RP-PPPoE 源码阅读报告

刘　迅

2023 年 11 月 23 日

目录

1 阅读之前：基本概念

在“下载 PPPoE 源码进行阅读”之前，首先要明确一些基本概念：

什么是 PPPoE？什么是 PPPoE 源码？ (1)

PPPoE（Point-to-Point Protocol over Ethernet）是一种网络协议，它建立在课程所讲授的 PPP 协议基础之上，并更进一步，将点对点协议扩展到以太网，在技术上则表现为将 PPP 帧封装到以太网帧中。PPPoE 运行在数据链路层上，在以太网的广播环境中提供点对点的连接服务。[1]结合现有网络的结构基础与 PPP 协议所提供的身份验证等功能，PPPoE 给出了一种低成本运营的方案。[2]

既然 PPPoE 是一种网络协议，说到底这是 RFC 文档所定义的一组规则，在实体上表现为若干手册[3][4][5][6]，至于具体的代码实现则有着不同的版本。通用的 Linux PPPoE 实现中较为流行的有 Roaring Penguin PPPoE（RP-PPPoE）[7]，这一项目的主页为RP-PPPoE。

本次阅读源码作业即基于 RP-PPPoE 这一具体代码实现以开展，使用代码分析工具 Understand[8]。

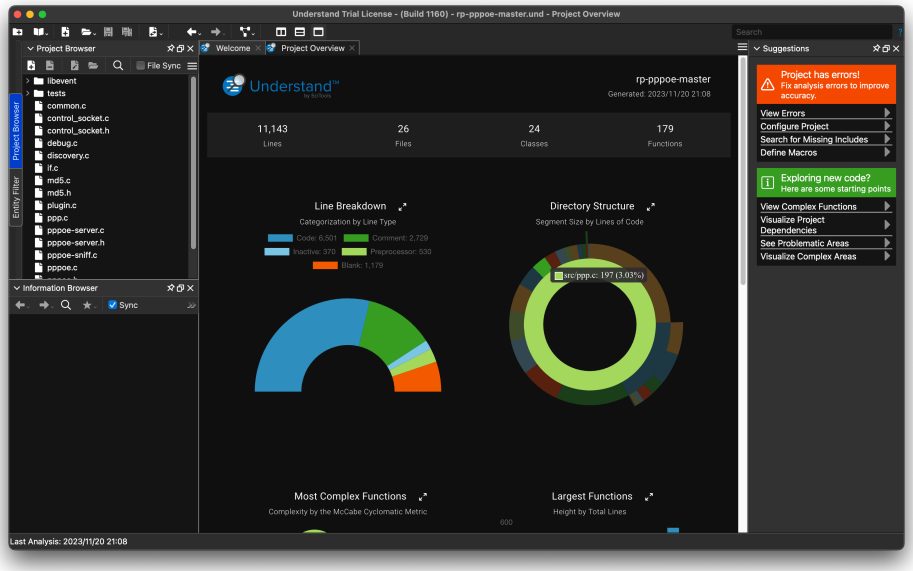


图 1: 代码分析工具 Understand

[1]MBA 智库百科 | PPPoE 协议：https://wiki.mbalib.com/wiki/PPPoE 协议

[2]华为 | 什么是 PPPoE? PPPoE 解决了哪些问题? ：https://info.support.huawei.com/info-finder/encyclopedia/zh/PPPoE.html

[3]RFC 2516 - A Method for Transmitting PPP Over Ethernet (PPPoE)：https://datatracker.ietf.org/doc/html/rfc2516

[4]RFC 3817 - Layer 2 Tunneling Protocol (L2TP) Active Discovery Relay for PPP over Ethernet (PPPoE)：https://datatracker.ietf.org/doc/html/rfc3817

[5]RFC 4638 - Accommodating a Maximum Transit Unit/Maximum Receive Unit (MTU/MRU) Greater Than 1492 in the Point-to-Point Protocol over Ethernet (PPPoE)：https://datatracker.ietf.org/doc/html/rfc4638

[6]RFC 4938 - PPP Over Ethernet (PPPoE) Extensions for Credit Flow and Link Metrics：https://datatracker.ietf.org/doc/html/rfc4938

[7]GitHub | RP-PPPoE - a PPPoE client, relay and server for Linux：https://github.com/dfskoll/rp-pppoe

