

FreeBSD? RACK ? ?? TCP ??

by Randall Stewart and Michael Tüxen

PDF : [Tuxen.pdf](#)

2017年 FreeBSD 的 TCP 堆栈 进行了 一次 大的 更新 。 这次 更新 主要 是 为了 提高 TCP 堆栈 的 性能 和 稳定性 。 更新 包括 了 对 SYN-Cookies 的 支持 ， 对 IP 堆栈 的 优化 ， 以及 对 TCP 堆栈 的 其他 改进 。 这些 改进 将 帮助 FreeBSD 的 TCP 堆栈 更好地 处理 高并发 的 网络 流量 ， 并 提高 了 TCP 堆栈 的 性能 和 稳定性 。

TCP RACK 是 tcp_do_segment() 函数 中 实现 的 一种 新的 排序 算法 。 它 是 为了 提高 TCP 堆栈 的 性能 和 稳定性 而 设计 的 。 RACK 是 一种 新的 排序 算法 ， 它 是 为了 提高 TCP 堆栈 的 性能 和 稳定性 而 设计 的 。 2021 年 RFC 8985 定义了 一种 新的 排序 算法 ， 它 是 为了 提高 TCP 堆栈 的 性能 和 稳定性 而 设计 的 。 TCP RACK 是 一种 新的 排序 算法 ， 它 是 为了 提高 TCP 堆栈 的 性能 和 稳定性 而 设计 的 。 RFC 8985 定义了 一种 新的 排序 算法 ， 它 是 为了 提高 TCP 堆栈 的 性能 和 稳定性 而 设计 的 。 (SACK) 是 一种 新的 排序 算法 ， 它 是 为了 提高 TCP 堆栈 的 性能 和 稳定性 而 设计 的 。 TCP RACK 是 一种 新的 排序 算法 ， 它 是 为了 提高 TCP 堆栈 的 性能 和 稳定性 而 设计 的 。 RFC 8985 定义了 一种 新的 排序 算法 ， 它 是 为了 提高 TCP 堆栈 的 性能 和 稳定性 而 设计 的 。 RACK 是 一种 新的 排序 算法 ， 它 是 为了 提高 TCP 堆栈 的 性能 和 稳定性 而 设计 的 。

TCP RACK ?? ?? ??

RACK 是 FreeBSD CURRENT 和 FreeBSD 14.0 中 实现 的 一种 新的 排序 算法 。 它 是 为了 提高 TCP 堆栈 的 性能 和 稳定性 而 设计 的 。

FreeBSD 14.0 中 实现 的 一种 新的 排序 算法 。

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

在 编译 时 需要 添加 一些 选项 。 在 编译 时 需要 添加 一些 选项 (HPTS) 和 一些 选项 (tcp_rack.ko) 。

```
tcp_rack_load="YES"
```

在 编译 时 需要 添加 一些 选项 /boot/loader.conf 中 实现 的 一种 新的 排序 算法 。

FreeBSD CURRENT 内核支持 TCP RACK 和 HPTS 功能，但需要手动加载 `tcp_hpts.ko` 和 `tcp_rack.ko` 内核模块。可以通过 `kldload` 命令加载，或者通过修改 `/boot/loader.conf` 文件来自动加载：

```
tcphpts_load="YES"
tcp_rack_load="YES"
```

修改完成后，需要重启系统。FreeBSD CURRENT 内核支持 TCP RACK 功能，但需要手动加载 `tcp_hpts.ko` 和 `tcp_rack.ko` 内核模块。

```
option TCPPHPTS
option TCP_RACK
```

TCP 连接。FreeBSD 14.0 内核支持 TCP RACK 功能，但需要手动加载 `tcp_hpts.ko` 和 `tcp_rack.ko` 内核模块。可以通过 `kldload` 命令加载，或者通过修改 `/boot/loader.conf` 文件来自动加载：

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

FreeBSD 14.0 内核支持 TCP RACK 功能，但需要手动加载 `tcp_hpts.ko` 和 `tcp_rack.ko` 内核模块。可以通过 `kldload` 命令加载，或者通过修改 `/boot/loader.conf` 文件来自动加载：

FreeBSD 14.1 内核支持 TCP RACK 功能，但需要手动加载 `tcp_hpts.ko` 和 `tcp_rack.ko` 内核模块。可以通过 `kldload` 命令加载，或者通过修改 `/boot/loader.conf` 文件来自动加载：

TCP RACK 功能。FreeBSD 14.0 内核支持 TCP RACK 功能，但需要手动加载 `tcp_hpts.ko` 和 `tcp_rack.ko` 内核模块。可以通过 `kldload` 命令加载，或者通过修改 `/boot/loader.conf` 文件来自动加载：

