中国科学院大学

University of Chinese Academy of Sciences

《软件与系统安全》

实验 1A & 1B 讲解

刘鹏

中国科学院大学 · UCAS

Contents

- **实验内容介绍**
  - 1A：嗅探器设计与实现
  - 1B：安全文件传输软件设计与实现
- **实验实施步骤**
- **作业提交方式 (2022年4月7日23:59前提交到课程网站)**
  - 不要压缩为 rar、7zip 等格式，只要 ZIP，不要加密。
  - 文件组织与命名方式见下页
  - 所有代码必须开源，GitHub/Gitee/GitLab 有完整记录。担心代码被抄袭的同学，可以选择在提交之后将 repo 设置为 public.
- **实验考核及评分准则**
  - 共有 46 人选课，最终将根据完成情况按排名打分。

Commit Guides

1A：嗅探器设计与实现

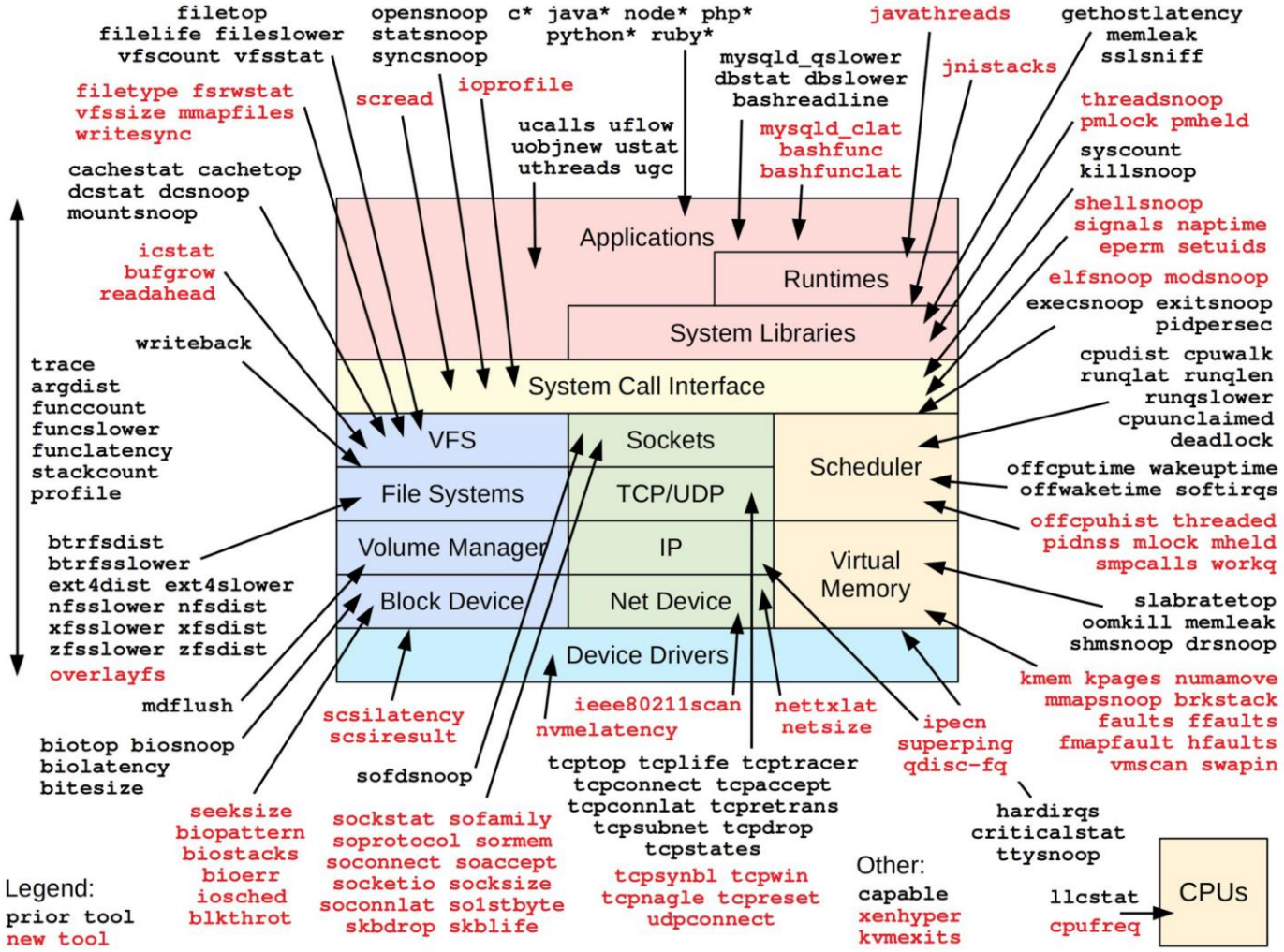
文件夹	张三_202018008829001_EX1A
1. 源码文件夹	张三_202018008829001_ex1A_src
2. 演示视频	张三_202018008829001_ex1A_play.mp4
3. 文档	张三_202118008829001_ex1A_report.pdf/docx

1B：安全文件传输软件设计与实现

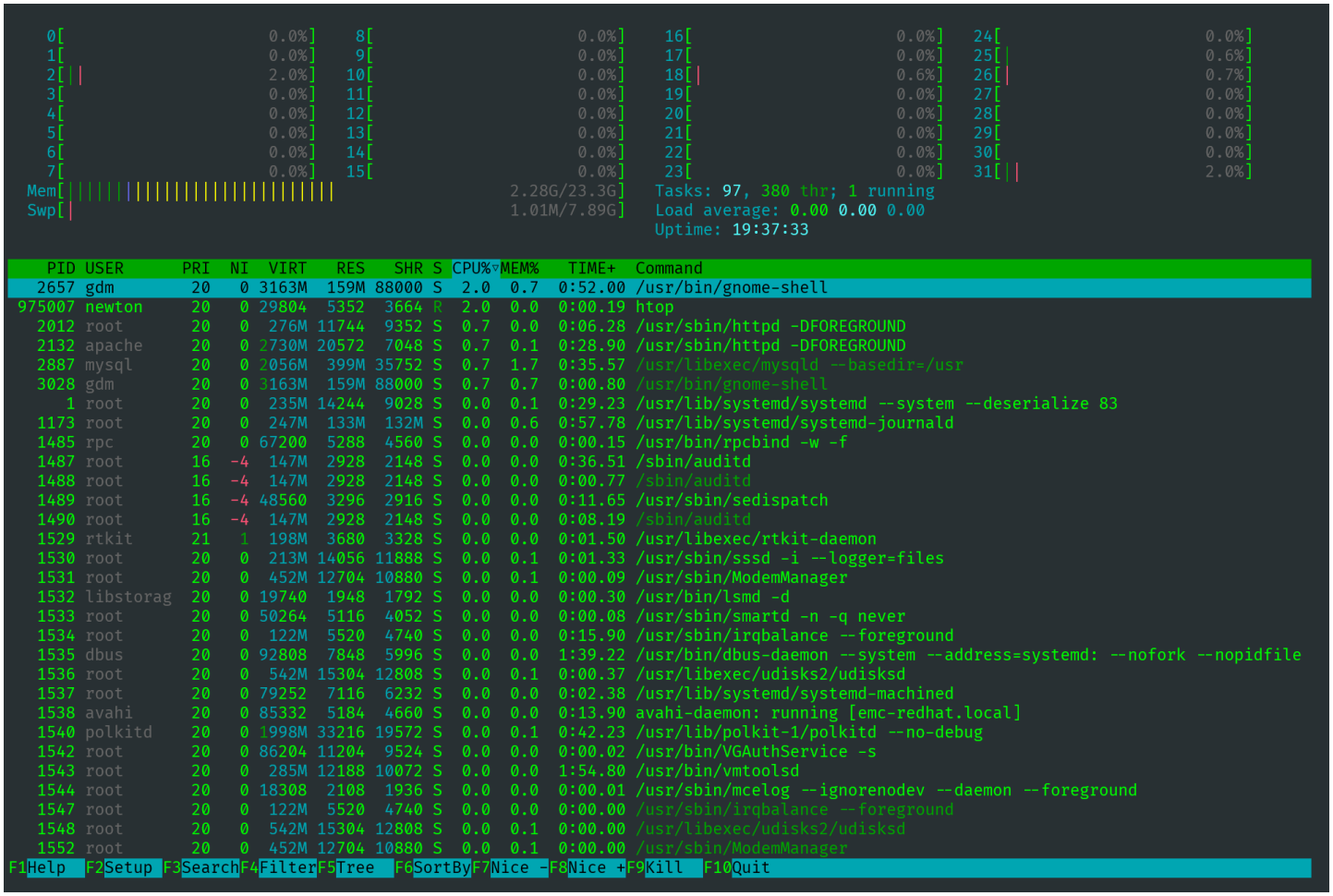
文件夹	张三_202018008829001_EX1B
1. 源码文件夹	张三_202018008829001_ex1B_src
2. 演示视频	张三_202018008829001_ex1B_play.mp4
3. 文档	张三_202118008829001_ex1B_report.pdf/docx