[8]Understand 是一款商用的代码分析工具，不过在官网上可以免费申请 7 天试用。这对于本次 PPPoE 代码阅读作业来说已经足够了。官网：https://scitools.com。

2 PPP 模块程序结构

RP-PPPoE 项目共有 26 个文件，程序文件一共 11143 行，代码 6501 行，注释 2729 行。其中 ppp.c 文件有 197 行代码，占总代码量的 3.03%。

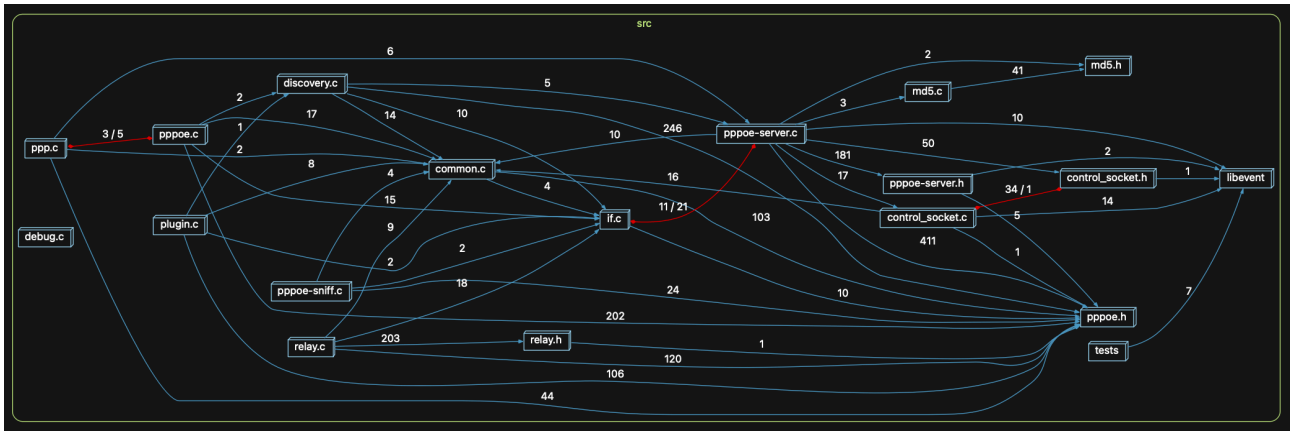


图 2: RP-PPPoE 项目各文件依赖情况

图中的蓝线表示单向依赖关系，线段上的数字指出了依赖数量；红线则表示双向的依赖关系。分析 ppp.c 在项目文件中的调用与被调用情况得到：

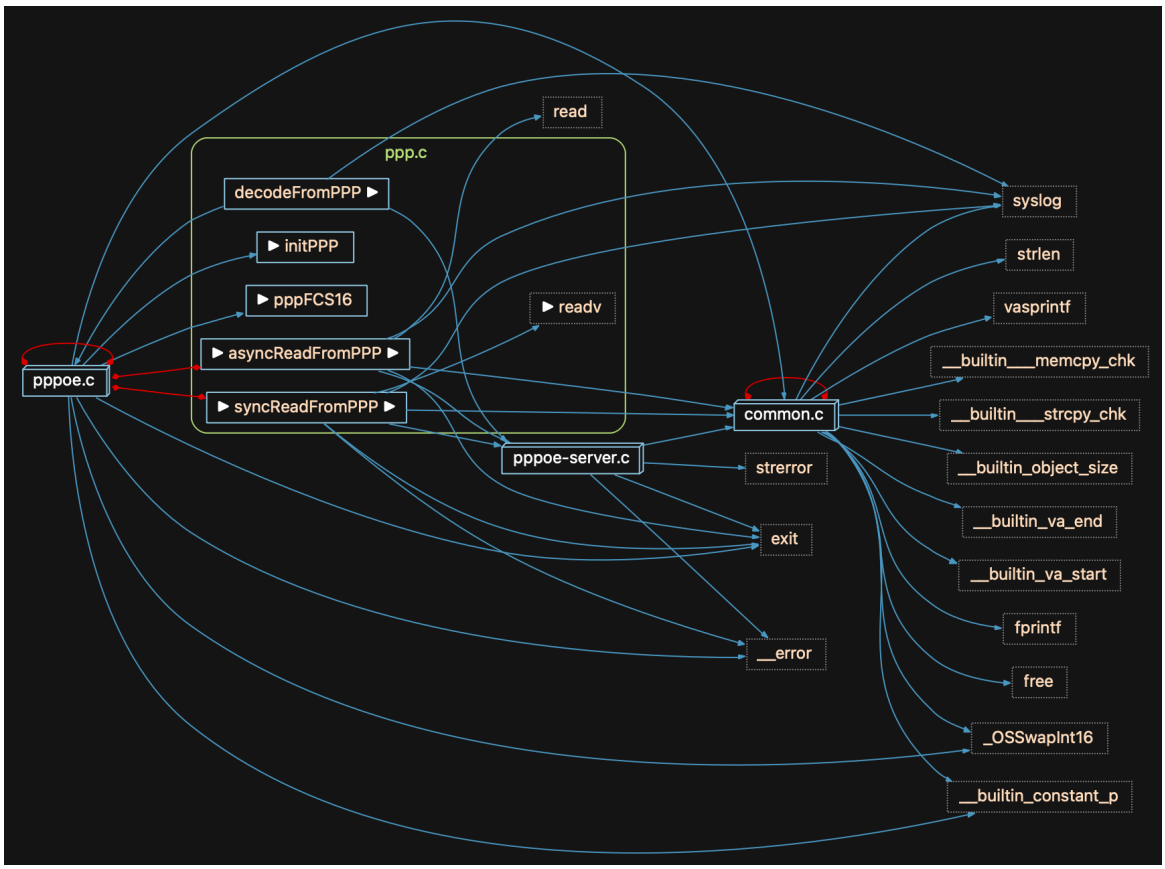


图 3: PPP 模块函数调用情况

