

# 2024 Jan/Feb

- [FreeBSD](#) [RACK](#)  [TCP](#)
- [FreeBSD 14](#) [TCP](#)
- [if\\_ovpn](#)  [OpenVPN](#)

# FreeBSD? RACK ? ?? TCP ??

*by Randall Stewart and Michael Tüxen*

 : [Tuxen.pdf](#)

2017年，FreeBSD的TCP实现进行了全面的优化。TCP实现进行了全面的优化，包括对SYN-Cookies的支持，以及对IP地址和TCP端口的优化。此外，还引入了SYN-Cache功能，以提高连接建立的速度。这些优化使得TCP连接建立更加快速和可靠，从而提升了网络性能。

[illegible]

# TCP RACK ?? ?? ??

RACK [ ] FreeBSD CURRENT [ ] FreeBSD 14.0 [ ] [ ] [ ] [ ] . [ ] [ ] [ ]  
[ ] [ ] FreeBSD [ ] [ ] [ ] .

FreeBSD 14.0                      .

```
option TCPHPTS
makeoptions WITH_EXTRA_TCP_STACKS=1
```

[illegible]

```
tcp_rack_load="YES"
```

```

[ ] [ ] [ ] [ ] /boot/loader.conf [ ] [ ] [ ] [ ] [ ] [ ] .

```

FreeBSD CURRENT 内核支持 TCP RACK 和 HPTS 选项，但需要手动加载。tcp\_hpts.ko 和 tcp\_rack.ko 内核模块，使用 kldload 命令加载。安装后，在 /boot/loader.conf 文件中添加以下配置：

```
tcp_hpts_load="YES"
tcp_rack_load="YES"
```

配置完成后，重启系统。FreeBSD CURRENT 内核支持 TCP RACK 和 HPTS 选项，但需要手动加载。

```
option TCPHPTS
option TCP_RACK
```

TCP 内核模块在 FreeBSD 14.0 版本中默认启用，但需要配置。在 FreeBSD 14.0 版本中，TCP 内核模块默认启用，但需要配置。在 FreeBSD 14.0 版本中，TCP 内核模块默认启用，但需要配置。

```
sysctl net.inet.tcp.functions_available
```

FreeBSD 14.0 版本中，TCP RACK 选项默认启用。在 FreeBSD 14.0 版本中，TCP RACK 选项默认启用。在 FreeBSD 14.0 版本中，TCP RACK 选项默认启用。

FreeBSD 14.1 版本中，TCP RACK 选项默认启用。在 FreeBSD 14.1 版本中，TCP RACK 选项默认启用。在 FreeBSD 14.1 版本中，TCP RACK 选项默认启用。

TCP RACK 选项在 FreeBSD 14.0 版本中默认启用。在 FreeBSD 14.0 版本中，TCP RACK 选项默认启用。在 FreeBSD 14.0 版本中，TCP RACK 选项默认启用。

sysctl-variable net.inet.tcp.functions\_default 选项在 FreeBSD 14.0 版本中默认启用。在 FreeBSD 14.0 版本中，sysctl-variable net.inet.tcp.functions\_default 选项在 FreeBSD 14.0 版本中默认启用。

```
sysctl net.inet.tcp.functions_default=rack
```

/etc/sysctl.conf 文件中添加以下配置：

```
net.inet.tcp.functions_default=rack
```

listener 选项在 FreeBSD 14.0 版本中默认启用。在 FreeBSD 14.0 版本中，listener 选项在 FreeBSD 14.0 版本中默认启用。在 FreeBSD 14.0 版本中，listener 选项在 FreeBSD 14.0 版本中默认启用。

tcpsso(8) 选项在 FreeBSD 14.0 版本中默认启用。在 FreeBSD 14.0 版本中，tcpsso(8) 选项在 FreeBSD 14.0 版本中默认启用。在 FreeBSD 14.0 版本中，tcpsso(8) 选项在 FreeBSD 14.0 版本中默认启用。

```

    if (tcp_function_blk) {
        tcp_function_set(tcp_function_blk, IPPROTO_TCP);
    } else {
        tcp_function_set(NULL, IPPROTO_TCP);
    }
}

```

```
struct tcp_function_set tfs;

strncpy(tfs.function_set_name, "rack", TCP_FUNCTION_NAME_LEN_MAX);
tfs.pcbcnt = 0;
setsockopt(fd, IPPROTO_TCP, TCP_FUNCTION_BLK, &tfs, sizeof(tfs));
```

```
TCP RACK [ ] [ ] TCP [ ] [ ] [ ] [ ] [ ] . [ ]  
[ ] [ ] net.inet.tcp.rack [ ] IPPROTO_TCP [ ] [ ] [ ] sysctl-variables [ ] [ ] [ ]  
[ ] .
```

# TCP RACK ??? ??

TCP RACK

# RACK/TLP

[illegible]

TCP RACK [ ] [ ][ ][ ] [ ][ ] [ ][ ][ ][ ] [ ][ ] [ ][ ][ ] RACK[ ] TLP[ ] [ ][ ] [ ][ ] [ ][ ] [ ][ ][ ][ ] .  
[ ][ ] [ ][ ][ ][ ] [ ][ ] [ ][ ][ ][ ] [ ][ ][ ] [ ][ ][ ][ ] [ ][ ][ ][ ] .

## Proportional Rate Reduction (PRR)

TCP RACK, RFC 6937  
IETF . PRR . RFC 5681

TCP 的 拥塞控制 算法 在 网络 拥塞 时 会 降低 发送 速率 。 这 导致 了 网络 延迟 的 增加 。 为 了解 决 这 个 问题 ， 网络 工程师 设计 了 许多 拥塞 控制 算法 ， 其中 最 著名 的 是 TCP SACK 和 TCP RACK 。 TCP SACK 是 一种 选择 性 确认 的 拥塞 控制 算法 ， 它 可以 提高 网络 的 吞吐量 和 降低 延迟 。 TCP RACK 是 一种 快速 恢复 的 拥塞 控制 算法 ， 它 可以 在 网络 拥塞 时 快速 恢复 发送 速率 ， 从而 降低 延迟 。

# RACK Rapid Recovery (RRR)

RACK Rapid Recovery(RRR) 是 一种 快速 恢复 的 拥塞 控制 算法 。 它 在 TCP RACK 的 基础上 进行 了 改进 ， 主要 改进 了 快速 恢复 的 逻辑 。 在 TCP RACK 中 ， 当 接收 窗口 被 填满 时 ， 发送 方 会 立即 停止 发送 数据 ， 直到 接收 方 发送 确认 为止 。 这 导致 了 网络 延迟 的 增加 。 RRR 通过 引入 快速 恢复 的 逻辑 ， 可以在 接收 窗口 被 填满 时 快速 恢复 发送 速率 ， 从而 降低 延迟 。 RRR 的 快速 恢复 逻辑 是 基于 接收 窗口 的 大小 和 接收 方 的 接收 速率 来 进行 判断 的 。 如果 接收 窗口 的 大小 大于 1ms ， 则 认为 网络 拥塞 已经 结束 ， 发送 方 可以 立即 恢复 发送 速率 。 如果 接收 窗口 的 大小 小于 1ms ， 则 认为 网络 拥塞 仍然 存在 ， 发送 方 需要 继续 降低 发送 速率 。 RRR 的 快速 恢复 逻辑 可以 显著 降低 网络 延迟 ， 提高 网络 的 吞吐量 。