无需 README，想说的全放在文档里即可。Git 链接必须写进去。

实验 1A：Introduction



实验 1A：Introduction

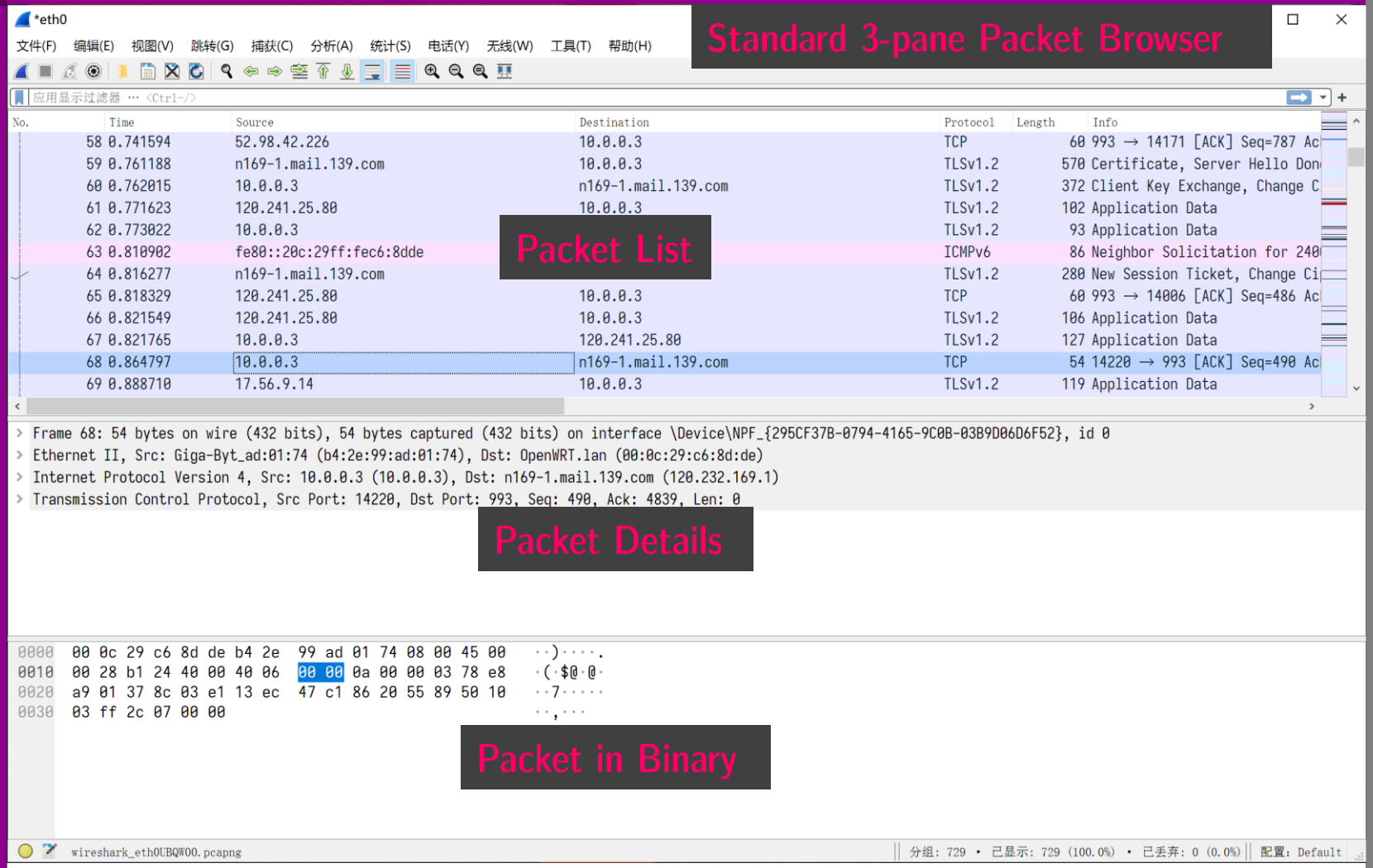


Linux *htop* 命令输出

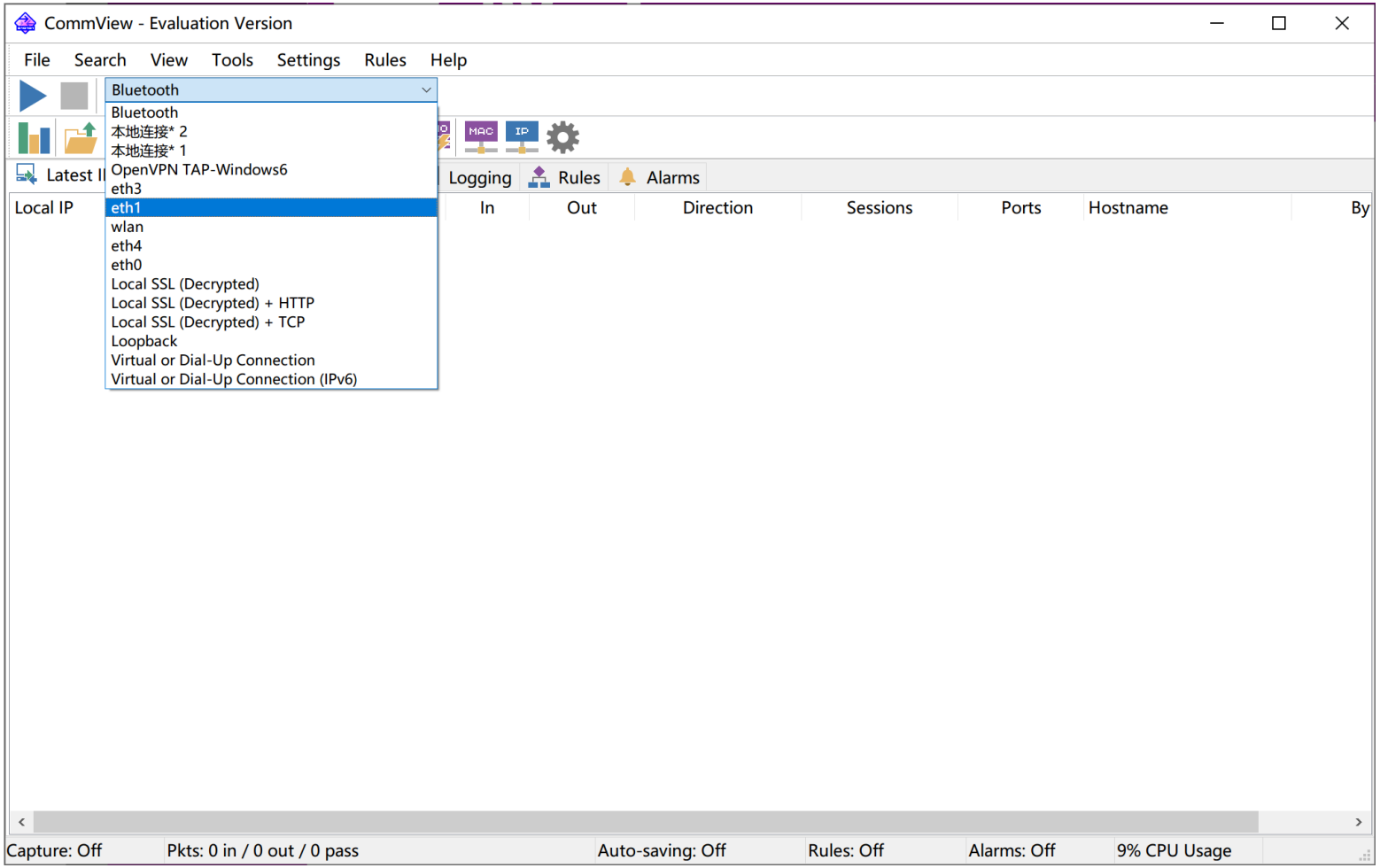
实验 1A：嗅探器设计与实现

- **网络嗅探器**
  - 介绍：网络嗅探器，Network Packet Sniffer (NPS)，是一种专门用来进行网络流量侦听的工具。数据包嗅探器是研究网络行为学的基础工具。
  - 本次实验中的嗅探器与常见的 Wireshark 等工具不同
    - **Wireshark 等工具实现了强大的数据包协议分析功能，属于 Network Protocol Analyzer (NPA).**
    - **请各位思考，如何识别某种特定类型的应用层报文。**
- **温馨提示**
  - 实验中必须实现 NPS 功能（否则不及格）
  - 实验中尽量完善 NPA 功能（否则分不高）
  - 如果有同学可以基于 Wireshark 做良好的二次开发，也会有加分。