绿框内是 ppp.c 文件内的函数；文件外部与 ppp.c 有关的函数展现在绿框之外。蓝色箭头表示函数调用关系，红色箭头则表明存在相互调用的情况。可以看出，ppp.c 文件与 pppoe.c 的客户端与服务端都存在紧密联系；并且这三个基于 ppp 的代码都依赖 common.c 提供的部分方法。

具体到 ppp.c 文件中各个函数与变量的声明定义、头文件的引用情况，Understand 也给出了图示分析。

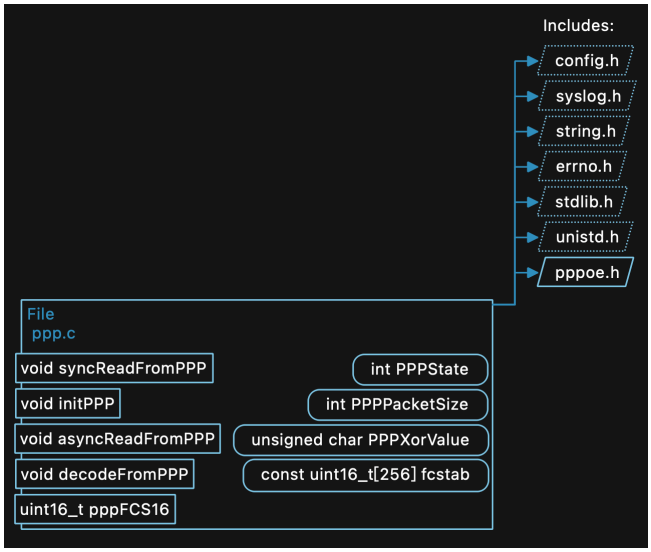


图 4: PPP 模块函数声明情况

从图4可以看出，ppp.c 定义了 PPPState,PPPPacketSize,PPPXorValue,fcstab 四个变量，syncReadFromPPP,initPPP,asyncReadFromPPP,decodeFromPPP 四个过程，以及 pppFCS16 函数。它们之间的数据流关系在图5中展现。

