第 3 章　CCNx 安装指导

3.1　CCNx 网络结构

在 CCNx 的网络结构中，每个节点都是平等的，不过扮演的角色可不尽相同：可能是数据源、中间节点、客户端等等。CCNx 的核心是 ccnd 进程，它支持了数据包的转发和缓存。组成 ccnd 最重要的 3 个数据结构分别是：FIB(Forwarding Information Base)、CS(Content Store)和 PIT(Pending Interest Table)。其中，FIB 表用来做路由表，CS 则用作缓存经过的数据包，PIT 表用来匹配、记录那些未处理的兴趣包。

图 3.1 展示了一个 ccnd 体系架构中各个节点之间的通信情况。

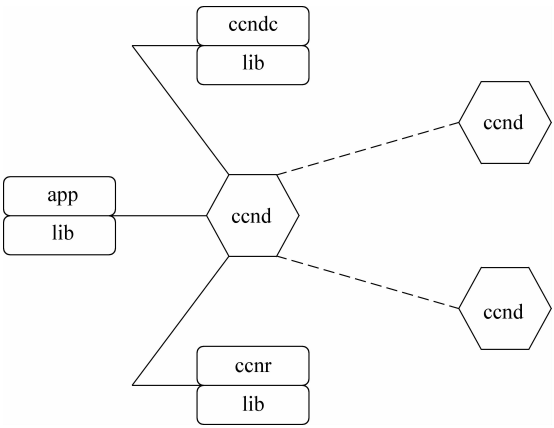

图 3.1　ccnd 间的通信模型

3.2　CCNx 代码安装

3.2.1　获取代码

如果想下载一个打包版本，最好下载最新的。如果对敏捷开发感兴趣，强烈建议从 Github 下载代码(详情请参见文献[1])。代码库正在不断增容，并随时间不断完善，但是却并不总是能够保证能够向下兼容旧的版本，因为这是一个研究项目而非产品。

详读下载的代码中的 README 文档。需要特别注意的是，在继续操作之前，可能需要为平台安装一些必需的第三方软件。

3.2.2　安装编译

如果有问题，可以先看看 ccnx-dev 或 ccnx-users 列表的文档中是否有人之前也遇到了

与你相同的麻烦。如果没有,可以留言请求帮助。过程记录得很详细,所有列出的步骤也是有原因的。例如,确定你的平台是否完成了所有前置环境的安装。

1. 准备工作

(1) 详细过程请阅读官方文件/ccnx-0.7.1/README。

(2) 安装平台为 Ubuntu Linux 13.04。

(3) 为方便安装,直接在 root 用户下进行安装。

① 安装 CCNx 项目中 C 语言的依赖包,主要包括 apt-get、install、git-core、python-dev、libssl-dev、libpcap-dev、libexpat1-dev 和 athena-jot。

安装 libcrypto(版本号为 0.9.8 或以后版本),用 apt-get 可以很方便地安装。

安装 libxml2,可以使用 apt-get,具体的包名字可以用 apt-cache search 查找一下。

如果要使用 vlc 和 wireshark 插件,请阅读相关文档(见 3.2.3 节,暂时可以不安装)。

② 安装 Java SDK。

安装 Sun 公司的 JDK,注意不能安装 openjdk。现在 Sun 被 Oracle 收购了,需要到 Oracle 的网站下载 JDK,最好安装 1.6 版本(网址 http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u29-download-513648.html)。直接运行下载的 jdk-6u29-linux-i586.bin,如果不能运行,需要修改文件的可执行权限。

③ 设置环境变量,需添加以下的环境变量:

```
export JAVA_HOME=/opt/jdk1.x
export JRE_HOME=/opt/jdk1.x/jre
export CLASSPATH=.:$JAVA_HOME/lib:$JRE_HOME/lib:$CLASSPATH
export PATH=$JAVA_HOME/bin:$JRE_HOME/bin:$PATH
```

为保证机器重启后,环境变量仍然存在,以上的这些变量需要写到 profile 里面去。也可以直接写到/root/.profile 文件的末尾。

④ 安装 ant。

从 http://ant.apache.org/bindownload.cgi 下载 ant 最新版 apache-ant-1.8.2-bin.zip。

下载后解压,然后添加环境变量。

假定 ant 安装目录是/usr/local/ant,以下设置环境变量:

```
export ANT_HOME=/usr/local/ant
export JAVA_HOME=/usr/local/jdk-1.5.0.05          //如果 JDK 正确安装,则不需要此设置
export PATH=${PATH}:${ANT_HOME}/bin
```

同理上面配置的环境变量也要写到启动脚本里(.profile),否则重启系统就不能正常工作了。

打开新的 shell 后,输入 ant,可以检查一下 ant 是否已安装,如果出现以下消息:

```
Buildfile: build.xml does not exist!
Build failed
```

这表明 ant 是工作着的。这条消息的意思是需要为用户的项目单独写一个 buildfile 文件。

运行“ant -version”,可以看到以下输出:

```
Apache Ant version 1.7.1 compiled on June 27 2008
```

2. 编译 ccnx

进入 ccnx 的目录：

```
./configure
make
make test                    //测试编译是否正确,虽然可以省略,但最好不要
make install
```

