

THINROUTER

计网联合实验 第 4 组

2019.12

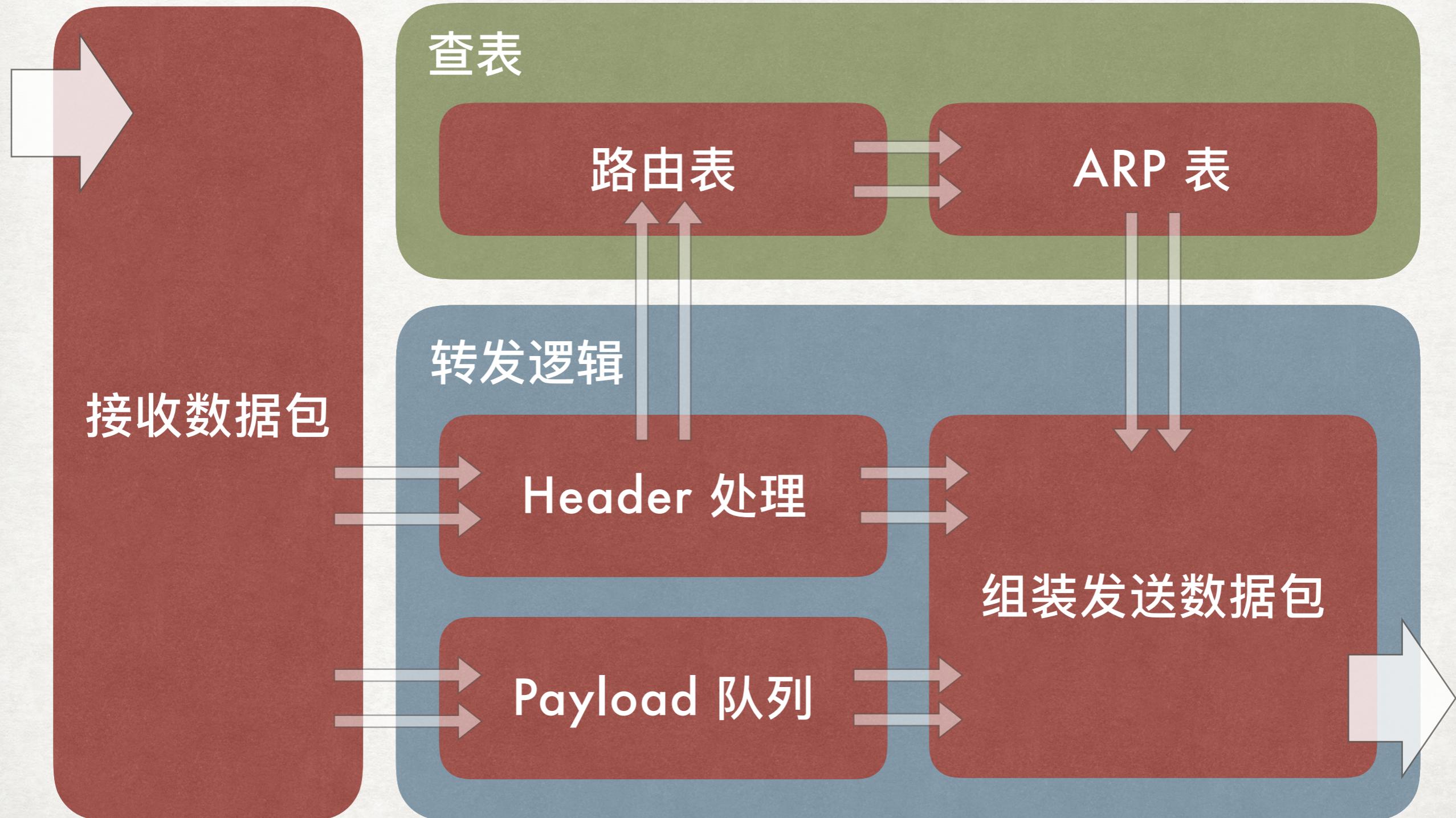
我们做了什么

- 硬件实现路由器功能
- CPU 提供 ThinDOS (假的)
- 对不起我们卷了

```
Console@ThinRouter.4 initialized
root@ThinRouter.4:~$ route
192.168.0.0/24    on port 1
192.168.1.0/24    on port 2
192.168.2.0/24    on port 3
192.168.3.0/24    on port 4
160.83.210.0/23   via 10.4.1.128 metric=2
184.0.0.0/5        via 10.4.1.128 metric=2
143.46.224.0/23   via 10.4.1.128 metric=2
149.198.0.0/15    via 10.4.1.128 metric=2
102.61.55.0/25    via 10.4.1.128 metric=2
103.192.0.0/11    via 10.4.1.128 metric=2
225.224.0.0/11    via 10.4.1.128 metric=2
11.172.158.0/23   via 10.4.1.128 metric=2
141.24.86.136/30  via 10.4.1.128 metric=2
130.80.0.0/14     via 10.4.1.128 metric=2
210.64.145.88/29  via 10.4.1.128 metric=2
130.59.178.53/32  via 10.4.1.128 metric=2
245.171.96.0/20   via 10.4.1.128 metric=2
110.128.0.0/10    via 10.4.1.128 metric=2
142.214.0.0/19    via 10.4.1.128 metric=2
184.142.234.104/29 via 10.4.1.128 metric=2
65.120.165.96/27  via 10.4.1.128 metric=2
147.212.117.64/26 via 10.4.1.128 metric=2
^C
root@ThinRouter.4:~$
```