在 网络 拥塞 时 ， 网络 工程师 需要 采取 一些 措施 来 降低 延迟 ， 提高 网络 的 吞吐量 。 这 包括 调整 网络 参数 ， 优化 网络 架构 等 。 RRR 是 一种 有效 的 拥塞 控制 算法 ， 可以 帮助 网络 工程师 降低 延迟 ， 提高 网络 的 吞吐量 。 RRR 的 快速 恢复 逻辑 是 基于 接收 窗口 的 大小 和 接收 方 的 接收 速率 来 进行 判断 的 。 如果 接收 窗口 的 大小 大于 12Mbps ， 则 认为 网络 拥塞 已经 结束 ， 发送 方 可以 立即 恢复 发送 速率 。 如果 接收 窗口 的 大小 小于 12Mbps ， 则 认为 网络 拥塞 仍然 存在 ， 发送 方 需要 继续 降低 发送 速率 。 RRR 的 快速 恢复 逻辑 可以 显著 降低 网络 延迟 ， 提高 网络 的 吞吐量 。

# SACK Attack Detection

SACK 是 一种 选择 性 确认 的 拥塞 控制 算法 ， 它 可以 提高 网络 的 吞吐量 和 降低 延迟 。 但是 ， SACK 也 存在 一些 安全 问题 ， 例如 SACK 攻击 。 SACK 攻击 是 一种 恶意 攻击 ， 攻击 者 可以 通过 伪造 SACK 报文 来 欺骗 发送 方 ， 从而 导致 发送 方 降低 发送 速率 ， 降低 网络 的 吞吐量 。 为 了解 决 这 个 问题 ， 网络 工程师 设计 了 许多 SACK 攻击 检测 算法 ， 其中 最 著名 的 是 TCP RACK 和 TCP SAD 。 TCP RACK 是 一种 快速 恢复 的 拥塞 控制 算法 ， 它 可以 在 网络 拥塞 时 快速 恢复 发送 速率 ， 从而 降低 延迟 。 TCP SAD 是 一种 SACK 攻击 检测 算法 ， 它 可以 检测 到 SACK 攻击 ， 从而 防止 发送 方 降低 发送 速率 。

TCP RACK 是 一种 快速 恢复 的 拥塞 控制 算法 ， 它 可以 在 网络 拥塞 时 快速 恢复 发送 速率 ， 从而 降低 延迟 。 TCP SAD 是 一种 SACK 攻击 检测 算法 ， 它 可以 检测 到 SACK 攻击 ， 从而 防止 发送 方 降低 发送 速率 。 TCP SAD 的 检测 逻辑 是 基于 SACK 报文 的 大小 和 接收 方 的 接收 速率 来 进行 判断 的 。 如果 SACK 报文 的 大小 大于 1500 ， 则 认为 SACK 攻击 已经 发生 ， 发送 方 需要 立即 停止 发送 数据 。 如果 SACK 报文 的 大小 小于 1500 ， 则 认为 SACK 攻击 没有 发生 ， 发送 方 可以 继续 发送 数据 。

```

option TCP_SAD_DETECTION

```

SACK 是 一种 选择 性 确认 的 拥塞 控制 算法 ， 它 可以 提高 网络 的 吞吐量 和 降低 延迟 。 但是 ， SACK 也 存在 一些 安全 问题 ， 例如 SACK 攻击 。 SACK 攻击 是 一种 恶意 攻击 ， 攻击 者 可以 通过 伪造 SACK 报文 来 欺骗 发送 方 ， 从而 导致 发送 方 降低 发送 速率 ， 降低 网络 的 吞吐量 。 为 了解 决 这 个 问题 ， 网络 工程师 设计 了 许多 SACK 攻击 检测 算法 ， 其中 最 著名 的 是 TCP RACK 和 TCP SAD 。 TCP RACK 是 一种 快速 恢复 的 拥塞 控制 算法 ， 它 可以 在 网络 拥塞 时 快速 恢复 发送 速率 ， 从而 降低 延迟 。 TCP SAD 是 一种 SACK 攻击 检测 算法 ， 它 可以 检测 到 SACK 攻击 ， 从而 防止 发送 方 降低 发送 速率 。

# Burst Mitigation

Linux 4.19 版本开始支持 TCP RACK 拥塞控制算法。该算法旨在通过更有效地利用接收缓冲空间来减少延迟，并提高吞吐量。它通过更积极地探测丢失的段来减少超时时间，从而加快了慢启动过程。此外，它还引入了新的拥塞控制参数，如 `tcp_rack_ssthresh` 和 `tcp_rack_retransmit_timeout`，以进一步优化性能。

# Support for TCP Blackbox Logging (BBLog)

TCP RACK 拥塞控制算法在 Linux 4.19 版本中得到了支持。它通过更有效地利用接收缓冲空间来减少延迟，并提高吞吐量。该算法通过更积极地探测丢失的段来减少超时时间，从而加快了慢启动过程。此外，它还引入了新的拥塞控制参数，如 `tcp_rack_ssthresh` 和 `tcp_rack_retransmit_timeout`，以进一步优化性能。

# Large Receive Offload (LRO) Integration for Burst Mitigation

TCP 接收卸载 (LRO) 是一种用于减少网络接口卡 (NIC) 处理大量接收到的数据包时的 CPU 负载的技术。它通过将多个连续的数据段合并成一个更大的段来减少接收缓冲空间的使用，从而提高吞吐量。LRO 的集成旨在通过减少接收缓冲空间的占用来缓解拥塞，从而提高网络性能。

Linux 4.19 版本开始支持 TCP RACK 拥塞控制算法。该算法旨在通过更有效地利用接收缓冲空间来减少延迟，并提高吞吐量。它通过更积极地探测丢失的段来减少超时时间，从而加快了慢启动过程。此外，它还引入了新的拥塞控制参数，如 `tcp_rack_ssthresh` 和 `tcp_rack_retransmit_timeout`，以进一步优化性能。

Linux 4.19 版本开始支持 TCP RACK 拥塞控制算法。该算法旨在通过更有效地利用接收缓冲空间来减少延迟，并提高吞吐量。它通过更积极地探测丢失的段来减少超时时间，从而加快了慢启动过程。此外，它还引入了新的拥塞控制参数，如 `tcp_rack_ssthresh` 和 `tcp_rack_retransmit_timeout`，以进一步优化性能。

# A Host of Alternate Features

Linux 4.19 版本开始支持 TCP RACK 拥塞控制算法。该算法旨在通过更有效地利用接收缓冲空间来减少延迟，并提高吞吐量。它通过更积极地探测丢失的段来减少超时时间，从而加快了慢启动过程。此外，它还引入了新的拥塞控制参数，如 `tcp_rack_ssthresh` 和 `tcp_rack_retransmit_timeout`，以进一步优化性能。

