스트림:	인터넷 엔지니어링 태스크포스(IETF)	
RFC:	9002	
카테고리:	표준 트랙	
게시됨:2021년	5월	
ISSN:	2070-1721	

저자:	J. Iyengar, Ed.	I. Swett, Ed.
	빠르게	*Google*

RFC 9002
QUIC 손실 감지 및 혼잡 제어

초록

이 문서에서는 QUIC의 손실 감지 및 혼잡 제어 메커니즘에 대해 설명합니다.

이 메모의 상태

이 문서는 인터넷 표준 트랙 문서입니다.

이 문서는 인터넷 엔지니어링 태스크포스(IETF)의 산물입니다. 이 문서는 IETF 커뮤니티의 합의를 나타냅니다. 이 문서는 공개 검토를 거쳤으며 인터넷 엔지니어링 운영 그룹(IESG)의 게시 승인을 받았습니다. 인터넷 표준에 대한 자세한 내용은 RFC 7841의 섹션 2에서 확인할 수 있습니다.

이 문서의 현재 상태, 오류 및 피드백 제공 방법( )에 대한 정보는 https://www.rfc-editor.org/info/rfc9002 에서 확인할 수 있습니다.

저작권 고지

목차

1. 소개

QUIC은 안전한 범용 전송 프로토콜로, [QUIC-TRANSPORT]에 설명되어 있습니다. 이 문서에서는 QUIC의 손실 감지 및 혼잡 제어 메커니즘에 대해 설명합니다.

2. 규칙 및 정의

이 문서에서 "**반드시**", "**반드시 하지 말아야 함**", "**필수**", "**해야** 함", "하지 **말아야 함**", "**권장**", "**권장하지 않음**", "할 **수 있음**" 및 "**선택적**"이라는 키워드는 다음과 같은 용도로 사용됩니다.
는 여기에 표시된 것처럼 모든 대문자로 표시되는 경우에만 BCP 14 [RFC2119] [RFC8174]에 설명된 대로 해석

해야 합니다.

이 문서에서 사용되는 용어의 정의입니다:

Ack 유도 프레임: ACK, 패딩, 연결_닫기를 제외한 모든 프레임은 응답을 유도하는 것으로 간주됩니다.

응답 유도 패킷: 응답 유도 프레임이 포함된 패킷은 수신자로부터 응답(ACK)을 유도합니다.
을 최대 승인 지연 시간 이내로 설정하고 이를 응답 유도 패킷이라고 합니다.

비행 중 패킷: 패킷이 응답을 유도하거나 다음을 포함하는 경우 비행 중으로 간주됩니다.
패딩 프레임으로 보내졌지만 받지 못했거나 분실 신고되었거나 기존 키와 함께 폐기된 경우입니다.

3. QUIC 변속기 설계

QUIC의 모든 전송은 암호화 수준을 나타내는 패킷 수준 헤더와 함께 전송되며, 여기에는 패킷 시퀀스 번호(이하 패킷 번호라고 함)가 포함됩니다. 암호화 수준은 [QUIC- 전송] 섹션 12.3에 설명된 대로 패킷 번호 공간을 나타냅니다. 패킷 번호는 연결이 유지되는 동안 패킷 번호 공간 내에서 반복되지 않습니다. 패킷 번호는 한 공간 내에서 단조롭게 증가하는 순서로 전송되어 모호성을 방지합니다. 일부 패킷 번호는 의도적으로 간격을 두고 사용하지 않는 것이 허용됩니다.

이 설계는 전송과 재전송을 구분할 필요가 없으므로 QUIC의 TCP 손실 감지 메커니즘 해석에 따른 상당한 복잡성을 제거합니다.

QUIC 패킷은 다양한 유형의 여러 프레임을 포함할 수 있습니다. 복구 메커니즘은 안정적인 전송이 필요한 데이터와 프레임을 확인하거나 손실된 것으로 선언하고 필요에 따라 새 패킷으로 전송합니다. 패킷에 포함된 프레임 유형은 복구 및 혼잡 제어 로직에 영향을 미칩니다:

- 모든 패킷이 승인되지만, 응답을 유도하는 프레임이 없는 패킷은 응답을 유도하는 패킷과 함께만 승인됩니다.
- 암호화 프레임이 포함된 긴 헤더 패킷은 QUIC 핸드셰이크의 성능에 중요하며, 승인을 위해 더 짧은 타이머를 사용합니다.
- ACK 또는 CONNECTION_CLOSE 프레임 외에 프레임이 포함된 패킷은 혼잡 제어 제한에 포함되며 비행 중인 것으로 간주됩니다.
- 패딩 프레임은 패킷이 직접적으로 확인을 전송하지 않고도 전송 중 바이트에 기여하도록 합니다.

4. QUIC과 TCP의 관련 차이점

TCP의 손실 감지 및 혼잡 제어에 익숙한 독자라면 잘 알려진 TCP 알고리즘과 유사한 알고리즘을 찾을 수 있습니다. 그러나 QUIC과 TCP 간의 프로토콜 차이로 인해 알고리즘에 차이가 있습니다. 이러한 프로토콜 차이점은 아래에 간략하게 설명되어 있습니다.

4.1. 패킷 번호 공백 분리

QUIC은 각 암호화 수준에 대해 별도의 패킷 번호 공간을 사용하지만, 0-RTT와 모든 세대의 1-RTT 키는 동일한 패킷 번호 공간을 사용합니다. 패킷 번호 공간을 분리하면 한 수준의 암호화로 전송된 패킷의 승인으로 인해 다른 암호화 수준으로 전송된 패킷의 허위 재전송이 발생하지 않습니다. 패킷 번호 공간에서 혼잡 제어 및 RTT(왕복 시간) 측정이 통합됩니다.

