计算机网络前沿领域的研究

徐雨航 520030910180

综述部分

摘要：本文结合计算机网络的五层模型，对计算机网络的一些前沿发展领域做了一定整理和研究。
关键词：网络内计算　可编程网络　拥塞控制　软件定义网络　确定性网络　无损以太网

概述

互联网经过几十年的发展，已经从最初的科研型网络转变为消费型网络，现在正向着生产型网络迈进。各种新型的需求也给传统互联网模型带来新的挑战：AR/VR、全息投影等技术对网络的低时延与高带宽提出更高要求；工业互联网则要求网络具有更好的稳定性和低抖动；分布式云计算、区块链等技术推动者网络向智能化、可定制化发展。

在流量爆炸式增长的大数据时代，为了解决传统互联网暴露出的种种问题，学术工业界展开了诸多创新性研究，涌现出许多新的议题：如分布式网络应用与计算、可编程交换机与软件定义网络已经是网络领域最热门的几个话题；智能化的视频流处理、拥塞控制协议这些传统网络领域的问题也有了长足进展。

下面将分层介绍计算机网络中一些新兴的研究领域和实际应用。

应用层

各种网络应用是计算机网络存在的原因，而应用层正是应用层协议得以存在和网络应用得以实现的地方。它直接与应用程序接口，将应用报文发送给下层，接收并交付来自下层的报文。应用层协议是计算机网络中数量最多的协议。

目前随着5G网络的兴起，一些应用也得以落实，如AR/VR（增强/虚拟现实）技术正受到越来越广泛的使用。然而当下VR行业还没有一个统一的行业标准，生成VR内容的数据格式众多，不同平台之间的格式、协议等兼容性差。我们期待统一的VR应用层协议的推出，或许这项协议也能像HTTP之于因特网一样，深刻改变我们的世界。

传输层

传输层是整个网络体系结构中的关键层次之一，主要负责向两个主机中进程之间的通信提供服务。由于一个主机同时运行多个进程，因此运输层具有复用和分用功能。传输层在给定的链路上通过流量控制、分段/重组和差错控制来保证数据传输的可靠性。

网络传输协议一直是计算机网络领域的热门话题。传统的TCP协议提供了面向连接的，端到端的可靠数据传输，UDP提供的则是无连接的不可靠的服务。然而当下低时延、高带宽的应用场景越来越多，TCP较高的时延和复杂的协议处理成为制约性能的瓶颈。

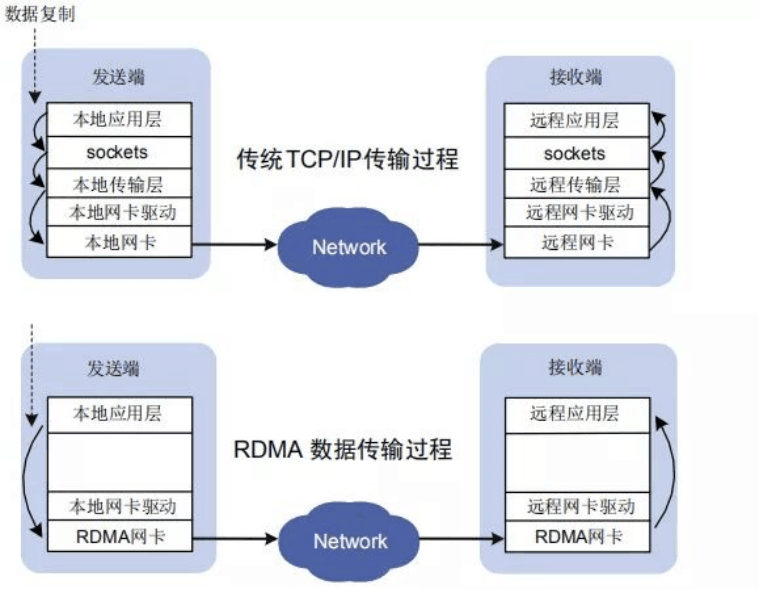
此外，一些传统问题，如拥塞控制也和当下流行的强化学习结合起来，取得了一定进展。

网络内计算与RDMA技术

随着高性能计算、大数据分析的发展，以及分布式网络和云数据库的普及，业务应用有越来越多的数据需要从网络中获取，这对数据传输协议的交换速度和传输性能要求越来越高。

传统的 TCP协议及应用存在着网络传输和数据处理的延迟过大、多次数据拷贝和中断处理、复杂的协议处理等问题。在HotNets 2021的一篇论文TCP is Harmful to In-Network Computing: Designing a Message Transport Protocol (MTP)中，作者还指出TCP协议难以解决多源的拥塞控制、以及合理的负载均衡与调度问题。并且他认为这是TCP内在的缺陷，难以通过扩展协议栈来解决。