# ???? TCP RACK ??? ????? ??

1. 在 Linux 中，使用 `tcpdump` 命令可以捕获网络数据包。例如，要捕获所有经过 eth0 接口的 TCP 数据包，可以使用以下命令：  
`tcpdump -i eth0 -s 0 -w tcp.pcap`

2. 在 Windows 中，使用 Wireshark 网络分析工具可以捕获网络数据包。例如，要捕获所有经过网卡的网络数据包，可以使用以下命令：  
`Wireshark -x tcp.pcap`

3. 在 FreeBSD 中，使用 `tcpdump` 命令可以捕获网络数据包。例如，要捕获所有经过 eth0 接口的 TCP 数据包，可以使用以下命令：  
`tcpdump -i eth0 -s 0 -w tcp.pcap`

4. 在 macOS 中，使用 `tcpdump` 命令可以捕获网络数据包。例如，要捕获所有经过 eth0 接口的 TCP 数据包，可以使用以下命令：  
`tcpdump -i eth0 -s 0 -w tcp.pcap`

[illegible]

TCP RACK [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]  
[ ] . [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]  
[ ] [ ] , [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] QoE [ ] [ ]  
[ ] .

?? ? ??

TCP RACK [11] FreeBSD [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90] [91] [92] [93] [94] [95] [96] [97] [98] [99] [100] [101] [102] [103] [104] [105] [106] [107] [108] [109] [110] [111] [112] [113] [114] [115] [116] [117] [118] [119] [120] [121] [122] [123] [124] [125] [126] [127] [128] [129] [130] [131] [132] [133] [134] [135] [136] [137] [138] [139] [140] [141] [142] [143] [144] [145] [146] [147] [148] [149] [150] [151] [152] [153] [154] [155] [156] [157] [158] [159] [160] [161] [162] [163] [164] [165] [166] [167] [168] [169] [170] [171] [172] [173] [174] [175] [176] [177] [178] [179] [180] [181] [182] [183] [184] [185] [186] [187] [188] [189] [190] [191] [192] [193] [194] [195] [196] [197] [198] [199] [200] [201] [202] [203] [204] [205] [206] [207] [208] [209] [210] [211] [212] [213] [214] [215] [216] [217] [218] [219] [220] [221] [222] [223] [224] [225] [226] [227] [228] [229] [230] [231] [232] [233] [234] [235] [236] [237] [238] [239] [240] [241] [242] [243] [244] [245] [246] [247] [248] [249] [250] [251] [252] [253] [254] [255] [256] [257] [258] [259] [260] [261] [262] [263] [264] [265] [266] [267] [268] [269] [270] [271] [272] [273] [274] [275] [276] [277] [278] [279] [280] [281] [282] [283] [284] [285] [286] [287] [288] [289] [290] [291] [292] [293] [294] [295] [296] [297] [298] [299] [300] [301] [302] [303] [304] [305] [306] [307] [308] [309] [310] [311] [312] [313] [314] [315] [316] [317] [318] [319] [320] [321] [322] [323] [324] [325] [326] [327] [328] [329] [330] [331] [332] [333] [334] [335] [336] [337] [338] [339] [340] [341] [342] [343] [344] [345] [346] [347] [348] [349] [350] [351] [352] [353] [354] [355] [356] [357] [358] [359] [360] [361] [362] [363] [364] [365] [366] [367] [368] [369] [370] [371] [372] [373] [374] [375] [376] [377] [378] [379] [380] [381] [382] [383] [384] [385] [386] [387] [388] [389] [390] [391] [392] [393] [394] [395] [396] [397] [398] [399] [400] [401] [402] [403] [404] [405] [406] [407] [408] [409] [410] [411] [412] [413] [414] [415] [416] [417] [418] [419] [420] [421] [422] [423] [424] [425] [426] [427] [428] [429] [430] [431] [432] [433] [434] [435] [436] [437] [438] [439] [440] [441] [442] [443] [444] [445] [446] [447] [448] [449] [450] [451] [452] [453] [454] [455] [456] [457] [458] [459] [460] [461] [462] [463] [464] [465] [466] [467] [468] [469] [470] [471] [472] [473] [474] [475] [476] [477] [478] [479] [480] [481] [482] [483] [484] [485] [486] [487] [488] [489] [490] [491] [492] [493] [494] [495] [496] [497] [498] [499] [500] [501] [502] [503] [504] [505] [506] [507] [508] [509] [510] [511] [512] [513] [514] [515] [516] [517] [518] [519] [520] [521] [522] [523] [524] [525] [526] [527] [528] [529] [530] [531] [532] [533] [534] [535] [536] [537] [538] [539] [540] [541] [542] [543] [544] [545] [546] [547] [548] [549] [550] [551] [552] [553] [554] [555] [556] [557] [558] [559] [560] [561] [562] [563] [564] [565] [566] [567] [568] [569] [570] [571] [572] [573] [574] [575] [576] [577] [578] [579] [580] [581] [582] [583] [584] [585] [586] [587] [588] [589] [590] [591] [592] [593] [594] [595] [596] [597] [598] [599] [600] [601] [602] [603] [604] [605] [606] [607] [608] [609] [610] [611] [612] [613] [614] [615] [616] [617] [618] [619] [620] [621] [622] [623] [624] [625] [626] [627] [628] [629] [630] [631] [632] [633] [634] [635] [636] [637] [638] [639] [640] [641] [642] [643] [644] [645] [646] [647] [648] [649] [650] [651] [652] [653] [654] [655] [656] [657] [658] [659] [660] [661] [662] [663] [664] [665] [666] [667] [668] [669] [670] [671] [672] [673] [674] [675] [676] [677] [678] [679] [680] [681] [682] [683] [684] [685] [686] [687] [688] [689] [690] [691] [692] [693] [694] [695] [696] [697] [698] [699] [700] [701] [702] [703] [704] [705] [706] [707] [708] [709] [710] [711] [712] [713] [714] [715] [716] [717] [718] [719] [720] [721] [722] [723] [724] [725] [726] [727] [728] [729] [730] [731] [732] [733] [734] [735] [736] [737] [738] [739] [740] [741] [742] [743] [744] [745] [746] [747] [748] [749] [750] [751] [752] [753] [754] [755] [756] [757] [758] [759] [760] [761] [762] [763] [764] [765] [766] [767] [768] [769] [770] [771] [772] [773] [774] [775] [776] [777] [778] [779] [780] [781] [782] [783] [784] [785] [786] [787] [788] [789] [790] [791] [792] [793] [794] [795] [796] [797] [798] [799] [800] [801] [802] [803] [804] [805] [806] [807] [808] [809] [810] [811] [812] [813] [814] [815] [816] [817] [818] [819] [820] [821] [822] [823] [824] [825] [826] [827] [828] [829] [830] [831] [832] [833] [834] [835] [836] [837] [838] [839] [840] [841] [842] [843] [844] [845] [846

TCP RACK 0000 0000 0000 000000 0000 00000 00000000 . 000 00 000  
000000 00000 00 00000 . 000 0000 000 00000 , 000 000  
00000 TCP RACK 000 0000 000 000000 . 000 00 0000 000 000  
net@freebsd.org 00 0 000 00000 000 000 . 000 0 00 0000 00 , TCP RACK  
000 00 FreeBSD 00 000 0 00 0000 .