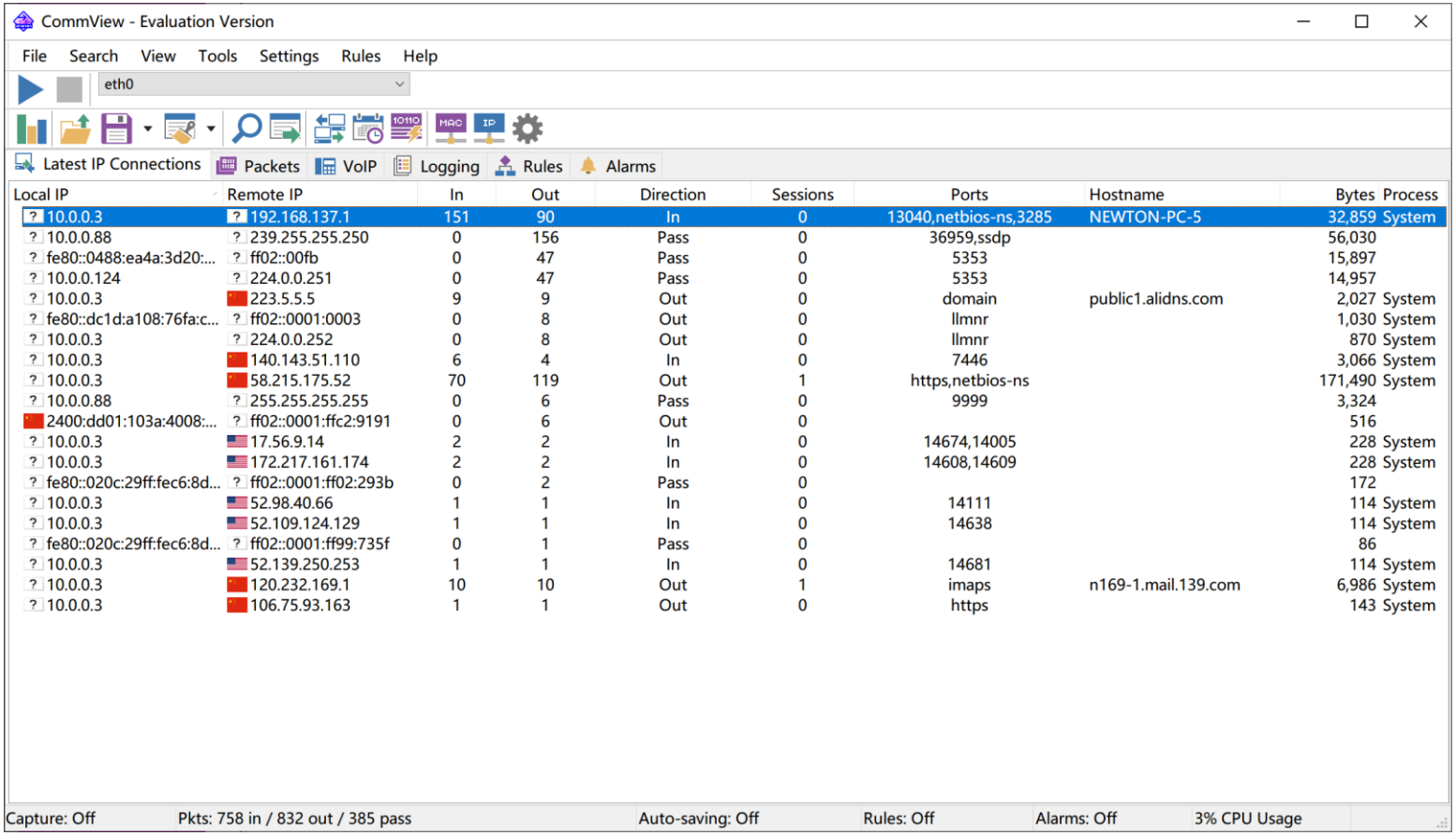
Demo: Wireshark



Demo: CommView



Demo: CommView



实验 1A 要求

- **务必理清 OSI 五层模型的概念**
  - 深入理解抓包层次与分析层次
    - **iOS 的 Thor App 可以在安装了系统描述文件之后，抓取 HTTP/HTTPS 的应用层包。同样因为安装了根证书性质的描述文件，故 Thor 能解析 HTTPS 中的内容。但具体到某些传输控制协议的内容，Thor 无能为力。这告诉了我们什么道理?**
  - 要有一定的协议过滤能力
    - **Wireshark 等软件的协议过滤器支持逻辑演算，所以该软件里含有逻辑推导的组件。基本的协议过滤需要支持筛选 HTTP、TCP/UDP、IPv4/v6、ICMP 等不同类型、层次的数据包。libpcap 提供了数据包筛选功能。**
  - 有一定的流追踪能力（加分项）
    - **基于 IP+Port 的 TCP 流**
    - **某进程产生的所有 TCP 流**