应运而生的是RDMA则是一种解决数据传输中处理时延的技术。RDMA是一种远程内存访问技术，它可以无需操作系统的介入，将数据从一台计算机的内存直接传输到另一台计算机的内存中。RDMA绕过了操作系统中数据的多次拷贝，大大节省了CPU资源，从而实现超低延时、超高吞吐量的数据传输。



拥塞控制：安全与公平性

TCP拥塞控制是计算机网络的十大课题之一，经典的拥塞控制算法是Reno算法，由慢启动、拥塞避免、快速恢复三个阶段组成。

除了经典算法外，利用强化学习解决拥塞控制也是一个前沿研究方向。拥塞控制的难点在于难以感知网络状况与其将来的变动；而强化学习可以对网络的复杂状况进行自适应，从而做出更好的决策。AI ECN（智能显示拥塞通知）是指通过智能算法，用实际的网络流量数据进行训练，以对流量变化进行预测，保障整网的最优性能。

拥塞控制算法(CCA)的安全性与公平性是备受瞩目的话题。在HotNets 2021的论文Counterfeiting Congestion Control Algorithms中，作者指出不稳定的拥塞控制算法不仅有损自身，并且会影响到网络中使用其他拥塞控制算法的用户，他因此提出一种基于逆向工程来检验未开源CCA安全性的方法。

在同期的另一篇论文Don't Hate the Player, Hate the Game: Safety and Utility in Multi-Agent Congestion Control中，作者认为拥塞控制中不公平的原因在于参与者不兼容的优化方式，即不同的CCA采用的根本策略不同，例如，低延时敏感度的应用和高优先级的应用之间就会产生冲突。他因此建议拥塞算法的设计者明确声明一个希望优化的效用函数，从而使对公平性的理论分析成为可能。

网络层

网络层是OSI参考模型中的第三层，介于传输层和数据链路层之间，它在数据链路层提供的两个相邻端点之间的数据帧的传送功能上，进一步管理网络中的数据通信，将数据设法从源端经过若干个中间节点传送到目的端，从而向传输层提供最基本的端到端的数据传送服务。

网络层分为数据平面和控制平面。软件定义网络(SDN)利用openflow协议将路由器的控制平面从数据平面分离，改以软件方式实现，可以说是当下最热门的的网络前沿技术之一。

网络层最主要的协议是IP协议，它是一种尽力而为的、不可靠的协议。然而，车联网、工业互联网这些新兴的技术对网络的时延、抖动、丢包率有着严格的要求，传统IP协议“尽力而为”的服务显然无法满足这种需求，而确定性网络可以解决这些问题。

软件定义网络

随着可编程交换机以及编程语言如 P4 的发展，网络数据面的可编程性大大增强。最新的方向是结合 SDN 与可编程数据面实现对网络的完全编程控制。这里面涉及到编程语言与网络的交叉，以及形式化方法（formal method）对网络程序正确性的验证。

SDN利用openflow协议，基于流（ Flow ）的概念来匹配转发规则，每一个交换机都维护一个流表（ Flow Table ），依据流表中的转发规则进行转发，而流表的建立、维护和下发都是由控制器完成的。

数据中心网络是 SDN 目前最为明确的应用场景之一，也是最有前景的应用场景之一。SDN 控制逻辑集中的特点可充分满足网络集中自动化管理、多路径转发等方面的要求。数据中心的建设和维护一般统一由数据中心运营商维护，具有相对的封闭性，可统一规划、部署和升级改造，SDN 在其中部署的可行性高。

确定性网络

传统IP网络用“尽力而为”的方式传输数据，只能将端到端的时延减少到几十毫秒，但许多的新兴业务需要将时延控制在微妙至毫秒级，将时延抖动控制在微秒级。因此需要一种“准时，准确，快速”的网络技术。

确定性网络（DetNet）就是以IP网络为基础，为需要的业务提供端到端可靠低延时服务的技术。它的特征是：低时延、低抖动、低丢包率、高带宽、高可靠。DetNet的前身是时间敏感网络（TSN），因此也沿用了TSN的时间同步、资源预留等机制，并通过边缘路由的时延抖动测量、骨干路由的确定路径与资源预留以及端到端显式路由与无缝冗余实现终端业务流的三层确定性传输。