4.2. 단조롭게 증가하는 패킷 수

TCP는 발신자의 전송 순서와 수신자의 전달 순서를 혼동하여 재전송 모호성 문제[재전송]를 일으킵니다. QUIC은 전송 순서와 배달 순서를 분리합니다. 패킷 번호는 전송 순서를 나타내며, 배달 순서는 스트림 프레임의 스트림 오프셋에 의해 결정됩니다.

QUIC의 패킷 번호는 패킷 번호 공간 내에서 엄격하게 증가하며 전송 순서를 직접 인코딩합니다. 패킷 번호가 높을수록 패킷이 나중에 전송되었음을 의미하고, 패킷 번호가 낮을수록 패킷이 더 일찍 전송되었음을 의미합니다. 응답을 유도하는 프레임이 포함된 패킷이 손실된 것으로 감지되면 QUIC은 새로운 패킷 번호를 가진 새 패킷에 필요한 프레임을 포함시켜 ACK 수신 시 어떤 패킷을 인정할지 모호성을 제거합니다.
따라서 보다 정확한 RTT 측정이 가능하고, 허위 재전송을 쉽게 감지할 수 있으며, 패킷 번호만을 기반으로 고속 재전송과 같은 메커니즘을 보편적으로 적용할 수 있습니다.

이 설계 포인트는 QUIC의 손실 감지 메커니즘을 크게 간소화합니다. 대부분의 TCP 메커니즘은 암묵적으로 TCP 시퀀스 번호를 기반으로 전송 순서를 추론하려고 시도하는데( ), 특히 TCP 타임스탬프를 사용할 수 없는 경우 이 작업은 쉽지 않은 일입니다.

4.3. 더 명확한 손실 에포크

패킷이 손실되면 QUIC은 손실 에포크를 시작합니다. 손실 에포크는 에포크가 시작된 후 전송된 패킷이 확인되면 종료됩니다. TCP는 시퀀스 번호 공간의 간격이 채워질 때까지 기다리므로 세그먼트가 연속으로 여러 번 손실되는 경우 손실 에포크가 여러 번 왕복하는 동안 종료되지 않을 수 있습니다. 둘 다 에포크당 한 번만 혼잡 기간을 줄여야 하므로, QUIC은 손실이 발생하는 모든 왕복에 대해 한 번만 감소시키는 반면, TCP는 여러 왕복에 걸쳐 한 번만 감소시킬 수 있습니다.

4.4. 리네깅 금지

QUIC ACK 프레임에는 TCP 선택적 승인(SACK) [RFC2018]의 정보와 유사한 정보가 포함되어 있습니다. 그러나 QUIC은 패킷 확인을 거부할 수 없으므로 양측의 구현이 크게 간소화되고 발신자의 메모리 부담이 줄어듭니다.

4.5. 더 많은 ACK 범위

QUIC은 TCP의 세 가지 SACK 범위와 달리 많은 ACK 범위를 지원합니다. 손실률이 높은 환경에서 복구 속도를 높이고, 허위 재전송을 줄이며, 시간 초과에 의존하지 않고 앞으로 나아갈 수 있도록 보장합니다.

4.6. 지연된 승인에 대한 명시적 수정

QUIC 엔드포인트는 패킷이 수신된 시점과 해당 확인이 전송된 시점 사이에 발생하는 지연을 측정하여 피어가 보다 정확한 RTT 추정치를 유지할 수 있도록 합니다([QUIC-TRANSPORT] 섹션 13.2 참조).

4.7. 프로브 타임아웃이 RTO 및 TLP를 대체합니다.

QUIC은 프로브 타임아웃(PTO, 6.2절 참조)을 사용하며, 타이머는 TCP의 재전송 타임아웃(RTO) 계산을 기반으로 합니다; [RFC6298] 참조. QUIC의 PTO는 고정된 최소 타임아웃을 사용하는 대신 피어의 최대 예상 승인 지연을 포함합니다.

TCP [RFC8985]의 RACK-TLP 손실 감지 알고리즘과 유사하게, QUIC은 꼬리 부분에서의 단일 패킷 손실이 지속적인 정체를 나타내지 않기 때문에 PTO가 만료될 때 혼잡 창을 축소하지 않습니다. 대신, QUIC은 지속적인 혼잡이 선언되면 혼잡 창을 축소합니다(섹션 7.6 참조). 이렇게 함으로써 QUIC은 불필요한 혼잡 윈도우 축소를 방지하여 순방향 RTO 복구(F- RTO) [RFC5682]와 같은 수정 메커니즘의 필요성을 없애줍니다. QUIC은 PTO 만료 시 혼잡 윈도우를 축소하지 않으므로, PTO 만료 후에도 사용 가능한 혼잡 윈도우가 남아 있는 경우 QUIC 발신자는 더 많은 기내 패킷을 전송하는 데 제한을 받지 않습니다. 이는 발신자가 애플리케이션이 제한되어 있고 PTO 타이머가 만료된 경우에 발생합니다. 이는 애플리케이션 제한이 있는 경우 TCP의 RTO 메커니즘보다 더 공격적이지만 애플리케이션 제한이 없는 경우와 동일합니다.

QUIC은 타이머( )가 만료될 때마다 프로브 패킷이 일시적으로 혼잡 창을 초과할 수 있도록 허용합니다.

4.8. 최소 혼잡 구간은 두 패킷입니다.

TCP는 패킷 한 개의 최소 혼잡 구간을 사용합니다. 그러나 이 단일 패킷이 손실되면 발신자는 PTO가 복구될 때까지 기다려야 하며(섹션 6.2), 이는 RTT보다 훨씬 더 길어질 수 있습니다. 또한 수신자가 수신 확인을 지연할 때 하나의 응답 유도 패킷을 전송하면 추가 지연 시간이 발생할 가능성이 높아집니다.

따라서 QUIC에서는 최소 혼잡 구간을 두 패킷으로 권장합니다. 이렇게 하면 네트워크 부하가 증가하지만, 지속적인 혼잡 상황에서도 발신자가 전송 속도를 기하급수적으로 줄일 수 있으므로 안전한 것으로 간주됩니다(섹션 6.2).

