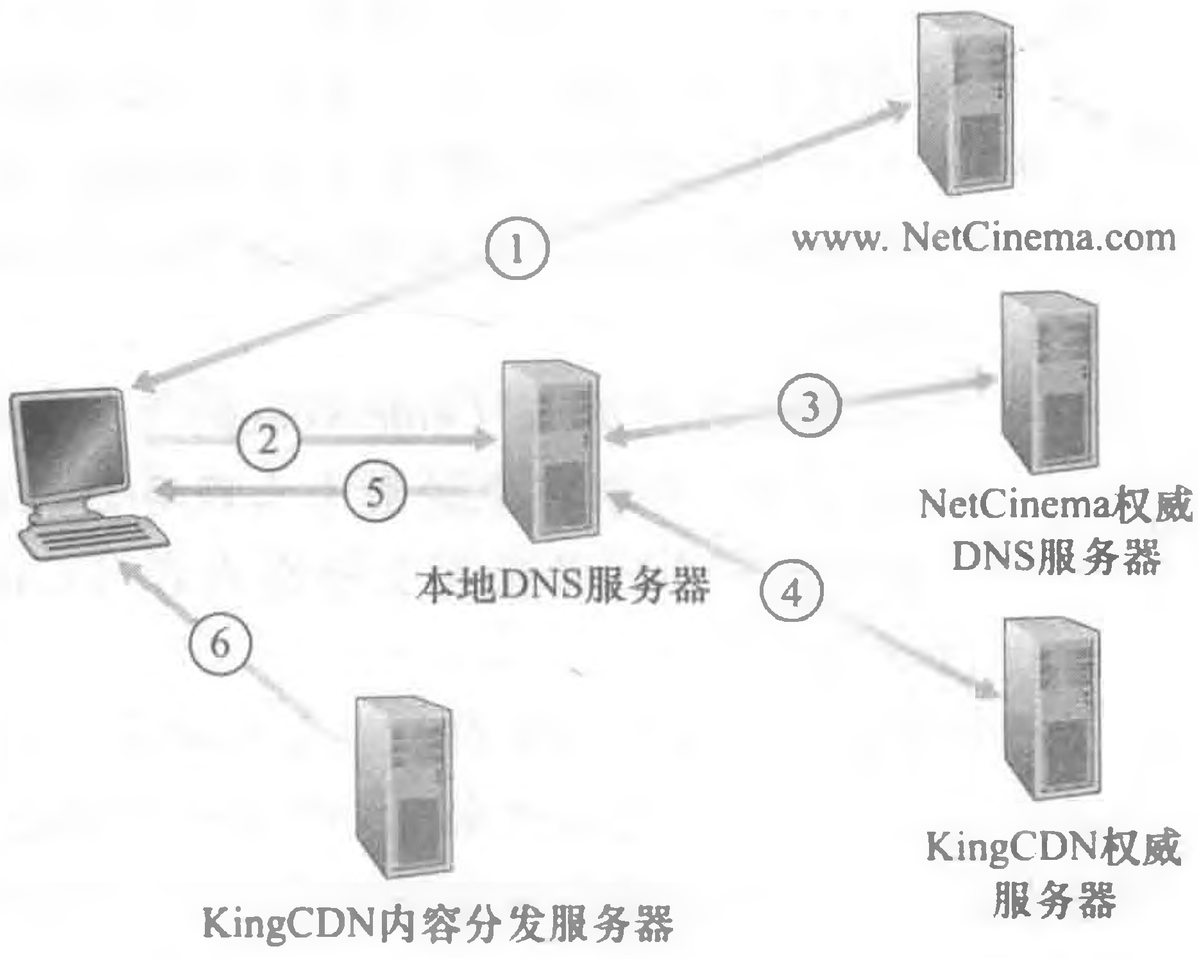

图 2-24　DNS 将用户的请求重定向到一台 CDN 服务器

1）用户访问位于 NetCinema 的 Web 网页。

2）当用户点击链接 http://video.netcinema.com/6Y7B23V 时，该用户主机发送了一个

对于 video. netcinema. com 的 DNS 请求。

3）用户的本地 DNS 服务器（LDNS）将该 DNS 请求中继到一台用于 NetCinema 的权威 DNS 服务器，该服务器观察到主机名 video. netcinema. com 中的字符串“video”。为了将该 DNS 请求移交给 KingCDN，NetCinema 权威 DNS 服务器并不返回一个 IP 地址，而是向 LDNS 返回一个 KingCDN 域的主机名，如 a1105. kingcdn. com。

4）从这时起，DNS 请求进入了 KingCDN 专用 DNS 基础设施。用户的 LDNS 则发送第二个请求，此时是对 a1105. kingcdn. com 的 DNS 请求，KingCDN 的 DNS 系统最终向 LDNS 返回 KingCDN 内容服务器的 IP 地址。所以正是在这里，在 KingCDN 的 DNS 系统中，指定了 CDN 服务器，客户将能够从这台服务器接收到它的内容。