我们 (rrs@freebsd.org) 40 个 网络 组织 , 10 个 FreeBSD 网络 组织  
 提供 . 的 TCP SCTP 的 网络 组织 的 网络 组织 的 网络 组织  
 提供 . 的 网络 组织 的 网络 组织 TCP 的 网络 组织 的 网络 组织 QoS 的 网络 组织  
 的 网络 组织 的 网络 组织 .

tuexen (tuexen@freebsd.org) 2009  
 FreeBSD 2.0.0, 2.0.1, 2.0.2, 2.0.3, 2.0.4, 2.0.5, 2.0.6, 2.0.7, 2.0.8, 2.0.9, 2.1.0, 2.1.1, 2.1.2, 2.1.3, 2.1.4, 2.1.5, 2.1.6, 2.1.7, 2.1.8, 2.1.9, 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.3.0, 2.3.1, 2.3.2, 2.3.3, 2.3.4, 2.3.5, 2.3.6, 2.3.7, 2.3.8, 2.3.9, 2.4.0, 2.4.1, 2.4.2, 2.4.3, 2.4.4, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.5.0, 2.5.1, 2.5.2, 2.5.3, 2.5.4, 2.5.5, 2.5.6, 2.5.7, 2.5.8, 2.5.9, 2.6.0, 2.6.1, 2.6.2, 2.6.3, 2.6.4, 2.6.5, 2.6.6, 2.6.7, 2.6.8, 2.6.9, 2.7.0, 2.7.1, 2.7.2, 2.7.3, 2.7.4, 2.7.5, 2.7.6, 2.7.7, 2.7.8, 2.7.9, 2.8.0, 2.8.1, 2.8.2, 2.8.3, 2.8.4, 2.8.5, 2.8.6, 2.8.7, 2.8.8, 2.8.9, 2.9.0, 2.9.1, 2.9.2, 2.9.3, 2.9.4, 2.9.5, 2.9.6, 2.9.7, 2.9.8, 2.9.9, 3.0.0, 3.0.1, 3.0.2, 3.0.3, 3.0.4, 3.0.5, 3.0.6, 3.0.7, 3.0.8, 3.0.9, 3.1.0, 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5, 3.1.6, 3.1.7, 3.1.8, 3.1.9, 3.2.0, 3.2.1, 3.2.2, 3.2.3, 3.2.4, 3.2.5, 3.2.6, 3.2.7, 3.2.8, 3.2.9, 3.3.0, 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6, 3.3.7, 3.3.8, 3.3.9, 3.4.0, 3.4.1, 3.4.2, 3.4.3, 3.4.4, 3.4.5, 3.4.6, 3.4.7, 3.4.8, 3.4.9, 3.5.0, 3.5.1, 3.5.2, 3.5.3, 3.5.4, 3.5.5, 3.5.6, 3.5.7, 3.5.8, 3.5.9, 3.6.0, 3.6.1, 3.6.2, 3.6.3, 3.6.4, 3.6.5, 3.6.6, 3.6.7, 3.6.8, 3.6.9, 3.7.0, 3.7.1, 3.7.2, 3.7.3, 3.7.4, 3.7.5, 3.7.6, 3.7.7, 3.7.8, 3.7.9, 3.8.0, 3.8.1, 3.8.2, 3.8.3, 3.8.4, 3.8.5, 3.8.6, 3.8.7, 3.8.8, 3.8.9, 3.9.0, 3.9.1, 3.9.2, 3.9.3, 3.9.4, 3.9.5, 3.9.6, 3.9.7, 3.9.8, 3.9.9, 4.0.0, 4.0.1, 4.0.2, 4.0.3, 4.0.4, 4.0.5, 4.0.6, 4.0.7, 4.0.8, 4.0.9, 4.1.0, 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.1.6, 4.1.7, 4.1.8, 4.1.9, 4.2.0, 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.5, 4.2.6, 4.2.7, 4.2.8, 4.2.9, 4.3.0, 4.3.1, 4.3.2, 4.3.3, 4.3.4, 4.3.5, 4.3.6, 4.3.7, 4.3.8, 4.3.9, 4.4.0, 4.4.1, 4.4.2, 4.4.3, 4.4.4, 4.4.5, 4.4.6, 4.4.7, 4.4.8, 4.4.9, 4.5.0, 4.5.1, 4.5.2, 4.5.3, 4.5.4, 4.5.5, 4.5.6, 4.5.7, 4.5.8, 4.5.9, 4.6.0, 4.6.1, 4.6.2, 4.6.3, 4.6.4, 4.6.5, 4.6.6, 4.6.7, 4.6.8, 4.6.9, 4.7.0, 4.7.1, 4.7.2, 4.7.3, 4.7.4, 4.7.5, 4.7.6, 4.7.7, 4.7.8, 4.7.9, 4.8.0, 4.8.1, 4.8.2, 4.8.3, 4.8.4, 4.8.5, 4.8.6, 4.8.7, 4.8.8, 4.8.9, 4.9.0, 4.9.1, 4.9.2, 4.9.3, 4.9.4, 4.9.5, 4.9.6, 4.9.7, 4.9.8, 4.9.9, 5.0.0, 5.0.1, 5.0.2, 5.0.3, 5.0.4, 5.0.5, 5.0.6, 5.0.7, 5.0.8, 5.0.9, 5.1.0, 5.1.1, 5.1.2, 5.1.3, 5.1.4, 5.1.5, 5.1.6, 5.1.7, 5.1.8, 5.1.9, 5.2.0, 5.2.1, 5.2.2, 5.2.3, 5.2.4, 5.2.5, 5.2.6, 5.2.7, 5.2.8, 5.2.9, 5.3.0, 5.3.1, 5.3.2, 5.3.3, 5.3.4, 5.3.5, 5.3.6, 5.3.7, 5.3.8, 5.3.9, 5.4.0, 5.4.1, 5.4.2, 5.4.3, 5.4.4, 5.4.5, 5.4.6, 5.4.7, 5.4.8, 5.4.9, 5.5.0, 5.5.1, 5.5.2, 5.5.3, 5.5.4, 5.5.5, 5.5.6, 5.5.7, 5.5.8, 5.5.9, 5.6.0, 5.6.1, 5.6.2, 5.6.3, 5.6.4, 5.6.5, 5.6.6, 5.6.7, 5.6.8, 5.6.9, 5.7.0, 5.7.1, 5.7.2, 5.7.3, 5.7.4, 5.7.5, 5.7.6, 5.7.7, 5.7.8, 5.7.9, 5.8.0, 5.8.1, 5.8.2, 5.8.3, 5.8.4, 5.8.5, 5.8.6, 5.8.7, 5.8.8, 5.8.9, 5.9.0, 5.9.1, 5.9.2, 5.9.3, 5.9.4, 5.9.5, 5.9.6, 5.9.7, 5.9.8, 5.9.9, 6.0.0, 6.0.1, 6.0.2, 6.0.3, 6.0.4, 6.0.5, 6.0.6, 6.0.7, 6.0.8, 6.0.9, 6.1.0, 6.1.1, 6.1.2, 6.1.3, 6.1.4, 6.1.5, 6.1.6, 6.1.7, 6.1.8, 6.1.9, 6.2.0, 6.2.1, 6.2.2, 6.2.3, 6.2.4, 6.2.5, 6.2.6, 6.2.7, 6.2.8, 6.2.9, 6.3.0, 6.3.1, 6.3.2, 6.3.3, 6.3.4, 6.3.5, 6.3.6, 6.3.7, 6.3.8, 6.3.9, 6.4.0, 6.4.1, 6.4.2, 6.4.3, 6.4.4, 6.4.5, 6.4.6, 6.4.7, 6.4.8, 6.4.9, 6.5.0, 6.5.1, 6.5.2, 6.5.3, 6.5.4, 6.5.5, 6.5.6, 6.5.7, 6.5.8, 6.5.9, 6.6.0, 6.6.1, 6.6.2, 6.6.3, 6.6.4, 6.6.5, 6.6.6, 6.6.7, 6.6.8, 6.6.9, 6.7.0, 6.7.1, 6.7.2, 6.7.3, 6.7.4, 6.7.5, 6.7.6, 6.7.7, 6.7.8, 6.7.9, 6.8.0, 6.8.1, 6.8.2, 6.8.3, 6.8.4, 6.8.5, 6.8.6, 6.8.7, 6.8.8, 6.8.9, 6.9.0, 6.9.1, 6.9.2, 6.9.3, 6.9.4, 6.9.5, 6.9.6, 6.9.7, 6.9.8, 6.9.9, 7.0.0, 7.0.1, 7.0.2, 7.0.3, 7.0.4, 7.0.5, 7.0.6, 7.0.7, 7.0.8, 7.0.9, 7.1.0, 7.1.1, 7.1.2, 7.1.3, 7.1.4, 7.1.5, 7.1.6, 7.1.7, 7.1.8, 7.1.9, 7.2.0, 7.2.1, 7.2.2, 7.2.3, 7.2.4, 7.2.5, 7.2.6, 7.2.7, 7.2.8, 7.2.9, 7.3.0, 7.3.1, 7.3.2, 7.3.3, 7.3.4, 7.3.5, 7.3.6, 7.3.7, 7.3.8, 7.3.9, 7.4.0, 7.4.1, 7.4.2, 7.4.3, 7.4.4, 7.4.5, 7.4.6, 7.4.7, 7.4.8, 7.4.9, 7.5.0, 7.5.1, 7.5.2, 7.5.3, 7.5.4, 7.5.5, 7.5.6, 7.5.7, 7.5.8, 7.5.9, 7.6.0, 7.6.1, 7.6.2, 7.6.3, 7.6.4, 7.6.5, 7.6.6, 7.6.7, 7.6.8, 7.6.9, 7.7.0, 7.7.1, 7.7.2, 7.7.3, 7.7.4, 7.7.5, 7.7.6, 7.7.7, 7.7.8, 7.7.9, 7.8.0, 7.