确定性网络被认为是6G网络的关键技术之一。近几年确定性网络已经逐步应用于局域网络，然而对于广域网的确定性，是当前业界的一大难题，成熟的方案有限。

在2022年2月份，全球首张确定性网络在山东济南发布，目前总长达5600公里。

物理与链路层

数据链路层和物理层是五层网络模型中的最低两层。物理层为设备之间的数据通信提供媒介，链路层则定义了在单条链路上如何传递数据，将来自物理层的数据可靠传输到目标节点。

无损以太网

以太网是目前运用最广泛的局域网技术，与IP协议一样，它提供的也是“尽力而为”的不可靠的服务。上文我们提到过，为了降低数据中心内部网络延迟，提高处理效率，RDMA 技术应用而生，它可以绕过内核直接读写远程内存，实现了高吞吐量、超低时延和低 CPU 开销的效果。

相应地，RDMA 技术也对网络性能提出了更高的要求，尤其是丢包和拥塞会极大影响它的性能，显然传统以太网尽力而为的服务是无法满足RDMA 的需要的。

基于 RDMA的无损链路应用而生。随着IETF发布DCB(Data Center Bridging)标准，以太网在专有网络领域内拥有了自己的标准，当下，RDMA在以太网上使用的传输协议是RCOEv2。

无损以太网的关键技术包括流量控制、拥塞控制等。在数据通信中，流量控制提供了一种机制，此机制用于解决发送端与接收端的速率匹配问题，做到无丢包。拥塞控制是指对进入网络的数据总量进行控制，使网络流量保持在可接受水平的一种控制方法。

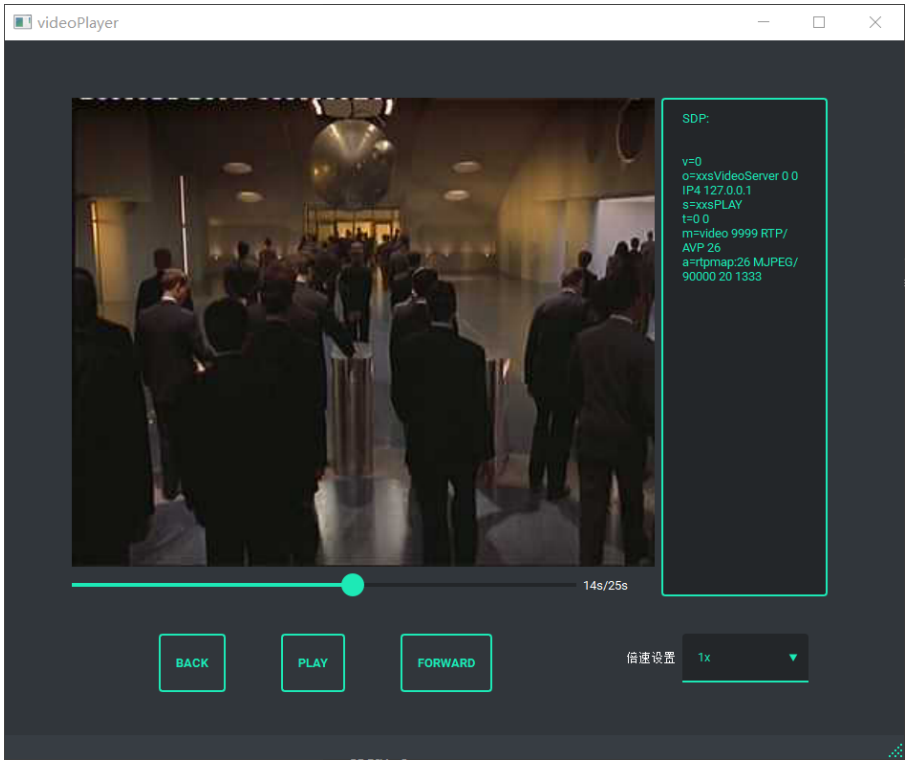
总结

各种崭新的应用场景对传统的计算机网络模型提出了新的挑战，更低时延、更高带宽是我们对网络不变地追求。网络深度可编程、网络确定性服务、网络计算存储一体化、网络与人工智能是当前热点的网络技术背景，在将来也将引领计算机网络领域的发展和变革。