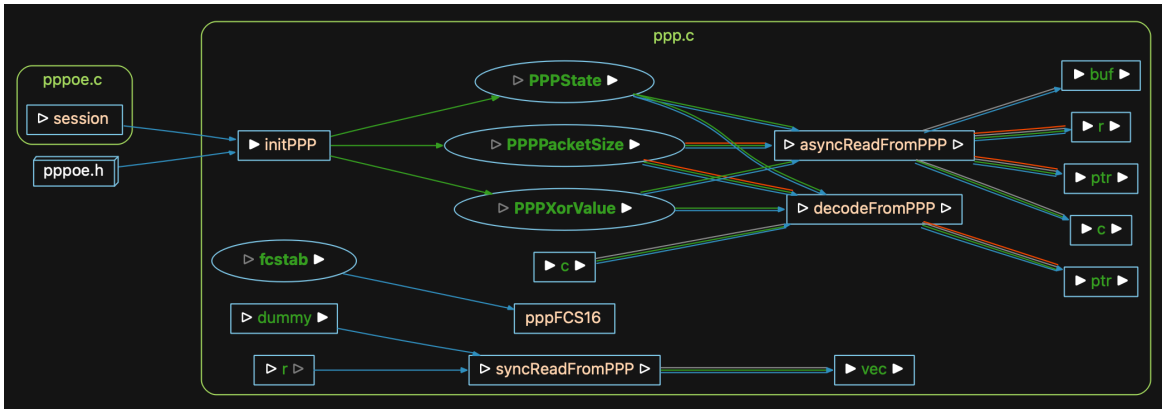


图 5: PPP 模块数据流情况

其中米色代表函数或过程，绿色代表变量。左侧出现的 pppoe.c 数据流关系则表明 ppp.c 作为功能化模块在 pppoe.c 中发挥作用。

四个过程的功能分别为

- `syncReadFromPPP()` 从一个同步的 PPP 设备读取数据，构建 PPPoE 数据报的时候丢弃帧地址字节，并在读到 EOF（即文件结束符）的时候执行 `sendPADT()` 发送 PADT 数据报，表示 PPP 连接终止。

- `initPPP()` 初始化 PPP 状态为 `STATE_WAITFOR_FRAME_ADDR`,也即等待 PPP 帧;初始化 `PPPPacketSize` 为零，目前收到的 PPP 包为空；初始化 `PPPXorValue` 为零，这是转义特殊字符所用到的异或“掩码”。
- `asyncReadFromPPP()` 从一个异步的 PPP 设备读取数据，与 `syncReadFromPPP()` 相对应，它在构建 PPPoE 数据报的时候逐字节处理。通信的时候将会使用状态机,在（`STATE_WAITFOR_FRAME_ADDR`、`STATE_DROP_PROTO`、`STATE_BUILDING_PACKET`）之间转换，以此解析 PPP 帧、处理转义序列和帧边界。
- `decodeFromPPP()` 与 `asyncReadFromPPP()` 功能类似，同样从一个异步的 PPP 设备读取数据，区别在于给定读取完成的 PPP 报文，而不需要在过程中读入。这一点在图5中可以得到印证，`asyncReadFromPPP()` 与 `decodeFromPPP()` 两个过程数据流都用到 PPP 状态机的变量，而在处理报文内容时所用的变量不同。