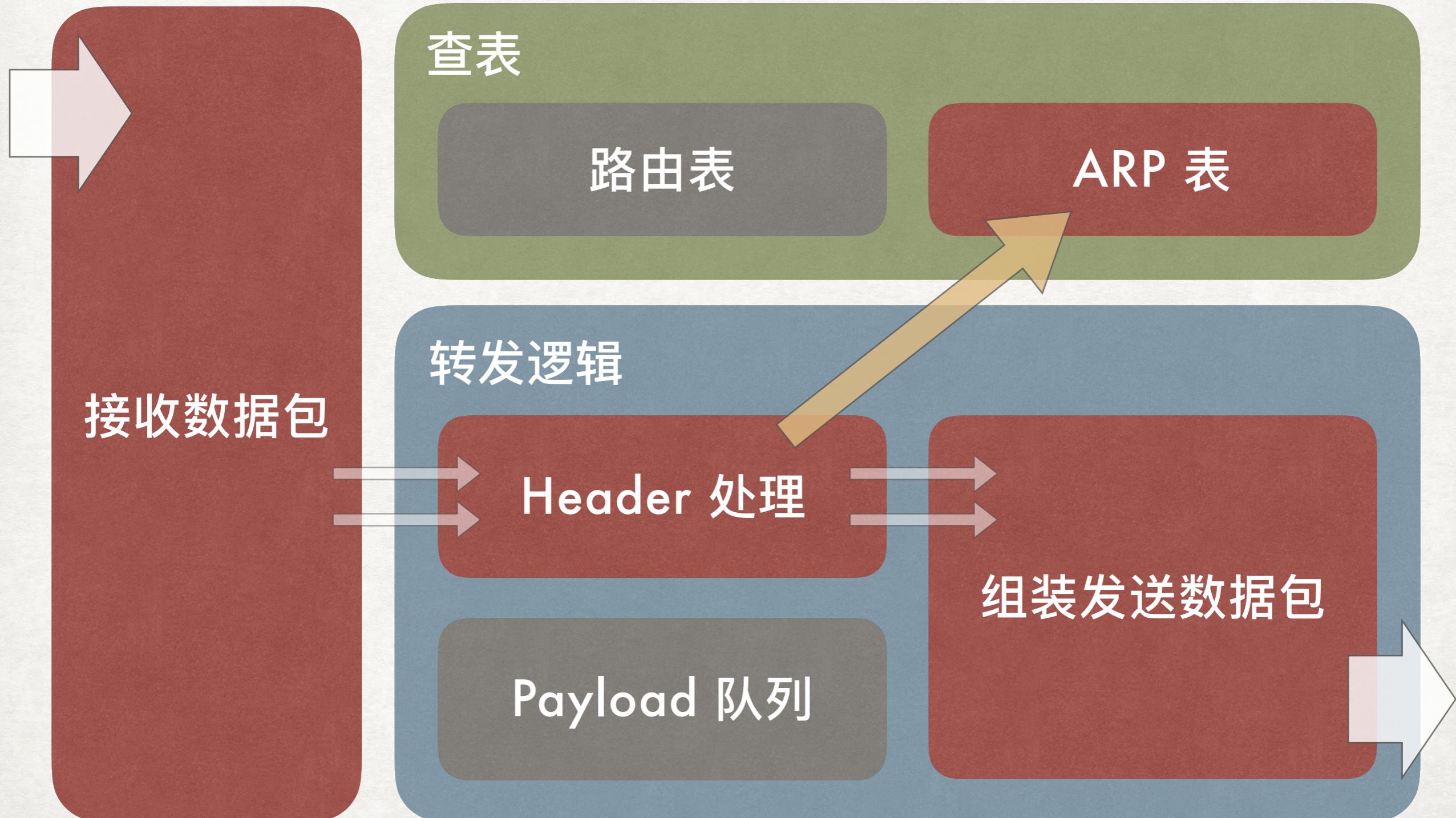
路由器部分

- 硬件流水转发 (IP 包)