实验 1A 提示

- **基于图形化编程的重要性**
  - 现代化的软件基本都提供图形化交互界面。
  - 图形化的界面对于分析流的时序图具有重要帮助作用。
  - 在现代图形 API 的帮助下开发图形化程序并不难。
  - 有关现代图形库的帮助，参见 Prompt II.
- **抓包的系统 API**
  - libpcap
  - winPcap: 基于 Windows NT 内核定制的 libpcap
  - 请大家自行学习上述 lib/dll 的用法，然后在代码中实现调用。

Linux 协议栈

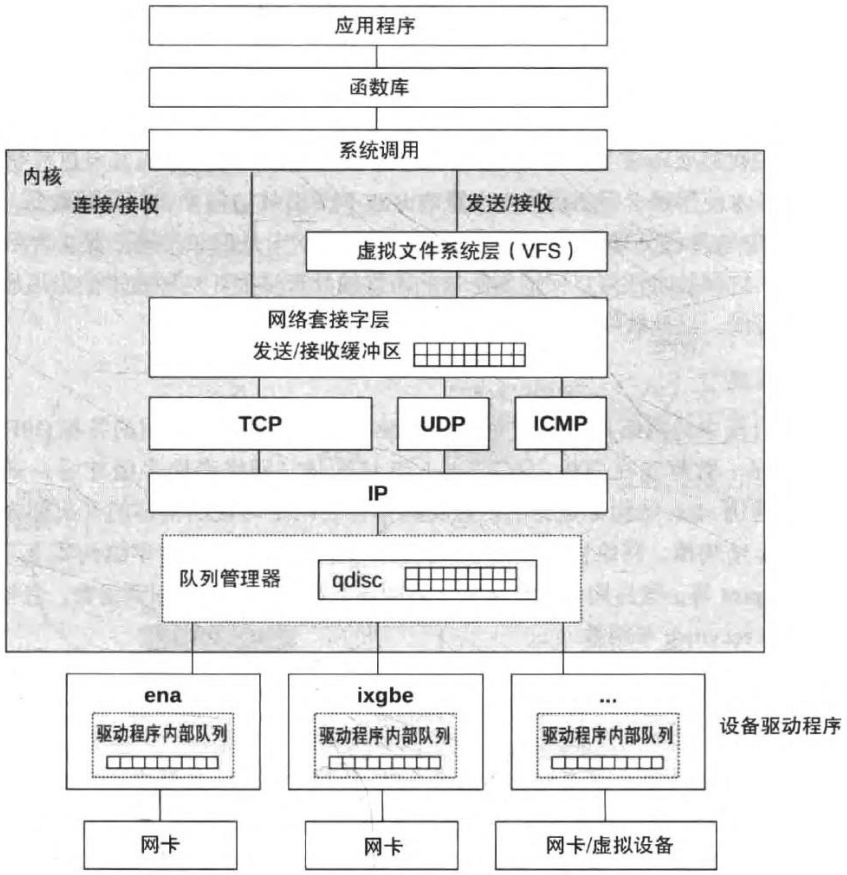


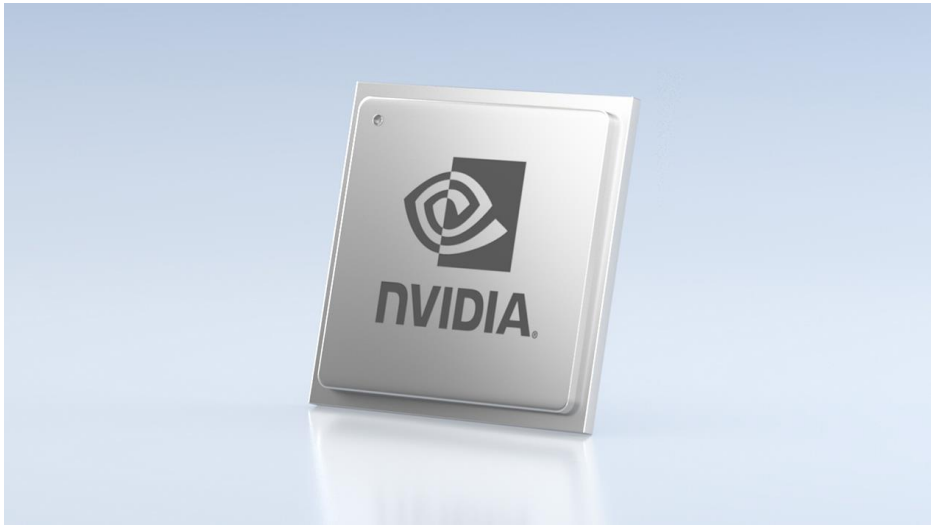
图. Linux 网络协议栈

VFS 抽象层会在 1B 中用到

“一切皆文件”是Linux的基本哲学之一，不仅是普通的文件，包括目录、字符设备、块设备、套接字等，都可以以文件的方式被对待。实现这一行为的基础，正是 Linux 的虚拟文件系统机制。

Linux BPF Internals

- **网络传输速度不断提升**
  - 以太网已经实现 100Gbps 甚至是 400Gbps 的速率
  - 如何让抓包工具能匹配网速呢？性能是个关键问题。



PRODUCT SPECIFICATIONS

Max Total Bandwidth	**400Gb/s**
Supported Ethernet Speeds	**10/25/40/50/100/ 200/400GbE**
Number of Network Ports	**1/2/4**
Network Interface Technologies	**NRZ (10/25G) / PAM4 (50/100G)**
Host Interface	**PCIe Gen5.0 x16/ x32**
Cards Form Factors	**PCIe FHHL/ HHHL, OCP3.0 SFF**
Network Interfaces	**SFP56, QSFP56, QSFP56-DD, QSFP112, SFP112**