4.9. 핸드셰이크 패킷은 특별하지 않습니다

TCP는 SYN 또는 SYN-ACK 패킷의 손실을 지속적인 혼잡으로 처리하고 혼잡 창을 하나의 패킷으로 줄입니다([RFC5681] 참조). QUIC은 핸드셰이크 데이터가 포함된 패킷의 손실을 다른 손실과 동일하게 처리합니다.

5. 왕복 시간 예상하기

높은 수준에서 엔드포인트는 패킷이 전송된 시점부터 RTT 샘플로 인식될 때까지의 시간을 측정합니다. 엔드포인트는 RTT 샘플과 피어 보고 호스트 지연([QUIC-TRANSPORT] 섹션 13.2 참조)을 사용하여 네트워크 경로의 RTT에 대한 통계적 설명을 생성합니다. 엔드포인트는 각 경로에 대해 다음 세 가지 값을 계산합니다: 일정 기간 동안의 최소값(min_rtt), 지수 가중 이동 평균(smoothed_rtt), 관찰된 RTT 샘플의 평균 편차(이 문서의 나머지 부분에서는 "변동"이라 함)(rttvar)입니다.

5.1. RTT 샘플 생성

엔드포인트는 다음 두 가지 조건을 충족하는 ACK 프레임을 수신하면 RTT 샘플을 생성합니다:

- 가장 큰 승인 패킷 번호가 새로 승인되고
- 새로 인식된 패킷 중 하나 이상이 응답을 유도하는 패킷이었습니다.

가장 큰 승인 패킷이 전송된 후 경과한 시간에 따라 RTT 샘플인 latest_rtt가 생성됩니다:

```
latest_rtt = ack_time - send_time_of_largest_acked
```

RTT 샘플은 수신된 ACK 프레임에서 가장 큰 승인 패킷만을 사용하여 생성됩니다. 이는 피어가 ACK 프레임에서 가장 큰 승인 패킷에 대해서만 승인 지연을 보고하기 때문입니다. 보고된 승인 지연은 RTT 샘플 측정에 사용되지는 않지만, 후속 smoothed_rtt 및 rttvar 계산에서 RTT 샘플을 조정하는 데 사용됩니다(5.3절).

단일 패킷에 대해 여러 RTT 샘플을 생성하지 않으려면, 가장 큰 승인 패킷을 새로 승인하지 않는 경우 RTT 추정치를 업데이트하는 데 ACK 프레임을 사용해서는 **안 됩니다.**

적어도 하나의 응답 유도 패킷을 새로 인식하지 않는 ACK 프레임을 수신할 때 RTT 샘플을 생성해서는 **안 됩니다.** 피어는 일반적으로 응답을 유도하지 않는 패킷만 수신하면 ACK 프레임을 보내지 않습니다. 따라서 응답을 유발하지 않는 패킷에 대한 확인만 포함된 ACK 프레임에는 임의로 큰 ACK 지연 값이 포함될 수 있습니다. 이러한 ACK 프레임을 무시하면 이후의 smoothed_rtt 및 rttvar 계산에서 복잡한 문제를 피할 수 있습니다.

발신자는 RTT 내에서 여러 개의 ACK 프레임이 수신될 때 RTT당 여러 개의 RTT 샘플을 생성할 수 있습니다. RFC6298]에서 제안한 바와 같이, 이렇게 하면 smoothed_rtt 및 rttvar의 히스토리가 부적절해질 수 있습니다. RTT 추정치가 충분한 히스토리를 유지하도록 하는 것은 현재 진행 중인 연구 질문입니다.

5.2. min_rtt 추정

min_rtt는 일정 기간 동안 주어진 네트워크 경로에서 관찰된 최소 RTT에 대한 발신자의 추정치입니다. 이 문서에서는 손실 감지에서 믿을 수 없을 정도로 작은 RTT 샘플을 거부하기 위해 min_rtt를 사용합니다.

min_rtt는 첫 번째 RTT 샘플에서 최신_rtt로 설정해야 합니다. 다른 모든 샘플에서는 min_rtt와 최신_rtt 중 작은 값으로 설정해야 합니다(섹션 5.1).

엔드포인트는 로컬에서 관측된 시간만 사용하여 최소_rtt를 계산하고 피어에서 보고한 승인 지연을 조정하지 않습니다. 이렇게 하면 엔드포인트가 전적으로 관찰한 것을 기반으로 smoothed_rtt의 하한을 설정할 수 있으며(섹션 5.3 참조), 피어에서 잘못 보고한 지연으로 인한 잠재적인 과소 평가를 제한할 수 있습니다.

네트워크 경로의 RTT는 시간이 지남에 따라 변경될 수 있습니다. 경로의 실제 RTT가 감소하는 경우, 첫 번째 낮은 샘플에 대해 min_rtt가 즉시 조정됩니다. 그러나 경로의 실제 RTT가 증가하면 min_rtt는 이에 적응하지 않으므로 새로운 RTT보다 작은 미래의 RTT 샘플이 smoothed_rtt에 포함될 수 있습니다.

엔드포인트는 지속적인 혼잡이 발생한 후 min_rtt를 최신 RTT 샘플로 설정해야 합니다. 이렇게 하면 RTT가 증가할 때 지속적 정체를 반복적으로 선언하는 것을 방지할 수 있습니다. 또한 네트워크 중단 이벤트가 발생한 후 연결이 min_rtt 및 smoothed_rtt의 추정치를 재설정할 수 있습니다(5.3절 참조).

엔드포인트는 트래픽 볼륨이 낮고 승인 지연이 짧은 상태에서 승인을 수신하는 경우와 같이 연결의 다른 시간에 min_rtt를 다시 설정할 **수 있습니다**.
경로의 실제 최소 RTT 를 자주 관찰할 수 없으므로 구현에서는 min_rtt 값을 너무 자주 새로 고치지 **않아야 합니다**.