至此，可以运行 ccnchat，验证安装是否成功。

打开一个终端，运行：

```
ccndstart
ccnchat ccnx:/test_room
```

可以再打开几个终端，运行 ccnchat ccnx:/test_room，这样就可以进行简单的聊天了。假设 ccnx 安装在 166.111.137.72 那么在浏览器中通过 http://166.111.137.72:9695 可以查看系统的运行状态信息。

3. 确认安装

(1) 运行 ccnd。

执行 ccndc，建立一些简单的路由。使用 ccnd 的内置 Web 服务器(http://localhost:9695)，可以看到有关 ccnd 的转送表和缓存的详细信息，如缓存的统计信息和转发表等。

通过命令行设置环境变量 CCN_LOG_DIR 以启动日志程序，并通过日志文件查看输出内容。

通过环境变量 CCND_DEBUG 来启动 ccnd 的日志程序。

(2) ccnChat。

运行一些简单的应用程序。最好上手的一个是 ccnChat。用户可以在一台机器上运行多个 ccnChat，它们可以相互通信。如果用户的转发设置是正确的，则可以利用 ccnChat 在多台机器上进行交谈。

如果知道机器之间的转发设置是正确的，可以考虑建一个节点仓库 repo，并加载一些视频数据进去，然后可以运行 VLC 将它们读取出来。阅读 README 文件中的应用程序列表，看看你能做些什么。

3.2.3 CCNx 工具

在这个版本的 README 文件中有一个非常全面的应用程序和工具列表。当用户准备开始时，阅读此文件对用户非常有帮助。

1. ccngetfile/ccnputfile

ccnputfile/ccngetfile 是一对应用程序，用于将文件系统或网页中的文件写入 CCNx，以及从 CCNx 中读取出面向文件的数据。如果同时启动，它们可以直接将数据写入对方；否则，ccnputfile 会将数据加载到仓库中，而 ccngetfile 会再从仓库中将数据读出。

2. ccngetfile

ccngetfile 获取一个发布的 CCNx 内容，并将其写入本地文件中。用法如下：

```
ccngetfile [-javaopts <options>]
[-debug <portno>]
[-unversioned]
[-timeout millis]
[-as pathToKeystore]
[-ac]
ccnxname filename
```

参数含义如下：

(1) unversioned 表示不查找版本，在路径 ccnxname 下进行非版本的内容获取。

(2) timeout 表示在流读取过程中的最大超时时间。

(3) log 表明日志等级，与 Java 的日志等级一致。

(4) as 设定用以解密内容的用户名称。

(5) ac 强制 ccngetfile 遵循访问权限控制。如果用户不允许在此命名空间中读取信息，那么解密失败。

(6) debug 允许 eclipse 远程调试器挂载到该端口进行调试。

(7) javaopts 允许使用 Java 的附加属性与选项。

3. ccnputfile

ccnputfile 将一个内容发布到 CCNx 上，注意这条命令必须在运行 ccnr 的主机上运行才能具有相关权限。用法如下：

```
ccnputfile [-javaopts <options>]
[-debug <portno>]
[-v]
[-raw]
[-unversioned]
[-local | -allownonlocal]
[-timeout millis]
[-log LEVEL]
[-as pathToKeystore]
[-ac]
ccnxname filename|url
```

大部分参数与 ccngetfile 相同，下面只解释不同的参数：

(1) raw 发布内容不上传到仓库。这个模式只有在有一个匹配的 ccngetfile 运行时才能成功。

(2) local 标识文件将上传到本地仓库。这是默认行为。

(3) allownonlocal 标识本地或非本地仓库均可用以保存该文件。

4. ccncat

ccncat 是用 Java 编写的一款简单的聊天程序，用来测试连通性的简单方法。ccncat 将通过流读取 CCNx 内容的数据并写入 stdout。ccncat 将获取路径下最新的版本，然后读取之。用法如下：

```
ccncat [-h]
```

```
[-d flags]
[-p pipeline]
[-s scope]
[-a]
ccnxnames filename|url
```

参数意义如下：

(1) h 打印帮助信息。

(2) d 标识调试信息，可以是以下任意标志位之和：

```
NoteGlitch=1, NoteAddRem=2, NoteNeed=4, NoteFill=8, NoteFinal=16, NoteTimeout=
32, NoteOpenClose=64;
```

(3) p 设置流水线的大小，默认为 4。

(4) s 设置 Interest 包的域，可以是 0(缓存)、1(本地)、2(邻居)或 3(无限制)，默认为 3。

(5) a 允许获取过期信息。

5. ccnrm

ccnrm 将任何符合前缀的本地缓存内容对象标记为已过期。用法如下：

```
ccnrm [-o outfile] ccnxname
```

参数只有一个-o，如果给出，则将所有此次被标记出的内容对象写入文件。

6. ccnls/ccnlsrepo

使用 ccnls 可以列出 ccnd 缓存的内容。ccnls 尝试列出某一命名层级下一级中所有可用的名字分量，用法如下：

```
ccnls ccnxname
```

使用 ccnlsrepo 的命令行可以列出仓库中的内容或其他名称枚举协议的响应内容。用法如下：

```
ccnlsrepo
```

7. ccnFileProxy

ccnFileProxy 是普通文件的 CCN 代理工具程序，使文件系统中的文件内容变得可被 CCNx 协议访问。该工具并没有进行很好的优化，但是建立 CCN 仓库并加载内容是让 CCNx 协议获取内容数据的一种简单的替代方法。

8. ccnsendchunks

一个简单的应用程序，收到 interests 时产生块或者每秒产生一次，两者取其快者。它的数据至少可以被 ccncatchunks 和 ccncatchunks2 读取。在一台机器上启动 ccnsendchunks，同时在另一台机器上启动 ccncatchunks2。

9. wireshark 插件

安装带 CCN 协议插件的 wireshark 网络分析软件，过程如下：

(1) 参考 ccn/apps/wireshark/README-wireshark-1.6.txt。

(2) 下载 wireshark 源码编译安装。编译 wireshark 需安装 yacc(语法分析)、flex(词法分析)、gtk+(图形库)。

(3) 安装完后,直接运行可能会报错,用 which 查看 wireshark 程序,是否在目录/usr/local/wireshark 中。

如果不在该目录下,可以直接在安装目录下运行,或者修改 $PATH 环境变量,修改符号链接等,即可成功运行 wireshark。

10. VLC 插件

一个可以读取 ccnx 数据的标准视频播放器插件。启动一个仓库,加载一些内容进去,然后用 VLC 播放这些内容。

11. ccnexplorer

ccnexplore 是一个图形界面的简单文件浏览器,用来浏览储存在 CCNx 中的数据。

3.3 代码开发

3.3.1 起步

用 Java 语言的话,可以从 ccnChat 和 ccnFileProxy 开始。它们都是非常简单的程序,而且它们中 CCNx 部分代码都非常小。ccnFileProxy 主要展示了登记过滤、Interest 处理和流 API 的使用。ccnChat 主要展示了称为"网络对象"的使用,它使用 CCNx 的版本化对象作为备份存储,而不是一个数据库。

用 C 语言的话,可以从 VoCCN 开始(在下载页面作为一个单独的 tar 文件分发)。它展示了 C 库的基本使用(虽然可能并不总是跟踪最新的 API),以及如何将 CCNx 和现有的 C 应用程序进行整合。CCNx 的代码量相对较小,并做了很好的本地化。另外,它还使用了一个 CCNx 封装的 RTP(Real-time Transport Protocol),或许在其他很多场合都会有用。

3.3.2 可能的问题

CCNx 的有些地方并不总是那么直观和明显,这里有一些最常遇到的情况的列表。

1. 用户的数据发送不了

CCNx 完全是面向 pull 的。除非有人发布了对数据的兴趣包,否则不允许写数据。这个约束在程序数据栈中实际上是非常广泛的,如果违反了它,可能会导致诸如 WaitForPutDrainException 等有趣的错误。如果正在编写的应用程序需要创建数据,那么为了使它们可以移动,还需要有一个应用程序来使用这些数据。一个应用程序如果知道如何向仓库写数据的话,那么最简单的方法就是运行一个可以作为普通消费者被排序的仓库。当然也可以量身订制一个消费程序来取用户想写的具体数据(如面向文件的数据,在运行 ccnputfile 的同时运行 ccngetfile)。如果仅仅是想让程序运行,那么也可以让程序不产生数据或仅仅是产生一些 interest。这里有个 C 程序 ccnslurp 在做类似的事情,还有一个 Java 类(Flosser,类似于 mental floss,通过 ccnd 取一些东西仅仅是为了这样做。)可以看看如何使用 Flosser 的 Java 测试以及 ccnslurp 的用户手册。

2. 遇到奇怪的验证错误,未能找到密钥

在 CCNx 中,所有的数据都是有签名的,而且公钥以另一种数据形式分发。我们进行了信任模型的实验,可以利用这些密钥去映射有意义的用户身份,但是在网络核心层,它们

就仅仅是密钥而已。我们不希望用户一开始就担心它们如何生存、是否需要把它们复制到别的地方等这类问题。因此用户会默认的发布用户密钥,并使得从程序中获取它们的过程相对自动(因此程序会知道特定的密钥签名了多种数据,尽管并不知道这个密钥是属于谁的)。当用户第一次使用 CCNx 程序时,函数库会生成一个密钥,而且如果不特别指出,Java 库会把公钥发布到默认的以 ccnx://ccnx.org 为前缀的命名空间,这可以通过多种 Java 属性和环境变量进行控制。如果仅仅添加了用户的应用程序的特定的命名空间到 ccnd 的路由表,而没有将用户发布的密钥的命名空间加入路由表中,那么客户端验证将会失败。在此期间,用来处理这个问题的选项包括:

指定发布的密钥的命名空间在能够全球路由的应用程序的命名空间之下,例如指定在以下命名空间中。

ccnx:/ccnx.org

或者

ccnx:/,

3. ContentExplorer 找不到东西

内容管理器使用名称枚举协议(Name Enumeration Protocol)查找和列出内容。它并不会显示 ccnd 缓存的任意数据,它只会列出那些适用于 NE 协议的内容。目前仓库和 ccnFileProxy 都包含在这个协议框架内。因此,如果没有运行仓库或者 ccnFileProxy,或者它们中没有内容,那么 contentExplorer 将找不到任何东西。如果只是想看看 ccnd 缓存中有什么内容,可以使用命令行程序 ccnls。

4. 遇到了奇怪的构建/运行时错误

请确定已按照 README 文档中的构建指令操作。如果没有满足一些依赖关系,将会导致一些难以追查的奇怪错误。已知的罪魁祸首包括没有最新版本的 OpenSSL,或在 XOS 中没有设置 JAVA_HOME(Java 控制面板会做这些,用户无须设置 JAVA_HOME,但是出现错误时很难想象这就是问题所在)或者在 ubuntu 上运行 Java 以外的东西(尽管 icedTea 支持很多东西)。Cygwin 上的构建失败,通常是由于缺少某个我们所列出的需要安装的包;Wireshark 也有类似情况。一旦确定是按照所有说明操作的,但是仍然有问题,可以给开发商发邮件。

3.4 CCNx 库

CCNx 发行版代码 8.0 中给出了 CCNx 类库中的一个不完全列表,让用户可以直观的知道库里有什么或者没有什么。这些库的发展非常迅速,所有这份列表总是要滞后于 github 上的代码,但是通常会早于最新的版本包。

3.4.1 共同组件

C 库和 Java 库共同的组件包括:

(1) 支持编码/解码的 ccnb。

(2) ccnb 使用一种紧凑的 XML 编写。ccnb 编码用于 CCNx 信息的传输,并有助于应用程序存储和传输内容。

(3) 核心的 CCNx interest-data 协议

具体体现在以下操作：

异步数据检索 API("express Interest")：登记一个 Interest，当收到响应数据时回调应用程序。当响应超时时，这两个库都会自动重发 Interest。当然也可以取消自动重发。

同步数据检索 API("GET")：提出一种阻塞的 get 方法，即程序可以发布 Interest 并阻塞，直到返回数据或超时。

异步 Interest 检索 API：注册一个过滤器，使得 ccnd 发送符合过滤条件的登记者的 Interest(而不是数据)。登记者则可以选择返回数据或生成响应数据。

支持自动密钥生成。

支持基本的签名和验证。

3.4.2 Java 库

Java 库包含了大量的高等级 API 和更先进的安全功能，旨在使 CCNx 编程变得更简单。很多这些组件还在积极的开发中，一旦完成，有用的还会移植到 C。

一组资料、数据和命名公约体现在下列功能中：

- 版本，包括最新版本的内容检索；
- 密钥发布；
- 名称枚举；
- 元数据发布；
- 命名空间管理(初步)；
- 基于加密的访问控制(一个特定的访问控制方案，也是初步)。

一组读写分段和使用 Java 输入/输出流接口的可控版本数据的抽象流。

使用一组数据格式使得版本化“网络对象”可以将自己序列化和反序列化到 CCNx 中，包括：

- ccnb 二进制编码；
- Java 序列化格式；
- 对象类型格式定义。

支持向仓库写入流和对象，但不能直接响应原始 interest。从仓库中检索数据和检索其他 CCNx 数据一样；它对请求者来说是透明的，无论数据是来自仓库、缓存或其他类型的服务器。

低级别支持利用程序提供的密钥对内容进行加解密。

初步支持利用可插拔密钥分发策略对内容进行自动加解密。

一个 CCNx 的链接概念封装，包括初步支持对分段特定内容自动取消引用链接。如果你通过链接检索到一个文件或一个网络对象，那么这个链接会被取消并且目标数据将会被检索，而仅仅保留链接信息用于验证。

自动签名和验证，同时支持每包签名和多包聚合签名。函数库目前使用 Merkle 哈希树作为默认的聚合技术，同时也支持其他聚合技术很好地工作。这个扩展机制并没有大量运用，可能还需要调整。

自动密钥检索和数据包验证。无法验证的数据包将会被丢弃。

挂接到包检索过程的程序级信任决策机制，使得程序只检索它们认为值得信任的数据包。

这些功能部件可以被组合来实现多种信任模型。我们期望函数库中很快会出现一些简单通用的信任模型以供程序使用。

3.4.3 C 库

除了上面介绍的核心功能，C 库还包括：

- 细分的基本支持；
- 版本化和内容的最新版本检索的基本支持；
- Interest 包和 Data 包签名生成库；
- Interest 包和 Data 包签名的验证，以及基于 Merkle 哈希树聚合的数据包签名验证；
- 一些元数据支持(头检索)。

注意：C 库无法对数据加密，也无法对用 Java 库进行加密的文件进行解密。一些基于 C 的 CCNx 应用，如 VoCCNN，支持程序级的加密。它同样也不支持向仓库写入数据，尽管它可以从仓库中读取数据。如上所述，从仓库中检索数据和检索其他 CCNx 数据一样；它对请求者来说是透明的，无论数据是来自仓库、缓存或其他类型的服务器。

参 考 文 献

[1] CCN wiki. https://www.ccnx.org/wiki.

[2] wireshark. http://www.wireshark.org.

第 4 章　内容中心网络流媒体

视频分发系统在视频流量飞速增长的今天，对降低网络负载，提升视频分发的速度具有重要的意义。本章节旨在阐述视频分发系统的意义，以及现有视频分发系统所面临的挑战。

4.1　视频分发需求及 Internet 架构现状

4.1.1　互联网视频内容

根据 Cisco 对 Internet 流量的分析报告[1]，近几年来，视频应用获得了飞速的发展，视频流量占总 Internet 流量的比例逐年攀升，从 2010 年的 4672PB 迅速攀升到 2012 年的 12146PB，占 Internet 流量的 57%，如图 4.1 所示。而且按照预测，这种趋势在未来的几年内还将继续下去，如图 4.2 所示。