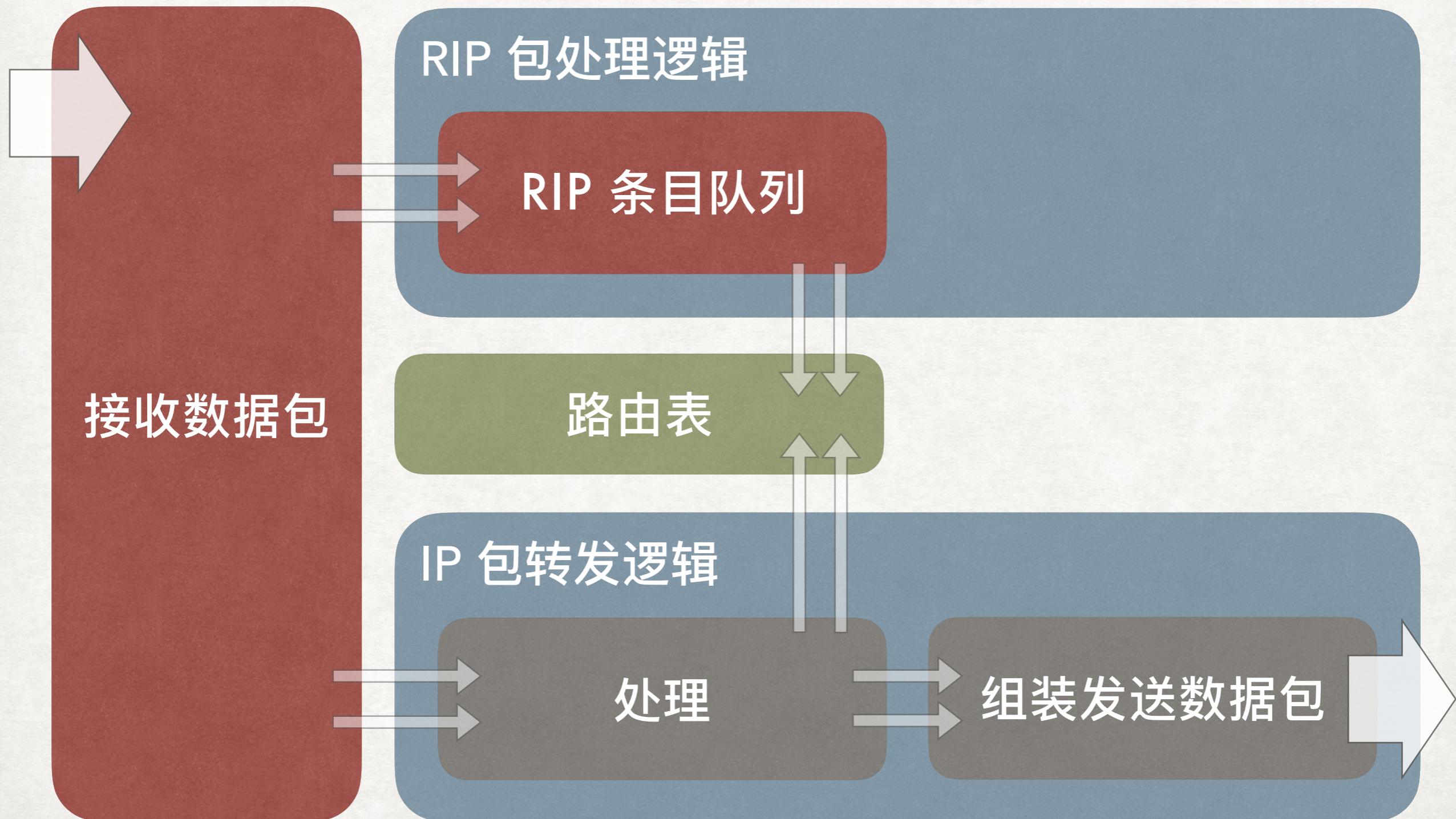
路由器部分

- ARP 表学习



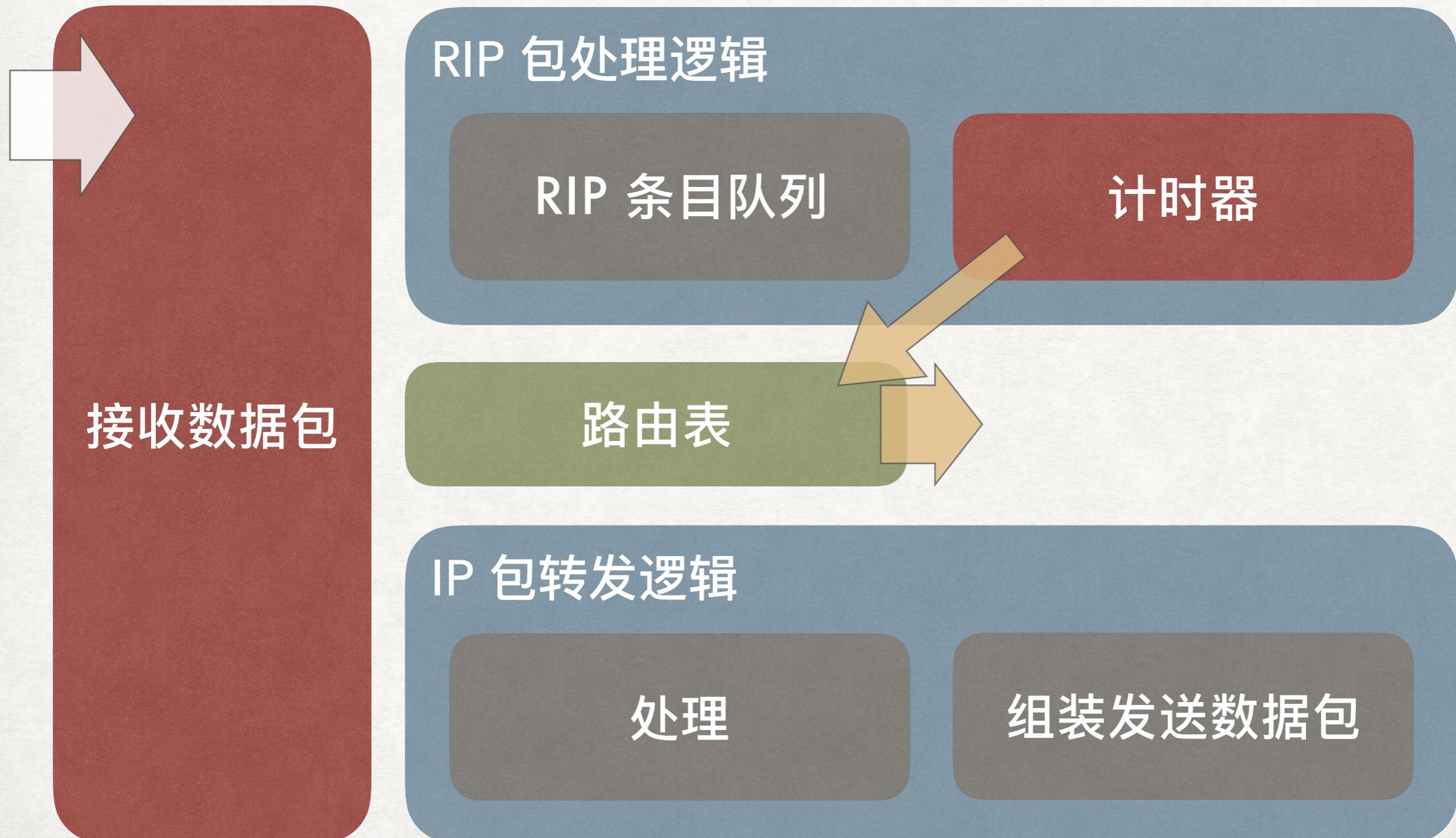
路由器部分

- 硬件处理 RIP 包



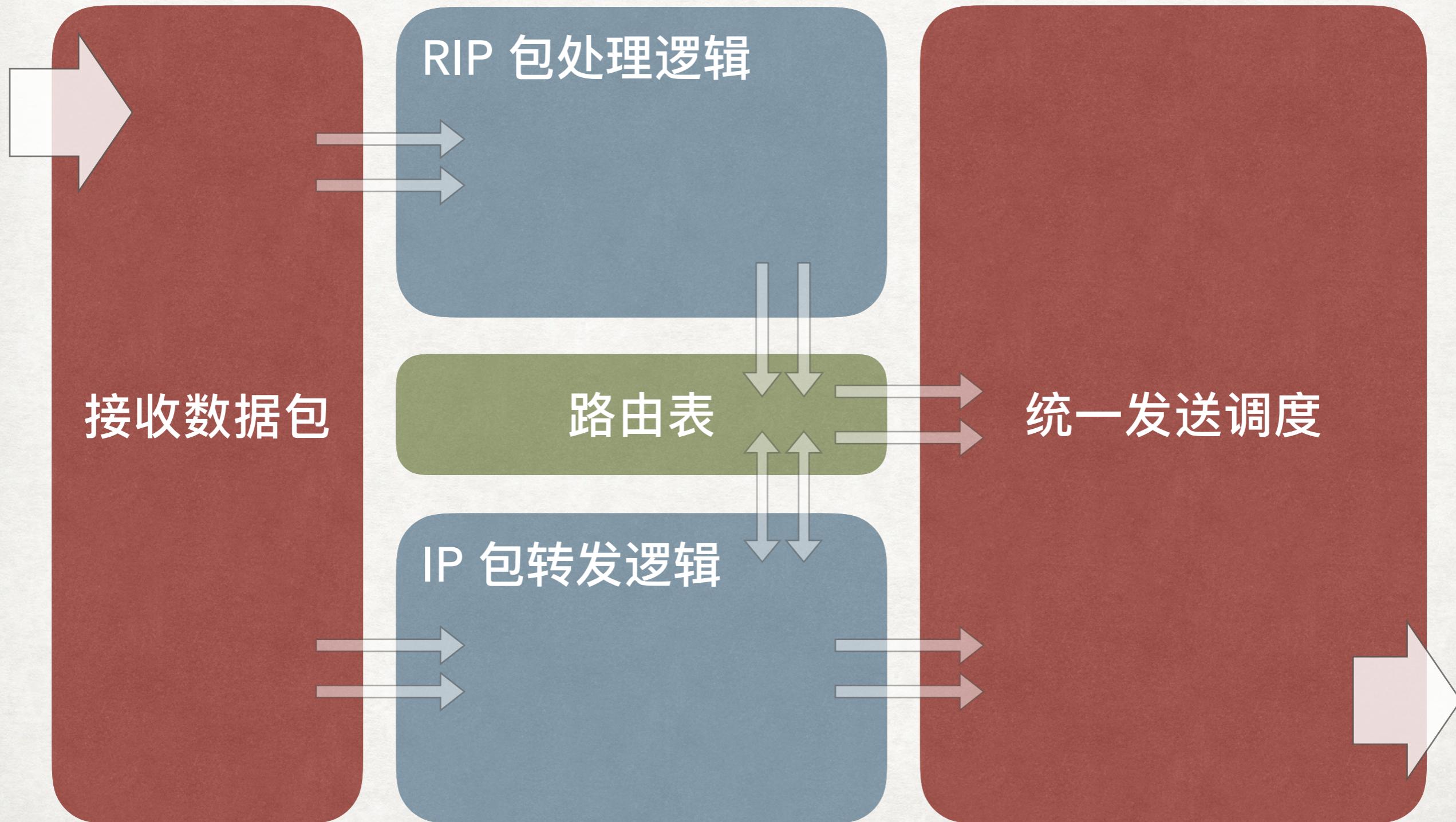
路由器部分

- 硬件发送 RIP 包



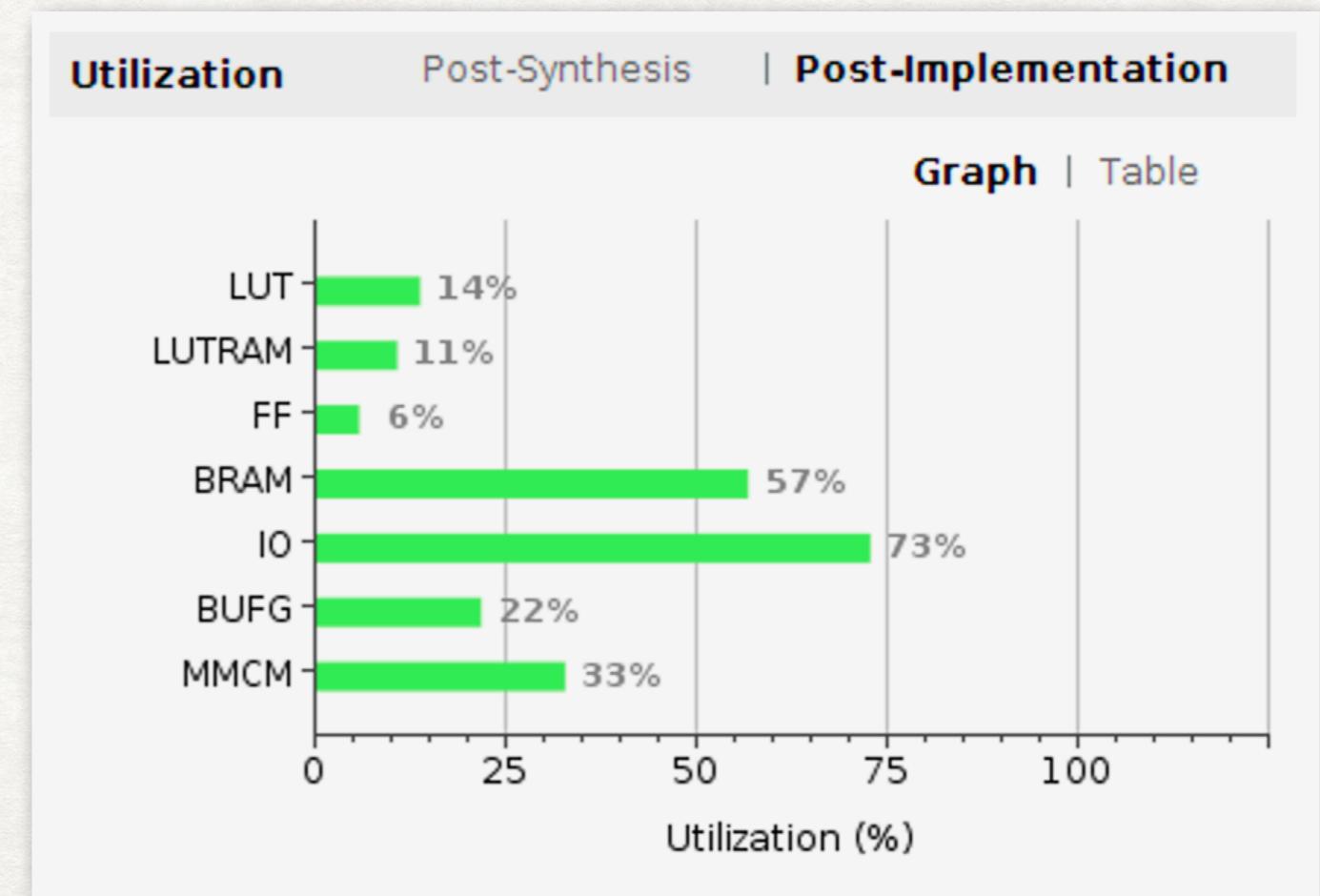
路由器部分

- 整体模块



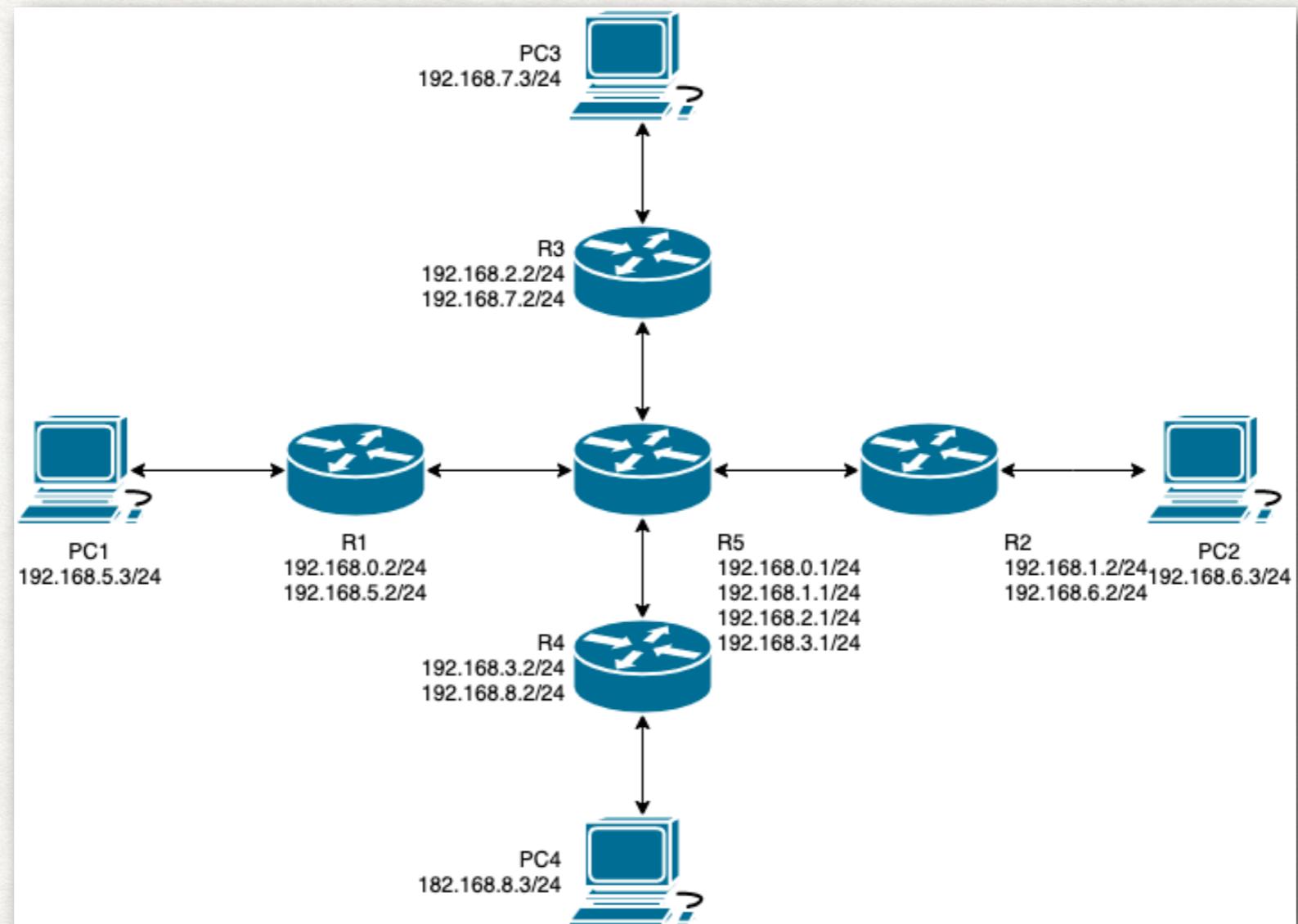
路由器资源占用

- Artix-7 的资源是非常富裕的
- ~40% BRAM 可以存下至少 8192 条路由
- ~10% LUT 可以实现全部路由器的逻辑



路由器性能

- @jiegec 提供测试



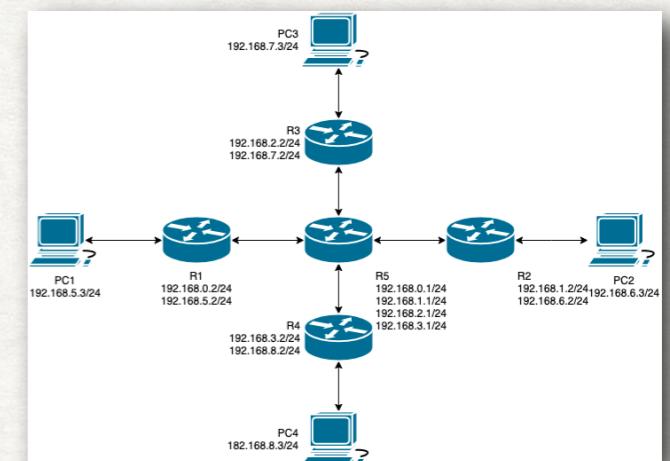
路由器性能

- 直连转发 <1ms
- 经过一层 Bird 转发 ≈1ms

```
lyricz — ping 10.0.4.5 -S 10.0.4.2 — ping — ping 10.0.4.5 -S 10.0.4.2 — 80×24