# FreeBSD 14? TCP ?? ???? ?

📄 : [scheffenegger.pdf](#)

📄 📄 📄 FreeBSD 📄 📄 , 📄 TCP 📄 📄 📄 📄 📄 📄 3 📄 📄 📄 . 📄 📄 📄 📄 📄 📄 , FreeBSD 📄 📄 TCP 📄 📄 📄 📄 📄 📄 📄 , 📄 RACK 📄 📄 📄 📄 📄 📄 . 📄 📄 📄 📄 📄 (📄 📄 ) 📄 📄 📄 BSD4.4 📄 📄 📄 . 📄 , 2018 📄 📄 📄 📄 ("RACK 📄 " - 📄 ACKnowledgement 📄 📄 📄 ) 📄 📄 📄 📄 📄 📄 📄 . 📄 📄 , RACK 📄 📄 📄 📄 📄 📄 . 📄 , 📄 📄 📄 📄 📄 📄 📄 📄 📄 📄 📄 . 📄 📄 📄 📄 📄 📄 📄 (CPU 📄 📄 📄 📄 📄 📄 📄 ) 📄 📄 📄 📄 📄 📄 . 📄 📄 📄 IO 📄 📄 📄 📄 📄 📄 📄 📄 📄 📄 . (RACK 📄 📄 📄 📄 📄 📄 , 📄 📄 Michael Tuexen 📄 Randall Stewart 📄 📄 📄 ).

📄 . 📄 📄 📄 📄 📄 📄 📄 📄 📄 📄 .

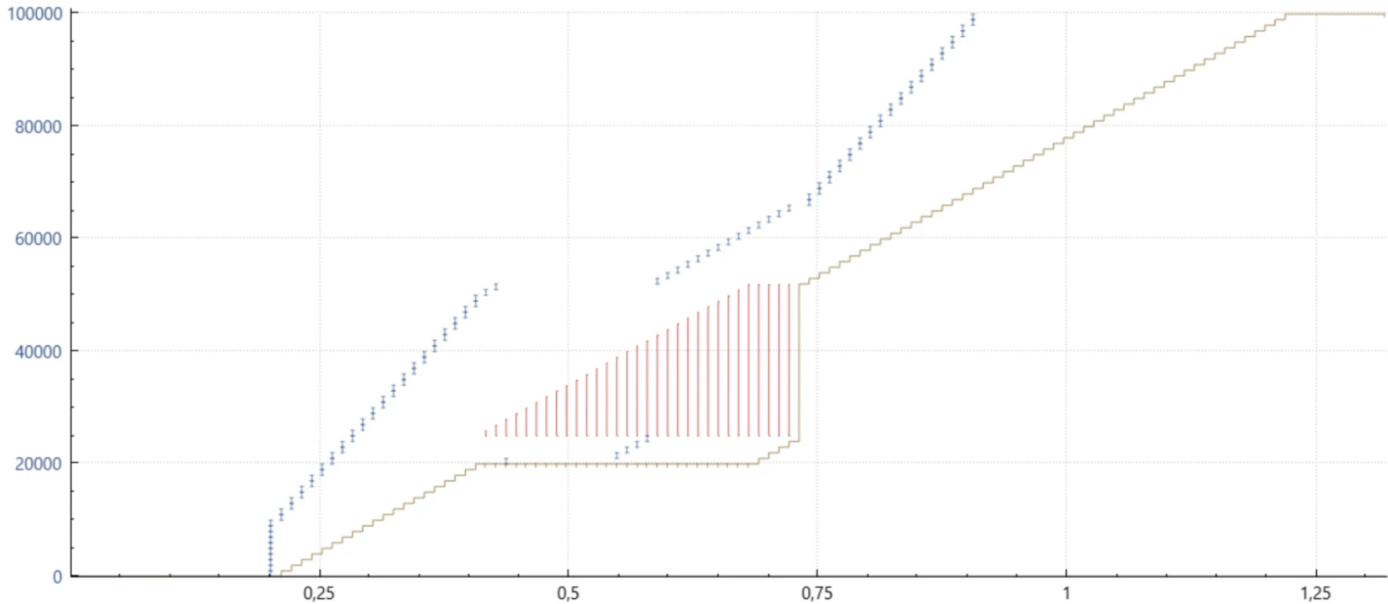
📄 📄 , FreeBSD 13.0 📄 📄 📄 📄 📄 📄 📄 sys/netinet 📄 📄 1033 📄 📄 . 📄 📄 📄 📄 📄 📄 📄 📄 :  
📄 :