`sysctl-variable net.inet.tcp.functions_default` 参数。FreeBSD 14.0 内核支持 TCP RACK 功能，但需要手动加载 `tcp_hpts.ko` 和 `tcp_rack.ko` 内核模块。可以通过 `kldload` 命令加载，或者通过修改 `/boot/loader.conf` 文件来自动加载：

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

`/etc/sysctl.conf` 文件。FreeBSD 14.0 内核支持 TCP RACK 功能，但需要手动加载 `tcp_hpts.ko` 和 `tcp_rack.ko` 内核模块。可以通过 `kldload` 命令加载，或者通过修改 `/boot/loader.conf` 文件来自动加载：

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

`listener` 参数。FreeBSD 14.0 内核支持 TCP RACK 功能，但需要手动加载 `tcp_hpts.ko` 和 `tcp_rack.ko` 内核模块。可以通过 `kldload` 命令加载，或者通过修改 `/boot/loader.conf` 文件来自动加载：

`tcpsso(8)` 命令。FreeBSD 14.0 内核支持 TCP RACK 功能，但需要手动加载 `tcp_hpts.ko` 和 `tcp_rack.ko` 内核模块。可以通过 `kldload` 命令加载，或者通过修改 `/boot/loader.conf` 文件来自动加载：

```

    if (tcp_function_blk == IPPROTO_TCP) {
        tcp_rack = TCP_RACK;
        tcp_function_set(tcp_rack);
    } else {

```

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

TCP RACK ??? ??

TCP RACK

RACK/TLP

[illegible][illegible]

Proportional Rate Reduction (PRR)

The (PRR) TCP RACK , RFC 6937 , 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 拥塞 窗口 (cwnd) 和 慢启动 阈值 (ssthresh) 。 这些 算法 通过 动态 调整 发送 速率 来 避免 网络 拥塞 ， 从而 提高 网络 性能 。

RACK Rapid Recovery (RRR)

RACK Rapid Recovery(RRR) 是 一种 快速 恢复 机制 ， 旨在 提高 TCP 在 网络 拥塞 时 的 恢复 速度 。 它 通过 引入 快速 重传 和 快速 恢复 算法 来 实现 。 在 传统 的 TCP 中 ， 当 发送 方 收到 接收 方 的 确认 时 ， 会 立即 增加 发送 窗口 并 继续 发送 数据 。 然而 ， 在 RRR 中 ， 发送 方 会 等待 一段时间 后 再 增加 发送 窗口 ， 这 可以 避免 在 网络 拥塞 时 发送 过多 数据 ， 从而 提高 网络 性能 。

在 网络 拥塞 时 ， 接收 方 会 立即 发送 快速 重传 确认 (RRR ACK) ， 这 可以 避免 发送 方 等待 接收 方 确认 的 时间 。 此外 ， RRR 还 引入 了 快速 恢复 算法 ， 该 算法 会 根据 接收 方 的 确认 来 动态 调整 发送 窗口 和 慢启动 阈值 ， 从而 提高 网络 性能 。

SACK Attack Detection

SACK 攻击 检测 是 一种 用于 检测 网络 拥塞 攻击 的 技术 。 在 网络 拥塞 时 ， 攻击 者 会 发送 大量 的 数据包 来 占用 网络 带宽 ， 从而 导致 网络 性能 下降 。 为了 检测 这 种 攻击 ， 网络 工程师 设计 了 许多 检测 算法 ， 其中 最 著名 的 是 拥塞 窗口 (cwnd) 和 慢启动 阈值 (ssthresh) 。 这些 算法 通过 动态 调整 发送 速率 来 避免 网络 拥塞 ， 从而 提高 网络 性能 。

TCP RACK 是 一种 快速 恢复 机制 ， 旨在 提高 TCP 在 网络 拥塞 时 的 恢复 速度 。 它 通过 引入 快速 重传 和 快速 恢复 算法 来 实现 。 在 传统 的 TCP 中 ， 当 发送 方 收到 接收 方 的 确认 时 ， 会 立即 增加 发送 窗口 并 继续 发送 数据 。 然而 ， 在 RACK 中 ， 发送 方 会 等待 一段时间 后 再 增加 发送 窗口 ， 这 可以 避免 在 网络 拥塞 时 发送 过多 数据 ， 从而 提高 网络 性能 。

```

option TCP_SAD_DETECTION

```

SACK 攻击 检测 是 一种 用于 检测 网络 拥塞 攻击 的 技术 。 在 网络 拥塞 时 ， 攻击 者 会 发送 大量 的 数据包 来 占用 网络 带宽 ， 从而 导致 网络 性能 下降 。 为了 检测 这 种 攻击 ， 网络 工程师 设计 了 许多 检测 算法 ， 其中 最 著名 的 是 拥塞 窗口 (cwnd) 和 慢启动 阈值 (ssthresh) 。 这些 算法 通过 动态 调整 发送 速率 来 避免 网络 拥塞 ， 从而 提高 网络 性能 。