Linux BPF

- **BPF/eBPF 可解决抓包开销问题**
  - BPF 已经是一个技术名称综合，与 LLVM 不是底层虚拟机一样，不可“望文生义”。利用 BPF 可以在内核态编程，并且有以下主要优势：
    - **User-defined programs. 用户可编程**
    - **Limited and secure kernel access. 内核依旧安全**
    - **A new type of software. 与 VM、Docker 类似，已经为一种新技术**

https://www.usenix.org/conference/lisa21/presentation/gregg-bpf

利用 Linux BPF 开发监控软件

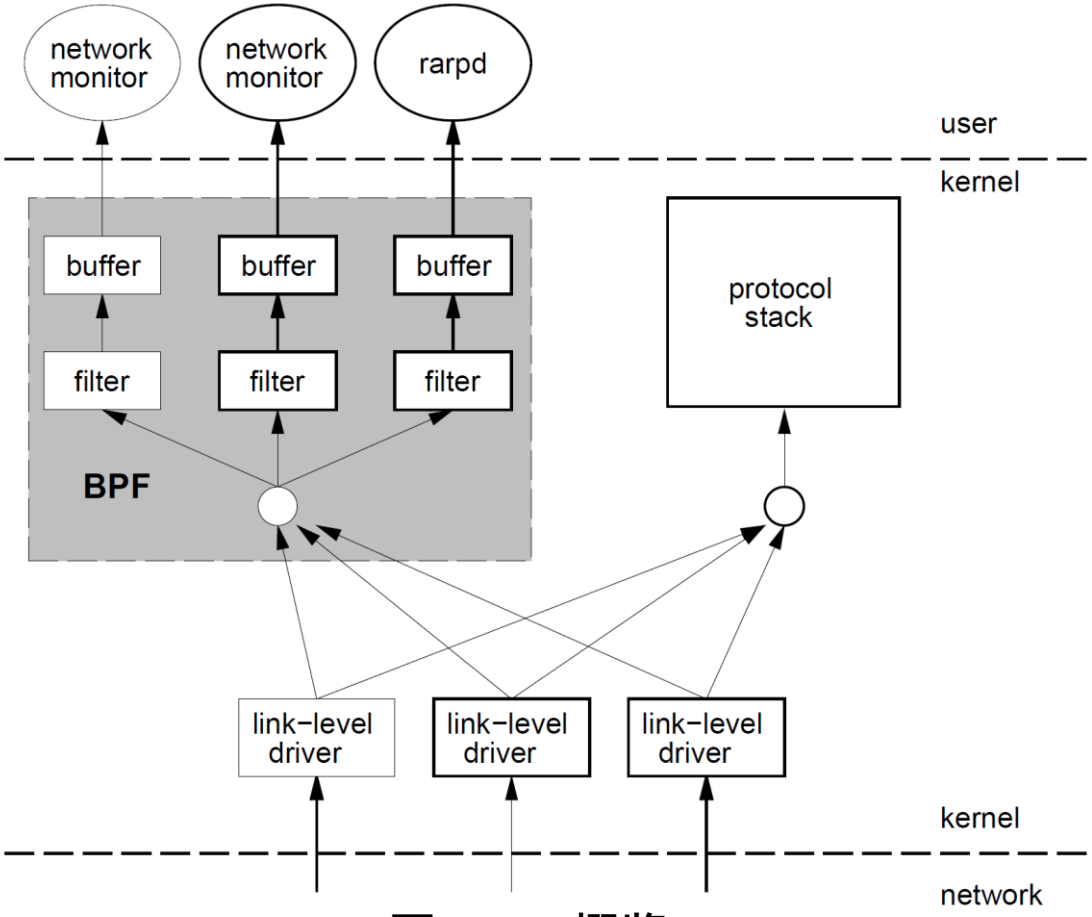


图. BPF 概览

BPF 与 传统编程对比

	Execution model	User defined	Compil-ation	Security	Failure mode	Resource access
User	task	yes	any	user based	abort	syscall, fault
Kernel	task	no	static	none	panic	direct
BPF	event	yes	JIT, CO-RE	verified, JIT	error message	restricted helpers

图 3. 用户态、内核态与 BPF 编程对比

BPF kernel Verifier

check_subprogs	check_helper_mem_access
check_reg_arg	check_func_arg
check_stack_write	check_map_func_compatibility
check_stack_read	check_func_proto
check_stack_access	check_func_call
check_map_access_type	check_reference_leak
check_mem_region_access	check_helper_call
check_map_access	check_alu_op
check_packet_access	check_cond_jmp_op
check_ctx_access	check_ld_imm
check_flow_keys_access	check_ld_abs
check_sock_access	check_return_code
check_pkt_ptr_alignment	check_cfg
check_generic_ptr_alignment	check_btf_func
check_ptr_alignment	check_btf_line
check_max_stack_depth	check_btf_info
check_tp_buffer_access	check_map_prealloc
check_ptr_to_btf_access	check_map_prog_compatibility
check_mem_access	check_struct_ops_btf_id
check_xadd	check_attach_modify_return
check_stack_boundary	check_attach_btf_id

Reference 1A

- https://www.youtube.com/watch?v=_5Z2AU7QTH4&ab_channel=USENIX
- libpcap官方仓库： https://github.com/the-tcpdump-group/libpcap

实验 1B：安全文件传输软件

- **文件传输是网络的基本功能之一**
  - 文件传输属于抽象需求，可细分为多种场景
    - **HTTP：通过构造 HTTP 请求实现文件传输（非典型）**
    - **WebDAV：Http 的扩展，用于在 80 端口进行文件共享**
    - **FTP：远比 HTTP 复杂，专用于文件传输，不加密**
    - **SFTP：FTP 的扩展，保持同样的接口，但是支持加密**
    - **SCP：Secure Copy，基于 SSH 协议在两台主机直接传输**
    - **SMB：微软独家的 Windows 电脑间的文件共享、打印共享协议，性能安全优异**
    - **NFS：Unix 系统间的文件系统级共享，权限控制比较弱**
  - Note：文件共享有别于块存储共享！
    - **块存储常见于 IDC，与本实验不同，但是功能比较相似**
    - **IP-SAN、FC-SAN：基于 SAN 存储网络的高性能块共享**
    - **iSCSI：一种广泛应用的 IP-SAN 技术**