## Proportional Rate Reduction (?? ??? ??)

📄 📄 📄 📄 📄 PRR - 📄 📄 📄 (RFC6937) 📄 . PRR 📄 📄 📄 📄 📄 📄 📄 SACK 📄 📄 📄 📄 . 📄 SACK 📄 📄 📄 📄 📄 📄 📄 📄 (📄 : NewReno 📄 📄 📄 50%, Cubic 📄 📄 70% ), 📄 📄 📄 📄 📄 📄 ACK 📄 📄 (NewReno) 📄 📄 30% 📄 📄 📄 📄 📄 📄 . 📄 📄 📄 📄 📄 ACK 📄 📄 📄 📄 📄 📄 📄 📄 📄 . 📄 📄 📄 📄 📄 . 📄 📄 📄 📄 (📄 📄 📄 📄 📄 📄 . 📄 📄 📄 📄 ). 📄 .

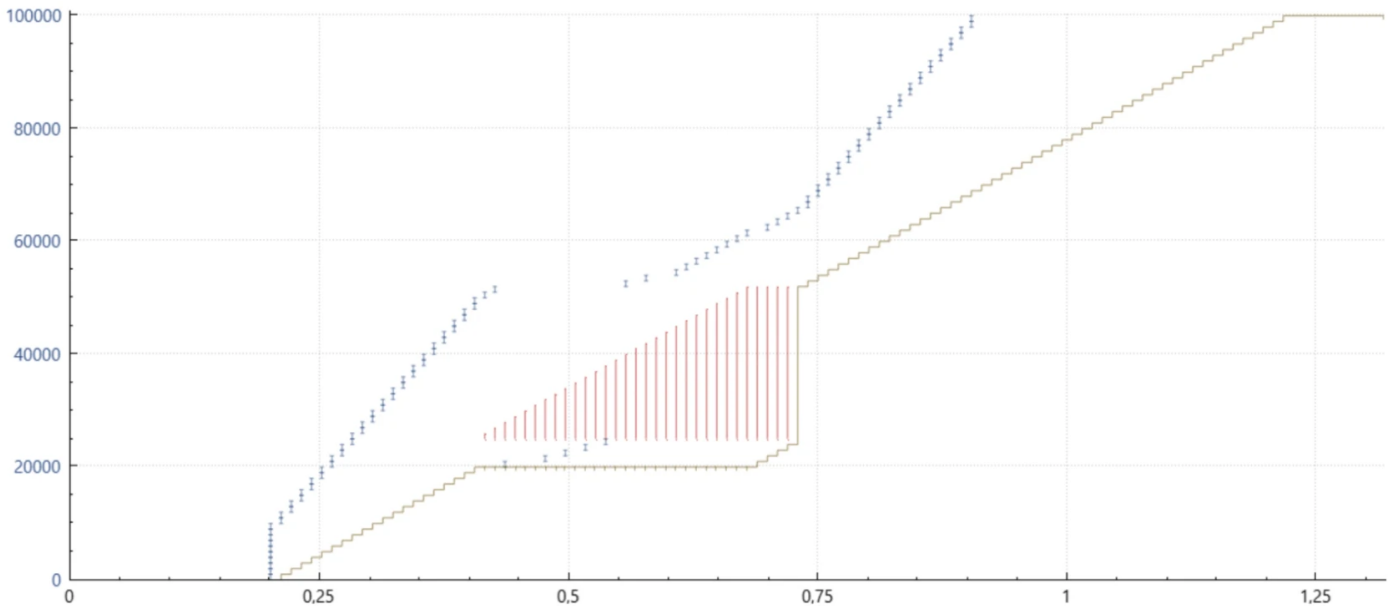
📄 📄 📄 📄 📄 📄 📄 📄 📄 ACK 📄 📄 📄 📄 📄 📄 PRR 📄 📄 📄 ACK 📄 📄 📄 📄 📄





Cubic with SACK, but no PRR

The graph shows the behavior of a Cubic congestion control algorithm with SACK but without PRR. The blue line represents the sender's buffer usage, which increases linearly until it reaches a ceiling of 50,000 at t=0.75. The red line represents the receiver's buffer usage, which also increases linearly until it reaches a ceiling of 50,000 at t=0.75. The yellow line represents the total data volume, which increases linearly until it reaches a ceiling of 100,000 at t=1.25. The graph illustrates that without PRR, the sender's buffer usage is limited by the receiver's buffer usage, which is limited by the total data volume.



Cubic with SACK (6675) and PRR

The graph shows the behavior of a Cubic congestion control algorithm with SACK (6675) and PRR. The blue line represents the sender's buffer usage, which increases linearly until it reaches a ceiling of 50,000 at t=0.75. The red line represents the receiver's buffer usage, which also increases linearly until it reaches a ceiling of 50,000 at t=0.75. The yellow line represents the total data volume, which increases linearly until it reaches a ceiling of 100,000 at t=1.25. The graph illustrates that with PRR, the sender's buffer usage is limited by the receiver's buffer usage, which is limited by the total data volume.

The graph shows the behavior of a Cubic congestion control algorithm with SACK (6675) and PRR. The blue line represents the sender's buffer usage, which increases linearly until it reaches a ceiling of 50,000 at t=0.75. The red line represents the receiver's buffer usage, which also increases linearly until it reaches a ceiling of 50,000 at t=0.75. The yellow line represents the total data volume, which increases linearly until it reaches a ceiling of 100,000 at t=1.25. The graph illustrates that with PRR, the sender's buffer usage is limited by the receiver's buffer usage, which is limited by the total data volume.