64 bytes from 10.0.4.5: icmp_seq=0 ttl=63 time=1.372 ms
64 bytes from 10.0.4.5: icmp_seq=1 ttl=63 time=1.027 ms
64 bytes from 10.0.4.5: icmp_seq=2 ttl=63 time=1.237 ms
64 bytes from 10.0.4.5: icmp_seq=3 ttl=63 time=0.958 ms
64 bytes from 10.0.4.5: icmp_seq=4 ttl=63 time=1.207 ms
64 bytes from 10.0.4.5: icmp_seq=5 ttl=63 time=1.421 ms
64 bytes from 10.0.4.5: icmp_seq=6 ttl=63 time=0.854 ms
64 bytes from 10.0.4.5: icmp_seq=7 ttl=63 time=0.963 ms
64 bytes from 10.0.4.5: icmp_seq=8 ttl=63 time=0.875 ms
64 bytes from 10.0.4.5: icmp_seq=9 ttl=63 time=0.811 ms
64 bytes from 10.0.4.5: icmp_seq=10 ttl=63 time=0.752 ms
64 bytes from 10.0.4.5: icmp_seq=11 ttl=63 time=0.722 ms
64 bytes from 10.0.4.5: icmp_seq=12 ttl=63 time=0.746 ms
64 bytes from 10.0.4.5: icmp_seq=13 ttl=63 time=0.711 ms
64 bytes from 10.0.4.5: icmp_seq=14 ttl=63 time=0.750 ms
64 bytes from 10.0.4.5: icmp_seq=15 ttl=63 time=0.795 ms
64 bytes from 10.0.4.5: icmp_seq=16 ttl=63 time=0.690 ms
64 bytes from 10.0.4.5: icmp_seq=17 ttl=63 time=0.754 ms
64 bytes from 10.0.4.5: icmp_seq=18 ttl=63 time=0.824 ms
64 bytes from 10.0.4.5: icmp_seq=19 ttl=63 time=0.703 ms
64 bytes from 10.0.4.5: icmp_seq=20 ttl=63 time=0.736 ms
64 bytes from 10.0.4.5: icmp_seq=21 ttl=63 time=0.869 ms
64 bytes from 10.0.4.5: icmp_seq=22 ttl=63 time=0.985 ms
64 bytes from 10.0.4.5: icmp_seq=23 ttl=63 time=0.716 ms
Captured 226 bytes on interface 0
```

路由器性能

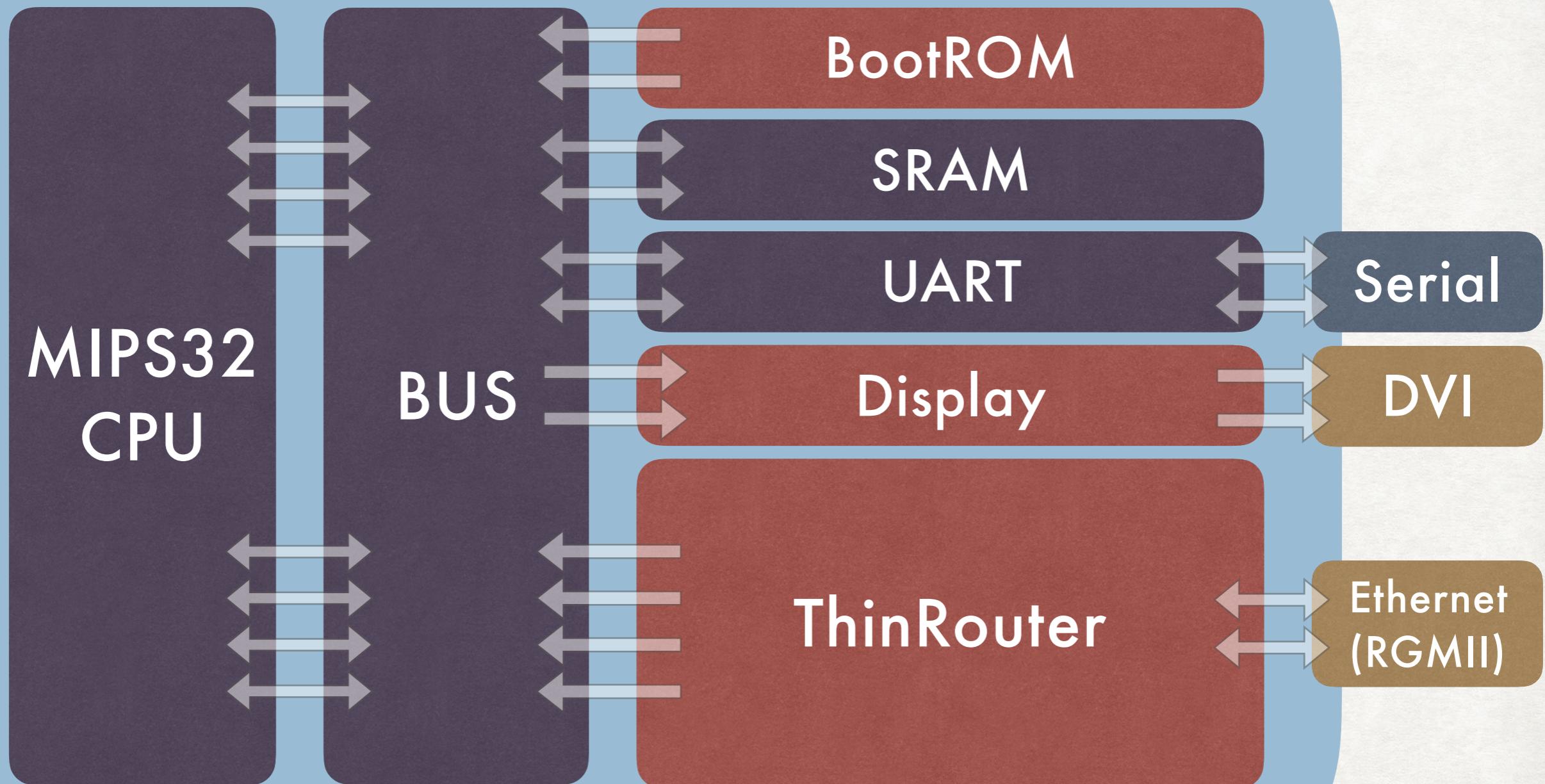
- @jiegec 提供测试
 - 酒井（9个设备）互联
 - PC [1-4] C_4^2 全联通测试
 - PC1 → PC2 单链接单工：94 Mbps (百兆网口)
 - PC1 → PC2 单链接双工：186 Mbps (百兆网口)
 - 上下左右 双链接双工：240Mbps
 - UDP 小包：130K packets/s
 - 通过 RIP 压力测试：
 - 246 条
 - 2048 条
 - 5000+ 条



路由器性能

- Wireshark 抓包测试
 - 8000 条随机路由，子网掩码随机
 - (实际只用了一半的 BRAM)
- RIP 性能
 - 8000 路由一次性插入完成
 - 广播需要加入延迟，否则发送 RIP 包会超过网口传输速率

整体架构



CPU 部分 + 联合的部分

- CPU 设计不再赘述 (毕竟是网原课)
 - MIPS32 Release 1
 - 五级流水线
 - 中断/异常的支持
 - 实现了编译器编译出来的全部常见指令，可以启 C 程序
 - 一个 make 可以直接把 C 编译好并把指令放到 BootROM 里面
- 实现了一个 BootROM (BlockRAM) , 无需烧指令到 SRAM, 可以直接自启动
- 实现了 DVI 显示
 - 支持打印字符 (26个字母 + \n + \r + Backspace) 、自动换行、滚屏、清屏
- 实现了把路由器挂到总线上
 - 路由表是一个双口 BlockRAM
 - CPU 可以在路由器工作的时候同时读取路由表的数据
- 于是我们实现了一个 ...

CPU 部分 + 联合的部分

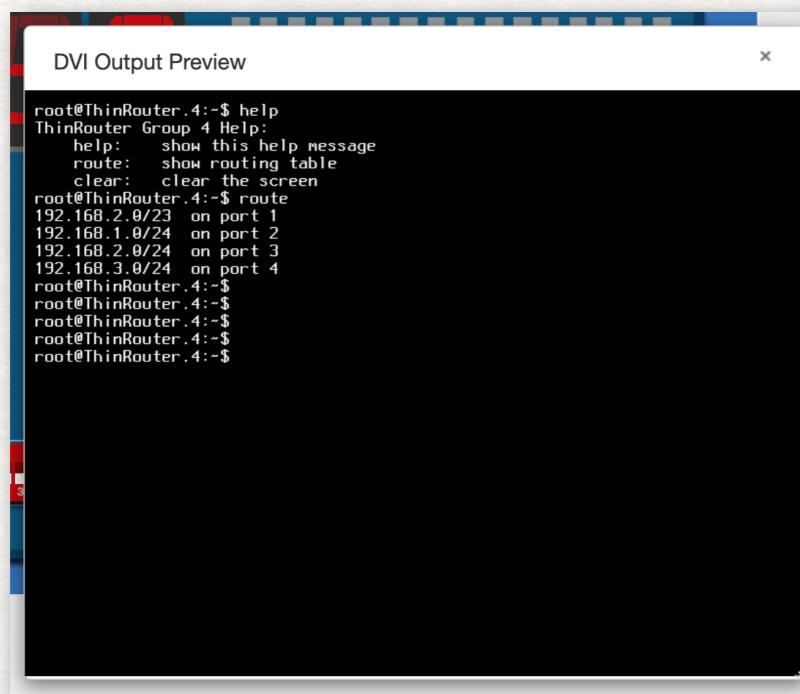
- (假的) 命令行
 - (支持显示帮助信息)
 - 支持清屏
 - 支持显示路由信息
 - 替换掉了监控程序
- 逻辑是用 C 写的
 - 在总线上给路由器数据划分一段地址
 - C 语言直接访问地址取数
 - 用 C 简直就是想干嘛干嘛
 - 之间把路由表的 Trie 树节点取出来
 - 然后用 C 解析
- 开箱（烧进去）即用
 - 这段逻辑之间放在了 BootROM 里面，作为电路存在
 - 不需要再点一下选择文件烧入 SRAM
- 没有实现 USB 键盘，如何交互？

DVI Output Preview

```
root@ThinRouter.4:~$ help
ThinRouter Group 4 Help:
    help:    show this help message
    route:   show routing table
    clear:   clear the screen
root@ThinRouter.4:~$ route
192.168.2.0/23  on port 1
192.168.1.0/24  on port 2
192.168.2.0/24  on port 3
192.168.3.0/24  on port 4
root@ThinRouter.4:~$
```

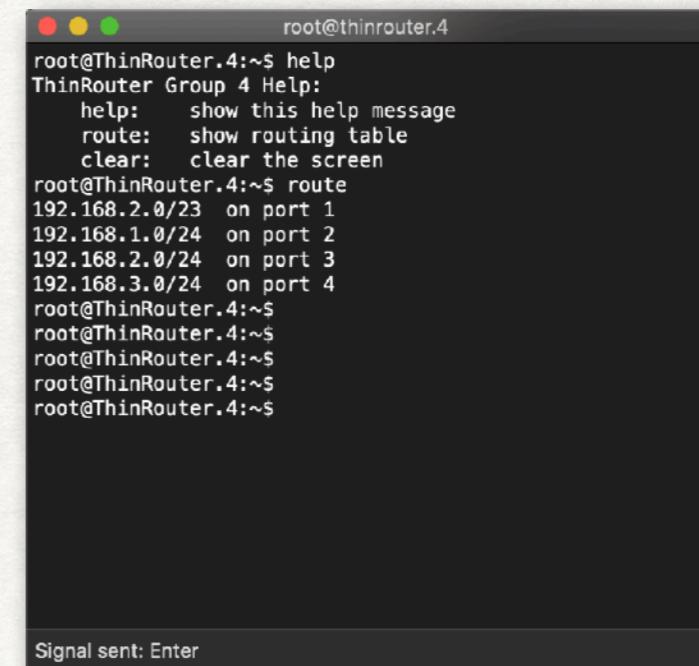
CPU 部分 + 联合的部分

- 用自己电脑的键盘，然后电脑通过
 - TCP 传给远程板子
 - UART 传给自己的板子
- 重写了 Term
 - (复习了大一小学期的 Python 和 Qt 知识)
 - 实现了一个 PyQt 的数据同传 Term



TCP
远程板子

UART
自己板子



ThinPad
(把输出信息打印屏幕 + 传回 UART)

PyQt 电脑控制
(把键盘数据传给CPU + 同步显示)