实验报告部分

概述

以下为成果图：



本实验基于RTSP和RTP协议，实现一个流媒体视频播放的服务器和客户端。在补充完整原有代码、实现基本播放功能外，本实验还做了以下扩展：

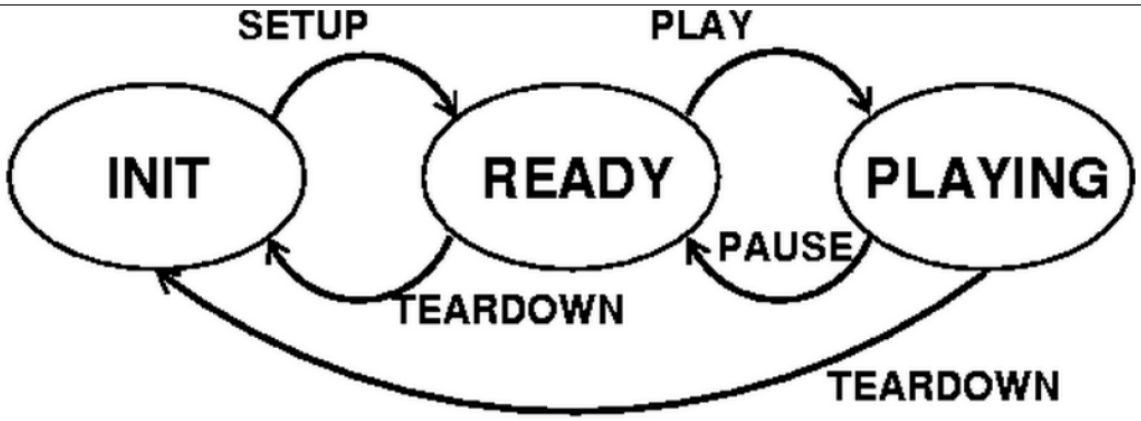
- 实现了RTSP的 `scale` 头部，并配套实现了视频的倍速播放功能。
- 实现了RTSP的 `range` 头部，并配套实现了视频的快进快退、拖拽进度条改变播放位置功能。
- 优化了 `setup`，`play`，`pause`，`teardown` 按钮的逻辑。将 `setup` 和 `teardown` 分别嵌入到启动和关闭窗口，将 `play` 和 `pause` 合并。
- 实现了RTSP的 `Request` 请求，并显示相关的 `SDP` 信息。
- 用 `PyQt5` 重写了客户端的界面。

实验中，一方面根据RFC 2326扩充了协议，另一方面在协议的基础上完善了播放功能。

实现

基本播放功能

- RTSP层面：
  RTSP协议负责服务器与客户端之前的沟通、播放的控制。
  在客户端层面，客户端可以理解成一个状态机，根据发送和接收的RTSP指令来改变自己的状态。

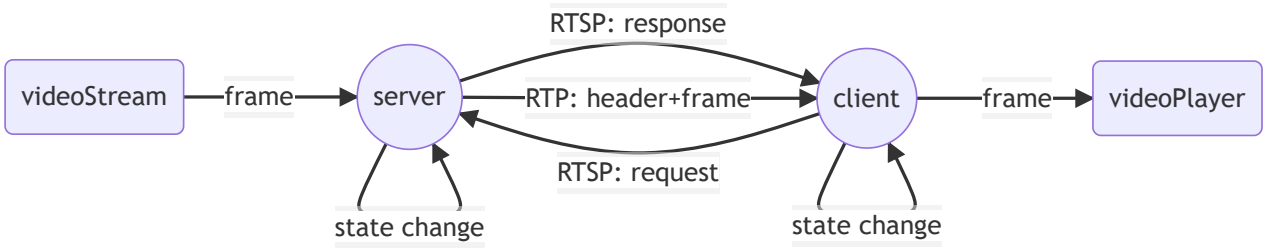


  在服务器层面，服务器对客户端的RTSP请求做出答复，并调整自己发送数据的层面。
- RTP层面：
  RTP负责有效载荷，即视频帧的传输。
  服务器以一定速率发送指定的视频帧，并在帧前加上RTP头部。
  客户端接收RTP包，根据头部的序列号等信息校验，并将帧显示在播放器。

这一部分的代码框架已由实验提供，具体完成的工作有两部分：

- 在客户端，完成四个按钮 `setup`，`play`，`pause`，`teardown` 对应的请求发送以及状态维护。
  - RTSP端：客户端时刻维护自己目前的状态 `INIT`，`READY`，`PLAYING`， 和上一条发送的请求类型 `SETUP`，`PLAY`，`PAUSE`，`TEARDOWN` 。并根据服务器的响应进行状态的转变。
  - RTP端: 客户端从收到的RTP包中取出头部和视频帧，检验头部的序列号后，将视频帧显示出来。
- 在服务器端，完成对RTP包的处理，具体是填充好RTP的头部，最重要的是序列号和时间戳这两个子段。
  - RTSP端：服务器对客户的成功响应基本是一致的。
  - RTP端：服务器根据请求类型，改变自己发包的状态。