Demo: WinSCP

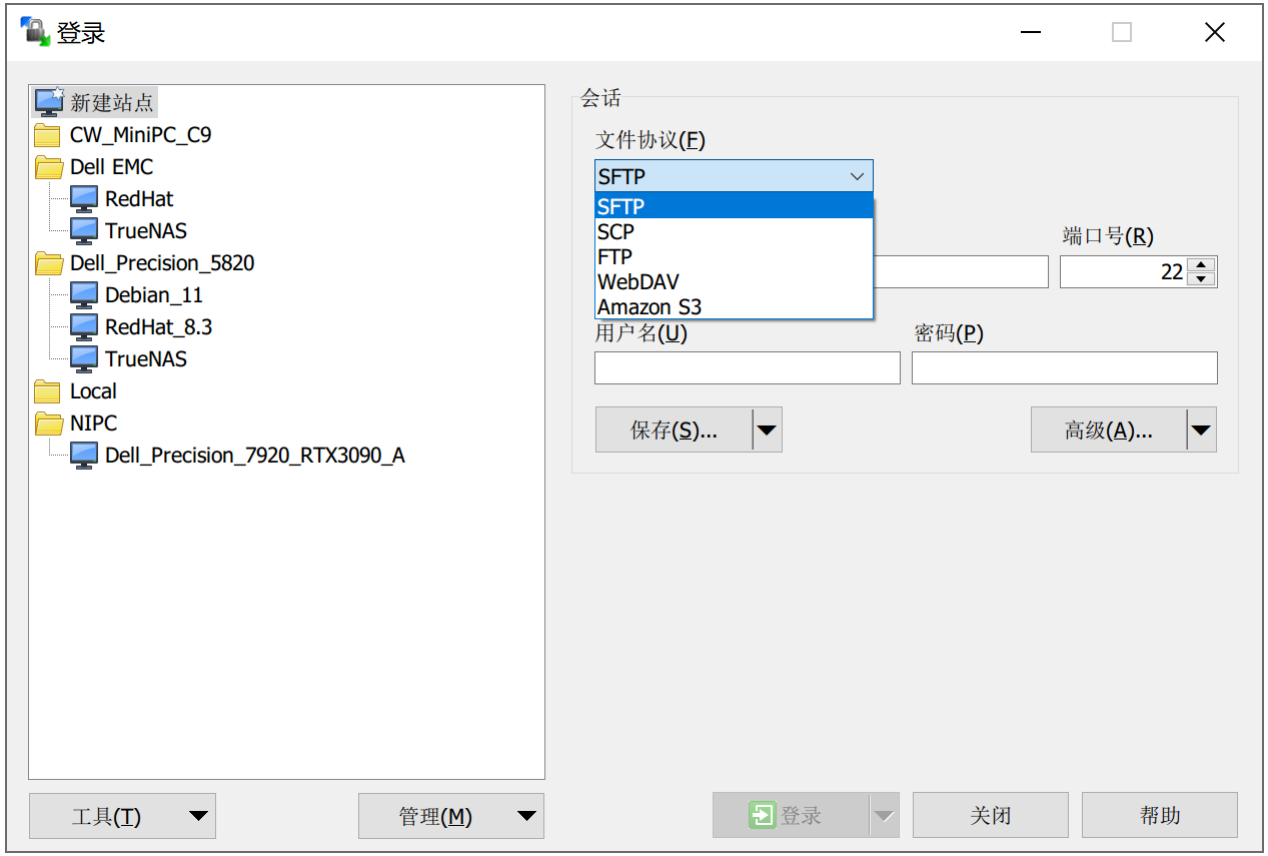


图. WinScp 软件配置界面

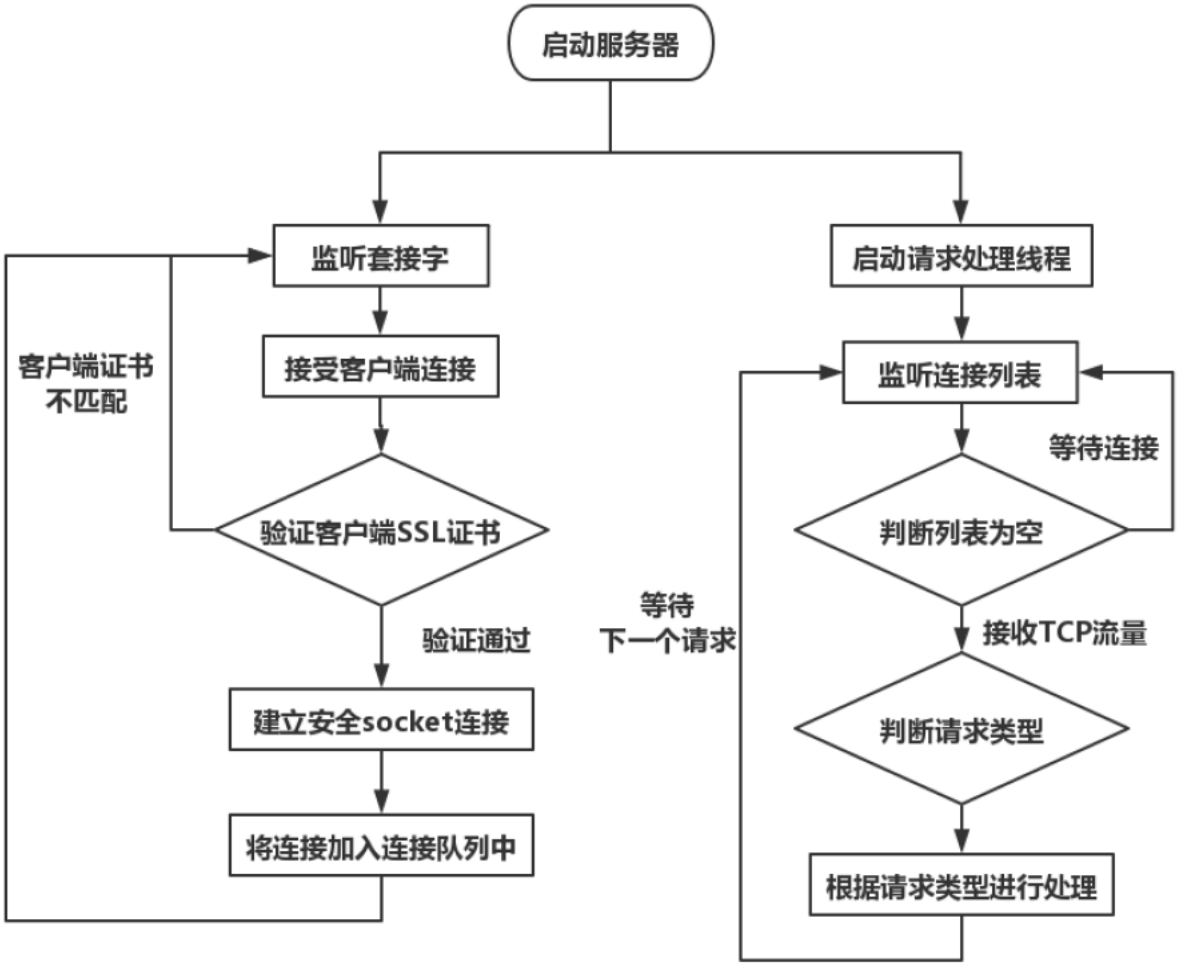
实验 1B：要求与提示

- **要求实现一种简单但安全的文件传输协议**
  - 能保证多用户并发访问时，文件可完整地成功传输
  - 通信全过程可抵御中间人、DDoS、SQL 注入等常见攻击
- **温馨提示**
  - 建议使用 Visual Studio 2022 实现图形化部分
    - **Windows 的 WinForm、WPF 图形库**
    - **macOS 的 Cocoa 图形库**
    - **Linux 的各种图形库（xxQT）**
    - **基于 Native API 的图形库封装：wxWdigets**
  - 由于用来判卷的电脑是 Windows 系统，因此用其他 OS 的同学，需要提供编译成功、运行成功的视频。
    - **视频里可以讲解一下自己的代码与思路，注意录制时长**
    - **视频要展示程序编译、部署、测试的全过程**