主要共性之处在于 PPP 状态机的构建与转移条件，这一问题可以通过 GPT-4 辅助绘图得到图6。

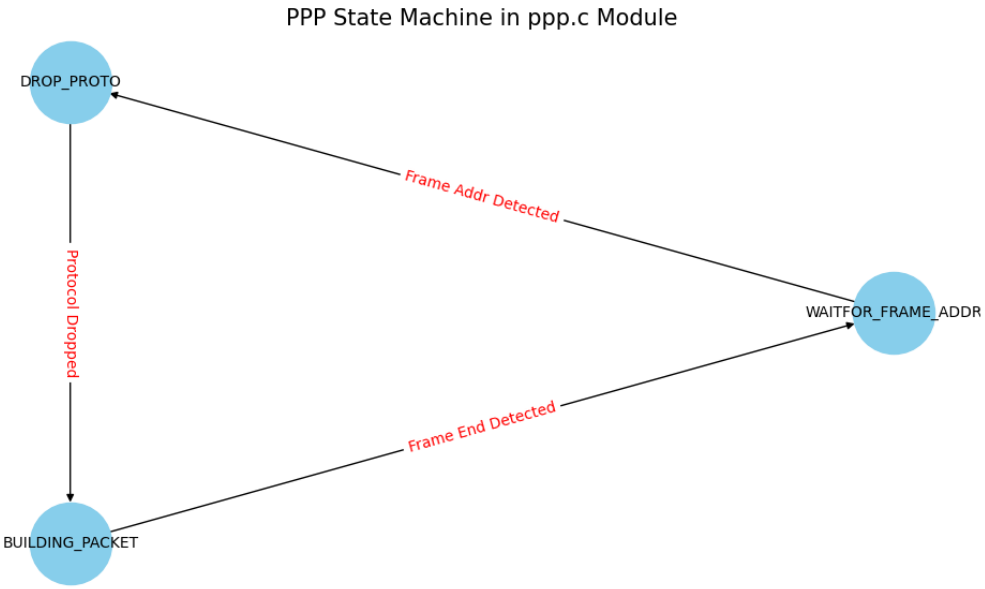


图 6: PPP 模块状态机转换图

异步读取时，状态机 `PPPState` 最初处于 `STATE_WAITFOR_FRAME_ADDR` 的等待状态；检测到 PPP 帧的起始字节时，进入 `STATE_DROP_PROTO` 处理状态，在这里读取并丢弃协议字段，进入读取状态；在 `STATE_BUILDING_PACKET` 读取状态中，程序开始构建数据报的有效载荷内容，并且处理转义字符和帧边界等问题，若遇到帧结束标记，则重新进入 `STATE_WAITFOR_FRAME_ADDR` 的等待状态，开启下一次 PPP 处理过程的循环。

3 PAP 和 CHAP 的工作流程

与上一节不同，二、三节使用 Distrotech 项目的 RP-PPPoE 项目代码，具体原因将在附录 A 中说明。

PAP 与 CHAP 的全称分别为 Password Authentication Protocol 和 Challenge-Handshake Authentication Protocol。

结合网络资料，这两个协议的大致流程分别为

PAP（Password Authentication Protocol）

- 发送认证请求：客户端发送一个认证请求给服务器，内容为用户的用户名和密码。
- 服务器验证：服务器接收到认证请求后，会根据数据库验证收到的用户名和密码。
- 认证：如果认证成功，服务器会发送一个认证成功的响应报文给客户端，允许建立 PPP 连接；如果认证失败，发送认证失败的响应，断开连接。

CHAP（Challenge-Handshake Authentication Protocol）

- 服务器发出挑战：服务器向客户端发送一个“挑战”消息，包含一个随机生成的值。
- 客户端响应挑战：客户端使用一个预共享的秘密（下图中的 secrets 文件），结合收到的挑战值，计算出响应值，并将其发送回服务器。
- 服务器验证：服务器使用相同的方法计算期望的响应值，并将其与客户端发送的响应值进行比较。
- 认证：如果两者匹配，则认证成功，服务器发送一个成功的响应给客户端；如果不匹配则认证失败，发送相应的失败响应。

在项目代码中，它们出现在 pppo-esetup.in 的配置文件内，可以看出 secrets 是预先硬编码在项目中的。

```
$ECHO "Adjusting /etc/ppp/pap-secrets and /etc/ppp/chap-secrets"
if [ -r /etc/ppp/pap-secrets ] ; then
    $ECHO "  (But first backing it up to /etc/ppp/pap-secrets-bak)"
    copy /etc/ppp/pap-secrets /etc/ppp/pap-secrets-bak
else
    cp /dev/null /etc/ppp/pap-secrets-bak
fi
if [ -r /etc/ppp/chap-secrets ] ; then
    $ECHO "  (But first backing it up to /etc/ppp/chap-secrets-bak)"
    copy /etc/ppp/chap-secrets /etc/ppp/chap-secrets-bak
else
    cp /dev/null /etc/ppp/chap-secrets-bak
fi
```

图 7: PAP 与 CHAP 配置，来源于项目中/scripts/pppoe-setup.in

4 PAP 和 CHAP 的底层调用

PAP 与 CHAP 两者都是 PPP 协议的一部分，在 TCP/IP 协议栈中主要用于网络层的身份验证，而都运行在数据链路层的层次上。

PAP 的所依赖的底层功能相对较少，它的特点有：

- 标准输入输出，用户名和密码以明文形式发送，没有加密。
- 单向认证，即只有客户端被验证，服务器不被验证。

CHAP 在上一节有所分析，依赖的底层功能相对较多，特点有：

- MD5 或其他哈希算法，确保了即使数据报被拦截，密码信息也不会暴露。
- 挑战-响应机制，服务器发送一个随机生成的挑战给客户端，客户端必须正确响应以验证其身份，增加安全性。

5 附录 A：谁是 RP-PPPoE 真身?