ACK 0.7 。

PRR 。

PRR ( ) SACK , SACK

ACK 。

ECN PRR

# SACK Handling

RFC6675 SACK

RFC6675

RACK

SACK

RACK

TCP ( : RPC)

FreeBSD 14 net.inet.tcp.do\_lrd

FreeBSD 15 net.inet.tcp.sack.lrd

IP





在 RFC2018(即 Nagle 算法)中，发送方在收到接收方的 ACK 之前，不会发送新的数据段。这导致在拥塞网络中，数据包在队列中等待的时间（RTO）可能会很长。为了解决这个问题，TCP 引入了 SACK（Selective Acknowledgment）机制。SACK 允许接收方选择性地确认已经收到的数据段，而不是必须按顺序确认。这有助于减少 RTO 的时间，提高网络的吞吐量。

---

在 2020 年 4 月，FreeBSD 项目宣布将支持 ECN（Explicit Congestion Notification）功能。ECN 是一种网络拥塞控制机制，通过在 IP 数据包的头中设置 ECN 标志来通知发送方网络中的拥塞情况。这有助于发送方调整其发送速率，避免网络拥塞。

# if\_ovpn ?? OpenVPN

PDF : [provost.pdf](#)

By Kristof Provost

OpenVPN 1 DCO 2.

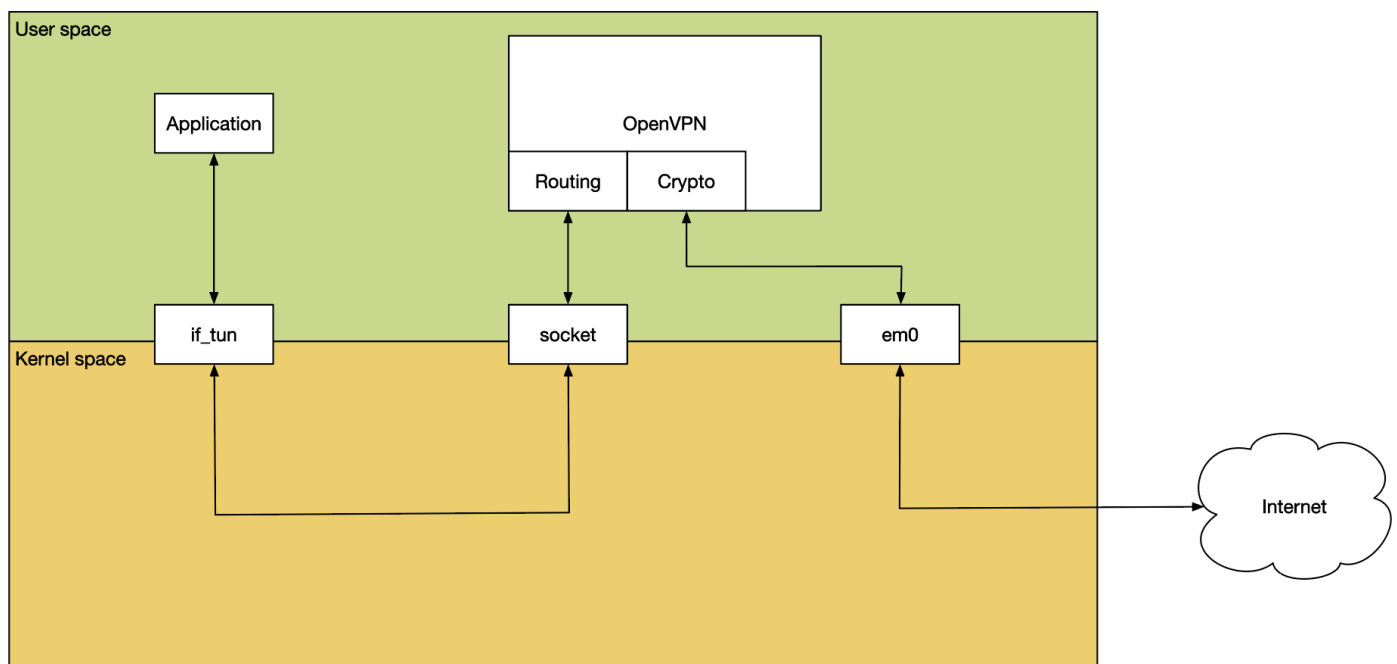
OpenVPN 2001 5 13 . . ( : FreeBSD, OpenBSD, Dragonfly, AIX, ...) (macOS, Linux, Windows) . . /

20 . ,

## ???

OpenVPN . . 3. .

if\_tun . . . .



OpenVPN DMA NIC .