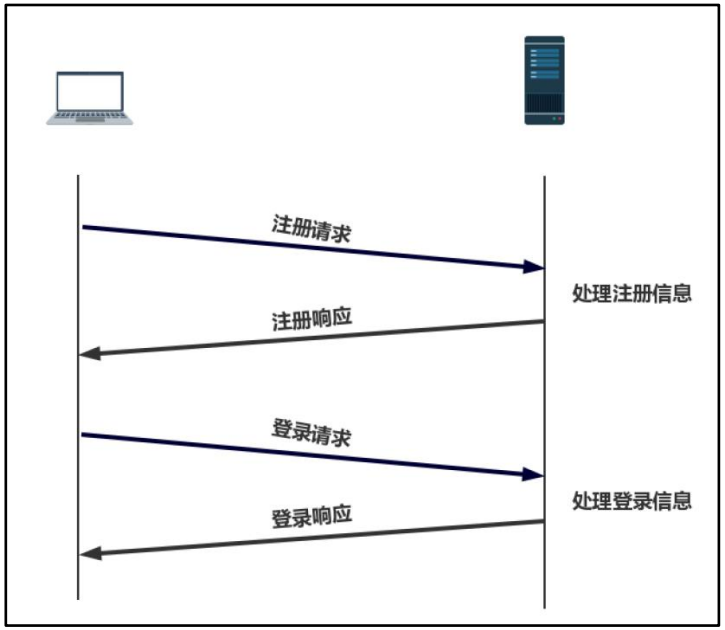
实验 1B：应用层协议设计

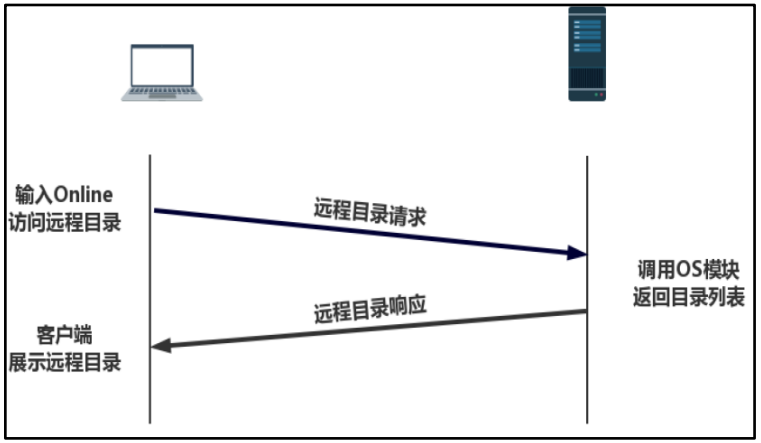
- **需要建立长连接，代码中要明确指示 keep alive**
  - 以下行为是不允许的：~~Client 发起一次请求并得到响应之后，即关闭 TCP 连接。~~需要有守护进程维护连接，直至 Client 被手动关闭。
- **需要在 Report 明确写出应用层协议的设计原理**
  - 图文并茂地叙述协议设计的思路
    - **时序图 (协议的每个模块都要画出时序图)**
    - **协议头部信息描述图（类似 TCP 头部）**
  - 解释协议能准确运行的合理性

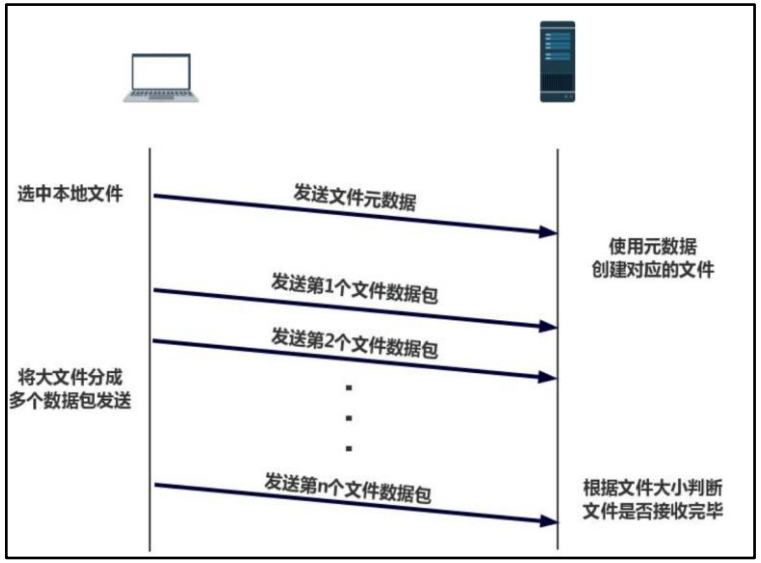
功能流程图



协议时序图







加分项

- **文件断点续传**
- **基于公钥的身份验证**
  - 双向证书验证亦可
- **支持同时上传、下载多个文件**
  - GUI 支持
- **支持服务端根据用户身份，控制传输带宽**
  - VIP

从 TCP 报头谈协议是什么

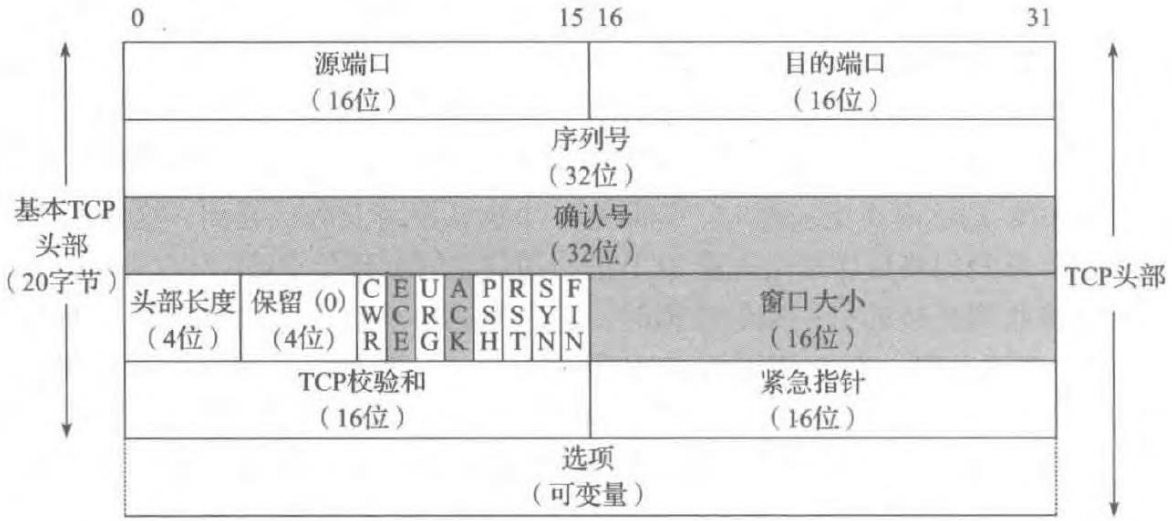


图 12-3 TCP 头部。它的标准长度是 20 字节，除非出现选项。头部长度（Header Length）字段以 32 位字为单位给出头部的大小（最小值是 5）。带阴影的字段（确认号（Acknowledgment Number）、窗口大小（Window Size）以及 ECE 位和 ACK 位）用于与该报文段的发送方关联的相反方向上的数据流

下知地理：TCP 可靠字节流

- **TCP 协议本质是带累积正向确认的滑动窗口协议**
  - TCP 虽然会给网络层提供 Segment，即报文段，但 TCP 本身是面向字节流的，并非是传输片段。
  - TCP 会自动分片
  - TCP 利用反馈、重传机制，在有损信道上实现可靠传输。
- **如何利用 TCP 传输一个固定大小的文件?**
  - 首先建立 TCP 连接：三次握手
  - 发送方用 Socket 发送文件：以 4 KiB 为单位读取本地文件，在读了 N 个 4KiB 之后，终于读取到了 EOF 标志，则最后发送 ($FileSize - 4 * N$) KiB的数据
  - 发送 FIN，关闭连接：四次挥手
  - 接收方只需要傻傻接收，然后把数据包组合起来即可恢复出原文件。

如何建立完善的应用层协议

- **上述场景的局限性在于传输的终结由发送方控制**
  - 在传输完成之前，只有发送方知道文件的具体大小。
  - 文件是否接收完是由发送方关闭连接所确定的。
- **如何克服上述弊端?**
  - 发送方通过自定义应用层协议，提前告知接收方需准备多少磁盘空间（比如 N kiB）来存储要接收的数据。
  - 接收方根据 N 的值，从某一个包开始，截取包内容，并往本地的文件描述符写入数据，一旦接收到了 N kiB 的数据，就完成接受，并回复接收完成的确认。此后也不需要关闭连接。
  - 接收方回复已完成接收后，发送方关闭打开的文件，并对其文件描述符引用计数减一。