5.3. smoothed_rtt 및 rttvar 추정하기

smoothed_rtt는 엔드포인트의 RTT 샘플에 기하급수적으로 가중치를 부여한 이동 평균이며, rttvar는 평균 변동을 사용하여 RTT 샘플의 변동을 추정합니다.

smoothed_rtt 계산은 승인 지연에 맞게 조정한 후 RTT 샘플을 사용합니다. 이러한 지연은 섹션에 설명된 대로 ACK 프레임의 ACK 지연 필드에서 디코딩됩니다.
퀵-트랜스포트] 19.3.

피어는 핸드셰이크 중에 피어의 max_ack_delay보다 큰 승인 지연을 보고할 수 있습니다([QUIC-

TRANSPORT] 섹션 13.2.1). 이를 고려하기 위해, 엔드포인트는 핸드셰이크가 확인될 때까지 최대 _ack_delay를 무시해야 합니다(섹션 4.1.2에 정의된 대로).

[QUIC-TLS]. 이러한 큰 승인 지연이 발생하면 반복되지 않고 핸드셰이크에 국한될 가능성이 높습니다. 따라서 엔드포인트는 최대_ack_delay로 제한하지 않고 이를 사용할 수 있으므로 불필요한 RTT 추정치 증가를 피할 수 있습니다.

피어의 승인 지연 보고 또는 엔드포인트의 최소_rtt 추정치에 오류가 있는 경우 승인 지연이 크면 smoothed_rtt가 상당히 부풀려질 수 있습니다. 따라서 핸드셰이크 확인 전에 승인 지연을 위해 RTT 샘플을 조정하면 샘플이 min_rtt보다 작아지는 경우 엔드포인트는 RTT 샘플을 무시할 **수 있습니다**.

핸드셰이크가 확인된 후 피어가 보고한 승인 지연이 피어의 최대_ack_delay보다 큰 경우, 이는 피어의 스케줄러 지연 또는 이전 승인 손실 등 의도하지 않았지만 잠재적으로 반복되는 지연이 원인일 수 있습니다. 초과 지연은 규정을 준수하지 않는 수신기로 인한 것일 수도 있습니다. 따라서 이러한 추가 지연은 사실상 경로 지연의 일부로 간주되어 RTT 예상치에 통합됩니다.

따라서 동료 보고 승인 지연을 사용하여 RTT 샘플을 조정할 때는 엔드포인트가 필요합니다:

- 초기 패킷에 대한 승인 지연은 피어에 의해 지연되지 않으므로 무시할 **수 있습니다**([QUIC-TRANSPORT] 13.2.1항);
- 핸드셰이크가 확인될 때까지 상대방의 max_ack_delay를 무시해야 합니다;
- 핸드셰이크가 확인된 후 승인 지연과 상대방의 max_ack_delay 중 더 작은 값을 사용해야 합니다.
- 결과값이 min_rtt보다 작을 경우 RTT 샘플에서 승인 지연을 빼지 **않아야 합니다**. 이렇게 하면 잘못된 보고로 인한 smoothed_rtt의 과소평가를 제한할 수 있습니다.

또한 엔드포인트는 해당 암호 해독 키를 즉시 사용할 수 없는 경우 승인 처리를 연기할 수 있습니다. 예를 들어, 클라이언트가 1-RTT 패킷 보호 키를 아직 사용할 수 없기 때문에 해독할 수 없는 0-RTT 패킷에 대한 승인을 받을 수 있습니다. 이러한 경우 엔드포인트는 핸드셰이크가 확인될 때까지 RTT 샘플에서 이러한 로컬 지연을 빼야 합니다.

RFC6298]과 유사하게 smoothed_rtt와 rttvar는 다음과 같이 계산됩니다.

엔드포인트는 연결 설정 중 및 연결 마이그레이션 중 추정기가 재설정될 때 RTT 추정기를 초기화합니다; [QUIC-TRANSPORT] 섹션 9.4 참조. 새 경로에 대해 RTT 샘플을 사용할 수 있기 전 또는 추정기가 재설정될 때 추정기는 초기 RTT를 사용하여 초기화됩니다(6.2.2절 참조).

smoothed_rtt와 rttvar는 다음과 같이 초기화되며, 여기서 kInitialRtt는 초기 RTT 값을 포함합니다:

```
smoothed_rtt = kInitialRtt
rttvar = kInitialRtt / 2
```

네트워크 경로에 대한 RTT 샘플은 최신_rtt에 기록됩니다(5.1절 참조). 초기화 후 첫 번째 RTT 샘플에서 추정기는 해당 샘플을 사용하여 재설정됩니다. 이렇게 하면 추정기가 과거 샘플의 기록을 유지하지 않습니다. 다른 경로로 전송된 패킷은 [QUIC-TRANSPORT] 섹션 9.4에 설명된 대로 현재 경로에 RTT 샘플을 기여하지 않습니다.

초기화 후 첫 번째 RTT 샘플에서 smoothed_rtt와 rttvar는 다음과 같이 설정됩니다:

```
smoothed_rtt = 최신_rtt
rttvar = 최신_rtt / 2
```

후속 RTT 샘플에서 smoothed_rtt와 rttvar는 다음과 같이 진화합니다:

```
ack_delay = (핸드셰이크가 확인된 경우) ACK 프레임에서 디코딩된 승
인 지연 시간입니다:
  ack_delay = min(ack_delay, max_ack_delay)
adjusted_rtt = latest_rtt
IF (LATEST_RTT >= MIN_RTT + AK_DELAY):
  ADJUSTED_RTT = LATEST_RTT - AK_DELAY
SMOOTHED_RTT = 7/8 * SMOOTHED_RTT + 1/8 * ADJUSTED_RTT
RTTVAR_SAMPLE = ABS(SMOOTHED_RTT - ADJUSTED_RTT)
RTTVAR = 3/4 * RTTVAR + 1/4 * RTTVAR_SAMPLE
```

6. 손실 감지

QUIC 발신자는 확인을 사용하여 손실된 패킷을 감지하고 PTO를 사용하여 확인이 수신되었는지 확인합니다(6.2절 참조). 이 섹션에서는 이러한 알고리즘에 대해 설명합니다.

패킷이 손실되면 QUIC 전송은 데이터를 재전송하거나 업데이트된 프레임을 전송하거나 프레임을 삭제하는 등의 방법으로 손실된 데이터를 복구해야 합니다. 자세한 내용은 다음 섹션을 참조하세요.
퀵-트랜스포트] 13.3.

손실 감지는 RTT 측정 및 혼잡 제어와 달리 패킷 수 공간별로 분리되어 있는데, 이는 RTT 및 혼잡 제어는 경로의 특성인 반면 손실 감지 는 키 가용성에 따라 달라지기 때문입니다.

6.1. 인증 기반 탐지

승인 기반 손실 감지는 TCP의 빠른 재전송 [RFC5681], 조기 재전송 [RFC5827], 순방향 승인 [FACK], SACK 손실 복구 [RFC6675], RACK-TLP [RFC8985]의 정신을 구현합니다. 이 섹션에서는 이러한 알고리즘이 QUIC에서 어떻게 구현되는지에 대한 개요를 제공합니다.

패킷이 다음 조건을 모두 충족하면 손실된 것으로 선언됩니다:

- 패킷이 승인되지 않은 상태로 전송 중이며 승인된 패킷보다 먼저 전송된 패킷입니다.
- 패킷이 승인된 패킷(섹션 6.1.1)보다 먼저 kPacketThreshold 패킷을 전송했거나(섹션 6.1.2) 과거에 충분히 오래 전송된 경우입니다(섹션 6.1.3).

확인은 나중에 전송된 패킷이 배달되었음을 나타내며, 패킷 및 시간 임계값은 패킷 재주문에 대한 어느 정도의 허용 오차를 제공합니다.

패킷을 허위로 손실로 선언하면 불필요한 재전송이 발생하고 손실 감지 시 혼잡 컨트롤러의 조치로 인해 성능이 저하될 수 있습니다. 구현을 통해 허위 재전송을 감지하고 패킷 또는 시간 재정렬 임계값을 높여 향후 허위 재전송 및 손실 이벤트를 줄일 수 있습니다. 적응형 시간 임계값을 사용하는 구현에서는 초기 재정렬 임계값을 더 작게 설정하여 복구 대기 시간을 최소화할 **수 있습니다.**

6.1.1. 패킷 임계값

패킷 재정렬 임계값(kPacketThreshold)의 **권장** 초기 값은 3이며, 이는 TCP 손실 감지 모범 사례 [RFC5681] [RFC6675]에 근거한 것입니다. TCP와 유사성을 유지하기 위해 구현에서는 3보다 작은 패킷 임계값을 사용해서는 안 됩니다([RFC5681] 참조).

일부 네트워크에서는 패킷 재정렬이 더 높은 수준으로 나타나 발신자가 가짜 손실을 감지할 수 있습니다. 또한, 패킷 재정렬은 TCP 패킷을 관찰하고 재정렬할 수 있는 네트워크 요소가 QUIC에서는 이를 수행할 수 없고 QUIC 패킷 번호가 암호화되어 있기 때문에 TCP보다 QUIC에서 더 일반적으로 발생할 수 있습니다. RACK [RFC8985]와 같이 허위로 손실을 감지한 후 재정렬 임계값을 높이는 알고리즘은 TCP에서 유용하다는 것이 입증되었으며, 최소한 QUIC에서도 유용할 것으로 예상됩니다.

6.1.2. 시간 임계값

동일한 패킷 번호 공간 내에서 나중에 전송된 패킷이 승인되면, 엔드포인트는 임계 시간보다 이전에 전송된 패킷이 있다면 손실된 것으로 선언해야 합니다. 패킷이 너무 일찍 손실된 것으로 선언되지 않도록 하려면 이 시간 임계값을 kGranularity 상수로 표시된 대로 로컬 타이머 단위 이상으로 설정해야 합니다. 시간 임계값은

```
max(kTimeThreshold * max(smoothed_rtt, latest_rtt), kGranularity)
```

가장 큰 승인 패킷 이전에 전송된 패킷을 아직 손실로 선언할 수 없는 경우 타이머(
남은 시간 동안 설정해야 **합니다.**

max(smoothed_rtt, latest_rtt)를 사용하면 다음 두 가지 경우를 방지할 수 있습니다:

- 의 경우 최신 RTT 샘플이 평활화된 RTT보다 낮은데, 이는 승인 경로가 더 짧은 경로를 만났을 때 순서를 다시 지정했기 때문일 수 있습니다;
- 의 경우 최신 RTT 샘플이 평활화된 RTT보다 높은데, 이는 실제 RTT가 지속적으로 증가했기 때문일 수 있지만 평활화된 RTT는 아직 따라잡지 못했기 때문일 수 있습니다.

RTT 승수로 표시되는 **권장** 시간 임계값(kTimeThreshold)은 9/8입니다. 그리고
타이머 세분성(k 세분성)의 **권장** 값은 1밀리초입니다.

> 참고: TCP의 RACK [RFC8985]는 비슷한 목적으로 5/4에 해당하는 약간 더 큰 임계값을 지정하고 있습니다. QUIC의 경험에 따르면 9/8이 잘 작동합니다.

구현에서는 절대 임계값, 이전 연결의 임계값, 적응형 임계값 또는 RTT 변동 포함을 실험해 볼 **수 있습니다. 임계값이** 작을수록 재정렬 복원력이 감소하고 스퓨리어스 재전송이 증가하며 임계값이 클수록 손실 감지 지연이 증가합니다.