我们使用 UDP 和 TCP 来传输数据。

我们使用 OpenVPN 来建立隧道，它使用 if\_tun 来创建隧道接口。我们使用 if\_tun 来创建隧道接口。

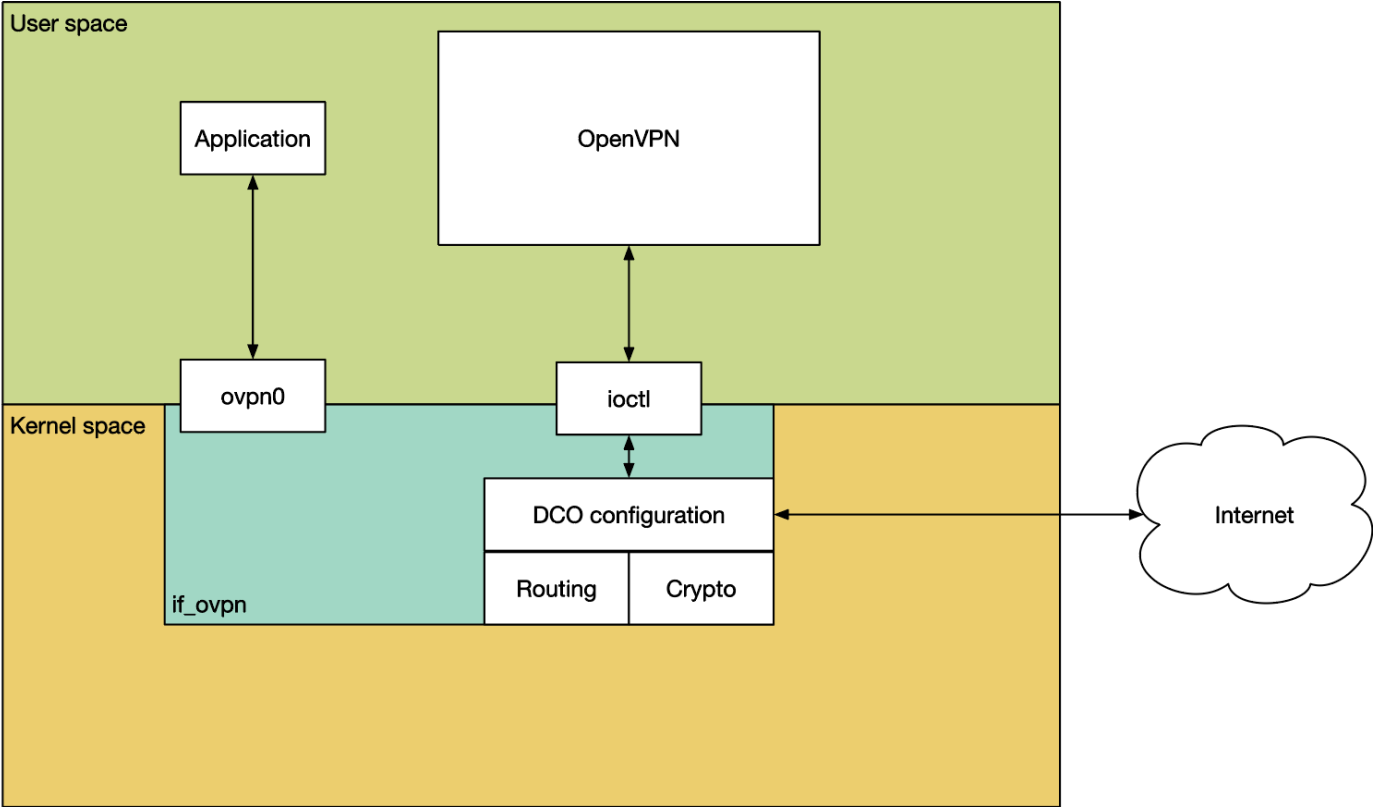
我们使用 if\_tun 来创建隧道接口。

我们使用 if\_tun 来创建隧道接口。

# DCO??

我们使用 DCO 来创建隧道接口。

我们使用 DCO 来创建隧道接口。



DCO 使用 if\_ovpn 来创建隧道接口。OpenVPN 使用 if\_ovpn 来创建隧道接口。我们使用 if\_ovpn 来创建隧道接口。

OpenVPN 使用 DCO 来创建隧道接口。我们使用 DCO 来创建隧道接口。我们使用 DCO 来创建隧道接口。我们使用 DCO 来创建隧道接口。我们使用 DCO 来创建隧道接口。

?? ??

# Multiplexing

1. 在 `ovpn.conf` 文件中添加以下配置：

```

client
remote ovpn.example.com 1195
remote-cert-tls client
remote-user ovpn
auth-user-pass ovpn

```

2. 保存并退出编辑器。

3. 使用 `ovpn` 命令启动 OpenVPN 客户端：

```

ovpn

```

4. 如果提示输入用户名和密码，请输入 `ovpn` 和 `ovpn`。

5. 成功连接后，您将获得一个 IP 地址。

`ovpn_new_peer()`<sup>6</sup>




























ovpn\_udp\_input()        .      

UDP       .

1. 在 Windows 10 上安装 OpenVPN 客户端  
 2. 下载并安装 OpenVPN 客户端  
 3. 配置 OpenVPN 客户端  
 4. 连接 OpenVPN 服务器  
 5. 验证连接  
 6. 使用 OpenVPN 客户端

OpenVPN 的 DCO 实现。

OpenVPN 的 DCO 实现。OpenVPN 使用 ioctl 系统调用，Linux 和 FreeBSD 都支持。OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。

OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。

OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。

# UDP

OpenVPN 使用 UDP 和 TCP 协议。OpenVPN 使用 UDP 和 TCP 协议。OpenVPN 使用 UDP 和 TCP 协议。

OpenVPN 使用 UDP 和 TCP 协议。OpenVPN 使用 UDP 和 TCP 协议。OpenVPN 使用 UDP 和 TCP 协议。

OpenVPN 使用 UDP 和 TCP 协议。OpenVPN 使用 UDP 和 TCP 协议。OpenVPN 使用 UDP 和 TCP 协议。

# Hardware Cryptography Offload

OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。

OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。

# Locking Design

OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。

OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。

OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。OpenVPN 使用 if\_ovpn 接口。



在 `ovpn_route_peer()` 中，我们使用 `ovpn_find_peer_by_ip()` 来查找 VPN 对端。如果找到对端，我们就会调用 `ovpn_route_peer()` 来为对端添加路由。如果找不到对端，我们就会调用 `ovpn_route_peer()` 来为对端添加路由。如果找不到对端，我们就会调用 `ovpn_route_peer()` 来为对端添加路由。

在 `if_ovpn` 中，我们使用 `ovpn_route_peer()` 来为对端添加路由。

# Key Rotation

OpenVPN 使用 `ovpn_route_peer()` 来为对端添加路由。在 `if_ovpn` 中，我们使用 `ovpn_route_peer()` 来为对端添加路由。

OpenVPN 使用 `OVPN_NEW_KEY` 来为对端添加路由。在 `if_ovpn` 中，我们使用 `ovpn_route_peer()` 来为对端添加路由。

在 `if_ovpn` 中，我们使用 `OVPN_SWAP_KEYS` 来为对端添加路由。

在 `if_ovpn` 中，我们使用 `OVPN_DEL_KEY` 来为对端添加路由。

# vnet

在 `vnet` 中，我们使用 `vnet` 来为对端添加路由。

在 `vnet` 中，我们使用 `IP` 来为对端添加路由。

在 `pfSense` 中，我们使用 `OpenVPN` 来为对端添加路由。

在 `FreeBSD` 中，我们使用 `FreeBSD` 来为对端添加路由。

# Performance

在 `FreeBSD` 中，我们使用 `FreeBSD` 来为对端添加路由。

在 `FreeBSD` 中，我们使用 `FreeBSD` 来为对端添加路由。

在 `FreeBSD` 中，我们使用 `FreeBSD` 来为对端添加路由。

if_tun	207.3 Mbit/s
DCO Software	213.1 Mbit/s
DCO AES-NI	751.2 Mbit/s
DCO QAT	1,064.8 Mbit/s

"if\_tun" DCO 100% 100% OpenVPN 100% . 100% 100% AES-NI 100% 100% 'DCO 100% ' 100% 100% 100% 100% 100% 100% . 100% 100% 100% 100% DCO 100% 100% 100% 100% . 100% 100% 100% ( 100% , DCO 100% AES-NI 100% 100% 100% ) 100% 100% 100% . DCO 100% 300% 100% 100% .

100% 100% 100% : 100% QuickAssist 100% 100% AES-NI 100% 100% 100% OpenVPN 100% 100% 500% 100% 100% .

# Future Work

100% 100% 100% 100% 100% 100% , 100% 100% 100% DCO 100% 100% 100% 100% . 100% OpenVPN 100% 3200% 100% 100% (IV) 100% , 100% 100% 100% <sup>11</sup> , 100% 100% 100% IV 100% 100% 100% 100% .

100% , 100% 100% 100% 100% . OpenVPN 100% 100% 100% 3600000% , 100% 100% 30% 100% 100%  $2^{32} * 0.7 / 3600$  , 100% 100% 835.000000% 100% 100% . 100% "100%" "8~9Gbit/s 100% (1300000% 100% 100% ) .

DCO 100% 100% 100% 100% 100% 100% 100% 100% 100% .

100% 100% 100% , 100% 100% 100% OpenVPN 100% 6400% IV 100% 100% 100% 100% 100% 100% 100% .

# Thanks

if\_ovpn 100% Rubicon Communications(Netgate 100% 100% ) 100% pfSense 100% 100% 100% . 22.05 pfSense plus 100% <sup>12</sup> 100% 100% . 100% 100% FreeBSD 100% 100% 100% 14.0 100% 100% . 100% OpenVPN 2.6.0 100% 100% .

100% , 100% FreeBSD 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% OpenVPN 100% 100% 100% 100% 100% .

Footnotes:

