2024 Jan/Feb

- FreeBSD용 RACK 및 대체 TCP 스택
- FreeBSD 14의 TCP 관련 업데이트
- if_ovpn 또는 OpenVPN

FreeBSD[illegible] RACK [illegible] TCP [illegible]

by Randall Stewart and Michael TÜxen

[illegible] ***: Tuxen.pdf***

2017[illegible] FreeBSD[illegible] TCP [illegible] TCP [illegible]. [illegible] TCP [illegible]. [illegible] TCP [illegible]. SYN-Cookies [illegible], [illegible], [illegible] IP [illegible] TCP [illegible] TCP [illegible] SYN-Cache [illegible]. [illegible] TCP [illegible] TCP [illegible], [illegible] TCP [illegible] TCP [illegible].

TCP RACK [illegible] tcp_do_segment() [illegible] TCP [illegible]. [illegible] (RACK)[illegible]. RACK[illegible], 2021[illegible] RFC 8985[illegible]. [illegible] TCP [illegible] RACK[illegible]. [illegible] TCP RACK [illegible] RFC 8985[illegible]. [illegible] (SACK) [illegible]. TCP RACK [illegible] RFC 8985[illegible] RACK [illegible]. [illegible].

TCP RACK [illegible]

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

FreeBSD 14.0[illegible].

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

[illegible]. [illegible] TCP [illegible] (HPTS)[illegible]. [illegible] TCP RACK [illegible] (tcp_rack.ko)[illegible]. TCP RACK [illegible].

```
tcp_rack_load="YES"
```

이 작업은 재부팅할 때마다 /boot/loader.conf 파일에 해당 줄을 추가하여 수행할 수 있습니다 .

FreeBSD CURRENT에서는 TCP RACK과 HPTS가 모두 기본적으로 커널 모듈로 빌드됩니다 . tcphpts.ko는 tcp_rack.ko의 종속성으로 자동으로 로드되므로 , 후자만 kldload를 사용하여 로드해야 합니다 . 재부팅할 때마다 TCP RACK 스택을 로드하려면 /boot/loader.conf 파일에 다음 두 줄을 추가해야 합니다 :

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

커널 구성 파일에 다음 두 줄을 추가하고 커널을 다시 빌드하면 FreeBSD CURRENT의 커널에 TCP RACK 스택을 정적으로 컴파일할 수도 있습니다 .

```
option TCPHPTS
option TCP_RACK
```

TCP 전송 개발자가 다양한 TCP 스택을 계측하고 디버깅하는 표준 방법이기 때문에 , 이제 FreeBSD 14.0 이상과 모든 64비트 플랫폼의 FreeBSD CURRENT에도 TCP 블랙박스 로깅 (옵션 TCP_BLACKBOX)이 기본으로 빌드되어 있습니다 .

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

위는 FreeBSD 시스템에서 TCP RACK 스택을 사용할 수 있게 하는 방법을 설명합니다 . 셸에서 실행하면 사용 가능한 모든 TCP 스택 목록이 표시됩니다 .

곧 출시될 버전인 FreeBSD 14.1 이상에서는 TCP RACK 스택의 사용법이 위에서 설명한 FreeBSD CURRENT 와 동일합니다 .

TCP RACK 스택을 실제로 사용하는 방법은 여러 가지가 있는데 , 일부는 애플리케이션의 소스 코드 변경을 포함하며 일부는 구성 변경만 포함합니다 .

sysctl-variable net.inet.tcp.functions_default는 socket(2) 시스템 호출을 사용하여 생성된 새 TCP 엔드포인트에 사용되는 기본 TCP 스택을 지정하는 데 사용됩니다 . 다음을 실행하면 기본 스택이 TCP RACK 스택으로 설정됩니다 .

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

/etc/sysctl.conf 파일에 이 줄을 추가하면 시스템 재부팅 후 TCP RACK 스택이 기본 TCP 스택이 됩니다 .

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

listener를 통해 TCP 엔드포인트가 생성되면 listener에서 TCP 스택이 상속되거나 sysctl-variable net.inet.tcp.functions_inherit_listen_socket_stack이 0이 아닌지 또는 0인지에 따라 기본 TCP 스택을

기반으로 합니다 . 이 변수의 기본값은 1입니다 .

이 도구의 매뉴얼 페이지에 설명된 대로 tcpsso(8) 명령줄 도구를 사용하여 개별 TCP 연결의 TCP 스택을 변경할 수도 있습니다 .

소스 코드를 변경할 수 있는 경우 , TCP_FUNCTION_BLK라는 이름의 IPPROTO_TCP 수준 소켓 옵션을 사용하여 소켓에 사용 중인 TCP 스택을 TCP RACK 스택으로 전환할 수 있습니다 . 옵션 값의 타입은 구조체 tcp_function_set입니다 . 예를 들어 , 다음 코드는 이를 수행합니다 :

```
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 스택을 사용하면 기본 TCP 스택이 현재 지원하지 않는 여러 기능을 사용할 수 있습니다 . 이러한 기능 중 상당수는 net.inet.tcp.rack의 IPPROTO_TCP 수준 소켓 옵션 또는 sysctl-variables를 통해 제어할 수 있습니다 .

TCP RACK 스택의 특징

다음 섹션에서는 TCP RACK 스택이 제공하는 가장 중요한 기능에 대해 설명합니다 .

RACK/TLP

Recent Acknowledgement (RACK)과 Tail Loss Probe (TLP)는 TCP RACK 스택에 통합된 두 가지 기능입니다 . RACK은 패킷 손실이 감지되고 재전송이 트리거되는 방식을 변경합니다 . FreeBSD 기본 스택에서 구현되고 RFC 5681에 명시된 손실 감지는 TCP 스택이 재전송을 보내도록 하기 위해 세 번의 중복 승인 또는 SACK을 통한 승인 도착을 필요로 합니다 . 예를 들어 패킷이 4개 미만으로 전송된 경우 , 이로 인해 재전송 시간 초과가 발생한 후에야 TCP 스택이 재전송을 전송하게 됩니다 . RACK은 이를 변경하여 SACK이 도착했을 때 손실된 패킷을 전송한 후 충분한 시간이 경과한 경우 즉시 재전송이 이루어지도록 합니다 . 충분한 시간이 경과하지 않은 경우 (일반적으로 현재 RTT보다 약간 큰 시간), 작은 RACK 타이머가 시작되고 이 타이머가 만료되면 재전송이 전송됩니다 . 이렇게 하면 재전송 시간 초과로 인해 데이터를 강제로 전송해야 하는 많은 경우 (전부는 아니지만)가 해결됩니다 . 마지막 경우는 TLP로 해결됩니다 . 이 경우 TCP RACK 스택이 데이터를 전송할 때마다 재전송 타이머 대신 TLP 타이머를 시작합니다 . TLP 타이머가 만료되면 TCP RACK 스택은 새 세그먼트 또는 마지막으로 전송된 세그먼트를 보냅니다 . 이 TLP 전송 세그먼트의 희망은 발신자가 모든 데이터가 수신되었음을 나타내는 확인을 다시 받거나 (마지막 확인이 손실된 경우) TLP가 SACK을 유도하여 재전송 시간 초과에 도달하지 않고 정상적인 빠른 복구 메커니즘이 이어받아 혼잡 창이 1 MSS로 축소되는 것입니다 .

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

Proportional Rate Reduction (PRR)

[illegible] (PRR)[illegible] TCP RACK [illegible] , RFC 6937[illegible] IETF[illegible] . PRR[illegible] RFC 5681[illegible] TCP [illegible] CPU [illegible] . PRR[illegible] '[illegible] '[illegible] . PRR[illegible] RTT [illegible] .

RACK Rapid Recovery (RRR)

RACK Rapid Recovery(RRR)[illegible] . [illegible] TCP RACK [illegible] SACK[illegible] RACK [illegible] RACK [illegible] . RACK [illegible] (RACK [illegible] 1ms[illegible]), TCP RACK [illegible] PRR[illegible] PRR [illegible] . [illegible] RRR[illegible] 3[illegible] , PRR[illegible] 6[illegible] .

[illegible] '[illegible] '[illegible] (QoE)[illegible] . [illegible] , [illegible] 12Mbps[illegible] RRR [illegible] . [illegible] RRR [illegible] [illegible] (MTU)[illegible] 1500[illegible] 12Mbps[illegible] .

SACK Attack Detection

[illegible] . TCP RACK [illegible] TCP RACK [illegible] TCP [illegible] CPU [illegible] . [illegible] SACK[illegible] .

TCP RACK [illegible] TCP_SAD_DETECTION[illegible] . SAD[illegible] SACK [illegible] (SAD)[illegible] . [illegible] TCP RACK [illegible]

이 기능을 활성화할 수 있습니다 .