在查找 RP-PPPoE 项目的时候，至少查找到了三个来源

- GitHub | dfskoll rp-pppoe，35 Stars，12 Forks。
- GitHub | Distrotech rp-pppoe，20 Stars，15 Forks。
- www.roaringpenguin.com，无法打开。

本着对 Stars 的认可度，最初选择的是第一个 GitHub 项目进行了本次报告第一板块中的分析部分。然而在分析 PAP 和 CHAP 的工作情况时，很惊讶地发现这个项目中竟然缺少 PAP 和 CHAP 的相关代码，只有在“如何连接”的注释文档中简单带过一句配置文件。

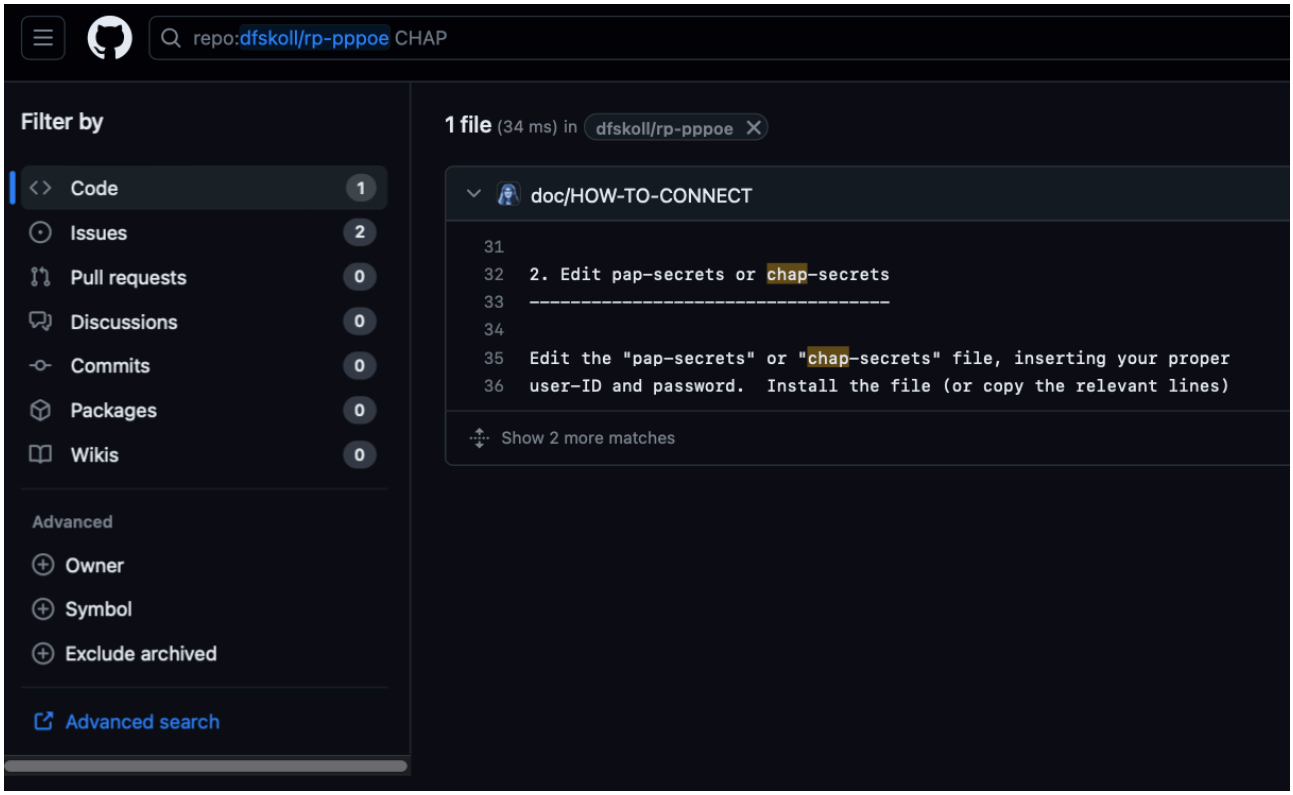


图 8: dfskoll 项目中查询 PAP 关键词，竟没有相关代码

这就非常令人疑惑。

为此，二三部分的阅读报告转而使用 Distrotech 的 GitHub 项目代码。

至于为什么在 dfskoll 项目中会出现这类问题；RP-PPPoE 项目的最新主分支究竟在何方……它们仍然是个开放问题。如果你对这些问题有答案，欢迎联系笔者。

6 附录 B：状态机绘制代码

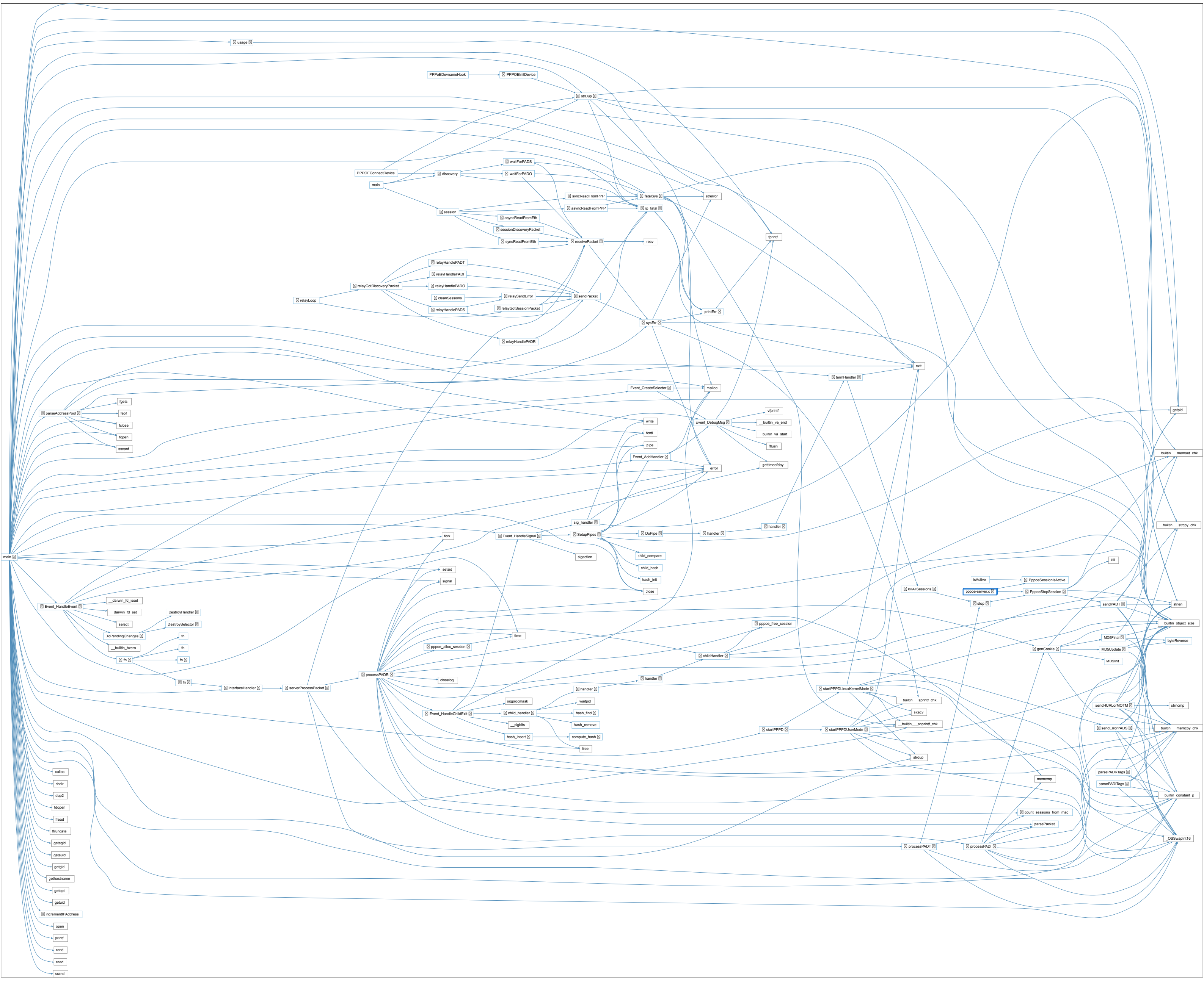
此处致谢 GPT-4 生成的用以绘制 PPP 状态机的 python 代码。

Code 1: 状态机绘制

```python
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import networkx as nx

def draw_state_machine_graph():
    # Create a directed graph
    G = nx.DiGraph()

    # Add nodes representing the states
    states = ["WAITFOR_FRAME_ADDR", "DROP_PROTO", "BUILDING_PACKET"]
    G.add_nodes_from(states)

    # Add edges representing the transitions
    G.add_edge("WAITFOR_FRAME_ADDR", "DROP_PROTO", label="Frame Addr Detected")
    G.add_edge("DROP_PROTO", "BUILDING_PACKET", label="Protocol Dropped")
    G.add_edge("BUILDING_PACKET", "WAITFOR_FRAME_ADDR", label="Frame End Detected")

