实验五　传输层协议实验—实验指导书

上次编辑时间	@2025年6月5日 22:24
创建人	陈 陈洑
创建日期	@2025年5月25日

实验目的

理解传输层的作用和工作机制，掌握网络数据可靠传输的概念、原理以及相关算法；

深入理解 TCP 握手协议（三次握手和四次挥手）、滑动窗口、数据确认、超时与重发机制的详细原理和流程；

锻炼学生将抽象的协议理论知识转化为具体代码的能力。

实验原理

一、传输层的核心功能

1. **分段与重组**

- 将应用层的数据分割成合适大小的 “数据段”（Segment），便于在网络中传输。
- 接收端将分段的数据重新组装成完整的消息。

2. **端到端通信**
   - 通过 “端口号”（Port Number）标识不同的应用进程（如 HTTP 用 80 端口，FTP 用 21 端口），实现同一主机上多应用的并发通信。
3. **流量控制**
   - 防止发送方发送数据过快，导致接收方处理不及而丢包。
4. **差错控制**
   - 检测传输过程中的数据错误，并通过重传等机制确保可靠性（仅针对可靠传输协议

二、主要传输层协议

1. 用户数据报协议（UDP，User Datagram Protocol）

- **特点**：
  - **无连接**：发送数据前无需建立连接，直接发送。
  - **不可靠**：不保证数据有序到达，不处理丢包、重复或错误。
  - **轻量级**：头部仅 8 字节（源端口、目的端口、长度、校验和），传输效率高。
- **应用场景**：
  - 对实时性要求高、允许少量丢包的场景，如：
    - 视频流（Netflix、抖音）、音频通话（Skype、微信语音）；
    - 在线游戏（低延迟优先）；
    - 网络管理协议（SNMP）、域名解析（DNS）。

2. 传输控制协议（TCP，Transmission Control Protocol）

- **特点**：
  - **面向连接**：通过 “三次握手” 建立连接，传输完成后 “四次挥手” 断开连接。
  - **可靠传输**：
    - **序列号与确认应答**：确保数据按序到达，丢失时重传；

        - **滑动窗口**：动态控制流量，避免拥塞；
        - **校验和**：检测数据错误，错误数据丢弃并请求重传。
    - **头部复杂**：至少 20 字节（包含端口、序列号、确认号、控制位等）。
- **三次握手过程**：

  1. 客户端发送 SYN包（序列号 Seq=x），请求建立连接；

  说明：客户端 → 服务器：发送 **SYN 包**（同步请求），包含：

  - **序列号（Seq=x）**：标识本次连接的初始序号（防止旧连接重复）。

  2.服务器回复 SYN+ACK 包（Seq=y，ACK=x+1），确认请求；

  说明：服务器 → 客户端：回复 **SYN+ACK 包**，包含：

  - **序列号（Seq=y）**：服务器的初始序号；
  - **确认号（ACK=x+1）**：确认客户端的 SYN 包已接收（期望下一个数据序号为 x+1）。

  3.客户端回复 ACK 包（Seq=x+1，ACK=y+1），连接建立完成。

  说明：客户端 → 服务器：发送 **ACK 包**，包含：

  - **序列号（Seq=x+1）**：客户端的下一个数据序号；
  - **确认号（ACK=y+1）**：确认服务器的 SYN 包已接收。
- TCP层可靠传输的策略包含序列号、确认应答、重传机制三部分：

1. 序列号与确认应答（ACK）

- **序列号（Seq）**：每个字节数据都有唯一编号，用于标记数据顺序（如 Seq=100 表示该段数据第一个字节是 100 号）。
- **确认号（ACK）**：接收方通过 ACK 告知发送方 “已成功接收某序号之前的数据”，例如：
    - 接收方收到 Seq=100、长度 = 200 的数据段，回复 ACK=300（表示期望下一个数据从 300 号开始）。