整体的行为逻辑可用下图表示：



补充代码后即可实现最基本的播放功能。

按钮逻辑优化

原始实现里窗口上有四个按钮， `setup`，`play`，`pause`，`teardown`， 它们包括了一次流视频播放从建立连接、进行播放/暂停，断开连接所需的动作。

而在我们的实现中，最终只显式的使用了一个按钮 `play/pause`， 进行播放和暂停的切换。

`setup` 按钮被删去，改为在客户端建立 `rtsp` 连接时就发送 `setup` 请求。

`teardown` 按钮被删去。这是因为在原设计中， `teardown` 操作和关闭窗口实际上完成的是相同的动作，因此我们改为确认关闭窗口即发送 `teardown` 请求。

SCALE头部

原理

RTSP支持 `scale` 头部来对播放速率进行控制。可见 `RFC2326` 中有关说明。
具体来说， `scale=1` 代表以正常速率播放， `scale=2` 代表以二倍速播放，为负数时代表倒放。

注意： `scale` 只控制**播放速率**，除非显式的使用 `speed` 头部，服务器不应该对视频帧的**发送速率**有任何修改。

比如，快于1的速率是在服务器端通过丢帧来实现的，2倍速时我们每隔一帧发送一帧(**而不是一秒中内发送两秒视频的内容**)；慢于1的速率可以通过发送重复帧来实现，0.5倍速时我们把每一帧重复发送两次。

> Implementation of scale changes depends on the
> server and media type. For video, a server may, for example, deliver only key frames or selected key frames. [RFC2326]

补充：RTSP也支持使用 `speed` 头部对视频发送速率进行修改，默认发送速率是支持视频播放的最小速率，即视频的码率。使用 `speed` 头部会改变传输的带宽。

代码思路

我们的实现支持以 0.5,1,2 三种速率播放。
客户端：首先要在 PLAY 请求中添加头部 scale 。每次改变速率都发送新的 PLAY 请求，如果是在播放的过程中改变速率，还要先发送 PAUSE 请求。以下是某次带 scale 的请求。

```
PLAY movie.Mjpeg RTSP/1.0
CSeq: 9
Session: 155344
Scale: 0.5
Range: npt=3.25-
```

服务器端：服务器端根据收到 PLAY 指令中的 scale 来控制帧的发送。具体是对 VideoStream 的 nextFrame 函数进行修改，如果速率为2，那么每次读两帧图片，但是只发送一帧；如果速率为0.5，那么每次首先判断是否要读取新的帧，否则直接发送上一帧。

RANGE头部

协议原理

RTSP支持 range 头部，代表一段时间。客户端在 PLAY 请求中设置 range 头部可以指定播放视频的哪一段。服务器在响应中设置 range 头部可以告知客户端视频的长度信息。
range 中时间可以采用各种格式，本实验中我们采用的是 npt 格式，它包含一个起点和一个终点，单位为秒，两者用 - 来分割，若不含终点则默认播放到视频结尾。如：

```
Range: npt=3.25-
```

该范围没有指定结束时间，代表希望从视频的3.25秒播放到结束。

在服务器对 SETUP 的响应中，也可以包含 range 头部，如 Range: npt=0.0-25.0 ，代表视频的时长为25s。

代码思路

实现 range 头部的主要困难点在服务器端。当客户发送带 range 的头部后，服务器需要定位并从指定时间的帧开始发送。

服务器将起始时间转换成对应的帧编号，如视频的帧速率(fps)是20帧每秒，那么从视频的3.5秒开始播放意味着从视频的第70帧开始播放。

具体代码实现在 VideoStream 中。服务器在建立连接时就对整个视频进行预处理，记录下每个帧在视频文件中的起始位置。当播放起始时间改变时就重定位到相应的帧，然后顺序播放。

利用range头部实现快进回退、拖拽进度条等功能

RTSP并没有直接改变播放位置的请求，这项功能实际上是通过 PAUSE 请求和 PLAY 请求结合来实现的。

当我们希望调到视频的另一个位置，首先发送 PAUSE 终止上一个视频流的传输，接着发送新的带有 range 头部的 PLAY 指令：

```
Data sent:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 4
Session: 863017

Data sent:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 5
Session: 863017
Scale: 1
Range: npt=11.316-
```