6.2. 프로브 시간 초과

프로브 타임아웃(PTO)은 예상 시간 내에 응답을 유도하는 패킷이 승인되지 않거나 서버가 클라이언트의 주소를 확인하지 못한 경우 하나 또는 두 개의 프로브 데이터그램을 전송하도록 트리거합니다. PTO를 사용하면 꼬리 패킷 또는 승인 손실로부터 연결을 복구할 수 있습니다.

손실 감지와 마찬가지로 PTO는 패킷 번호 공간당입니다. 즉, 패킷 번호 공간당 PTO 값이 계산됩니다.

PTO 타이머 만료 이벤트는 패킷 손실을 나타내지 않으며 이전에 승인되지 않은 패킷이 손실된 것으로 표시되어서는 **안 됩니다.** 패킷을 새로 승인하는 승인이 수신되면 패킷 및 시간 임계값 메커니즘에 따라 손실 감지가 진행됩니다(섹션 6.1 참조).

QUIC에서 사용되는 PTO 알고리즘은 테일 손실 프로브 [RFC8985], RTO [RFC5681], TCP용 F-RTO 알고리즘 [RFC5682]의 신뢰성 기능을 구현합니다. 타임아웃 계산은 TCP의 RTO 주기 [RFC6298]를 기반으로 입니다.

6.2.1. 컴퓨팅 PTO

응답 유도 패킷이 전송되면 발신자는 다음과 같이 PTO 기간에 대한 타이머를 예약합니다:

```
PTO = smoothed_rtt + max(4*rttvar, kGranularity) + max_ack_delay
```

PTO 기간은 발신자가 전송한 패킷의 확인을 기다려야 하는 시간입니다. 이 기간에는 수신자가 확인 전송을 지연할 수 있는 최대 시간을 고려하기 위해 예상 네트워크 RTT(smoothed_rtt), 예상치의 변동(4*rttvar), 최대 _ack_delay가 포함됩니다.

PTO가 초기 또는 핸드셰이크 패킷 번호 공간에 대해 무장된 경우, 피어가 이러한 패킷을 의도적으로 지연시키지 않을 것으로 예상되므로 PTO 주기 계산의 max_ack_delay는 0으로 설정됩니다([QUIC-TANSPORT] 섹션 13.2.1 참조).

타이머가 즉시 만료되는 것을 방지하려면 PTO 기간이 최소 k 단위 이상이어야 합니다.

여러 패킷 번호 공간에서 응답을 유도하는 패킷이 전송 중인 경우, 타이머는 초기 및 핸드셰이크 패킷 번호 공간 중 더 빠른 값으로 설정해야 합니다.

엔드포인트는 핸드셰이크가 확인될 때까지 애플리케이션 데이터 패킷 번호 공간에 대한 PTO 타이머를 설정해서는

안 됩니다. 이렇게 하면 피어에 패킷을 처리할 키가 아직 없거나 엔드포인트가 패킷의 정보를 재전송할 수 없습니다.

아직 승인을 처리할 키를 가지고 있지 않습니다. 예를 들어, 클라이언트가 서버에 0-RTT 패킷을 전송할 때 서버가 이를 해독할 수 있는지 여부를 알지 못한 상태에서 이러한 상황이 발생할 수 있습니다. 마찬가지로 서버가 클라이언트가 서버의 인증서를 확인하여 1-RTT 패킷을 읽을 수 있는지 확인하기 전에 1-RTT 패킷을 전송할 때도 이러한 문제가 발생할 수 있습니다.

발신자는 수신 확인 패킷이 전송되거나 확인될 때마다 또는 초기 키 또는 핸드셰이크 키가 삭제될 때마다 PTO 타이머를 다시 시작해야 합니다([QUIC-TLS] 4.9절). 이렇게 하면 PTO가 항상 최신 추정 RTT를 기반으로 패킷 번호 공간에 걸쳐 올바른 패킷에 대해 설정됩니다.

PTO 타이머가 만료되면 PTO 백오프가 증가해야 하며, 그 결과 PTO 기간이 현재 값의 두 배로 설정됩니다. PTO 백오프 계수는 다음과 같은 경우를 제외하고는 확인을 받으면 재설정됩니다. 서버가 핸드셰이크 중에 패킷에 응답하는 데 시간이 오래 걸릴 수 있습니다. 이러한 서버를 반복되는 클라이언트 프로브로부터 보호하기 위해 서버가 클라이언트 주소의 유효성 검사를 완료했는지 아직 확실하지 않은 클라이언트에서는 PTO 백오프가 재설정되지 않습니다. 즉, 클라이언트는 초기 패킷에서 확인을 수신할 때 PTO 백오프 계수를 재설정하지 않습니다.

발신자 전송률의 기하급수적인 감소는 심각한 혼잡으로 인한 패킷 손실 또는 승인으로 인해 연속적인 PTO가 발생할 수 있기 때문에 중요합니다. 여러 패킷 번호 공간에서 수신 확인을 유발하는 패킷이 전송 중인 경우에도 네트워크의 과도한 부하를 방지하기 위해 모든 공간에서 PTO가 기하급수적으로 증가합니다. 예를 들어, 초기 패킷 번호 공간의 타임아웃은 핸드셰이크 패킷 번호 공간의 타임아웃 길이를 두 배로 늘립니다.

연속적인 PTO가 만료되는 총 시간은 유휴 시간 제한에 의해 제한됩니다. 시간 임계값 손실 감지를 위해 타이

머를 설정한 경우 PTO 타이머를 설정해서는 **안 됩니다(다음** 섹션 참조).
6.1.2. 시간 임계값 손실 감지를 위해 설정된 타이머는 다음에서 PTO 타이머보다 일찍 만료됩니다.
대부분의 경우 데이터를 허위로 재전송할 가능성이 적습니다.

6.2.2. 악수와 새로운 길

동일한 네트워크에서 재개된 연결은 이전 연결의 최종 평활화된 RTT 값을 재개된 연결의 초기 RTT로 사용할 **수 있습니다.** 이전 RTT를 사용할 수 없는 경우, 초기 RTT는 333밀리초로 설정해야 합니다. 이렇게 하면 TCP의 초기 RTO에 권장되는 대로 1초의 PTO로 핸드셰이크가 시작됩니다([RFC6298] 섹션 2 참조).