    # Graph layout
    pos = nx.circular_layout(G)

    # Draw the graph
    fig, ax = plt.subplots(figsize=(8, 8))
    nx.draw(G, pos, with_labels=True, node_size=3000, node_color="skyblue",
    ↪  font_size=10, ax=ax)
    edge_labels = nx.get_edge_attributes(G, 'label')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='red',
    ↪  ax=ax)

    # Title
    plt.title("PPP State Machine in ppp.c Module", size=15)

    return fig

fig = draw_state_machine_graph()
plt.show()
```

7 附录 C：pppoe-server.c 函数调用关系图（含 pppd）

通过对 pppoe-server.c 的分析，可以得到 pppd（PPP Daemon）这一进程的调用关系，从而分析 PAP 与 CHAP 认证协议等动作。


main
usage
PPPoEDevnameHook
PPPOEInitDevice
strDup
PPPOEConnectDevice
discovery
waitForPADS
waitForPADO
session
syncReadFromPPP
asyncReadFromPPP
fatalSys
rp_fatal
strerror
asyncReadFromEth
sessionDiscoveryPacket
syncReadFromEth
receivePacket
recv
fprintf
relayHandlePADT
relayHandlePADI
relayGotDiscoveryPacket
relayHandlePADO
relayLoop
cleanSessions
relaySendError
sendPacket
relayHandlePADS
relayGotSessionPacket
printErr
sysErr
relayHandlePADR
exit
termHandler
Event_CreateSelector
malloc
parseAddressPool
fgets
feof
fclose
fopen
sscanf
Event_DebugMsg
vfprintf
__builtin_va_end
__builtin_va_start
fflush