```
option TCP_SAD_DETECTION
```

일단 추가하면 기본적으로 켜져 있습니다 . 악의적인 피어를 감시하고 , 탐지되면 해당 피어에서 SACK 처리를 비활성화합니다 . 이렇게 하면 해당 피어의 성능이 저하되지만 연결이 진행되는 것을 막지는 못합니다 . 사실상 SACK이 활성화되지 않은 것처럼 응답하는 연결이 됩니다 . 이렇게 하면 손실 복구에 불이익이 발생하지만 연결은 계속할 수 있습니다 .

Burst Mitigation

버스트 완화 기능은 사용자 개입 없이 TCP RACK 스택에 내장되어 있습니다 . 버스트를 완화하기 위해 스택은 전송 기회에 설정된 크기 (최대 버스트 크기)만 전송하고 작은 타이머를 시작하거나 (더 많이 전송하기 위해) 반환되는 확인 스트림에 따라 더 많은 데이터를 전송하도록 유도합니다 . 이렇게 하면 과도한 손실을 유발할 수 있는 대규모 버스트를 완화하는 데 도움이 됩니다 .

Support for TCP Blackbox Logging (BBLog)

TCP RACK 스택의 흥미로운 측면 중 하나는 디버깅과 일반적인 통계 분석 및 계측을 위한 TCP 블랙박스 로깅을 광범위하게 지원한다는 점입니다 . 이를 통해 문제를 추적하고 연결 동작에 대한 분석을 훨씬 쉽게 얻을 수 있습니다 .

Large Receive Offload (LRO) Integration for Burst Mitigation

TCP 대용량 수신 오프로드 (LRO)는 수신된 여러 개의 TCP 세그먼트를 하나의 세그먼트로 합쳐서 TCP 스택으로 전달하기 전에 수신기에 필요한 CPU 리소스를 줄이는 기능입니다 . 이로 인해 개별 수신 세그먼트에 대한 정보가 손실되는 경우가 많지만 , TCP 스택에서 처리해야 하는 TCP 세그먼트 수가 줄어들기 때문에 필요한 CPU 리소스를 줄일 수 있습니다 .

흥미로운 기능 상호 작용은 TCP RACK 스택에서 페이싱을 더 잘 지원하기 위해 LRO 코드에 적용된 일련의 변경 사항입니다 . TCP 연결이 버스트 완화를 수행하는 경우 , 전송 경로를 더 자주 통과하여 더 작은 버스트를 보내는 경향이 있습니다 . 이 때문에 패킷 도착 정보에 대한 모든 타이밍 데이터가 손실 없이 TCP RACK 스택으로 전달될 수 있도록 LRO 코드가 변경되었습니다 . 기본적으로 패킷을 처리하는 동안 LRO 코드는 패킷이 TCP RACK 스택에 직접 패킷을 큐에 넣을 수 있는 연결과 연결되어 있는지 조회합니다 . 그렇다면 패킷은 연결에 직접 큐에 대기하고 연결 상태에 따라 연결이 깨어날 수 있습니다 . TCP RACK 스택이 버스트 완화 또는 페이싱을 수행하는 경우 , 타이머가 만료되고 인바운드 승인을 통해 무언가를 수행할 수 있을 때까지 웨이크업이 연기됩니다 . 이러한 단계는 또한 IP 스택 처리를 우회하므로 필요한 CPU 리소스를 추가로 약간 줄일 수 있습니다 .

A Host of Alternate Features

이 외에도 다양한 소켓 옵션과 sysctl-variables를 통해 TCP RACK 스택에서 많은 기능을 사용할 수 있습니다. 현재 TCP RACK 스택은 페이싱, 버스트 완화 옵션, 복구 응답 수정 등 다양한 기능을 가능하게 하는 58개의 소켓 옵션을 지원합니다. 소켓 옵션 외에도, 소켓 옵션을 모든 연결에 적용하거나 다양한 TCP RACK 스택 기본 구성을 수정하기 위해 약 150개의 sysctl-variables가 존재합니다. 이러한 모든 기능과 구성을 통해 네트워크 조건과 요구사항에 맞게 TCP RACK 스택을 조정할 수 있습니다.

넷플릭스가 TCP RACK 스택을 발전시키는 방법

넷플릭스는 현재 TCP RACK 스택만 사용하고 있으며, FreeBSD 기본 스택은 존재하지만 사용되지 않습니다. 넷플릭스가 TCP RACK 스택을 사용하는 방식은 약간 새롭고 주목할 가치가 있습니다. 넷플릭스는 실제로 릴리스 번호로 명명된 여러 세대의 TCP RACK 스택을 보관하고 있습니다. 항상 전송 그룹에서 개발 중인 모든 최첨단 기능이 포함된 "최신" TCP RACK 스택을 유지합니다.

주기적으로 릴리스가 중단되면 릴리스 번호에 따라 개발 중인 최신 TCP RACK 스택이 복사되어 지원됩니다. 그런 다음 이 TCP 스택은 사용 중인 기본 TCP 스택인 이전 릴리스와 비교하여 QoE 및 CPU 성능을 기준으로 평가됩니다. 최신 TCP RACK 스택이 이전 TCP RACK 스택보다 최소한 동등하거나 더 나은 경우, 기본값은 다음 릴리스에서 최신 TCP RACK 스택으로 전환됩니다. 이전 TCP RACK 스택은 여러 릴리스 동안 유지되다가 결국 제거됩니다.

TCP RACK 스택의 새로운 기능도 이러한 방식으로 테스트하여 해당 기능이 가치를 더하는지 여부를 결정할 수 있습니다. 넷플릭스 사용자의 체감 품질 저하 없이 네트워크에 미치는 영향을 줄이는 것이 전송 팀의 주요 목표 중 하나이며, 이를 통해 넷플릭스는 더 나은 네트워크 시민이 되는 동시에 전반적으로 우수한 QoE를 제공할 수 있습니다.

결론 및 전망

TCP RACK 스택은 FreeBSD 기본 스택에 대한 강력한 대안을 제공합니다. 더 많은 기능과 옵션을 추가하여 애플리케이션 개발자가 사용자에게 더 나은 TCP 경험을 제공할 수 있는 다양한 대안을 제공합니다.