연결은 새 경로에 대한 초기 RTT(부록 A.2의 kInitialRtt 참조)를 설정하기 위해 PATH_CHALLENGE 전송과 PATH_RESPONSE 수신 사이의 지연을 사용할 **수** 있지만, 지연을 RTT 샘플로 간주해서는 **안 됩니다**.

초기 키와 핸드셰이크 키가 폐기되면(6.4절 참조), 초기 패킷과 핸드셰이크 패킷은 더 이상 인식할 수 없으므로 비행 중 바이트에서 제거됩니다.
초기 키 또는 핸드셰이크 키를 삭제할 경우, 키를 삭제하면 순방향 진행이 표시되고 손실 감지 타이머가 현재 삭제된 패킷 번호 공간에 설정되었을 수 있으므로 PTO 및 손실 감지 타이머를 **반드시** 재설정해야 합니다.

6.2.2.1. 주소 유효성 검사 전

서버가 경로에서 클라이언트의 주소를 확인할 때까지 서버가 전송할 수 있는 데이터 양은 [QUIC- 전송] 섹션 8.1에 명시된 대로 수신된 데이터 양의 3배로 제한됩니다. 추가 데이터를 전송할 수 없는 경우, PTO로 전송된 패킷은 증폭 방지 제한에 포함되므로 클라이언트로부터 데이터그램을 수신할 때까지 서버의 PTO 타이머가 작동해서는 **안 됩니다.**

서버가 클라이언트로부터 데이터그램을 수신하면 증폭 제한이 증가되고 서버는 PTO 타이머를 재설정합니다. 그런 다음 PTO 타이머가 과거의 시간으로 설정되어 있으면 즉시 실행됩니다. 이렇게 하면 핸드셰이크 완료에 중요한 패킷 이전에 새로운 1-RTT 패킷이 전송되는 것을 방지할 수 있습니다. 특히 0-RTT가 수락되었지만 서버가 클라이언트의 주소를 검증하지 못한 경우에 이런 문제가 발생할 수 있습니다.

클라이언트로부터 더 많은 데이터그램이 수신될 때까지 서버가 차단될 수 있으므로, 서버가 주소 검증을 완료한 것이 확실할 때까지 서버의 차단을 해제하기 위해 패킷을 보내는 것은 클라이언트의 책임입니다([QUIC-TRANSPORT] 섹션 8 참조). 즉, 클라이언트가 핸드셰이크 패킷에 대한 확인을 받지 못했고 핸드셰이크가 확인되지 않은 경우 클라이언트는 전송 중인 패킷이 없더라도 PTO 타이머를 설정해야 합니다([QUIC-TLS] 섹션 4.1.2 참조). PTO가 발동되면 클라이언트는 핸드셰이크 키가 있는 경우 핸드셰이크 패킷을 보내야 하며, 그렇지 않으면 페이로드가 최소 1200바이트인 UDP 데이터그램의 초기 패킷을 보내야 합니다( ).

6.2.3. 핸드셰이크 완료 속도 향상

서버가 중복된 암호화 데이터가 포함된 초기 패킷을 수신하면, 클라이언트가 초기 패킷으로 전송된 서버의 암호화 데이터를 모두 수신하지 못했거나 클라이언트의 예상 RTT가 너무 작다고 가정할 수 있습니다. 클라이언트가 핸드셰이크 키를 받기 전에 핸드셰이크 또는 1-RTT 패킷을 수신하면 서버의 초기 패킷의 일부 또는 전부가 손실된 것으로 간주할 수 있습니다.

이러한 조건에서 핸드셰이크 완료 속도를 높이기 위해, 엔드포인트는 연결당 제한된 횟수 동안 [QUIC-TRANSPORT] 섹션 8.1의 주소 유효성 검사 제한에 따라 PTO 만료보다 일찍 승인되지 않은 암호화 데이터가 포함된 패킷을 전송할 **수 있습니다.** 각 연결에 대해 최대 한 번만 재전송하면 단일 패킷 손실로부터 신속하게 복구하기에 충분합니다. 처리할 수 없는 패킷을 수신하면 항상 패킷을 재전송하는 엔드포인트는 패킷이 무한히 교환될 위험이 있습니다.

엔드포인트는 또한 합쳐진 패킷([QUIC-TRANSPORT] 섹션 12.2 참조)을 사용하여 각 데이터그램이 적어도 하나의 승인을 유도하도록 할 수 있습니다. 예를 들어, 클라이언트는 PING 및 PADDING 프레임이 포함된 초기 패킷을 0-RTT 데이터 패킷과 병합할 수 있고, 서버는 PING 프레임이 포함된 초기 패킷을 첫 번째 비행에서 하나 이상의 패킷과 병합할 수 있습니다.

6.2.4. 프로브 패킷 보내기

PTO 타이머가 만료되면 발신자는 패킷 번호 공간에 최소 하나의 응답 유도 패킷을 프로브로 보내야 합니다. 엔드포인트는 하나의 데이터그램 손실로 인해 비용이 많이 드는 연속적인 PTO 만료를 방지하거나 여러 패킷 번호 공간에서 데이터를 전송하기 위해 응답 유도 패킷이 포함된 풀사이즈 데이터그램을 최대 두 개까지 보낼 **수 있습니다.** PTO에서 전송되는 모든 프로브 패킷은 **반드시** Ack를 유도해야 합니다.