DEMO 1

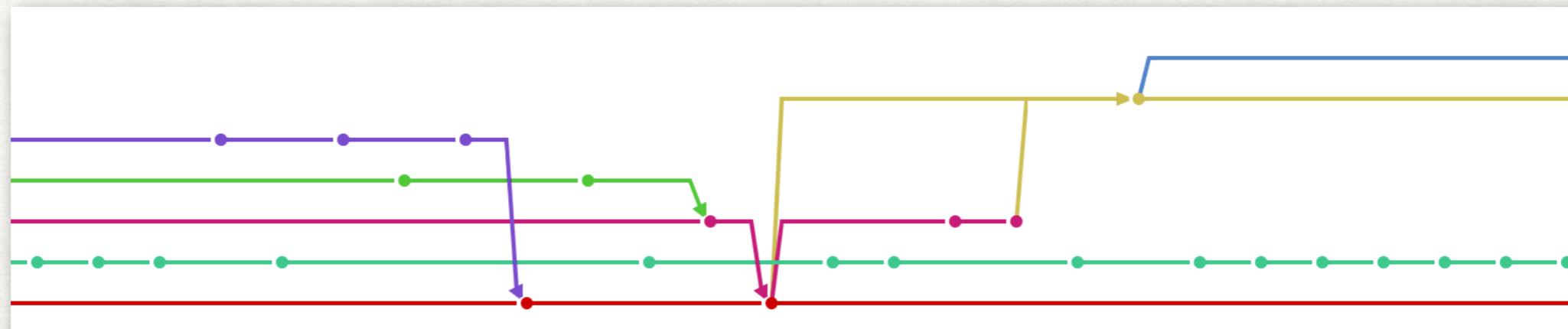
- 命令行：PC 的键盘输入通过 UART 给 CPU
- 可以通过指令查询路由表的状态
- 电脑通过一个 Qt for Python 的 Term 控制端同步显示数据



软工般的体验

- 即使计网联合，开发过程也（比较）软工
(不，我们不写文档)
- 全部代码 System Verilog
 - 学习了很多奇怪的语法和特性
 - (@Harry 说代码写得跟 C 一样)
- 开发过程比较合规范
 - 8000+ LoC (~3k Router, ~2.5k CPU, ~2.5k Tests)
 - 420 Commits, 65 Issues, 3 Milestones, 50 Merge Requests

```
15      typedef enum logic [47:0] {
16          RouterMAC1 = 48'ha8_88_08_18_88_88,
17          RouterMAC2 = 48'ha8_88_08_28_88_88,
18          RouterMAC3 = 48'ha8_88_08_38_88_88,
19          RouterMAC4 = 48'ha8_88_08_48_88_88,
20          McastMAC = 48'h01_00_5e_00_00_09,
21          BcastMAC = 48'hff_ff_ff_ff_ff_ff
22      } _mac_constants;
23
24
25      function logic [2:0] port;
26          input logic [31:0] ip;
27      begin
28          case (ip[31:8])
29              RouterIP1[31:8]: port = 1;
30              RouterIP2[31:8]: port = 2;
```



软工般的体验

- 非常完备的测试
- 7 个 Testbench
 - CPU 自启动
 - CPU 和 Term 交互测试
 - CPU 和监控程序交互测试
 - ARP 表测试
 - 路由表测试
 - 整个 RIP 测试（先模拟其他路由发 RIP，然后 ping）
 - RIP 打包测试
- 5 个 Python 测例生成器可生成任意规模测例
 - 生成 ARP 插入/删除请求
 - 生成路由表插入/删除/查询请求
 - 生成以太网帧给 RIP 测试
 - 生成监控程序的命令模拟串口给 CPU
 - 生成 Term 的命令模拟串口给 CPU

软工般的体验

- 并非为了写而写，而是很大程度上提高了 Debug 效率
- 包括用 enum 代替 hardcoded 等技巧可以直接在仿真看到每个周期在运行的指令的名字
- Testbench 可以模拟 PC 和监控程序/自制命令行交互
- 路由器的 Testbench 直接会 Assert 结果对不对

```
RECV: b5 (e)
RECV: 64 (d)
RECV: 2e (.)
SEND: Command G
SEND: Addr (G): 0x8000200c
RECV: (G) Start running
RECV: Running (G): 4f (0)
RECV: Running (G): 4b (K)
RECV: (G) End running
```

Tcl Console x Messages Log

295570 ns get 209.109.53.124
correct
1275.
295590 ns query 24.239.110.236
295730 ns get 0.0.0.0
correct
1276.
295750 ns insert 194.88.0.0/13 -> 4.114.148.154
296250 ns insert done
1277.
296250 ns insert 136.74.224.0/21 -> 38.199.197.42
296850 ns insert done
1278.



分工

- 涂轶翔：路由器部分，DVI 字符矩阵显示，部分单元测试
- 王征翊：CPU 部分，RIP Response 封装模块
- 赵成钢：CPU 部分，（伪）操作系统，部分单元测试

总结

- 路由器
 - 纯硬件 RIP，相当于把正常路由器实验的「全部逻辑 + HAL 后端」用硬件重写，可脱离 CPU 独立运行
 - 高效
 - 整体架构三级流水线
 - 路由表：路径压缩 Trie 树 / 256 路压缩 Trie 树 + 插入/删除 + 极限 ~8000 条
 - ARP 表：线性查表
- MIPS32 Release 1 CPU
 - 五级流水线 + 中断/异常的支持
 - 实现了编译器编译出来的（几乎）全部指令，可以启 C 程序
 - 实现了一个 BootROM，无需烧指令到 SRAM，可以直接自启动
- 路由器挂在总线上，实现了和 CPU 的交互
 - 实现了命令行，串口给 CPU 发送键盘输入，可以像真的命令行一样交互
 - 可以随时查看路由表中的数据
- 即使计网联合，开发过程也（比较）软工
 - System Verilog Fancy Code: 8k+ LoC (~2.5k CPU, ~3k Router, ~2.5k Tests)
 - 420 Commits, 65 Issues, 3 Milestones, 50 Merge Requests
 - 任意规模测例仿真直接 Assert: 7 Testbenches, 6 Python Testcase Generators
 - 监控程序和自制命令行都可以在仿真里面直接交互

谢谢

欢迎提问 @第四组