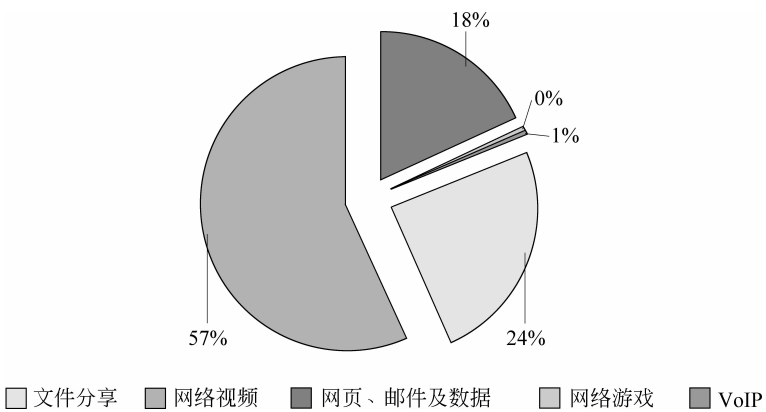

图 4.1　2012 年 Internet 流量成分

数据来源：Cisco VNI，2012

然而，在 Internet 流量的主要增长动力来源于视频之时，Internet 体系结构对于视频分发的需求却没有有效的应对措施[2]。所谓视频分发，指的是 Internet 上，将视频从源传输到终端用户的过程的有效性可以通过延时、可用性以及速率等指标来进行测量。一种视频分发方式的延时越低、可用性越高、速率越高，该方式进行视频分发的效果就越好。

4.1.2　Internet 架构问题

然而，传统的 Internet 体系结构 TCP/IP，提供的是一种端到端的网络体系，这对于视频以及其他内容分发来讲，并不是一种有效的方法。TCP/IP 就如同电话机一般，在进行内容分发时，有几个终端用户，就需要建立几条连接到源头的链接，势必带来极大的负载。而视频分发的需求在电视机这样的广播网络下，能够获得有效的传输性能，增加一个终端用户对广播源头带来的负载开销，几乎可以忽略不计。因此，如何在现有网络的基础上进行扩

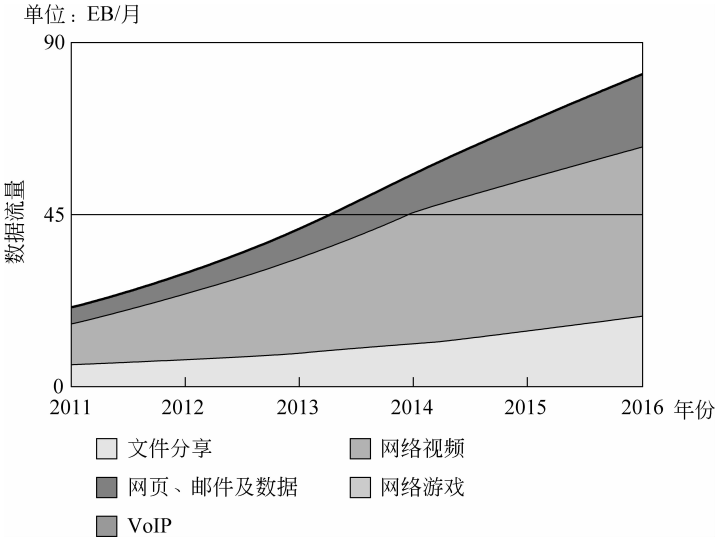

图 4.2　Internet 流量趋势及组成

数据来源：Cisco VNI，2012

展，使之适应于日益增长的内容分发需求，成为摆在人们面前亟待解决的重要课题。

针对这一问题，工业界和学术界都在努力进行尝试，各自提出了不同的解决方案。

4.2　HLS 标准概述

HLS(HTTP Live Streaming)协议是基于 HTTP 协议的流媒体传输协议[3]。该协议将媒体流切分成足够小的片段，而后通过 HTTP 协议进行传输。

和传统的流媒体传输协议相比，HLS 具有良好的业界标准支持，广泛支持不同的系统平台以及设备，尤其是 iOS 系统等移动设备平台，HLS 是少数具有良好支持的流媒体协议。目前，HLS 的建议稿草案已经提交给 IETF。

4.2.1　HLS 客户端请求流程

HLS 的客户端请求流程如下：

(1) 客户端请求 HTML 页面，该页面中包含 HTML5 的 video 标签，指向视频索引 M3U8 文件。下面以 http://example.com/M3U8 为例；

(2) M3U8 文件中并不包含实际的视频内容，它索引了视频的每一个片段，以及相应的附加描述。通过 M3U8 文件，客户端可以进一步找到视频片段文件，这些片段都是 ts 格式的文件；

(3) 客户端对 ts 格式文件解码，进行播放。

基于 HTTP 的设计，使得该系统能充分发挥 HTTP 成熟的技术优势。例如在客户端获取 HLS 视频的过程中，由于视频的请求是通过 HTTP 请求完成的，中间服务器能够自然地缓存视频片段，以供未来的请求使用，而这一过程由于完全通过 HTTP 实现，并不需要额外的系统支持。

鉴于 HLS 协议具有上述优良的特性，我们的视频分发系统将基于 HLS 协议来实现。

4.2.2 M3U8 格式

M3U8 格式[4]又名 M3U 格式，是一种存储媒体播放列表的文件格式。它是纯文本文件，其中标明了一个或多个媒体文件的位置。一个 M3U8 文件的样例如下。

```
#EXTM3U
#EXTINF:123, Sample artist-Sample title
C:\Documents and Settings\I\My Music\Sample.mp3
#EXTINF:321,Example Artist-Example title
C:\Documents and Settings\I\My Music\Greatest Hits\Example.ogg
```