安全要点：Server 端 ACL 配置

- **自定义服务端业务代码**
  - 业务端进程需具备严格的访问控制。服务端只允许某个目录被 Client 读取、写入，以减少 Server 进程被黑客控制后产生进一步的损失
  - ACL 主要针对以下项目进行权限控制
    - **用户 User**
    - **用户组 Group**
    - **默认属性 Mask**
  - 另有 setfacl 之外的方法可实现服务器端的访问控制。
  - Server 端的可读写目录配置需以 shell 脚本或其他形式给出明确体现。

提示：协议的实现方法

- **仿照 TCP 报头的样式进行实现**
  - 划分字段，不同的值代表不同的行为
  - 收到报头，按照 struct 进行解析，然后按照对应的逻辑完成对应的任务。
  - 特别注意，报头将出现在每一个应用层 packet 里，要注意协议的传输开销。
- **用 JSON 编码/解析器完成**
  - 这种方法与 TCP/IP 的 bitmap + struct 报头方案在理论上是等价的，但是更加现代化、更灵活
  - 优势：JSON 是服务器端和客户端都理解的描述语言，因此只要定义了双方的通信解析方法，就可以通过 JSON 格式的数据来传输控制报文，无需做字节级的包头解析。

仿 TCP 等报文头的设计

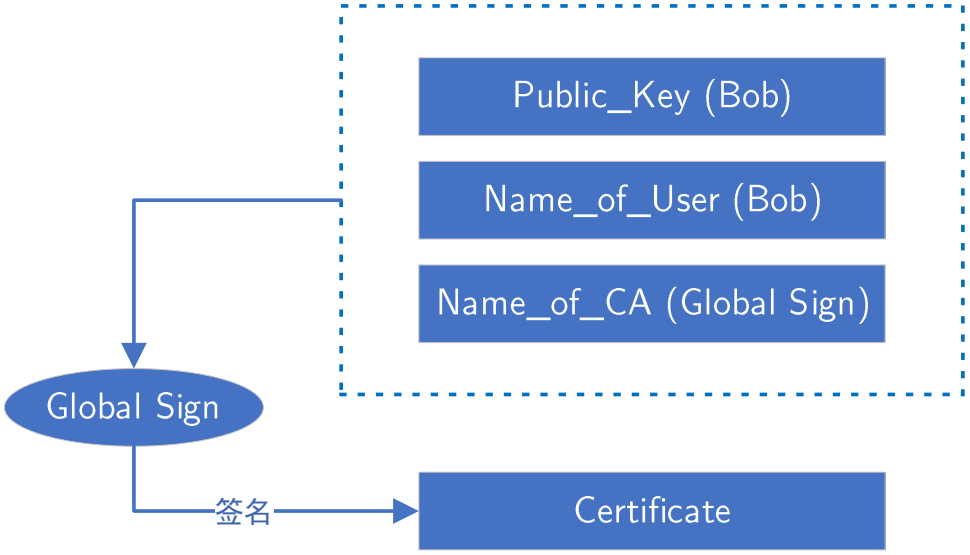
包类型名称	类型解释	包类型定义
REGISTER_REQUEST	注册请求	00
REGISTER_RESPONSE	注册响应	01
LOGIN_REQUEST	登陆请求	11
LOGIN_RESPONSE	登陆响应	12
CATALOG_REQUEST	远程文件目录请求	21
CATALOG_RESPONSE	远程文件目录响应	22
FILE_REQUEST	文件下载请求	31
FILE_METADATA	文件元数据通知	32
FILE_CONTENT	文件内容	33

PKI Internals

- **PKI 技术：解决公钥密码算法中的信任问题**
  - 要使用公钥算法，必须明确公钥的归属问题
  - Diffie and Hellman 首次公开公钥密码算法时便已意识到此问题，他们设想了一种 online 的可信资料库，但面临许多问题：
    - **频繁查询带来的性能开销问题**
    - **离线使用问题**
  - 1978 年，Kohnfelder 提出数字证书 (Certificate) 的概念，由证书认证中心 (Certification Authority, CA) 签发证书来解决公钥归属问题。PKI 中的三个实体：
    - **CA：负责为他人签发证书 (CA 的私钥至关重要！)**
    - **证书持有者 (Certificate Holder)：有 CA 签发的证书和自己的对应私钥 (而非 CA 的私钥)**
    - **依赖方 (Relying Party)：安装了 CA 的证书 (CA 的公钥)**

CA 签名过程

- **签名过程很严格、很繁琐，需要大量人工参与**
  - CA 保证证书申请者与证书所标识的是同一个实体
  - 证书的身份信息也要进行仔细审查
  - 检查 Optional 的信息：电子邮件、IP 地址、DNS 域名



CA 给自己签发证书的过程


```
# 创建 CA 的信息
# 生成 CA 私钥
openssl genrsa -out ca-key.pem 1024
# 生成 CA 请求文件
openssl req -out ca-req.csr -key ca-key.pem -new
# 自签署 CA 的根证书
openssl x509
        -out ca-cert.cer
        –req –in ca-req.csr
        -signkey ca-key.pem
        -days 365
```

CA 给 Server 签发证书的过程


```
# CA 签发服务器的证书
# 生成服务器私钥
openssl genrsa -out server-key.pem 1024
# 生成服务器请求文件 (公钥 + 信息)
openssl req –out server-req.csr -key server-key.pem -new
# 用 CA 的根证书、CA 的私钥签署服务器证书
openssl x509
        –out server-cert.cer
        –req -in server-req.csr
        -signkey server-key.pem
        -CA ca-cert.pem
        -CAkey ca-key.pem
        -CAcreateserial
        -days 365
```

CA 给 Client 签发证书的过程


```
# 创建客户端证书
# 生成客户端私钥
openssl genrsa  -out client-key.pem 1024
# 生成客户端请求文件
openssl req -out client-req.csr -key client/client-key.pem -new
# 用 CA 的根证书、CA 私钥签发客户端证书
openssl x509
        -out client-cert.cer
        -req -in client-req.csr
        -signkey client-key.pem
        -CA ca-cert.cer
        -CAkey ca-key.pem
        -CAcreateserial
        -days 365
```

证书使用说明

- **证书使用者不限于计算机**
  - 进程、线程等运行时对象之间也可以用证书校验身份
  - 以邮件程序为例：
    - **Alice 给 Bob 发送邮件时，在写好邮件之后，用自己的证书给邮件进行签名 (证书持有者)**
    - **Alice 准备发送时用 Bob 的证书给邮件加密 (证书依赖者)**
- **证书需要能撤销**
  - 证书持有者丢失了自己的私钥
    - **需要重新申请新证书**
    - **废除旧的证书**
  - 绕过 CA 撤销别人的证书将直接导致服务不可用

技术分享建议

- **先描述好一个问题。**
  - 描述一个能让大家感同身受的编程、设计问题，不要一上来就讲你怎么做的
- **How 比 What 重要。**
  - 要有不同技术的比较
- **一定要有 Best Practice 或方法论总结**

Thanks

- **Thank you for your listening!**
- **contact information**
  - Email: liup@nipc.org.cn