需要注意的是：**不能不发送 PAUSE 指令就直接发送新的 PLAY 指令**。因为RTSP规定，对于连续的多个 PLAY 请求，服务器应该依次响应，即等上一个 PLAY 请求的视频段播放完后再顺序播放下一个，因此需要用 PAUSE 来终止上次 PLAY 请求。

代码实现中的一个同步问题

我们通过依次发送 PAUSE 和 PLAY 来实现视频的重定位。但是如果在 PAUSE 指令的响应到来前就发送 PLAY ,由于没有收到 PAUSE 的响应，此时客户端的状态仍然是播放中，会忽略掉后来的 PLAY 命令。
解决这个问题的方法有两种：

1. 直接在发送 PLAY 命令前等待恰当的时间，保证 PAUSE 的回复已经收到。
2. 发送 PAUSE 后给主线程上锁，只有接收响应的线程收到回复后才释放锁。

拖拽进度条

在这种情况下，我们点击进度条的某个位置，这个位置占进度条长度的比例乘以视频总时长，即是新的 PLAY 指令中 range 的起点。而视频总时长，是在服务器对 SETUP 请求的响应的 range 头部中获取的。

快进与回退

在这种情况下，我们将视频的播放位置相对的前移或后移。因此客户端需要能知道当前视频的位置。
这里我们采用了一种比较简单的机制：将已播放的视频帧数除以视频的帧速率，即为已经播

放的秒数。那视频的帧速率(fps)怎么获取呢？事实上，**可以发送 DESCRIBE 请求，服务器返回的响应中包含SDP描述，其中就会包含视频的帧速率等信息**，这也是我们下面将介绍的。

DESCRIBE请求与SDP

describe 是RTSP中一个非强制要求实现的请求，用于对话双方之间的媒体协商。其响应一般包含SDP格式的媒体描述，可以包含视频编码格式、长度、码率、帧率等信息。

客户在向服务器发送 SETUP 请求之前，可以发送 DESCRIBE 请求以获取有关媒体信息。
以下是实验中一次 DESCRIBE 请求和响应：

```
Data sent:
DESCRIBE movie.Mjpeg RTSP/1.0
CSeq: 1
Session: sdp

Receive:
RTSP/1.0 200 OK
CSeq: 1

v=0
o=xxsVideoServer 0 0 IP4 127.0.0.1
s=xxsPLAY
t=0 0
m=video 9999 RTP/AVP 26
a=rtpmap:26 MJPEG/90000 20 1333
```

可以看到在收到的响应中，包含了SDP格式的媒体描述。
从 v=0 到 t=0 0 是会话描述，它包含了会话双方的一些信息。如使用的是 IPV4 ，服务器的地址是 127.0.0.1 等等。

m=... 和 a=... 这两行是媒体描述，一个会话中可能有多个媒体(视频，音频等)，因此可能有多个媒体描述。但是我们的会话中只有一个媒体，即播放的视频。
这两行具体含义是：
第一行： video 指明了媒体的类型(video/audio)是视频类型； 9999 指明了载荷将通过哪个端口号发送； RTP/AVP 是使用的传输协议，即 RTP OVER UDP ,如果是通过 TCP 则是 TCP/RTP/AVP 。26表示了有效载荷类型，即JPEG。
第二行： rtpmap 代表接下来的是 RTP 属性的映射表。 26 MJPEG 代表编码类型； 90000 代表视频的采样率是90000； 20 代表视频的帧速率； 1333 代表视频的码率，单位为 kbps 。

总结

- RTSP协议的实现较为复杂，主要体现在服务器端上。本次实验中，客户端只需要发送指令和接收图片，而对各种播放的具体实现都要靠服务器来完成。这也是RTSP的缺点之一。

RTSP的优点是时延较低，对视频的控制比较方便。因为RTP对视频的发送是以帧为单位的，因此可以方便地实现快进回退等功能

- 为了完成实验的扩展部分，我较为细致地阅读了RFC2326文档的有关部分。对计算机网络中“协议”的重要性有了更加细致的认识。
  具体来说，协议是双方沟通应遵循的格式，但没有规定协议支持功能的具体实现。以RTSP为例，其支持用 `Scale` 来协商视频的播放速率，但是如何实现依赖于服务器自己。
- 在代码方面，难点在于让服务器端支持倍速和视频的重定位，需要对视频帧进行较精细的操作，并且要和RTSP请求响应保持同步。