Burst Mitigation

TCP RACK 是 TCP 接收端的一个特性，它通过限制接收窗口的大小来防止接收端被突发流量淹没。
 在接收端，TCP 接收窗口的大小是由接收缓冲区的大小决定的。如果接收缓冲区的大小有限，那么接收窗口的大小也会有限。
 如果接收端收到大量的突发流量，那么接收窗口的大小就会变得很小，从而导致接收端无法接收更多的数据。
 这就是 TCP RACK 的作用，它通过限制接收窗口的大小来防止接收端被突发流量淹没。

Support for TCP Blackbox Logging (BBLog)

TCP RACK 是 TCP 接收端的一个特性，它通过限制接收窗口的大小来防止接收端被突发流量淹没。
 在接收端，TCP 接收窗口的大小是由接收缓冲区的大小决定的。如果接收缓冲区的大小有限，那么接收窗口的大小也会有限。
 如果接收端收到大量的突发流量，那么接收窗口的大小就会变得很小，从而导致接收端无法接收更多的数据。
 这就是 TCP RACK 的作用，它通过限制接收窗口的大小来防止接收端被突发流量淹没。

Large Receive Offload (LRO) Integration for Burst Mitigation

TCP 接收端的大接收卸载 (LRO) 是一种技术，它可以将多个接收窗口合并成一个大的接收窗口，从而提高接收效率。
 在接收端，TCP 接收窗口的大小是由接收缓冲区的大小决定的。如果接收缓冲区的大小有限，那么接收窗口的大小也会有限。
 如果接收端收到大量的突发流量，那么接收窗口的大小就会变得很小，从而导致接收端无法接收更多的数据。
 这就是 TCP RACK 的作用，它通过限制接收窗口的大小来防止接收端被突发流量淹没。

TCP RACK 是 TCP 接收端的一个特性，它通过限制接收窗口的大小来防止接收端被突发流量淹没。
 在接收端，TCP 接收窗口的大小是由接收缓冲区的大小决定的。如果接收缓冲区的大小有限，那么接收窗口的大小也会有限。
 如果接收端收到大量的突发流量，那么接收窗口的大小就会变得很小，从而导致接收端无法接收更多的数据。
 这就是 TCP RACK 的作用，它通过限制接收窗口的大小来防止接收端被突发流量淹没。

A Host of Alternate Features

sysctl-variables 是 TCP RACK 的一个特性，它通过限制接收窗口的大小来防止接收端被突发流量淹没。
 在接收端，TCP 接收窗口的大小是由接收缓冲区的大小决定的。如果接收缓冲区的大小有限，那么接收窗口的大小也会有限。
 如果接收端收到大量的突发流量，那么接收窗口的大小就会变得很小，从而导致接收端无法接收更多的数据。
 这就是 TCP RACK 的作用，它通过限制接收窗口的大小来防止接收端被突发流量淹没。

???? TCP RACK ??? ????? ??

本文档描述了 TCP RACK 的实现，FreeBSD 中该实现位于 `netinet/tcp_rack.c`。该实现旨在提高 TCP 连接在丢包情况下的性能，通过快速重传和快速关闭连接来减少延迟。本文档还介绍了 TCP RACK 的实现细节，包括如何配置和测试。

本文档描述了 TCP RACK 的实现，FreeBSD 中该实现位于 `netinet/tcp_rack.c`。该实现旨在提高 TCP 连接在丢包情况下的性能，通过快速重传和快速关闭连接来减少延迟。本文档还介绍了 TCP RACK 的实现细节，包括如何配置和测试。

TCP RACK 的实现旨在提高 TCP 连接在丢包情况下的性能，通过快速重传和快速关闭连接来减少延迟。本文档还介绍了 TCP RACK 的实现细节，包括如何配置和测试。

?? ? ??

TCP RACK 的实现旨在提高 TCP 连接在丢包情况下的性能，通过快速重传和快速关闭连接来减少延迟。本文档还介绍了 TCP RACK 的实现细节，包括如何配置和测试。

TCP RACK 的实现旨在提高 TCP 连接在丢包情况下的性能，通过快速重传和快速关闭连接来减少延迟。本文档还介绍了 TCP RACK 的实现细节，包括如何配置和测试。

本文档描述了 TCP RACK 的实现，FreeBSD 中该实现位于 `netinet/tcp_rack.c`。该实现旨在提高 TCP 连接在丢包情况下的性能，通过快速重传和快速关闭连接来减少延迟。本文档还介绍了 TCP RACK 的实现细节，包括如何配置和测试。

本文档描述了 TCP RACK 的实现，FreeBSD 中该实现位于 `netinet/tcp_rack.c`。该实现旨在提高 TCP 连接在丢包情况下的性能，通过快速重传和快速关闭连接来减少延迟。本文档还介绍了 TCP RACK 的实现细节，包括如何配置和测试。