其中，#EXTM3U 是文件头，标识该文件是 M3U8 文件。此后每一个片段，都指示了某一片段的媒体。例如：

```
#EXTINF:123, Sample artist-Sample title
C:\Documents and Settings\I\My Music\Sample.mp3
```

此片段中，#EXTINF 头注明了媒体片段的长度(123)和标题(Sample artist Sample title)，此后一行不以“#”开头，标明了媒体文件的路径。这里的路径，既可以是文件系统的路径，也可以是一个 URL。既可以是绝对路径，也可以是相对于播放列表位置的相对路径。

在 HLS 中，每一个媒体流，都是由一个或多个视频片段组成的“播放列表”，因而可以通过 M3U8 格式进行索引。一个标准的 HLS 索引文件[3]，包含了所有媒体片段的路径，它们按照播放顺序排列。除此之外，还需要在文件开头包含几个 HLS 特有的标识：

(1) #EXT-X-TARGETDURATION　该标识注明了媒体片段的最大长度。所有视频片段的长度都必须不大于这一标识。这一标识在 M3U8 文件中出现且仅出现一次，并应用到整个 M3U8 文件中。

(2) #EXT-X-MEDIA-SEQUENCE　注明了第一个出现在 M3U8 中的媒体片段的序列号。此后每一个片段的序列号等于其之前的序列号加 1。这一标识的作用是，在 HLS 进行流媒体播放(如现场直播)时，不断递增来注明当前播放的位置。

一个 HLS 标准的 M3U8 文件如下。

```
#EXTM3U
#EXT-X-TARGETDURATION:8
#EXT-X-MEDIA-SEQUENCE:2680
#EXTINF:8,
https://priv.example.com/fileSequence2680.ts
#EXTINF:8,
https://priv.example.com/fileSequence2681.ts
#EXTINF:8,
https://priv.example.com/fileSequence2682.ts
```

在 HLS 的标准草稿[3]中，为了更好地支持 HLS 播放的体验，还为 M3U8 格式添加了如下扩展标识：EXT-X-BYTERANGE、EXT-X-TARGETDURATION、EXT-X-MEDIA-

SEQUENCE、EXT-X-KEY、EXT-X-PROGRAM-DATE-TIME、EXT-X-ALLOW-CACHE、EXT-X-PLAYLIST-TYPE、EXT-X-STREAM-INF、EXTX-I-FRAME-STREAM-INF、EXT-X-I-FRAMES-ONLY、EXT-X-MEDIA、EXT-XENDLIST、EXT-X-DISCONTINUITY，以及EXT-X-VERSION。

4.2.3 TS文件格式

TS文件格式，即MPEG Transport Stream[5]，是一种用于传输和存储音频、视频以及PSIP协议的标准格式，它同时也被广泛应用在诸如DVB和ATSC的广播系统之中。TS文件标准描述了用于包装基于数据包的基础流的容器格式，并提供纠错和流同步的支持，以在信号缺失时维护数据的完整性。

Data包是组成TS文件最基本的单元，在通常的系统中，每个包的长度都是188B，包含了一些控制位以及媒体数据。而在特殊的系统中，传输介质可能会在数据包上添加一些。数据包的控制位描述如表4.1所示。

表4.1 部分MPEG TS流数据包的格式

名　　称	位　　数	描　　述
同步位	8	永远为0x47
传输错误指示器	1	如果不能修正错误，则由解调器设置该位，以告知分路器这个数据包包含了不可修正的错误
载荷单元起始标志	1	1表示为PES数据的起始或是PSI，否则为0
传输优先级	1	1表示该数据包比起其他相同PID的数据包具有更高的优先级
PID	13	数据包ID
混淆控制	2	00为无混淆，01为反转，10为偶键混淆，11为奇键混淆
适应项存在性	2	01只有载荷，没有适应项；10只有适应项；11同时有适应项和载荷
连续性计数器	4	只有在存在载荷时才会增加①
适应项	0或更多	取决于标志位
载荷数据	0或更多	取决于标志位

① 此处以上总的位数为32，被称为TS的4字节前缀或是TS头。

在HLS系统中，协议使用TS格式来作为媒体文件的格式，但是比起普通的TS文件，HLS的片段有更多严格的限制。例如，每一个片段的起始位置都需要包含一个节目关联表(Program Association Table)和节目映射表(Program Map Table)，且每个片段必须包含单一的MPEG-2节目。除此之外，连续的两个视频片段，必须具有连续性，这意味着时间戳以及连续性计数器(Continuity Counters)都必须没有跳变。连续性的保证，只能在两种情况下可以忽略：该片段是M3U8的第一个片段，或者该片段在M3U8中标明了#EXT-X-DISCONTINUITY标识。

有鉴于现有视频分发系统所面临的机遇和挑战，本节提出借助内容中心网络设计的视频分发系统。接下来将对分发系统的系统架构进行详细的阐述。

4.3 CCNHLS 系统

本节主要描述了 CCNHLS 系统的架构，以及各个模块的职责。系统的整体架构图如图 4.3 所示。

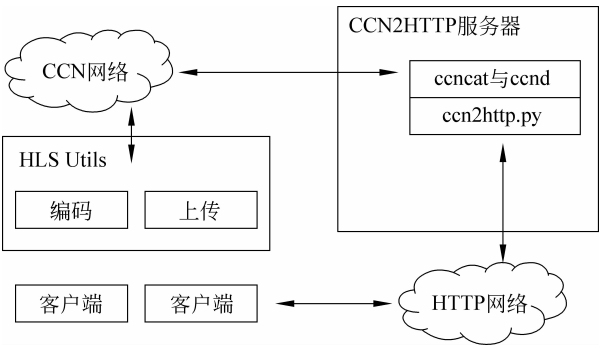

图 4.3 CCNHLS 系统整体架构图

CCNHLS 主要包含如下几个部分：

(1) 内容源转码及切分处理模块　负责提供对不同视频格式的支持，通过将各类视频格式转换为标准 HLS 所支持的 ts 格式，并对其进行切分来完成。

(2) 内容中心网络的架设与配置模块　整合利用现有的工具，在本地服务器上搭建纯内容中心网络的服务，提供路由、获取内容的功能支持。

(3) CCN2HTTP 中间服务器则将任何捕获的 HTTP　请求转换为内容中心网络的请求，并将返回的响应相反地转换为 HTTP 的响应，以此达到内容中心网络和现有 IP 网络的衔接。下面将就各个部分作具体介绍。

4.3.1 内容源转码及切分处理

并不是所有的视频格式都支持通过 HLS 协议播放，只有经过苹果公司提供的工具 mediafilesegmenter 转换的视频流才能获得支持。mediafilesegmenter 提供了广泛的格式支持，几乎能够将任意的视频文件转换成 HLS 所需的 M3U8 文件和 ts 文件。

然而，想要使用苹果公司提供的工具，必须加入苹果公司的开发者计划。曾经有人在 2009 年尝试使用开源工具完成 mediafilesegmenter 的工作。首先使用 ffmpeg 对视频进行了转码，然后使用自行编写的切分器进行视频的片段切分。

然而，由于切分器依赖的 ffmpeg 软件接口的变更，时至今日，切分器已经不能正常工作。因此，我们需要寻找替代的方法，进行视频的切分。

除此之外，使用 ffmpeg 进行编解码也有多种参数的选择[6,7]，经过尝试以后，系统选取了兼容性最好的一种。

4.3.2 内容源转码

Linux 下的内容源转码由开源工具 ffmpeg 完成。如果系统没有自带 ffmpeg，则需要重

新编译一份。

首先从 ffmpeg 官网下载 ffmpeg 的最新源码，然后使用以下命令进行编译配置[6]。

```
configure --enable-gpl
--enable-nonfree
--enable-pthreads
--enable-libfaac
--enable-libfaad
--enable-libmp3lame
--enable-libx264
```

上述编译选项会安装 MPEG-2 编解码、x264 编解码等模块，这些模块是 HLS 编解码所必需的。然后使用 make 指令进行编译。

```
make
make install
```

在完成 ffmpeg 的编译和安装后，就可以使用 ffmpeg 进行转码。使用如下指令可以编码成 HLS 所需的、适合于 iPhone 播放的视频流[7]。

```
ffmpeg -i <input video>
-acodec libmp3lame
-ac 1
-vcodec libx264
-s 320x240
-level 30
-f mpegts
<output video>
```

此后，就可以获得完成的 ts 文件，以供下一步的切分。

4.3.3 内容切分

内容切分的工作同样使用 ffmpeg 来完成。ffmpeg 提供了如下的命令行选项，使用户可以在每次转码时指定转码的片段。

(1) -ss 指定转码的起始时间，用 HH:MM:SS 的方式表示。例如，00:00:10。

(2) -t 指定转码的时间长度，用 HH:MM:SS 的方式表示。

用户可以将编解码与切分两个步骤结合起来，使用如下命令，就能够从任意视频源获得所需的第一个长度为 10s 的片段。

```
ffmpeg -i <input video>
-acodec libmp3lame
-ac 1
-vcodec libx264
-s 320x240
-level 30
-f mpegts
```

```
-ss 00:00:00
-t 00:00:10
segment1.ts
```

4.3.4 M3U8 文件的生成

Linux 下没有任何工具进行 HLS 下的 M3U8 文件生成，但是由于 M3U8 是纯文本格式，加上具有官网的样例，我们可以参照其样式自行进行 M3U8 文件的生成。M3U8 的具体格式参照 4.2.2 一节对 M3U8 格式的描述。

HLS 的 M3U8 文件头中，需要标明如下内容：

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
```

在实验系统中，片段的长度一律取为 10s。由于系统并没有有关直播的需求，起始序列号恒定为 0 即可。

另外，按照 HLS 的标准，前后片段存在不连续的情况，就必须提供 # EXTX-DISCONTINUITY 标签。由于使用 ffmpeg 以时间为标度提取片段，在两个视频片段之间，可能出现一定的跳帧现象，MPEG-2 的连续性计数器可能会因此有所改变，因此应该提供 # EXT-X-DISCONTINUITY 标签。然而，# EXTX-DISCONTINUITY 标签的另一个副作用是，导致客户端重置编码器和解析器[3]，这就带来了性能上的损失（片段中虽然有跳帧现象，但是编解码器是完全相同的）。

考虑到多数的编解码器对于轻微的连续性计数器抖动有良好的容错性，并不会带来系统问题，在 M3U8 中，没有加入 # EXT-X-DISCONTIUITY 标签。

最后，为了方便系统的迁移（不同域名、不同 URL、不同文件系统），所有的路径皆使用相对路径标明，使用时只需将 M3U8 和 TS 文件放置在同一目录下即可。

4.3.5 自动化脚本

一部电影长度大致在 120min，也就是 720 个 10s 的片段，不可能全部通过人工来进行编解码切片。因此，使用自动化脚本完成上述步骤是极为必要的。

Python 脚本语言提供了对文件以及字符串良好的操作支持，其简洁的语法也有助于逻辑的表达，因而使用其进行脚本的编写。其命名为 seg.py，完成从视频编解码、切片到生成 M3U8 的所有工作。

命令格式如下：