2. 超时重传（Retransmission）

- 发送方启动定时器，若超时未收到 ACK，重新发送未确认的数据段。
- 重传时间（RTO）动态调整：基于往返时间（RTT）的统计值计算。

3. 去重与排序

- 接收方通过序列号丢弃重复数据，并缓存乱序到达的数据，等待缺失数据补全后再提交给应用层。

应用场景：

- 对可靠性要求高的场景，如：
  - 文件传输（FTP、SFTP）；
  - 网页浏览（HTTP/HTTPS）；
  - 电子邮件（SMTP、POP3、IMAP）；
  - 远程登录（SSH、Telnet）

3、UDP 与 TCP 对比

特性	UDP	TCP
连接方式	无连接（无状态）	面向连接（需三次握手）
可靠性	不可靠（尽力而为）	可靠（保证交付、按序）
传输效率	高（头部小、无额外开销）	低（头部大、拥塞控制等开销）
适用场景	实时性、少量丢包可接受的业务	可靠性优先的业务
典型端口	DNS（53）、DHCP（67/68）	HTTP（80）、HTTPS（443）

4、其他传输层协议（扩展）

1. **流控制传输协议（SCTP，Stream Control Transmission Protocol）**
   - 结合 TCP 的可靠性与 UDP 的多流特性，支持多宿主（同一连接使用多个 IP 地址）和多流传输（如视频会议同时传输音频和视频流）。
   - 应用于电信网络（如 VoIP）、实时消息系统。
2. **数据报拥塞控制协议（DCCP，Datagram Congestion Control Protocol）**
   - 基于 UDP，增加拥塞控制机制，适用于实时数据传输（如流媒体），在保证低延迟的同时避免网络拥塞。

5、传输层与其他层次的关系

- **下层（网络层）**：传输层依赖 IP 协议实现跨网络的数据传输，IP 负责 “主机到主机” 的寻址，传输层负责 “进程到进程” 的通信。
- **上层（应用层）**：为应用程序提供传输服务，应用层协议（如 HTTP）需指定使用 TCP 或 UDP。

实验内容

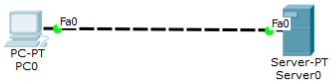
一、验证性实验

捕获PDU数据包（TCP）并分析

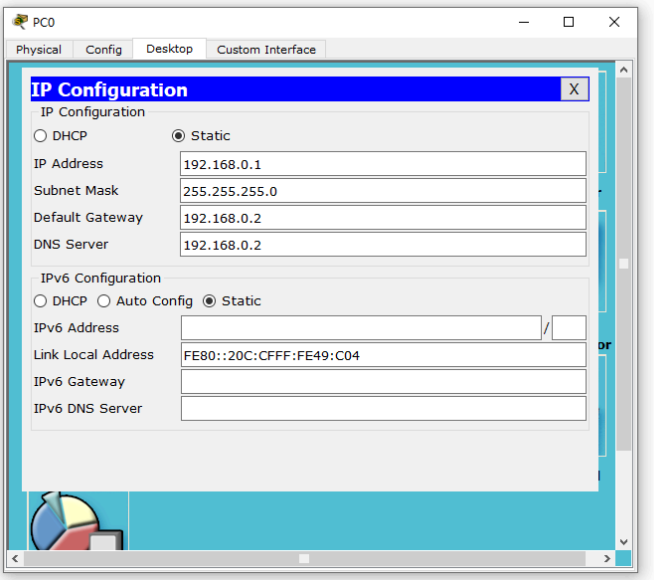
Wireshark可以捕获和显示通过网络接口进出其所在 PC 的所有网络通信。Packet Tracer 的模拟模式可以捕获流经整个网络的所有网络通信，但支持的协议数量有限。这里我们使用一台PC直连至服务器网络，捕获相关数据包并分析。