TCP RACK 스택은 넷플릭스 설정과 워크로드를 사용하여 광범위하게 테스트되었습니다. 하지만 다른 설정과 워크로드에서도 테스트하는 것이 중요합니다. 따라서 사용자가 자신의 하드웨어에서, 자신의 설정과 워크로드에서 TCP RACK 스택을 테스트해 주시면 감사하겠습니다. 테스트 중에 발견되는 문제가 있으면 net@freebsd.org 또는 이 문서의 작성자에게 보고해 주세요. 피드백 및 추가 테스트에 따라, TCP RACK 스택은 향후 FreeBSD의 기본 스택이 될 수도 있습니다.

랜달 스튜어트 (rrs@freebsd.org)는 40년 이상 운영체제 개발자로 , 10년 이상 FreeBSD 개발자로 활동해 왔습니다 . 그는 TCP와 SCTP를 포함한 전송을 전문으로 하지만 운영체제의 다른 영역에도 관심을 갖고 있습니다 . 현재 넷플릭스에서 전송 팀에서 근무하며 TCP 스택을 지원하는 동시에 사용자 QoE를 지속적으로 개선하기 위해 혁신하고 있습니다 .

마이클 투젠 (tuexen@freebsd.org)은 뮌스터 응용과학대학의 교수이자 넷플릭스의 파트타임 계약자이며 2009년부터 FreeBSD 소스 커미터로 활동하고 있습니다 . 그의 관심 분야는 SCTP와 TCP와 같은 전송 프로토콜과 IETF에서의 표준화 , 그리고 FreeBSD에서의 구현입니다 .

FreeBSD 14 TCP

□ : *scheffenegger.pdf*

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

, FreeBSD 13.0 sys/netinet 1033 :

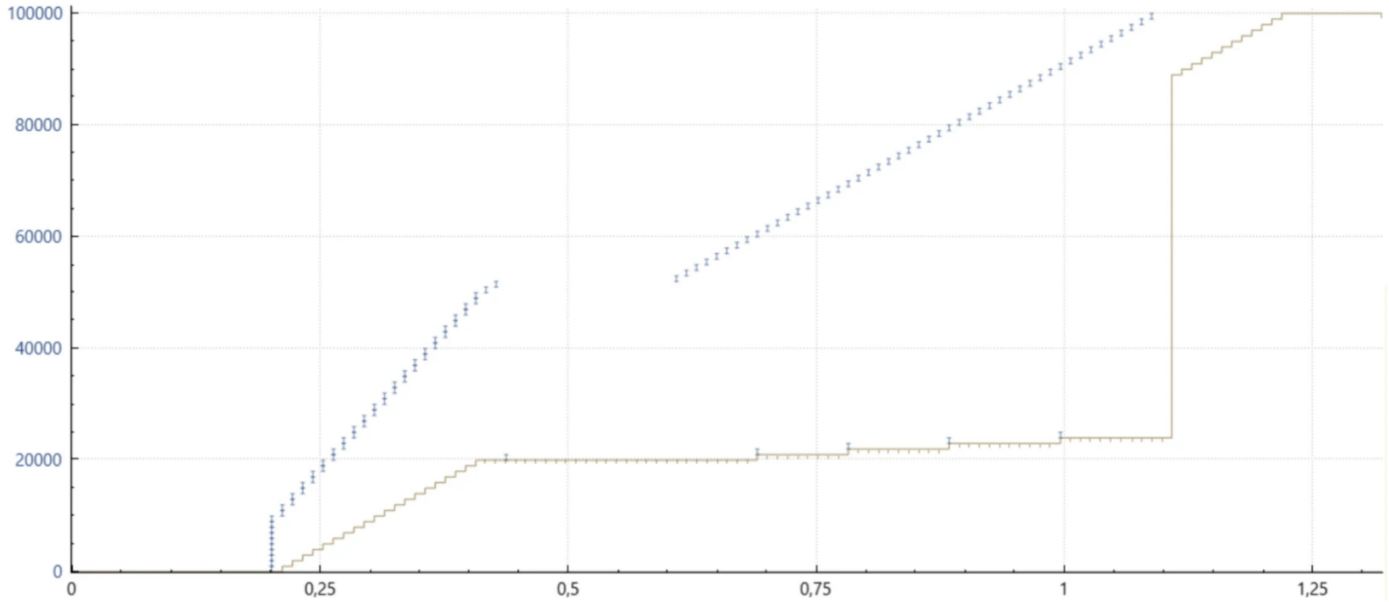
Proportional Rate Reduction ( )

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

역할을 할 수 있습니다 . 이러한 동작은 종종 후속 손실 (재전송된 패킷의 손실까지 포함할 수 있습니다 . 자세한 내용은 나중에 설명합니다 .)로 이어집니다 .

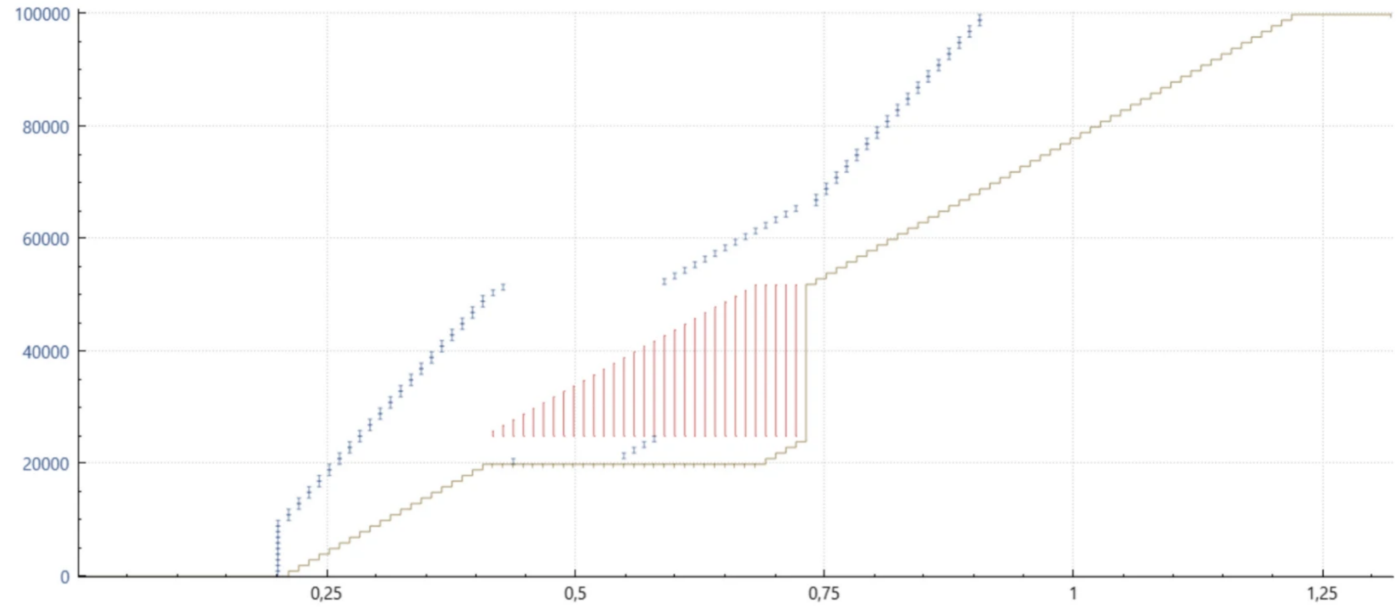
효과적인 전송 속도를 신속하게 조정하고 데이터 패킷의 손실이 여러 번 발생하거나 ACK 패킷이 손실되는 경우에도 보다 적절하게 대처하기 위해 PRR은 새로 들어오는 모든 ACK에 대해 전송해야 하는 데이터 양을 계산하고 그 시점에 적절한 만큼의 풀사이즈 패킷을 전송합니다 . 혼잡 구간이 절반으로 줄어들고 손실이 한 번만 발생하는 NewReno의 간단한 예에서 , 이렇게 하면 두 개의 ACK가 반환될 때마다 하나의 새 패킷이 전송됩니다 . 따라서 전송 속도가 이전의 절반 수준으로 즉시 조정되어 혼잡한 디바이스에 과부하가 걸리지 않습니다 . 여러 번의 데이터 손실 또는 버려진 ACK가 있는 경우 , PRR은 최종적으로 ACK가 수신될 때 한 개 이상의 패킷을 주입하여 놓친 전송 기회를 잘 활용할 수 있습니다 . 전반적으로 이 동작은 손실 복구가 발생하는 창 (RTT)의 끝에서 유효 혼잡 창이 예상 혼잡 창에 최대한 가깝게 유지되도록 하며 , 여러 패킷 손실 또는 ACK 손실과 같은 문제가 있는 시나리오에서도 전송 기회를 놓치지 않도록 보장합니다 .

몇 가지 그래프를 통해 이 세부 사항을 더 잘 설명할 수 있기를 바랍니다 . 아래에는 와이어샤크 또는 tcptrace 와 관련 xplot의 조합을 통해 얻을 수 있는 시간 순서 그래프가 있습니다 . 작은 파란색 세로 막대는 왼쪽 축에서와 같이 데이터 시퀀스를 포함하는 특정 패킷이 전송된 시점을 나타냅니다 . 아래의 녹색 가로줄은 수신자가 연속적으로 수신한 데이터를 나타냅니다 . 빨간색 세로선은 수신기에 도달한 데이터의 불연속적인 범위를 나타냅니다 .



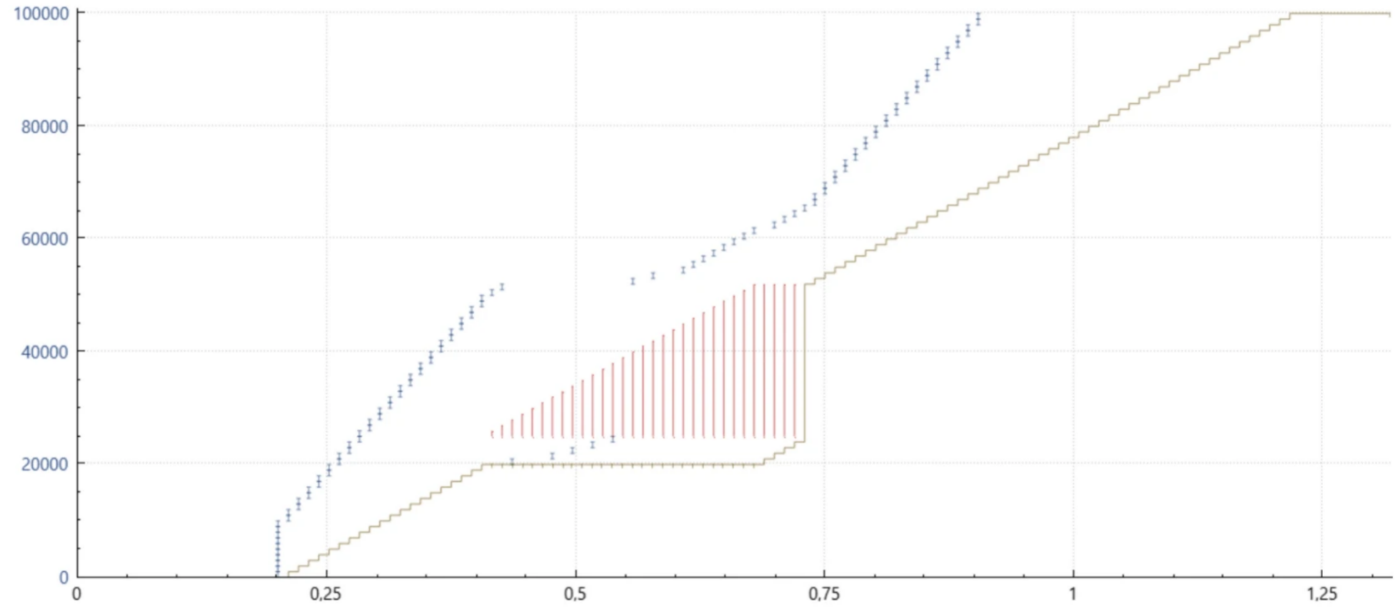
Cubic Without SACK or PRR, Classic NewReno Loss Recovery

하나의 창 (또는 왕복 시간) 내에 하나의 데이터 패킷만 복구할 수 있으며 , 가로로 길게 늘어진 녹색 선은 수신 애플리케이션이 추가 데이터를 처리하기까지 발생하는 지연 시간을 나타냅니다 .



Cubic with SACK, but no PRR

이 예시에서 볼 수 있듯이 , SACK은 손실된 모든 패킷을 (일반적으로) 하나의 RTT 내에 재전송할 수 있기 때문에 상황을 극적으로 개선합니다 . 그러나 나중에 각 ACK에서 전송이 일시 중지되었다가 다시 시작된다는 점에 유의하세요 . 이 동작은 데이터를 유효 속도로 전송하기 때문에 일부 패킷이 네트워크에서 삭제될 수 있습니다 . 이로 인해 하나 이상의 재전송이 너무 빨리 도착하여 네트워크에서 재전송을 삭제하는 경우가 종종 있습니다 . 이 경우 유일한 방법은 재전송 시간 초과 (RTO)를 기다리는 것입니다 .



Cubic with SACK (6675) and PRR

여기에 표시된 PRR의 개선 사항은 미묘합니다 . 이전에는 절반의 시간 동안 데이터가 전송되지 않고 후반부에는 너무 높은 전송률로 전송되었다면 , PRR은 새로운 전송률에 도달할 때까지 거의 매번 수신되는 ACK에 패킷을 주입한 다음 , 이후 들어오는 거의 모든 ACK에 패킷을 주입합니다 . 이는 재전송의 유효 전송률을 낮추고 네트워크에서 재전송이 버려질 가능성을 낮추는 역할을 합니다 . 결과적으로 RTO가 줄어들고 지연 시간이 개선됩니다 .

여기에 표시된 그래프는 완전히 정확하지는 않지만 수신된 ACK를 적절히 디더링하여 전송하는 PRR의 패킷 전송 측면을 전달하려고 시도한 것으로 , 이 경우 네트워크에서 버려졌을 수 있는 패킷을 포함하여 평균적으로

모든 ACK당 0.7개의 패킷이 전송됩니다 .

이 분야의 마지막 업데이트는 이제 손실 복구에 추가적인 손실이 발생하지 않는 한 PRR이 자동으로 덜 보수적인 모드로 전환된다는 것입니다 . 이는 혼잡 회피 단계에서 정상 작동할 때와 유사하게 손실 복구 중 전송 속도를 효과적으로 개선합니다 . PRR은 (당연히) SACK과 함께 사용할 때 가장 잘 작동하지만 , SACK이 아닌 중복 ACK만 사용할 수 있는 경우에도 마찬가지입니다 . ECN 피드백만 있어도 PRR은 전송 타이밍을 개선합니다 .

SACK Handling

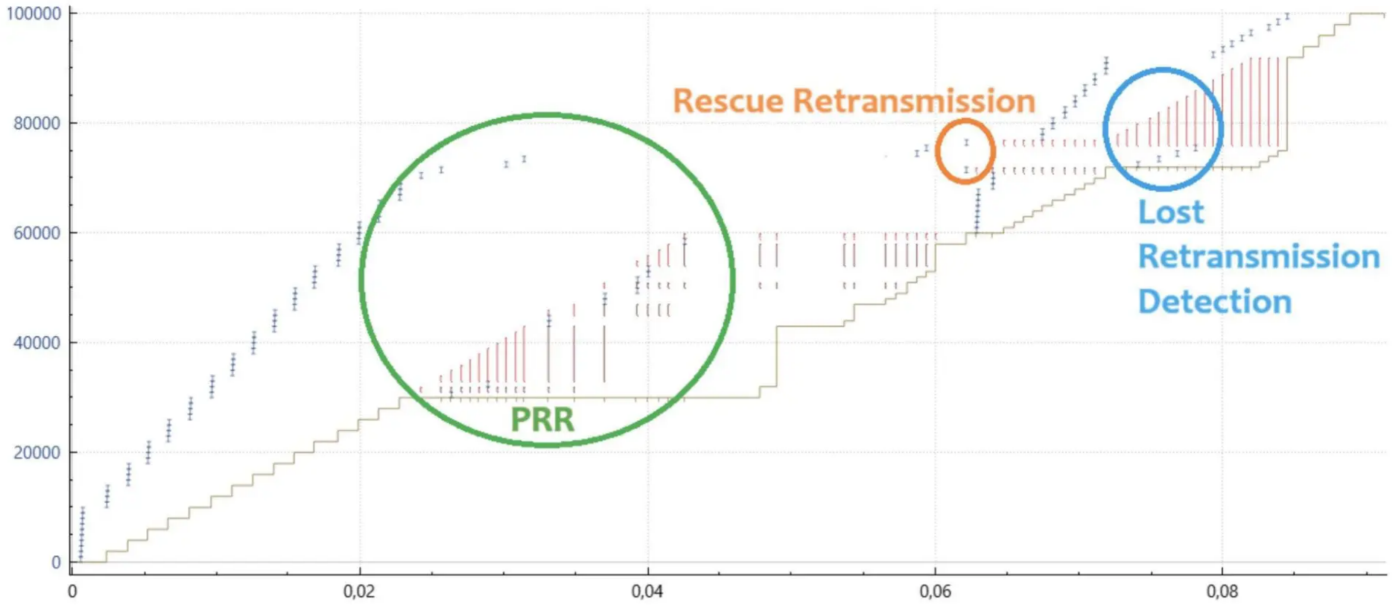
최근 몇 년 동안 RFC6675에 명시된 SACK 손실 복구에 대한 기본 스택의 준수 여부가 개선되었습니다 . 그러나 네트워크에 아직 남아 있는 데이터의 양에 대한 추정의 일부가 개선되었지만 , RFC6675의 다른 측면이 누락되었습니다 .

이제 이 분야의 개선 사항에는 RACK 스택에서 구현되는 꼬리 손실 프로브의 선구자인 구조 재전송의 사용이 포함됩니다 . 즉 , 이전 패킷 손실에 더해 전송의 마지막 몇 개의 패킷이 손실되면 스택이 문제를 감지하고 최종 패킷을 재전송하여 적시에 손실 복구를 수행합니다 .

또한 스택은 들어오는 SACK 블록을 처리할 때 추가 계정을 구현하여 특정 패킷이 수신되었거나 삭제되었을 가능성이 매우 높은 패킷이 네트워크를 떠나야 하는지 여부를 더 잘 추적합니다 .

마지막으로 개선된 기능은 재전송도 네트워크에서 삭제되었는지 여부를 추적하는 것이었지만 , 시간 도메인을 사용하는 RACK과 달리 기본 스택은 시퀀스 도메인을 살펴봅니다 . 이러한 손실된 재전송 감지는 RFC 시리즈에 명시되어 있지는 않지만 , TCP 스택을 사용하는 모든 요청 -응답 (예 : RPC) 프로토콜의 흐름 완료 시간 /IO 서비스 응답 시간을 줄이는 데 매우 유용한 추가 기능입니다 . 손실된 재전송을 추적하고 복구하는 기능은 아직 기본적으로 제공되지 않습니다 . FreeBSD 14에서는 net.inet.tcp.do_lrd로 활성화할 수 있지만 , FreeBSD 15에서는 net.inet.tcp.sack.lrd로 이동하여 기본적으로 활성화할 수 있습니다 .

전반적으로 , 이러한 변경 사항은 IP 네트워크의 혼잡과 관련하여 자주 발생하는 병리적 문제에서 기본 스택의 복원력을 향상시킵니다 .



마지막으로 , 기본 스택 (및 RACK 스택)은 가짜 중복 데이터 패킷을 수신할 때 DSACK(RFC2883) 응답을 생성합니다 . 이러한 DSACK 정보를 수신해도 스택 동작에는 영향을 미치지 않지만 , 이 정보를 원격 발신자에게 제공하면 해당 발신자가 특정 네트워크 경로 동작에 더 잘 적응할 수 있습니다 (예 : Linux는 경로 왕복 시간 (RTT)의 급격한 증가로 인해 듀프스레시를 높이거나 허위 재전송을 감지할 수 있음).

Logging and Debugging

수십 년 동안 베이스 스택은 라이브 시스템에서 디버깅할 수 있는 여러 가지 메커니즘을 축적해 왔습니다 . 가장 잘 알려지지 않은 도구 중 하나인 trpt와 그 지원은 FreeBSD 14에서 제거되었습니다 . 하지만 여전히 수많은 다른 옵션이 존재합니다 (dtrace, siftr, bblog, ...).

블랙박스 로깅은 RACK 스택과 함께 도입되었고 점점 더 많은 기본 스택을 포함하도록 확장되었습니다 . 실행 중인 시스템에서 내부 상태 변화를 추출하고 패킷 추적 자체와 함께 코어 덤프에서 추출하는 도구가 준비되고 있습니다 . (https://github.com/Netflix/tcplog_dumper 및 https://github.com/Netflix/read_bbrlog 참조)

Cubic

이전 글에서 설명한 것처럼 , TCP 큐빅은 거의 모든 곳에서 사용되는 사실상 표준 혼잡 제어 알고리즘입니다 . 최근에는 어떤 TCP 스택을 사용하든 상관없이 FreeBSD의 기본값으로 큐빅이 지정되기도 했습니다 .

여기서 주목할 만한 확장 기능 중 하나는 HyStart++의 추가입니다 . TCP 세션이 시작되면 혼잡 제어 메커니즘은 슬로우 스타트라고 불리는 단계에서 대역폭을 빠르게 증가시킵니다 . 일반적으로 슬로우 스타트 단계는 혼잡 패킷 손실의 첫 징후 또는 명시적 혼잡 알림 (ECN) 피드백이 수신되면 종료됩니다 . 큐빅 모듈의 일부로 구현되어 항상 활성화되어 있는 HyStart++를 사용하면 RTT가 모니터링됩니다 . 네트워크 대기열이 형성되기 시작하여 RTT가 상승하기 시작하면 덜 공격적인 단계 (보수적인 슬로우 스타트)로 진입하고 타이밍 기반 신호는 안정적으로 확보하기 어렵기로 악명이 높기 때문에 RTT는 계속 모니터링됩니다 . 이 보수적

슬로우 스타트 단계에서 RTT가 다시 감소하면 일반 슬로우 스타트가 재개됩니다 . 그렇지 않은 경우 , CSS의 덜 공격적인 전송 속도는 소위 오버슈트 (불가피한 손실로 인해 복구해야 하는 데이터의 양)를 제한합니다 .

Accurate Explicit Congestion Notification

앞서 언급했듯이 ECN은 혼잡 이벤트를 나타내는 유일한 신호로서 패킷 손실을 방지하는 메커니즘입니다 . 지난 10년 동안 인터넷 엔지니어링 태스크포스 (IETF)에서는 이 시그널링을 개선하기 위해 많은 노력을 기울여 왔습니다 . 원래 ECN은 패킷 손실과 "동일한 " 신호로 간주되었지만 , 의미가 다른 더 빈번한 신호가 넓은 대역폭에서 얕은 (빠른) 대기열을 유지하는 데 더 효과적이라는 것이 밝혀졌습니다 . 전체 아키텍처의 이름은 저지연 , 저손실 , 확장성 (L4S)입니다 . 현재 FreeBSD의 모든 부분이 제대로 된 "TCP Prague" 구현을 구현할 준비가 되어 있지는 않지만 , DCTCP 혼잡 제어 모듈과 여기서는 정확도 높은 ECN(AccECN)과 같은 많은 개별 기능이 이제 FreeBSD 14에서 스택의 일부가 되었습니다 .

기존 ECN에서는 RTT당 하나의 혼잡 경험 마크만 시그널링할 수 있었습니다 . 따라서 혼잡 제어 모듈에 의한 집중적인 관리가 필요했습니다 . 실제로 CE 마크는 RFC3168 모드에서 작동하는 동안 TCP 대역폭을 조정할 때 패킷 손실 표시와 동일한 것으로 간주됩니다 . 이와는 대조적으로 , AccECN을 사용하면 수신자가 임의의 수의 명시적 혼잡 표시를 데이터 발신자에게 다시 보낼 수 있습니다 . 이를 통해 네트워크에서 보다 변조되고 세분화된 신호를 추출할 수 있습니다 . 이는 중간 스위치에 의해 훨씬 더 공격적인 마킹 임계값이 수정된 DCTCP를 사용해야 하는 환경에서 특히 유용합니다 . 또한 TCP Prague라고도 알려진 저지연 , 저손실 , 확장성 (L4S) 아키텍처의 핵심 요소 중 하나이기도 합니다 .

Authentication and Security

최근 RACK 스택은 TCP 패킷의 MD5 인증을 완벽하게 처리할 수 있는 기능을 갖추게 되었습니다 . 이는 RACK 스택에서 BGP를 사용할 수 있도록 개선된 기능으로 , RACK 스택이 모든 일반적인 상황에서 완전한 기능을 갖추고 사용할 수 있도록 하는 또 다른 단계입니다 .

오랫동안 RFC7323(RFC1323)의 두 가지 기능인 윈도우 스케일링과 타임스탬프 옵션 사이에는 긴밀한 결합이 있었습니다 . 이제 이 공간에서 두 기능 중 하나를 독립적으로 활성화할 수 있도록 허용하는 한편 , 기본값은 여전히 둘 다 활성화할 수 있도록 허용합니다 . 이제 net.inet.tcp.rfc1323을 켜기 (1) 또는 끄기 (0)뿐만 아니라 2(창 크기만) 및 3(타임스탬프만)으로 설정하여 이 기능을 구현할 수 있습니다 . 또한 RFC7323에 따라 이제 모든 상황에서 TCP 타임스탬프를 적절히 사용하도록 요구함으로써 TCP 세션을 더욱 안전하게 보호할 수 있습니다 . 이는 net.inet.tcp.tolerate_missing_ts를 0으로 설정하면 됩니다 .

What’s Next?

TCP 기능의 다양한 측면에 대한 개선이 반환 감소 단계에 접어들었지만 , 아직 몇 가지 추가 개선 사항이 논의 중입니다 .

예를 들어 , RFC2018(선택적 승인)의 부칙에 따라 이전과 달리 재전송 시간 초과 (RTO) 동안 정보를 보관할 수 있게 되었습니다 . 원래 표준 당시의 주된 동기는 수신자의 '리니게이팅 '을 허용하는 것이었습니다 . 명시적으로 승인하지 않는 한 , 메모리 압박 등의 이유로 후속 데이터를 버릴 수 있었습니다 . 실제로 이러한 리니게이팅은 거의 발생하지 않지만 , SACK 손실 복구 단계에서 재전송 시간 초과가 발생하는 경우는 꽤 자주 발생합니다 . 이 정보를 유지하면 RTO 이후에도 보다 효율적으로 재전송할 수 있습니다 . 문제는 기본 스택이 재전송 시간 초과 후 발생해야 하는 다른 측면 (예 : 매우 작은 혼잡 창에서 느리게 시작)과 암묵적으로 긴밀하게 연결되어 있다는 것입니다 . 또한 더미넷 경로 에뮬레이터에 몇 가지 추가 기능을 추가하여 손실을 보다 제어 가능한 방식으로 모델링하는 등 RTO 이후 이러한 변경의 영향을 평가해야 합니다 .

리차드 셰페네거는 2020년 4월부터 FreeBSD 커미터로 활동하고 있으며 , 주로 느린 경로 (손실 복구 , 혼잡 제어 처리)에 중점을 두고 TCP 스택의 기능을 개선하는 데 관심을 갖고 있으며 , IETF와 함께 정확한 ECN과 같은 개선 사항을 적극적으로 개발하고 있습니다 .

if_ovpn 또는 OpenVPN

원본 : *provost.pdf*

By Kristof Provost

오늘 [1]은 OpenVPN의 DCO에 대해 알아보겠습니다 [2].

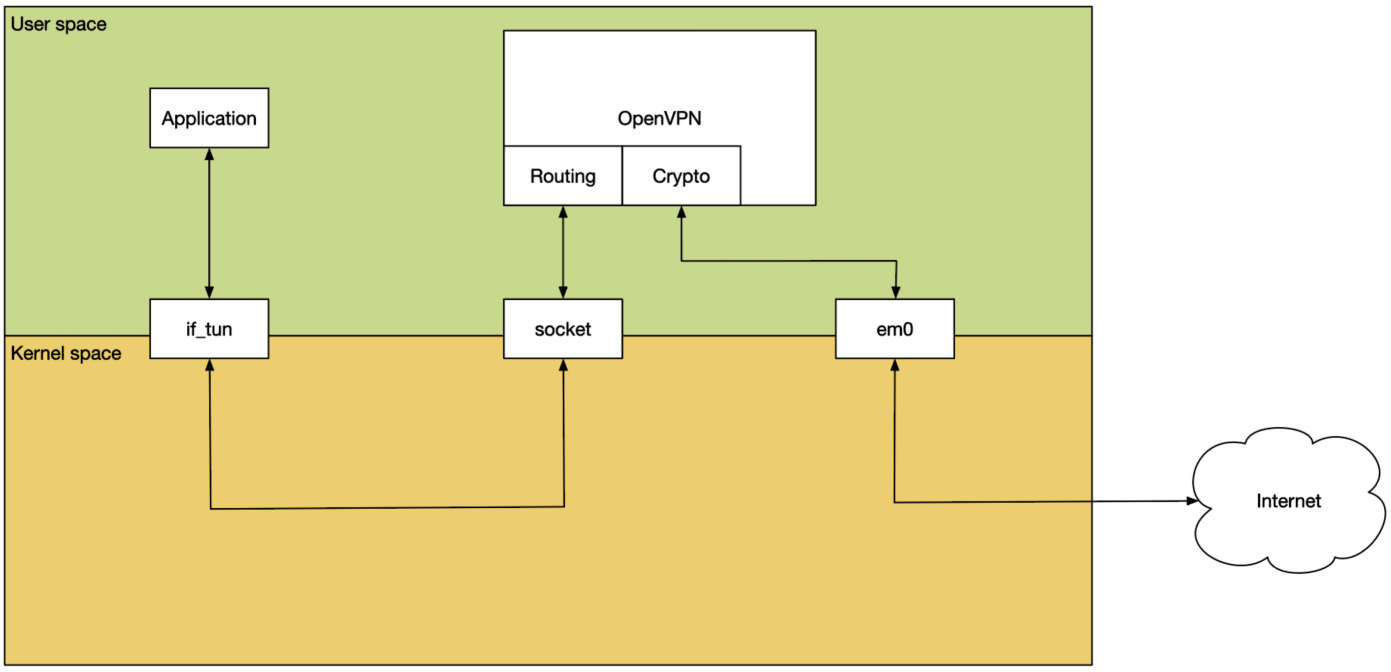
제임스 요난이 처음 개발한 OpenVPN은 2001년 5월 13일에 처음 출시되었습니다 . 이 서비스는 많은 일반적인 플랫폼 (예 : FreeBSD, OpenBSD, Dragonfly, AIX, ...)과 덜 일반적인 플랫폼 (macOS, Linux, Windows)도 지원합니다 . 사전 공유 키 , 인증서 또는 사용자 이름 /비밀번호 기반 인증으로 피어 투 피어 및 클라이언트 서버 모델을 지원합니다 .

20년 이상 된 프로젝트에서 기대할 수 있듯이 , 다양한 사용 사례를 위해 많은 기능이 추가되었습니다 .

문제점

OpenVPN은 매우 훌륭하지만 분명히 문제가 있습니다 . 문제가 없다면 이 글은 별로 흥미롭지 않을 것입니다 [3]. 실제로 문제가 하나 있는데 , 바로 OpenVPN이 단일 스레드의 사용자 공간 프로세스로 구현된다는 점입니다 .

이 프로세스는 if_tun을 사용하여 네트워크 스택에 패킷을 주입합니다 . 그 결과 성능이 현재의 연결 속도를 따라가지 못합니다 . 또한 최신 멀티코어 하드웨어나 암호화 오프로드 하드웨어를 활용하기 어렵습니다 .



OpenVPN 성능의 주요 문제는 사용자 공간 특성입니다 . 들어오는 트래픽은 일반적으로 커널 메모리로 패킷을 DMA하는 NIC에 의해 자연스럽게 수신됩니다 . 그런 다음 네트워크 스택에서 패킷이 속한 소켓을 파악하여 사용자 공간으로 전달할 때까지 추가 처리를 거칩니다 . 이 소켓은 UDP 또는 TCP일 수 있습니다 .

사용자 공간으로 패킷을 전달하려면 패킷을 복사해야 하며 , 이때 사용자 공간 OpenVPN 프로세스는 패킷을 확인하고 복호화하여 if_tun을 사용하여 네트워크 스택에 다시 삽입합니다 . 이는 추가 처리를 위해 일반 텍스트 패킷을 커널로 다시 복사하는 것을 의미합니다 .

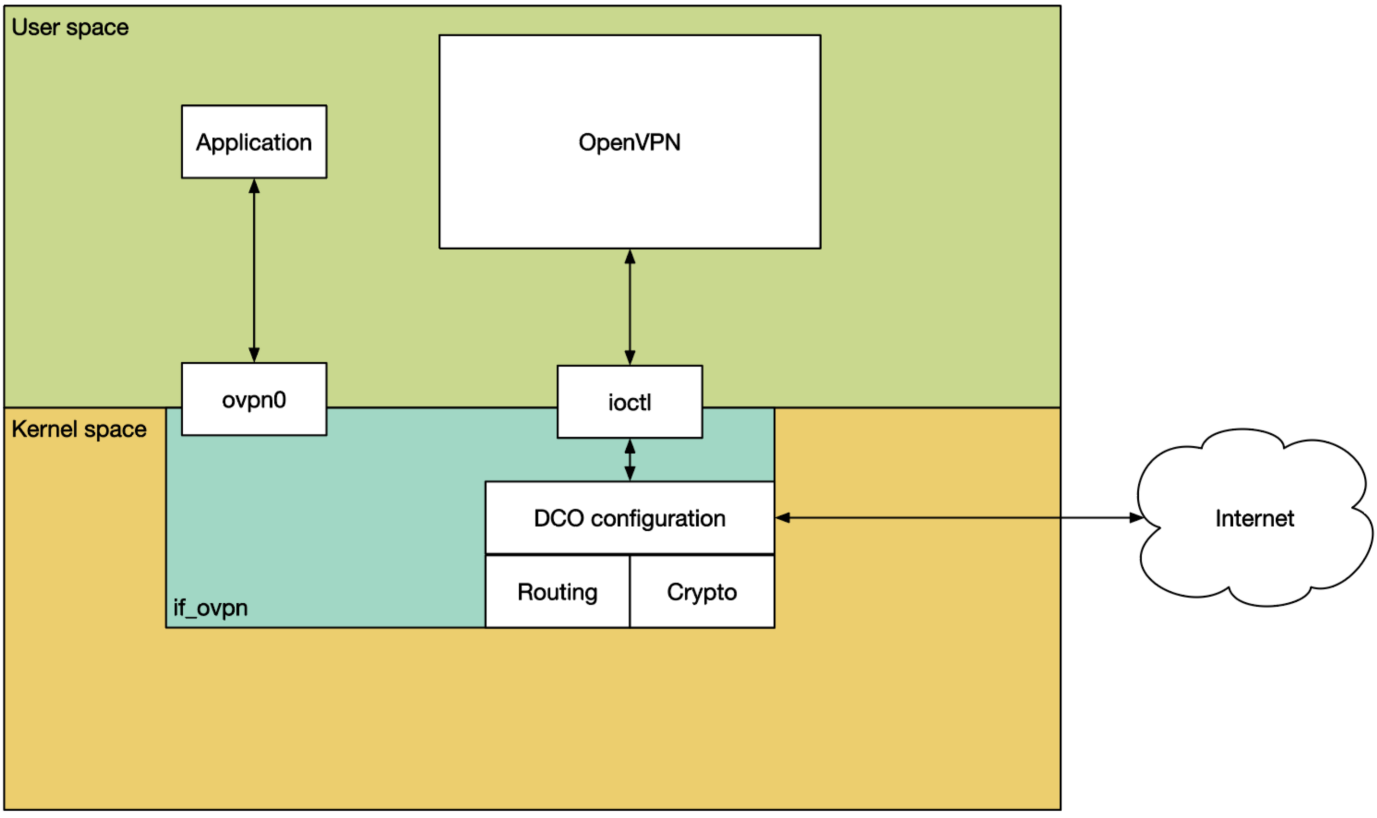
이러한 모든 컨텍스트 전환과 복사는 필연적으로 성능에 상당한 영향을 미칩니다 .

현재 아키텍처에서는 성능을 크게 개선하기가 매우 어렵습니다 .

DCO란 ?

이제 문제가 무엇인지 확인했으니 해결책을 생각해 볼 수 있습니다 [4].

컨텍스트가 사용자 공간으로 전환되는 것이 문제라면 작업을 커널 내부에 유지하는 것이 그럴듯한 해결책 중 하나이며 , 이것이 바로 DCO(데이터 채널 오프로드)가 하는 일입니다 .



DCO는 데이터 채널 , 즉 암호화 작업과 트래픽 터널링을 커널로 이동합니다 . 이 작업은 새로운 가상 장치 드라이버인 if_ovpn을 통해 수행됩니다 . OpenVPN 사용자 공간 프로세스는 여전히 연결 설정 (인증 및 옵션 협상 포함)을 담당하며 , 새로운 ioctl 인터페이스를 통해 if_ovpn 드라이버와 조정합니다 .

OpenVPN [illegible] DCO[illegible] . [illegible] . AES-GCM [illegible] ChaCha20/Poly1305 [illegible] . [illegible] . [illegible] DCO[illegible] , [illegible] 2 [illegible] , [illegible] [5] [illegible] .

[illegible] DCO[illegible] OpenVPN [illegible] . [illegible] . [illegible] , [illegible] .

[illegible]

[illegible] . [illegible] ? [illegible] !

[illegible] :

Multiplexing

[illegible] OpenVPN[illegible] . [illegible] OpenVPN [illegible] .

[illegible] . [illegible] OpenVPN [illegible] . [illegible] . [illegible] ([illegible] , if_ovpn)[illegible] .

[illegible] , [illegible] if_ovpn[illegible] . [illegible] . OpenVPN [illegible] , [illegible] . [illegible] .

[illegible] , ovpn_new_peer()[6] [illegible] .

[illegible] udp_set_kernel_tunneling()[illegible] . [illegible] ovpn_udp_input()[illegible] OpenVPN[illegible] .

[illegible] . [illegible] . [illegible] https://cgit.freebsd.org/src/commit/?id=742e7210d00b359d81b9c778ab520003704e9b6c [illegible] .

ovpn_udp_input() 함수는 수신 경로의 주요 진입점입니다 . 네트워크 스택은 설치된 소켓에 도착하는 모든 UDP 패킷에 대해 이 함수에 패킷을 넘깁니다 .

이 함수는 먼저 패킷이 커널 드라이버에서 처리될 수 있는지 확인합니다 . 즉 , 패킷이 데이터 패킷이고 알려진 피어 ID로 향하는 패킷인지 확인합니다 . 그렇지 않은 경우 필터 함수는 필터 기능이 없는 것처럼 패킷을 정상적인 흐름으로 통과시키도록 UDP 코드에 지시합니다 . 즉 , 패킷이 소켓에 도착하여 OpenVPN의 사용자 공간 프로세스에 의해 처리됩니다 .

초기 버전의 DCO 드라이버에는 제어 메시지를 읽고 쓰기 위한 별도의 ioctl 명령이 있었지만 , Linux 및 FreeBSD 드라이버는 모두 소켓을 대신 사용하도록 조정되었습니다 . 이렇게 하면 제어 패킷과 새 클라이언트의 처리가 모두 간소화됩니다 .

반면에 패킷이 알려진 피어에 대한 데이터 패킷인 경우에는 해독하고 서명의 유효성을 검사한 다음 추가 처리를 위해 네트워크 스택으로 전달합니다 .

자세한 내용은 https://cgit.freebsd.org/src/tree/sys/net/if_ovpn.c?id=da69782bf06645f38852a8b23af#n1483 에서 확인할 수 있습니다 .

UDP

OpenVPN은 UDP와 TCP 모두에서 실행할 수 있습니다 . UDP는 레이어 3 VPN 프로토콜을 위한 당연한 선택이지만 , 일부 사용자는 방화벽을 통과하기 위해 TCP를 통해 실행해야 합니다 .

FreeBSD 커널은 UDP 소켓을 위한 편리한 필터 기능을 제공하지만 , TCP에 해당하는 기능이 없기 때문에 FreeBSD if_ovpn은 현재 UDP만 지원하고 TCP는 지원하지 않습니다 .

리눅스 DCO 드라이버 개발자는 용기를 내어 TCP 지원도 구현하기로 결정했습니다 . 이 개발자는 예상과 달리 실제로 이 경험에서 살아남았고 지금은 훨씬 더 현명해졌습니다 .

Hardware Cryptography Offload

if_ovpn은 암호화 작업을 위해 커널 내 OpenCrypto 프레임워크에 의존합니다 . 즉 , 시스템에 존재하는 모든 암호화 오프로드 하드웨어를 활용할 수도 있습니다 . 이를 통해 성능을 더욱 향상시킬 수 있습니다 .

이미 인텔의 퀵어시스트 기술 (QAT), SafeXcel EIP-97 암호화 가속기 및 AES-NI로 테스트되었습니다 .

Locking Design

잠금에 대해 이야기할 필요 없이 커널 코드에 대해 논의할 수 있을 거라고 생각했다면 무슨 말을 해야 할지 모르겠네요 . 그건 순진하게 낙관적인 생각이었죠 .

CPU , .
, .

if_ovpn ( ). .

.

.

"" "" , , counter(9) . CPU .

Control Interface

OpenVPN DCO OpenVPN .

Linux netlink , if_ovpn FreeBSD netlink .

if_ovpn ioctl . SIOCSDRVSPEC/SIOCGDRVSPEC .

ifdrv . ifd_cmd , ifd_data ifd_len .

if_ovpn nvlist . , . . [7] .

nvlist . .

Routing Lookups

if_ovpn은 라우팅 결정에 대해 걱정할 필요가 없다고 생각할 수 있습니다 . 패킷이 네트워크 드라이버에 도착할 때쯤이면 커널의 네트워크 스택이 이미 라우팅 결정을 내렸을 테니까요 . 틀린 생각입니다 . 저도 이 사실을 알아내는 데 시간이 좀 걸렸습니다 .

문제는 주어진 if_ovpn 인터페이스에 잠재적으로 여러 개의 피어가 있을 수 있다는 것입니다 (예 : 서버 역할을 하고 있고 여러 클라이언트가 있는 경우). 커널은 문제의 패킷이 이들 중 한 곳으로 가야 한다는 것을 알아냈지만 , 커널은 이러한 모든 클라이언트가 단일 브로드캐스트 도메인에 있다는 가정 하에 작동합니다 . 즉 , 인터페이스에서 전송된 패킷은 모든 클라이언트가 볼 수 있습니다 . 여기서는 그렇지 않으므로 if_ovpn은 패킷이 어느 클라이언트로 가야 하는지 알아내야 합니다 .

이 작업은 ovpn_route_peer()가 처리합니다 . 이 함수는 먼저 피어 목록을 살펴보고 피어의 VPN 주소가 목적지 주소와 일치하는지 확인합니다 . (주소 패밀리에 따라 ovpn_find_peer_by_ip() 또는 ovpn_find_peer_by_ip6()에 의해 수행됩니다). 일치하는 피어를 찾으면 이 피어에게 패킷이 전송됩니다 . 그렇지 않은 경우 ovpn_route_peer()는 경로 조회를 수행하고 결과 게이트웨이 주소로 피어 조회를 반복합니다 .

if_ovpn이 패킷을 전송할 피어를 찾은 경우에만 패킷을 암호화하여 전송할 수 있습니다 .

Key Rotation

OpenVPN은 때때로 터널을 보호하는 데 사용되는 키를 변경합니다 . 이는 if_ovpn이 사용자 공간에 맡기는 어려운 작업 중 하나이므로 OpenVPN과 if_ovpn 간의 약간의 조정이 필요합니다 .

OpenVPN은 OVPN_NEW_KEY 명령을 사용하여 새 키를 설치합니다 . 각 키에는 ID가 있으며 , 모든 패킷에는 암호화에 사용된 키 ID가 포함됩니다 . 즉 , 키가 교체되는 동안에도 이전 키와 새 키가 모두 커널에 알려져 있고 활성 상태로 유지되므로 모든 패킷을 해독할 수 있습니다 .

새 키가 설치되면 OVPN_SWAP_KEYS 명령을 사용하여 활성화할 수 있습니다 . 즉 , 새 키는 발신 패킷을 암호화하는 데 사용됩니다 .

나중에 OVPN_DEL_KEY 명령을 사용하여 이전 키를 삭제할 수 있습니다 .

vnet

네 , 브이넷에 대해 이야기해야 할 것 같습니다 . 이 글을 쓰다 보니 어쩔 수 없는 일이죠 .

제가 너무 게을러서 전부 설명하기는 어렵기 때문에 훨씬 더 뛰어난 저자인 올리비에 코샤르 -라베가 쓴 "감옥 : 사례로 보는 vnet"[8]이라는 글을 소개해드리겠습니다 .

vnet은 감옥을 자체 IP 스택을 갖춘 가상 머신으로 전환하는 것으로 생각하면 됩니다 .

pfSense 사용 사례에 반드시 필요한 것은 아니지만 테스트를 훨씬 더 쉽게 해줍니다 . 즉 , 외부 도구 없이도 단일 머신에서 테스트할 수 있다는 뜻입니다 (OpenVPN 자체는 매우 당연한 이유 때문에 제외).

이 방법이 궁금하신 분들께는 FreeBSD 저널의 다른 기사도 도움이 될 수 있습니다 : "자동화된 테스트 프레임워크 ", 작성자 ... 잠깐만요 , 크리스토프 프로보스트라는 사람을 알 것 같습니다 .

Performance

이 모든 과정을 거치고 나면 "그래도 정말 도움이 될까 ?"라는 의문이 드실 겁니다 .

다행히도 제게는 그렇습니다 .

넷게이트의 동료 중 한 명이 넷게이트 4100^{10} 디바이스에 iperf3을 설치하여 테스트한 결과 다음과 같은 결과를 얻었습니다 :

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"은 DCO가 없는 기존 OpenVPN 방식입니다 . 사용자 공간에서 AES-NI 명령어를 사용했고 'DCO 소프트웨어 ' 설정은 사용하지 않았다는 점에 주목할 필요가 있습니다 . 이러한 노골적인 치팅 시도에도 불구하고 DCO는 여전히 약간 더 빨랐습니다 . 공평한 경쟁의 장 (즉 , DCO가 AES-NI 명령어를 사용하는 경우) 에서는 경쟁이 없습니다 . DCO가 3배 이상 빠릅니다 .

인텔에게도 좋은 소식이 있습니다 : 인텔의 QuickAssist 오프로드 엔진은 AES-NI보다 훨씬 빨라서 OpenVPN 이 이전보다 5배 더 빨라졌습니다 .

Future Work

개선할 수 없을 정도로 좋은 것은 없지만 , 어떤 면에서 이번 개선은 DCO 설계의 성공으로 인한 결과입니다 . 유선 OpenVPN 프로토콜은 32비트 초기화 벡터 (IV)를 사용하며 , 여기서는 설명하지 않겠습니다 [11], 암호화상의 이유로 동일한 키로 IV를 재사용하는 것은 좋지 않은 생각입니다 .

즉 , 키를 재협상해야 한다는 뜻입니다 . OpenVPN의 기본 재협상 간격은 3600초이며 , 안전을 위해 30%의 여유를 두면 2^32 * 0.7 / 3600, 즉 초당 약 835.000개의 패킷이 전송됩니다 . 이는 "겨우 " 8~9Gbit/s입니다 (1300바이트 패킷 가정).

DCO를 사용하면 최신 하드웨어로는 이미 어느 정도 도달할 수 있는 수준입니다 .

좋은 문제이긴 하지만 , 여전히 문제이기 때문에 OpenVPN 개발자들은 64비트 IV를 사용하는 패킷 포맷을 업데이트하기 위해 노력하고 있습니다 .

Thanks

if_ovpn 작업은 Rubicon Communications(Netgate로 거래)의 후원으로 pfSense 제품 라인에 사용되었습니다 . 22.05 pfSense plus 릴리스 [12]부터 사용되었습니다 . 이 작업은 FreeBSD로 업스트림되었으며 최근 14.0 릴리스의 일부입니다 . 사용하려면 OpenVPN 2.6.0 이상이 필요합니다 .

또한 , 초기 FreeBSD 패치가 나왔을 때 매우 환영해 주셨고 도움이 없었다면 이 프로젝트가 이만큼 잘 진행되지 못했을 OpenVPN 개발자들에게도 감사의 말씀을 전하고 싶습니다 .

Footnotes:

1. 아니면 이 글을 읽을 때마다 .
2. 좋아요 글쓰기 . 읽고 이봐요 , 현학적으로 얘기할 거면 하루 종일 이 얘기를 할 거예요 .
3. DCO에 관심이 없다면 다음 기사를 읽으면 돼요 . 아주 좋은 글일 겁니다 .
4. 제가 "저희 "라고 말했지만 , 이 솔루션에 대한 공로를 인정받고 싶은 만큼 DCO 아키텍처를 고안하고 Windows와 Linux용으로 구현한 것은 OpenVPN 개발자들입니다 . 저는 그들이 한 일만 했을 뿐입니다 . 다만 FreeBSD용입니다 .
5. OpenVPN에서는 DCO는 OS의 트래픽 쉐이핑 (즉 , 더미넷)과 결합할 수 있습니다 .
6. https://cgit.freebsd.org/src/tree/sys/net/if_ovpn.c?id=da69782bf06645f38852a8b23af#n490
7. 구조가 뚱뚱해졌기 때문이라고 말할 수도 있습니다 . 그럴 수도 있습니다만 , 전 너무 예의가 없습니다 .
8. https://freebsdfoundation.org/wp-content/uploads/2020/03/Jail-vnet-by-Examples.pdf
9. https://freebsdfoundation.org/wp-content/uploads/2019/05/The-Automated-Testing-Framework.pdf
10. https://shop.netgate.com/products/4100-base-pfsense
11. 대부분 제가 직접 이해하지 못하기 때문입니다 .
12.

> https://www.netgate.com/blog/pfsense-plus-software-version-22.05-now-available

---

크리스토프 프로보스트는 네트워크 및 비디오 애플리케이션을 전문으로 하는 프리랜서 임베디드 소프트웨어 엔지니어입니다 . 그는 FreeBSD 커미터이자 FreeBSD의 pf 방화벽 유지 관리자입니다 . 그는 현재 대부분의 시간을 Netgate의 pfSense 작업에 할애하고 있습니다 .

크리스토프는 안타깝게도 uClibc 버그에 걸려 넘어지는 경향이 있고 , FTP에 대한 증오심이 강합니다 . 그에게 IPv6 조각화에 대해 이야기하지 마세요 .