```
seg.py <input-file><seg-interval-in-seconds><output-dir>
```

参数意义如下：

(1) input-file 输入的视频文件，可以是 ts 以及其他任意格式。

(2) seg-interval-in-seconds 每一视频片段的长度，时间单位为 s(秒)。

(3) output-dir M3U8 文件以及 TS 文件的输出位置。

4.4 CCN2HTTP 中间服务器

CCN2HTTP 中间服务器采用 Tornado 服务器[8]，CCN2HTTP 基于 Python 2.6 的 Tornado 2.2 框架编写。能够在本地的 8888 端口开设 HTTP 服务器，将 HTTP 请求转换为 CCN 请求。

例如，如果我们访问架设了 CCN2HTTP 服务器的主机，访问如下 URL：

```
http://localhost:8888/ec2/videostream-1.ts
```

那么，CCN2HTTP 将会将其转换为 CCNx 的请求 ccnx:/ec2/videostream-1.ts，获取指定命名的内容。

CCN2HTTP 使用 ccncat 来完成获取内容这一任务，ccncat 能够将指定 ccnx 命名的内容输出到 stdout 之中，而 CCN2HTTP 通过 Python 的 subprocess 库捕获 stdout 的输出，并将其重定向到 HTTP 请求上。之所以选择 ccncat 而不使用 ccngetfile，是因为 ccngetfile 还涉及了文件操作，可能进行磁盘读写，而通过将 stdout 重定向的方式，ccncat 将完全运行在内存中，效率更高。而且除此之外，利用 ccncat 工具，避免了对 CCNx 细节的深究，在 CCNx 版本更新时，同样具有完整的兼容支持。

除此之外，为了遵循 HLS 标准的约定，CCN2HTTP 将探测 URL 的后缀并添加相应的 MIME Type。对于以.ts 后缀名结尾的文件，CCN2HTTP 会添加 video/MP2T 的 MIME Type；对于以.M3U8 后缀名结尾的文件，则会添加 application/x-mpegURL 的 MIME Type。这种添加，严格遵循内容的后缀名，因此对于特殊情况，还需要进行特殊处理（如不以.ts 后缀结尾的视频片段，则需要手动添加）。

4.5 小　　结

通过对 CCNx 支持流媒体系统 CCNHLS 的设计以及实现，完成了基于统一标准内容中心网络的视频分发系统。

在设计中，有一个关于内容中心网络数据大小设定的问题值得注意。在目前版本的内容中心网络协议中，每一个内容包的大小固定为 4KB。然而，未来将获得广泛应用的视频数据却需要比 4KB 长得多的长度。一段 10s 的 HLS 视频片段，大小大约为 256KB，如果使用内容中心网络进行请求，就意味着需要进行 256KB/4KB＝64 次内容包的发送，多了 63 个名字，就给客户端以及所有中间节点（PIT、FIB、CS 表的维护）带来 63 倍的额外负担。

而目前大多数的内容中心网络，都是建立在 IP 层之上的覆盖网，对内容的分拆和重新组合的逻辑，已经在 IP 层中有了成熟的实现，并且在内容中心网络下，并不能实现更多的改进。既然如此，为什么不将这些逻辑下移，简化内容中心网络的实现呢？如果未来底层有所改进——例如，网络设备的 MTU 提升到了 256KB——内容中心网络的架构也不需要改动。考虑到视频是内容中心网络的主要负载，将内容包的大小提升至 32KB，乃至 256KB，可能是必要的。

参 考 文 献

[1] Cisco Visual Networking Index: Forecast and Methodology, 2010-2015. 2012, http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html.

[2] Zhang L, Estrin D, Burke J, et al. Named Data Networking (NDN) Project. Technical report, University of California, Los Angeles, October, 2010. http://www.named-data.net/ndn-proj.pdf.

[3] Roger Pantos J. HTTP Live Streaming draft-pantos-http-live-streaming-07, September, 2011. http://tools.ietf.org/html/draft-pantos-http-live-streaming-07.

[4] M3U-Wikipedia, the free encyclopedia, 2012. http://en.wikipedia.org/wiki/M3U.

[5] MPEG transport stream-Wikipedia, the free encyclopedia, 2012. http://en.wikipedia.org/wiki/MPEG_transport_stream.

[6] McDonald C. iPhone HTTP Streaming with FFMpeg and an Open Source Segmenter, 2012. http://www.ioncannon.net/programming/452/iphone-http-streaming-with-ffmpeg-and-an-open-source-segmenter.

[7] Vries N D. HTTP Live Streaming, FFMPEG and FFSERVER, and iPhone OS 3, 2012. http://stackoverflow.com/questions/1093667/http-live-streaming-ffmpeg-ffserver-and-iphone-os-3.

[8] Tornado Web Server, www.tornadoweb.org.