gettimeofday
write
fcntl
pipe
Event_AddHandler
__error
getpid
__builtin___memset_chk
sig_handler
SetupPipes
DoPipe
handler
__builtin___strcpy_chk
fork
Event_HandleSignal
sigaction
child_compare
child_hash
hash_init
close
setsid
signal
Event_HandleEvent
__darwin_fd_isset
__darwin_fd_set
select
DoPendingChanges
DestroyHandler
DestroySelector
__builtin_bzero
fn
time
pppoe_alloc_session
processPADR
closelog
InterfaceHandler
serverProcessPacket
sigprocmask
Event_HandleChildExit
child_handler
waitpid
hash_find
__sigbits
hash_remove
hash_insert
compute_hash
free
pppoe_free_session
childHandler
isActive
PppoeSessionIsActive
pppoe-server.c
PppoeStopSession
kill
killAllSessions
stop
sendPADT
strlen
__builtin_object_size
genCookie
MD5Final
MD5Update
MD5Init
byteReverse
startPPPDLinuxKernelMode
__builtin___sprintf_chk
execv
startPPPD
startPPPDUserMode
__builtin___snprintf_chk
sendHURLorMOTM
strncmp
sendErrorPADS
__builtin___memcpy_chk
strdup
parsePADRTags
parsePADITags
memcmp
__builtin_constant_p
count_sessions_from_mac
parsePacket
processPADT
processPADI
_OSSwapInt16
calloc
chdir
dup2
fdopen
fread
ftruncate
getegid
geteuid
getgid
gethostname
getopt
getuid
incrementIPAddress
open
printf
rand
read
srand