5）LDNS 向用户主机转发内容服务 CDN 节点的 IP 地址。

6）一旦客户收到 KingCDN 内容服务器的 IP 地址，它与具有该 IP 地址的服务器创建了一条直接的 TCP 连接，并且发出对该视频的 HTTP GET 请求。如果使用了 DASH，服务器将首先向客户发送具有 URL 列表的告示文件，每个 URL 对应视频的每个版本，并且客户将动态地选择来自不同版本的块。

2. 集群选择策略

任何 CDN 部署，其核心是**集群选择策略**（cluster selection strategy），即动态地将客户定向到 CDN 中的某个服务器集群或数据中心的机制。如我们刚才所见，经过客户的 DNS 查找，CDN 得知了该客户的 LDNS 服务器的 IP 地址。在得知该 IP 地址之后，CDN 需要基于该 IP 地址选择一个适当的集群。CDN 一般采用专用的集群选择策略。我们现在简单地介绍一些策略，每种策略都有其优点和缺点。

一种简单的策略是指派客户到**地理上最为邻近**（geographically closest）的集群。使用商用地理位置数据库（例如 Quova［Quova 2016］和 Max-Mind［MaxMind 2016］），每个 LDNS IP 地址都映射到一个地理位置。当从一个特殊的 LDNS 接收到一个 DNS 请求时，CDN 选择地理上最为接近的集群，即离 LDNS 最少几千米远的集群，“就像鸟飞一样”。这样的解决方案对于众多用户来说能够工作得相当好［Agarwal 2009］。但对于某些客户，该解决方案可能执行的效果差，因为就网络路径的长度或跳数而言，地理最邻近的集群可能并不是最近的集群。此外，一种所有基于 DNS 的方法都内在具有的问题是，某些端用户配置使用位于远地的 LDNS［Shaikh 2001；Mao 2002］，在这种情况下，LDNS 位置可能远离客户的位置。此外，这种简单的策略忽略了时延和可用带宽随因特网路径时间而变化，总是为特定的客户指派相同的集群。

为了基于当前流量条件为客户决定最好的集群，CDN 能够对其集群和客户之间的时延和丢包性能执行周期性的**实时测量**（real-time measurement）。例如，CDN 能够让它的每个集群周期性地向位于全世界的所有 LDNS 发送探测分组（例如，ping 报文或 DNS 请求）。这种方法的一个缺点是许多 LDNS 被配置为不会响应这些探测。

2.6.4 学习案例：Netflix、YouTube 和“看看”

通过观察三个高度成功的大规模部署 Netflix、YouTube 和“看看”，我们来总结对流式存储视频的讨论。我们将看到，这些系统采用的方法差异很大，但却应用了在本节中讨论的许多根本原则。

1. Netflix

Netflix 在 2015 年产生了 37% 的北美住宅 ISP 中的下载流量，它已经成为美国首屈一

指的在线电影和 TV 节目的服务提供商［Snadvine 2015］。如我们下面讨论的那样，Netflix 视频分发具有两个主要部件：亚马逊云和它自己的专用 CDN 基础设施。

Netflix 有一个 Web 网站来处理若干功能，这些功能包括用户注册和登录、计费、用于浏览和搜索的电影目录以及一个电影推荐系统。如图 2-25 所示，这个 Web 网站（以及它关联的后端数据库）完全运行在亚马逊云中的亚马逊服务器上。此外，亚马逊云处理下列关键功能：

- 内容摄取。在 Netflix 能够向它的用户分发某电影之前，它必须首先获取和处理该电影。Netflix 接收制片厂电影的母带，并且将其上载到亚马逊云的主机上。
- 内容处理。亚马逊云中的机器为每部电影生成许多不同格式，以适合在桌面计算机、智能手机和与电视机相连的游戏机上运行的不同类型的客户视频播放器。为每种格式和比特率都生成一种不同的版本，允许使用 DASH 经 HTTP 适应性播放流。
- 向其 CDN 上载版本。一旦某电影的所有版本均已生成，在亚马逊云中的主机向其 CDN 上载这些版本。

