清華大學
Tsinghua University

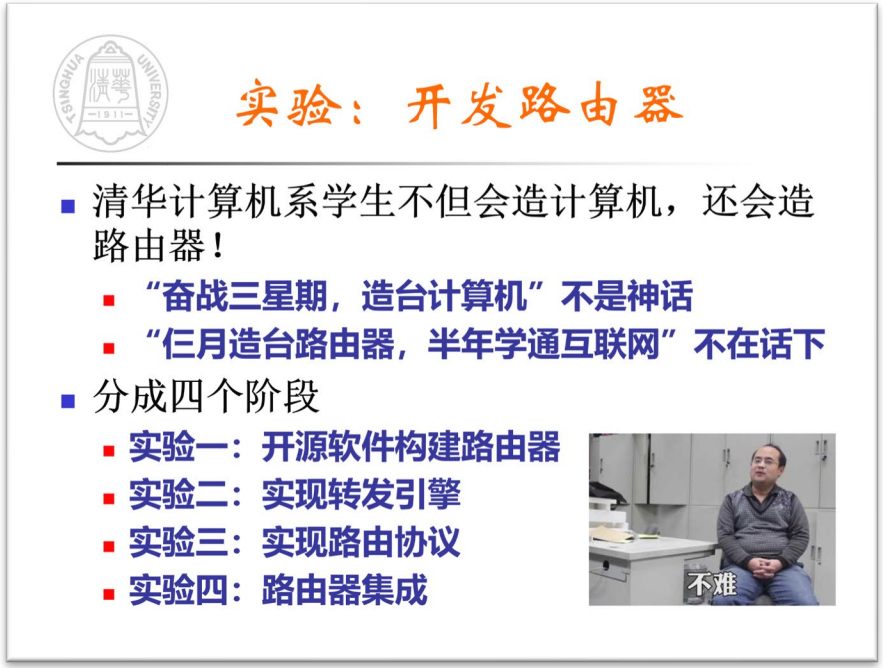
计算机网络原理实验系统简介

陈晟祺

SHENGQI.CHEN@TUNA.TSINGHUA.EDU.CN

背景

- 2018年秋季学期起，《计算机网络原理》课程进行实验改革
- 由吴建平院士提出，徐明伟、全成斌老师全程参与指导
  - “仨月造台路由器，半年学通互联网”
- 2019年秋季学期起，计算机系学生团队接手实验平台
  - 陈晟祺、陈嘉杰、刘晓义、高一川



为什么要接手?

- 原本的实验由某外包公司负责，代码质量低劣
- 细节繁多，实现困难，学习到的内容并非本质
- 文档粗糙，格式混乱
- 助教对于实验了解甚少，无法帮助
- 实验室的电脑实在太慢了
- 吐槽详见：
  - https://harrychen.xyz/2020/06/20/cs-course-reform/
  - “现实部分”

```
main : main.o  fib.o checksum.o arp_query.o ether_encapsulate.o route_manage.o
        cc -o main  main.o  fib.o checksum.o arp_query.o ether_encapsulate.o route_manage.o -lpthread -g
main.o      : main.c      fib.h checksum.h arp_query.h ether_encapsulate.h route_manage.h
        cc -c main.c -g
fib.o : fib.c fib.h
        cc -c fib.c -g
checksum.o  : checksum.c  checksum.h
        cc -c checksum.c -g
arp_query.o    : arp_query.c   arp_query.h
        cc -c arp_query.c -g
ether_encapsulate.o : ether_encapsulate.c ether_encapsulate.h
        cc -c ether_encapsulate.c -g
route_manage.o : route_manage.c route_manage.h
        cc -c route_manage.c -g
clean :
        rm -rf main  main.o  fib.o checksum.o arp_query.o ether_encapsulate.o route_manage.o
SUBDIR = $(shell ls ./ -R | grep /)
SUBDIRS  = $(subst :,/,$(SUBDIR))
SOURCE = $(foreach dir, $(SUBDIRS),$(wildcard $(dir)*.o))
#clean:
#       rm  -rf $(SOURCE)
```

1）安装 g+

yum install g

2）安装 gav

yuminstall g

3）安装 ncu

yum install r

yum install r

4）安装 libr

第一种方法

如果第一种方法编译 quagga-0.99.21 时出现 vtysh 这个目录下不能 make 过的情况，请使用第二种方法安装 readline。

第二种方法：

下载 readline-6.2.tar.gz

wget -c ftp://ftp.gnu.org/gnu/readline/readline-6.2.tar.gz

解压 readline-6.2.tar.gz

tarxzf readline-6.2.tar.gz

编译安装

进入解压出来的目录中

./configure

make&& make install

ldconfig

5）安装 quagga-0.99.21

1.解压压缩包：

tarxzf quagga-0.99.21_20131218.tar.gz

2.编译安装

./configure--enable-vtysh --enable-zebra --enable-ripd--disable-doc --disable-babeld --enable-isisd=no --disable-bgpd --disable-ospfd --disable-ospf6d --disable-ospfclient--enable-user=root --enable-group=root -enable-vty-group=root

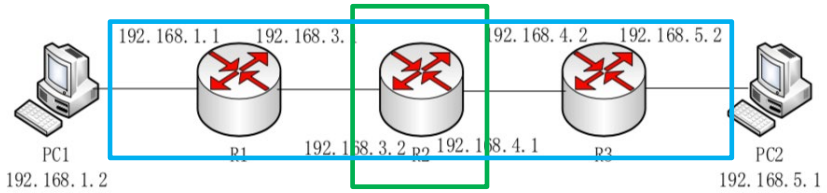
make&& make install

下一跳信息结构体如下：

```
structnexthop
{
structnexthop*next;
char*ifname;
unsignedintifindex;/* Interface index. */
structin_addrnexthopaddr;/* Nexthop address */
}
```

实验构成

- 40% 编程作业
  - IP checksum、IP 头更新、路由表查询、RIP 协议处理
  - GitLab 仓库 CI 自动运行+评分
- 30% 个人实验
  - 基于 GitLab CI 和硬件实验平台
  - 正确性测试 + 性能测试
- 30% 组队实验
  - 与个人实验相同，但使用三个同学的代码



要做什么？

- 一套接口简单易用的实验框架：https://github.com/thu-cs-lab/Router-Lab
- 一份清晰科学的实验文档：https://github.com/thu-cs-lab/Router-Lab-Docs
- 一套方便且准确的评测系统：TanLabs

ROUTER-LAB 的 HAL 部分

- HAL：硬件抽象层
  - 宇翔总是对的
  - 屏蔽实现不一致，提供易用的接口（raw socket 太难用了！）
  - 在 Linux / macOS 上使用 `libpcap` 同时支持基于文件的后端和基于真实网络接口的后端
  - 嵌入式系统（如 FPGA SoC）：直接读写 MMIO 寄存器

```
int HAL_Init(HAL_IN int debug, HAL_IN uint32_t if_addrs[N_IFACE_ON_BOARD]);
uint64_t HAL_GetTicks();
int HAL_ArpGetMacAddress(HAL_IN int if_index, HAL_IN uint32_t ip, HAL_OUT macaddr_t o_mac);
int HAL_GetInterfaceMacAddress(HAL_IN int if_index, HAL_OUT macaddr_t o_mac);
int HAL_ReceiveIPPacket(HAL_IN int if_index_mask, HAL_OUT uint8_t *buffer, HAL_IN size_t length, HAL_OUT macaddr_t src_mac,
                        HAL_OUT macaddr_t dst_mac, HAL_IN int64_t timeout, HAL_OUT int *if_index);
int HAL_SendIPPacket(HAL_IN int if_index, HAL_IN uint8_t *buffer, HAL_IN size_t length, HAL_IN macaddr_t dst_mac);
```

ROUTER-LAB 的作业部分

- 编程作业：checksum、forwarding、lookup、protocol
  - 使用 libpcap 的文件后端和手写 Python 脚本，自动比对学生结果
  - 使用 GitLab CI，在学生修改代码时自动运行测试
- 路由器实现：router
  - 使用 libpcap 的网络接口后端，编译出可执行文件
  - 同样使用 CI，编译得到 artifact 供测试服务器使用
- .gitlab-ci.yml 检查自己签名，防止学生修改
  - 有效防止完全复制去年的代码

真机实验部分

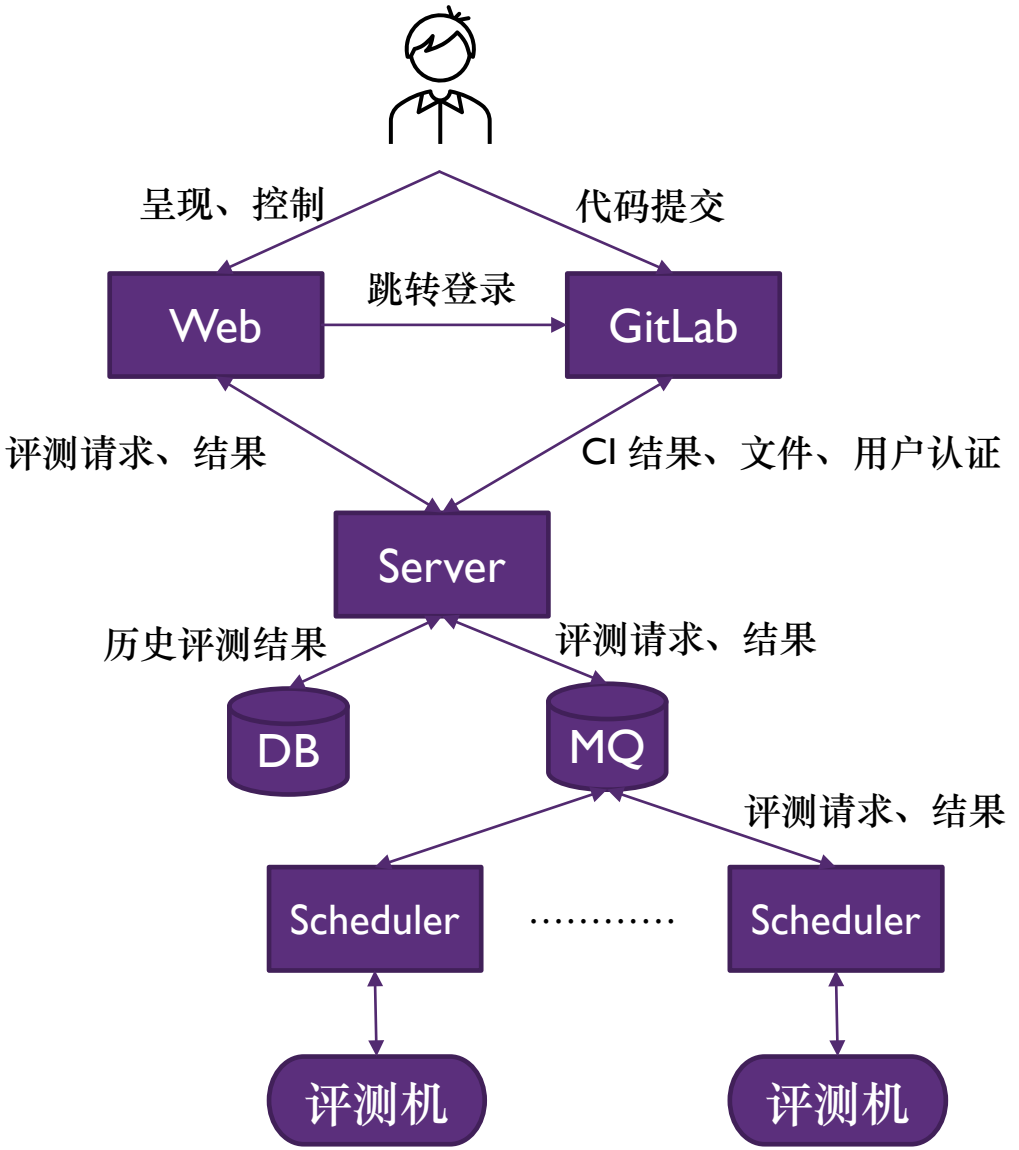
- 使用树莓派 + Linux 作为实验平台（学生调试、评测均使用）
- 标准实现：
  - 四层以下：Linux 网络栈
  - 路由协议：BIRD Internet Routing Daemon
  - 应用软件：ping、curl、iperf3 等
- 学生路由器要求实现：
  - 三层路由转发
  - RIP 路由协议

实验文档

- https://lab.cs.tsinghua.edu.cn/router/doc/
- mkdocs-material 真香
- 使用杰哥的 webhookd 进行 CI
- 涵盖了实验的完整内容：
  - 实验要求、测试方法、评分标准
  - 调试方法、真机测试方法（netns）
  - Linux、Git、Make 等简易教程

软件实验平台：TANLABS

- 分为多个解耦合的组件：
  - TanLabs-Web
  - TanLabs-Server
  - TanLabs-Scheduler
  - TanLabs-RPi
- 以及自己设计的硬件评测机



软件平台（WEB 部分）

- TanLabs-Web
  - 基于 React 的纯前端
  - 使用 GitLab OAUTH 认证用户，显示作业成绩、评测结果
- TanLabs-Server
  - 基于 Rust actix 的后端，读写 MySQL、消息队列
  - 为前端提供 JSON API
  - 同时接受 GitLab HTTP 请求（web hook），记录 CI 结果写入数据库（不可否认）
  - 将评测请求（以及 binary url）推入消息队列，拉取评测结果写入数据库

SQL 建表一时爽……

- ……期末 join 火葬场
- 特殊情况需要手工调整表内容
  - 换队友
  - 拆队伍

带评测结果：

```sql
SELECT student_id, real_name,
       s.build_id as stage1,
       s.late_build_id as stage1_late,
       s.run_id as stage2,
       s.late_run_id as stage2_late,
       g.run_id as stage3,
       g_late.run_id as stage3_late,
       b.grade_checksum * 2.5 + b.grade_forwarding * 2.5 + b.grade_lookup * 2.5 + b.grade_protocol / 12.0 * 10.0 as stage1_grade,
       b_late.grade_checksum * 2.5 + b_late.grade_forwarding * 2.5 + b_late.grade_lookup * 2.5 + b_late.grade_protocol / 12.0 * 10.0 as stage1_late_gr
       i.result->"$.ping_1to2" as stage2_ping_1to2, i.result->"$.ping_2to1" as stage2_ping_2to1,
       i.result->"$.nc_1to2" as stage2_nc_1to2, i.result->"$.nc_2to1" as stage2_nc_2to1,
       i.result->"$.ping_ttl_1to2" as stage2_ping_ttl_1to2, i.result->"$.ping_ttl_2to1" as stage2_ping_ttl_2to1,
       i.result->"$.route_r1" as stage2_route_r1, i.result->"$.route_r3" as stage2_route_r3,
       i.result->"$.iperf3_bitrate" as stage2_iperf3_bitrate,
       i_late.result->"$.ping_1to2" as stage2_late_ping_1to2, i_late.result->"$.ping_2to1" as stage2_late_ping_2to1,
       i_late.result->"$.nc_1to2" as stage2_late_nc_1to2, i_late.result->"$.nc_2to1" as stage2_late_nc_2to1,
       i_late.result->"$.ping_ttl_1to2" as stage2_late_ping_ttl_1to2, i_late.result->"$.ping_ttl_2to1" as stage2_late_ping_ttl_2to1,
       i_late.result->"$.route_r1" as stage2_late_route_r1, i_late.result->"$.route_r3" as stage2_late_route_r3,
       i_late.result->"$.iperf3_bitrate" as stage2_late_iperf3_bitrate,
       g.result->"$.ping_1to2" as stage3_ping_1to2, g.result->"$.ping_2to1" as stage3_ping_2to1,
       g.result->"$.nc_1to2" as stage3_nc_1to2, g.result->"$.nc_2to1" as stage3_nc_2to1,
       g.result->"$.ping_ttl_1to2" as stage3_ping_ttl_1to2, g.result->"$.ping_ttl_2to1" as stage3_ping_ttl_2to1,
       g.result->"$.ping_unreachable_1" as stage3_ping_unreachable_1, g.result->"$.ping_unreachable_2" as stage3_ping_unreachable_2,
       g.result->"$.iperf3_bitrate" as stage3_iperf3_bitrate, g.result->"$.iperf3_datagram" as stage3_iperf3_datagram,
       g.result->"$.small_ping_1" as stage3_small_ping_1, g.result->"$.small_ping_2" as stage3_small_ping_2,
       g.result->"$.medium_ping_1" as stage3_medium_ping_1, g.result->"$.medium_ping_2" as stage3_medium_ping_2,
       g.result->"$.large_ping_1" as stage3_large_ping_1, g.result->"$.large_ping_2" as stage3_large_ping_2,
       g_late.result->"$.ping_1to2" as stage3_late_ping_1to2, g_late.result->"$.ping_2to1" as stage3_late_ping_2to1,
       g_late.result->"$.nc_1to2" as stage3_late_nc_1to2, g_late.result->"$.nc_2to1" as stage3_late_nc_2to1,
       g_late.result->"$.ping_ttl_1to2" as stage3_late_ping_ttl_1to2, g_late.result->"$.ping_ttl_2to1" as stage3_late_ping_ttl_2to1,
       g_late.result->"$.ping_unreachable_1" as stage3_late_ping_unreachable_1, g_late.result->"$.ping_unreachable_2" as stage3_late_ping_unreachable_
       g_late.result->"$.iperf3_bitrate" as stage3_late_iperf3_bitrate, g_late.result->"$.iperf3_datagram" as stage3_late_iperf3_datagram,
       g_late.result->"$.small_ping_1" as stage3_late_small_ping_1, g_late.result->"$.small_ping_2" as stage3_late_small_ping_2,
       g_late.result->"$.medium_ping_1" as stage3_late_medium_ping_1, g_late.result->"$.medium_ping_2" as stage3_late_medium_ping_2,
       g_late.result->"$.large_ping_1" as stage3_late_large_ping_1, g_late.result->"$.large_ping_2" as stage3_late_large_ping_2
       FROM users
    LEFT JOIN submissions s on users.user_id = s.user_id
    LEFT JOIN groups g1 on g1.user1_student_id = users.student_id
    LEFT JOIN groups g2 on g2.user2_student_id = users.student_id
    LEFT JOIN groups g3 on g3.user3_student_id = users.student_id
    LEFT JOIN builds b on b.build_id = s.build_id
    LEFT JOIN builds b_late on b_late.build_id = s.late_build_id
    LEFT JOIN runs i on i.run_id = s.run_id
    LEFT JOIN runs i_late on i_late.run_id = s.late_run_id
    LEFT JOIN runs g on g.run_id = COALESCE(g1.final_run_id, g2.final_run_id, g3.final_run_id)
    LEFT JOIN runs g_late on g_late.run_id = COALESCE(g1.late_final_run_id, g2.late_final_run_id, g3.late_final_run_id);
```

软件平台（硬件管理）

- TanLabs-RPi
  - 统一管理树莓派，配置 PXE，DHCP
  - 基于 buildroot 的 rootfs
  - Grafana 监控树莓派状态
- TanLabs-Scheduler
  - 使用 Rust 编写
  - 从消息队列拉取真机评测请求统一调度
  - 运行真机评测
    - 动态修改交换机配置以控制网络拓扑（通过 NETCONF 协议）
    - 在树莓派上分别部署学生实现（从 CI 下载）和标准实现（BIRD、Linux 网络栈）
    - 运行各类测试（ping、curl、iperf、BIRD 等），报告结果

硬件评测机

- H3C S5000 系列交换机
  - 48 * RJ45 1000 Mbps + 4 * SFP+ 10 Gbps
- 18 * Raspberry Pi 4
  - PoE 供电，PXE 无卡启动（但需要逐个打开 PXE）
- 定制硬件
  - 拆卸交换机，分离 PoE 供电板，诱骗机箱风扇电源
  - 固定在 2U 机箱中（高一川设计固定组件）
- 共制作 4 套相同的评测机
  - 耗费约 10 博士生 * 天
- 实测温度稳定在 30°C 以下

实验集群部署

- 普通 X86 服务器 + 评测机，rackmount
- TanLabs 组件
  - 支持使用 Docker / Kubernetes 自动化运维，组件天然解耦合
  - 运行多个 scheduler 管理多个评测集群
- GitLab
  - 可使用已有部署，新部署也很简单（Docker）
  - 需要 CI Runner，可与 TanLabs 部署在同一台服务器上
  - 本地使用泰山 200 系列 aarch64 服务器

使用情况

- 2020-2021 秋季学期《计算机网络原理》
  - 两个班级，共约 250 人，平均上线三台评测机（最后阶段四台）
  - 评测基本正常，队列没有拥堵
  - 杰哥手工修复若干次 race 等问题
- 2020-2021 春季学期《计算机网络原理》

未来展望

- 与 TanLabs Speed Tester 集成，使用 FPGA 作为测速实现
  - https://laekov.com.cn/view/3b42dc
  - https://mp.weixin.qq.com/s/CpeO8kxeLnEPomIC8P_8Mg
- 集成自动查重系统（jieplag）
  - 妈妈再也不用担心我看 MOSS 到老眼昏花了
- 整理代码、（可能）开源
  - 考虑使用 double license 进行授权

THANKS & QA