1、分别配置PC与server端并设置DNS环境

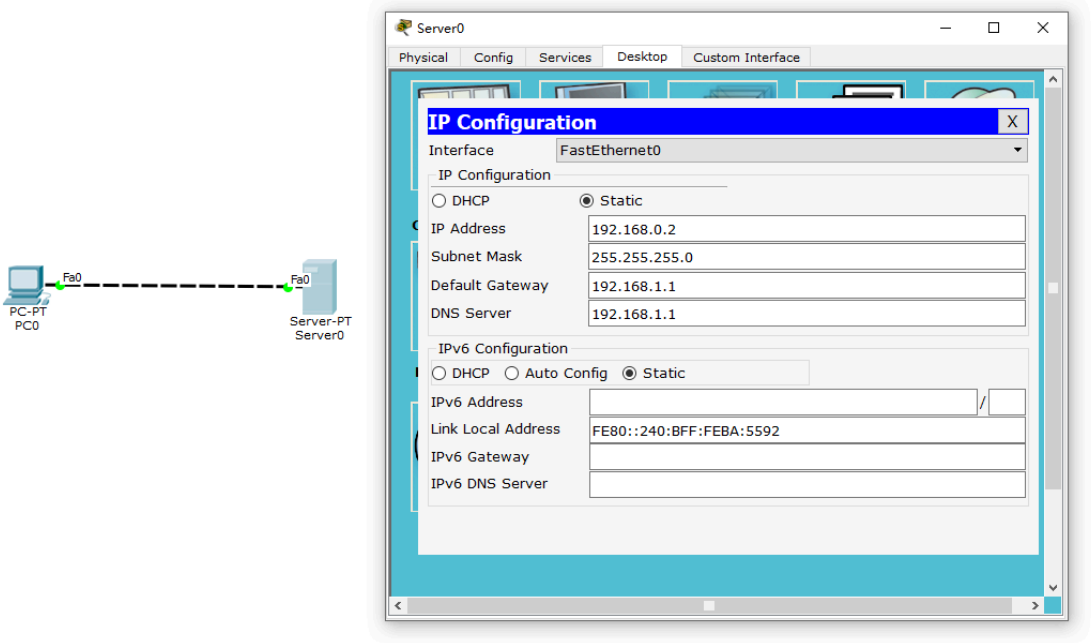
实验拓扑图



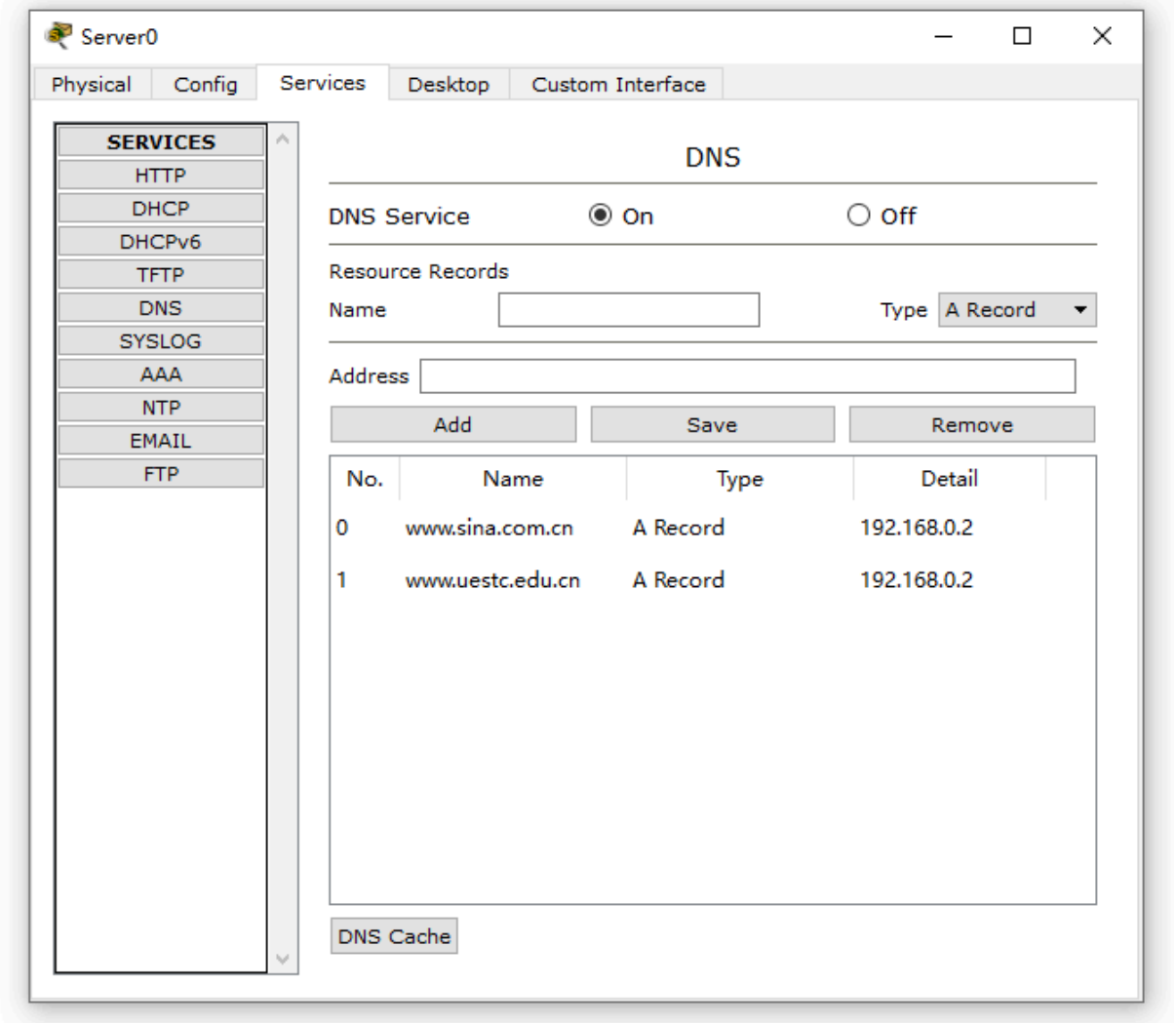
设置PC的IP地址



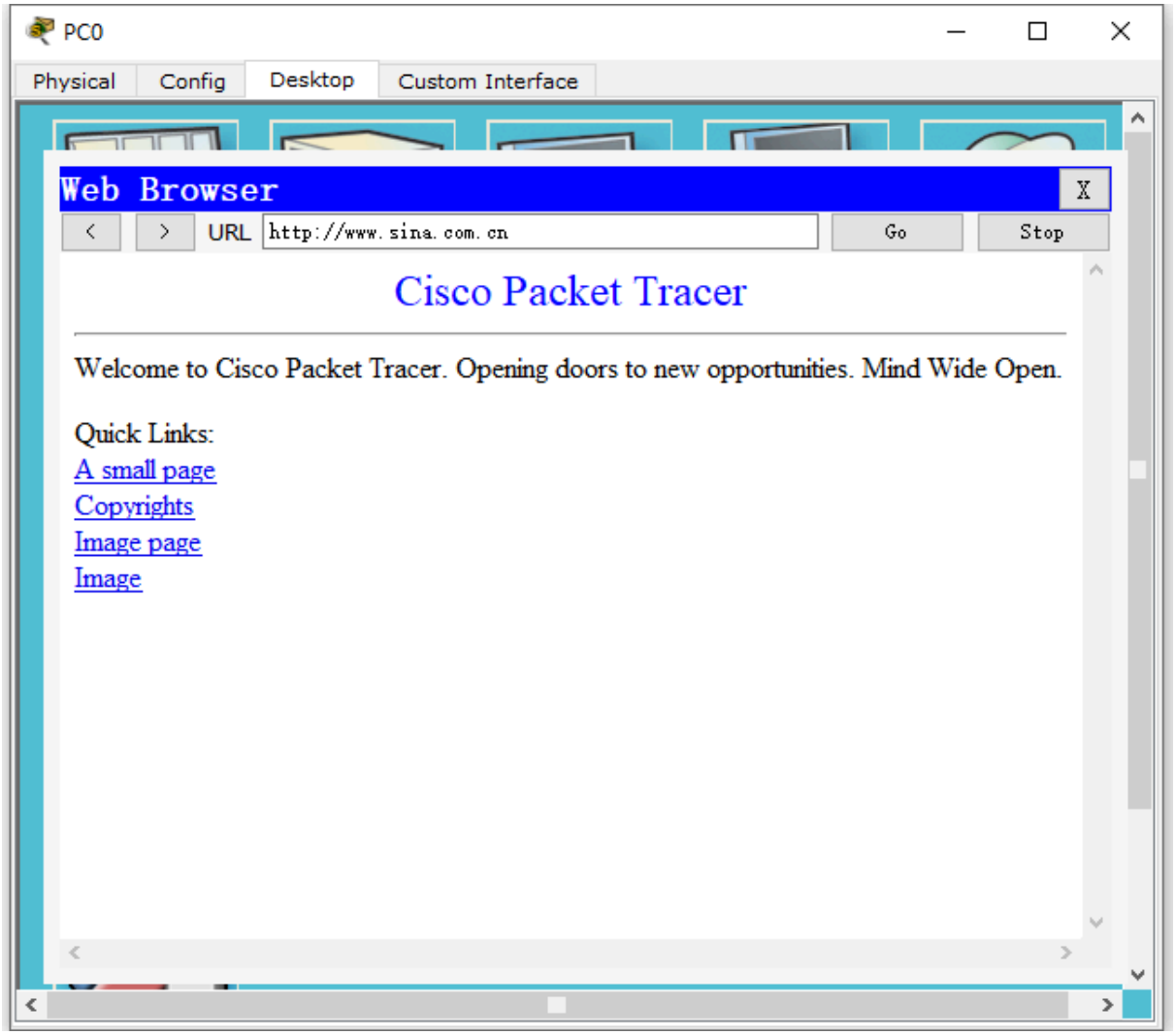
设置服务器的IP地址

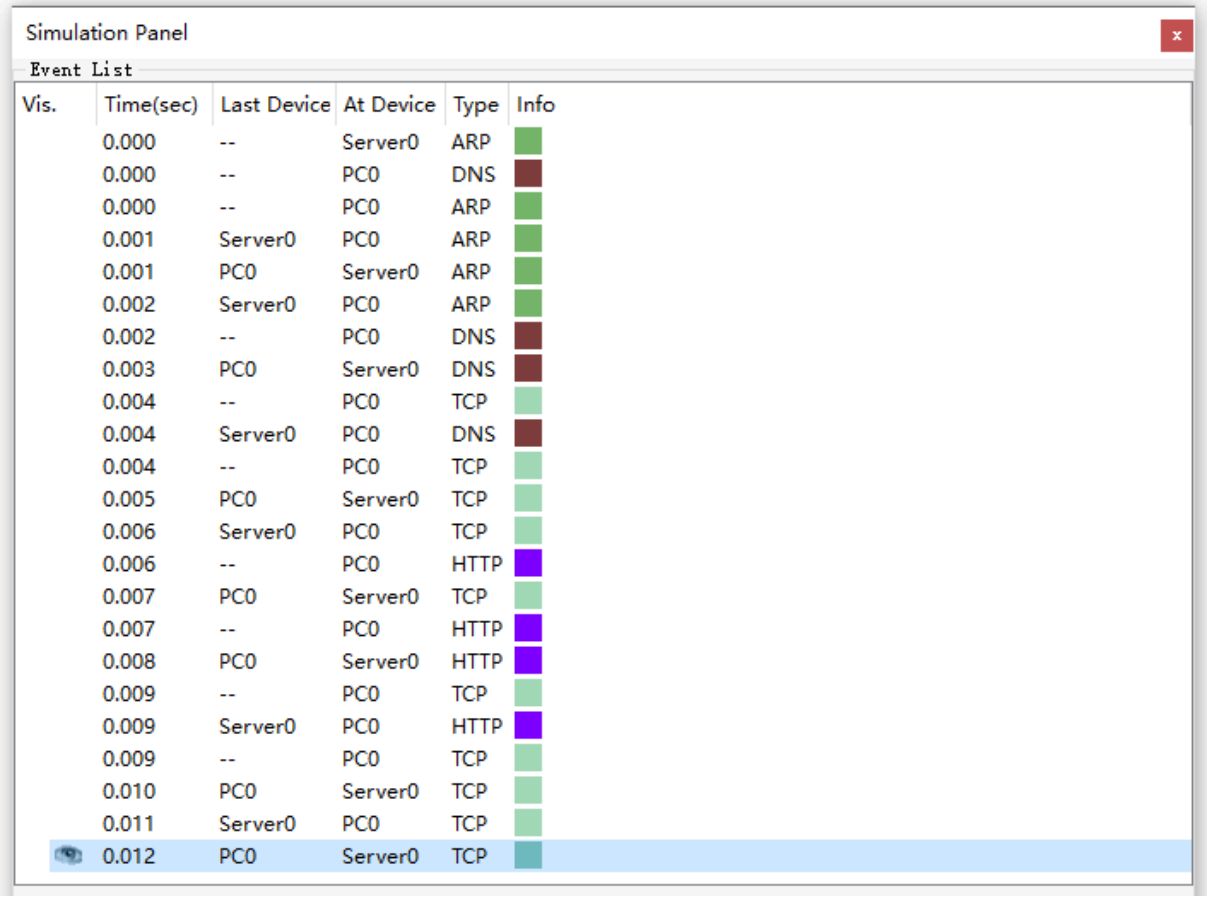

Server0
Physical
Config
Services
Desktop
Custom Interface
IP Configuration
X
Interface
FastEthernet0
IP Configuration
DHCP
Static
IP Address
192.168.0.2
Subnet Mask
255.255.255.0
Default Gateway
192.168.1.1
DNS Server
192.168.1.1
IPv6 Configuration
DHCP
Auto Config
Static
IPv6 Address
