路由器实验报告

李晨昊 2017011466

2020-1-5

目录

我没有直接使用 `boilerplate`，最终完整的路由器代码在 `Homework/router/main.cpp` 中。

1 遇到的问题和解决方法

1.1 端序不一致

网络包中的大于一个字节的整数都按照大端序来传递，而主机有可能是大端序或者小端序的。大端序的主机可以直接读写内存中的整数；小端序的主机需要在读出整数后和写入整数前进行端序的转换，而计算过程中使用小端序。

一种很简便的进行端序转换的方法是使用 GCC 提供的 builtin 函数，即 `__builtin_bswap16` 和 `__builtin_bswap32`，它们分别可以交换 16 位整数和 32 位整数的端序。不过在代码中不应该直接使用它们，因为大端序的主机不需要（也不能）进行端序转换。虽然实际上 PC 和树莓派都是小端序的，但写代码的时候还是有考虑兼容性的必要，为此我写了一个宏，利用预编译器的 `__BYTE_ORDER__` 变量来判断当前主机是大端序还是小端序，从而决定读写内存时的行为。它是在编译时生效的，所以必须保证编译时的主机和运行时的主机端序一致，因为我们并没有做交叉编译，所以这不会造成问题。

顺便提一句，IP 头的前两个字段，即 `Version` 和 `IHL`，标准规定 `Version` 为高 4 字节，`IHL` 为低 4 字节。如果希望用位域来访问它们的话，显然也必须和这个顺序一致。这其实不算端序的问题，因为它是一个字节内部的顺序，不过这两个问题很相似。经实验将 `IHL` 写在 `Version` 前

面才能达到这个效果。我查阅了相关资料，位域中字段的具体分布是 implementation-defined，而且也没有什么通用的方法来得知编译器是怎么排布它们的，所以现在只能照这样写着，兼容性可能不是很好。

1.2 路由器代码搭建

如我上面所说，我没有直接使用 `boilerplate`。这主要是因为当时我对 `boilerplate` 中更新路由表这一部分的逻辑没有完全理解，而且为了发 RIP 包，必须能够访问整个路由表，而原来的代码中并没有这个接口。总之，最后我另写了一份，将所有代码都写在一个文件里了，对于我来说，我认为这样其实更加利于理解和维护，能够更清晰的看出信息是在哪里获取，哪里使用的。

我省去了 `protocal` 中从 RIP 包中提取 `RipPacket` 和用 `RipPacket` 构造 RIP 包这两步 (但错误检查仍然保留)，这主要是为了简单，毕竟多一层间接总是会增加需要考虑的东西。

除了添加新字段外，我还对原来的路由表进行了一点修改，即不再存储 `len`，直接存储 `mask`。由于 `len` 的大小关系和 `mask` 的大小关系相同 (`len` 越长，1 就越多，`mask` 就越大)，所以进行最长匹配的时候也不会出错。这样可以在查询的时候直接拿去和地址做与，减少一点运算量，收到 RIP 的时候也不用做额外的计算把 `mask` 转成 `len` 了。

1.3 测试环境配置

需要配置主机的 IP 地址和网关。

用 PC 运行压力测试的时候，脚本中有多处要修改网口名称，我们组因为不熟悉 BIRD 的使用和原理，在这里花了一些时间来摸索。

1.4 IP 头校验和的增量更新

我采用了 IP 头校验和的增量更新算法，在修改了包的 TTL 后不必全部重新计算校验和，只需要几步简单的更新即可。但是 RFC1141 给出的原始的算法有一定问题，关于这个已经有很多相关的讨论了，参考jiege.ch/networking/2019/05/30/ip-and-udp-checksum-incremental-update/，特判一下即可。非常良心的是，确实有对应情况的测例，这节省了后续调试的时间。