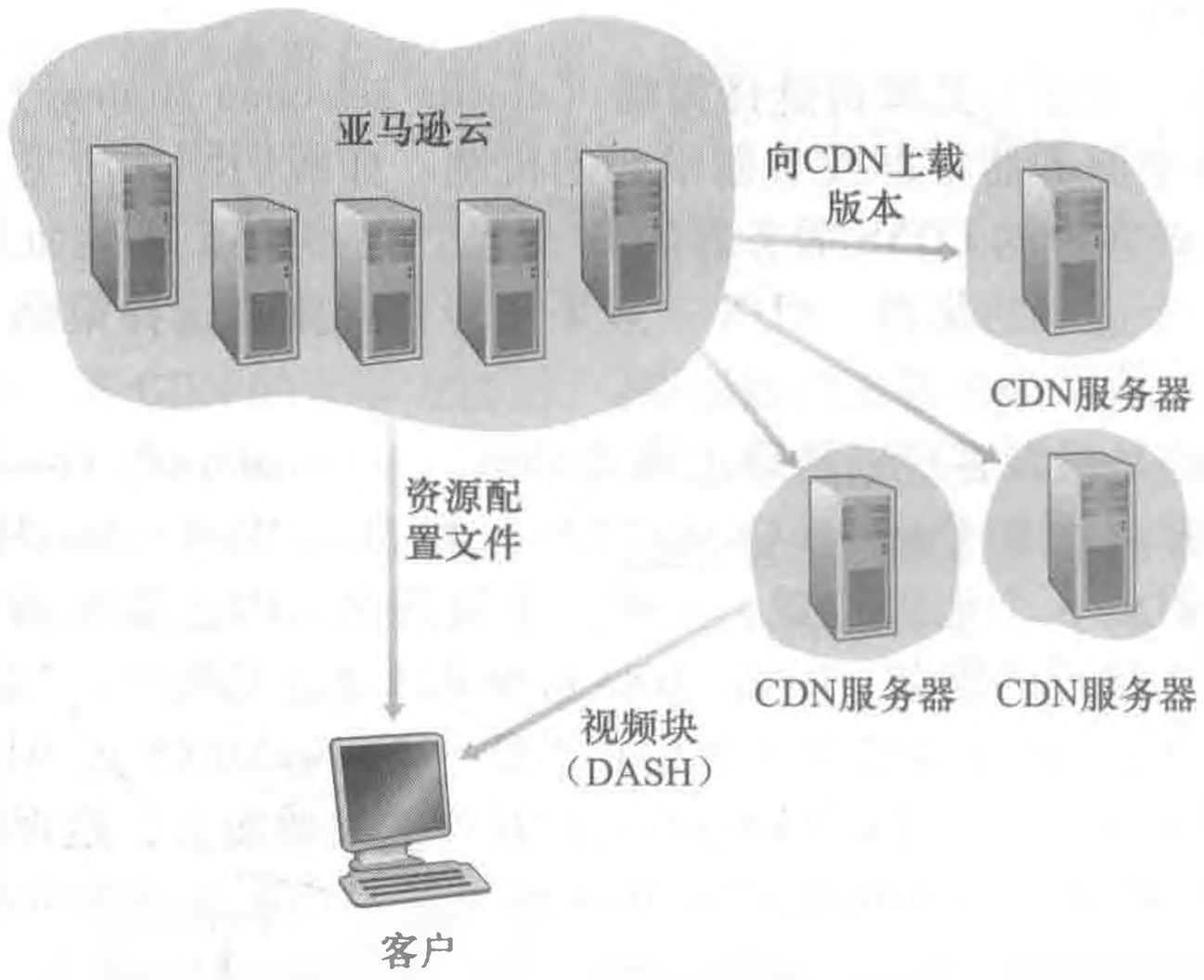

图 2-25　Netflix 视频流平台

当 Netflix 于 2007 年首次推出它的视频流式服务时，它雇佣了 3 个第三方 CDN 公司来分发它的视频内容。自那时起，Netflix 创建了自己专用的 CDN，现在它从这些专用 CDN 发送它所有的视频。（然而，Netflix 仍使用 Akamai 来分发它的 Web 网页。）为了创建它自己的 CDN，Netflix 在 IXP 和它们自己的住宅 ISP 中安装了服务器机架。Netflix 当前在超过 50 个 IXP 位置具有服务器机架；存放 Netflix 机架的 IXP 清单可参见［Netflix Open Connect 2016］。也有数百个 ISP 位置存放 Netflix 机架，同样可参见［Netflix Open Connect 2016］，其中 Netflix 向潜在的 ISP 合作伙伴提供了在其网络中安装一个（免费）Netflix 机架的操作指南。在机架中每台服务器具有几个 10Gbps 以太网端口和超过 100TB 的存储。在一个机架中服务器的数量是变化的：IXP 安装通常有数十台服务器并包含整个 Netflix 流式视频库（包括多个版本的视频以支持 DASH）；本地 IXP 也许仅有一台服务器并仅包含最为流行的视频。Netflix 不使用拉高速缓存（2.2.5 节）以在 IXP 和 ISP 中扩充它的 CDN 服务器，反而在非高峰时段通过推将这些视频分发给它的 CDN 服务器。对于不能保存整个库的那些