Link Local Address
FE80::240:BFF:FEBA:5592
IPv6 Gateway
IPv6 DNS Server
Fa0
Fa0
PC-PT
PC0
Server-PT
Server0

设置服务器工作类型



2、模拟HTTP访问并截取数据包


PC0
Physical
Config
Desktop
Custom Interface
Web Browser
X
URL
http://www.sina.com.cn
Go
Stop
Cisco Packet Tracer
Welcome to Cisco Packet Tracer. Opening doors to new opportunities. Mind Wide Open.
Quick Links:
A small page
Copyrights
Image page
Image



3、截取TCP协议包并分析

三次握手过程

四次挥手过程

根据上述过程分析并回答问题：

- PC与服务器连接何时、以何种方式建立，试分析建立过程；
- DNS协议与HTTP协议的通信端口分别是什么，请举例说明；

二、编程实验

给出源程序，用C语言模拟实现滑动窗口协议，包含发送端和接收端的核心逻辑。这个实现模拟了基本的滑动窗口机制，包括序列号管理、确认应答、超时重传等功能。具体核心如下：

1、程序说明

通过模拟 TCP 自定义序号机制的 C 语言实现，支持按序号发送数据和确认应答功能。这个程序模拟了客户端和服务器之间的通信过程，包括自定义序号、滑动窗口、超时重传和确认应答等核心机制。

程序模拟TCP 协议的自定义序号机制，主要包含以下功能：

1. **数据包结构**：包含序列号、确认号、数据内容和时间戳
2. **滑动窗口**：发送窗口和接收窗口的实现
3. **超时重传**：当数据包超时未确认时进行重传
4. **确认应答**：接收方按序接收数据并发送 ACK
5. **乱序处理**：接收方处理乱序到达的数据包

2、关键数据结构

```
// 数据包结构
typedef struct {
int seq_num;            // 序列号
int ack_num;            // 确认号
char data[MAX_DATA_SIZE]; // 数据
int data_size;          // 数据大小
bool is_ack;            // 是否为ACK包
int send_time;          // 发送时间（用于超时判断）
} Packet;

// 发送窗口结构
typedef struct {
Packet packets[WINDOW_SIZE]; // 窗口内的数据包
int base;                   // 窗口基序号
int next_seq_num;           // 下一个可用的序列号
} SendWindow;

// 接收窗口结构
typedef struct {
Packet packets[WINDOW_SIZE]; // 窗口内的数据包
int expected_seq;           // 期望接收的序列号
} ReceiveWindow;
```

3、需补充函数

1）// 发送窗口是否已满

bool is_window_full(*) {

****

}

2）// 发送方处理确认

void process_ack（*）{

****

}

3）// 检查超时并重传

void check_and_retransmit（*）{

****

}