발신자는 타이머가 만료된 패킷 번호 공간에서 데이터를 전송하는 것 외에도 다른 패킷 번호 공간에서 전송 중인 데이터와 함께 수신 확인을 유도하는 패킷을 전송하여 가능하면 패킷을 합쳐야 합니다. 이는 서버가 초기 데이터와 핸드셰이크 데이터를 모두 가지고 있거나 클라이언트가 핸드셰이크와 애플리케이션 데이터를 모두 가지고 있을 때 특히 유용합니다. 왜냐하면 피어가 두 패킷 번호 공간 중 하나에 대한 수신 키만 가지고 있을 수 있기 때문입니다.

발신자가 PTO에서 더 빠른 승인을 유도하려는 경우 패킷 번호를 건너뛰어 승인 지연을 제거할 수 있습니다.

엔드포인트는 PTO 만료 시 전송되는 패킷에 새 데이터를 포함해야 합니다. 새 데이터를 전송할 수 없는 경우 이전에 전송된 데이터가 전송될 **수 있습니다.** 구현은 애플리케이션의 우선순위에 따라 새 데이터 또는 재전송된 데이터 전송을 포함하여 프로브 패킷의 내용을 결정하기 위해 다른 전략을 사용할 **수 있습니다.**

발신자에게 전송할 새 데이터나 이전에 전송한 데이터가 없을 수 있습니다. 예를 들어, 새 애플리케이션 데이터가 스트림 프레임으로 전송된 후 손실된 것으로 간주되어 새 패킷으로 재전송된 다음 원래 전송이 승인되는 일련의 이벤트를 생각해 보세요. 전송할 데이터가 없는 경우, 발신자는 단일 패킷으로 PING 또는 기타 응답 유도 프레임을 전송하여 PTO 타이머를 다시 **활성화해야 합니다.**

또는 발신자는 수신 거부 패킷을 보내는 대신 아직 전송 중인 패킷을 손실된 것으로 표시할 **수 있습니다.** 이렇게 하면 추가 패킷을 보내지 않아도 되지만 손실이 너무 공격적으로 선언되어 혼잡 컨트롤러가 불필요하게 속도를 낮출 위험이 높아집니다.

연속적인 PTO 기간은 기하급수적으로 증가하며, 그 결과 네트워크에서 패킷이 계속 삭제됨에 따라 연결 복구 지연 시간도 기하급수적으로 증가합니다. PTO 만료 시 두 개의 패킷을 전송하면 패킷 드롭에 대한 복원력이 증가하여 연속적인 PTO 이벤트가 발생할 확률이 줄어듭니다.

PTO 타이머가 여러 번 만료되어 새 데이터를 전송할 수 없는 경우, 구현은 매번 동일한 페이로드를 전송할지 아니면 다른 페이로드를 전송할지 선택해야 합니다. 동일한 페이로드를 전송하는 것이 더 간단할 수 있으며 우선순위가 가장 높은 프레임이 먼저 도착할 수 있습니다. 매번 다른 페이로드를 보내면 허위 재전송의 가능성이 줄어듭니다.

6.3. 재시도 패킷 처리

재시도 패킷은 클라이언트가 다른 초기 패킷을 전송하여 연결 프로세스를 효과적으로 다시 시작하게 합니다. 재시도 패킷은 초기 패킷이 수신되었지만 처리되지 않았음을 나타냅니다. 재시도 패킷은 패킷이 처리되었음을 나타내거나 패킷 번호를 지정하지 않으므로 승인으로 취급할 수 없습니다.

재시도 패킷을 수신한 클라이언트는 대기 중인 타이머를 재설정하는 등 혼잡 제어 및 손실 복구 상태를 재설정합니다. 기타 연결 상태, 특히 암호화 핸드셰이크 메시지는 유지됩니다; [QUIC-TRANSPORT] 17.2.5항을 참조하세요.

클라이언트는 첫 번째 초기 패킷이 전송된 시점부터 재시도 또는 버전 협상 패킷이 수신될 때까지의 기간으로 서버에 대한 RTT 추정치를 계산할 **수 있습니다**. 클라이언트 는 초기 RTT 예상값으로 기본값 대신 이 값을 사용할 수 있습니다.

6.4. 키 및 패킷 상태 삭제

초기 및 핸드셰이크 패킷 보호 키가 삭제되면([QUIC-TLS] 4.9절 참조), 해당 키로 전송된 모든 패킷은 승인을 처리할 수 없으므로 더 이상 승인할 수 없습니다. 발신자는 해당 패킷과 관련된 모든 복구 상태를 삭제하고 전송 중 바이트 수에서 해당 패킷을 제거해야 합니다.

엔드포인트는 핸드셰이크 패킷 교환을 시작하면 초기 패킷 송수신을 중지합니다([QUIC-TRANSPORT] 섹션 17.2.2.1 참조). 이 시점에서 모든 전송 중 초기 패킷의 복구 상태는 삭제됩니다.

0-RTT가 거부되면 모든 비행 중 0-RTT 패킷에 대한 복구 상태가 삭제됩니다.

서버가 0-RTT를 수락하지만 초기 패킷보다 먼저 도착하는 0-RTT 패킷을 버퍼링하지 않는 경우, 초기 0-RTT 패킷은 손실로 선언되지만 이런 경우는 드물 것으로 예상됩니다.

키로 암호화된 패킷이 승인되거나 손실이 선언된 후 어느 시점에 키가 폐기될 것으로 예상됩니다. 그러나 초기 및 Handshake 비밀은 핸드셰이크 및 1-RTT 키가 클라이언트와 서버 모두에서 사용 가능하다는 것이 입증되는 즉시 삭제됩니다([QUIC-TLS] 4.9.1절 참조).

7. 혼잡 제어

이 문서는 TCP 뉴레노[RFC6582]와 유사한 QUIC용 발신자 측 혼잡 컨트롤러를 지정합니다.

QUIC이 혼잡 제어를 위해 제공하는 신호는 일반적이며 다양한 발신자 측 알고리즘을 지원하도록 설계되었습니다. 발신자는 일방적으로 CUBIC [RFC8312]와 같은 다른 알고리즘을 선택할 수 있습니다.

발신자가 이 문서에 지정된 컨트롤러와 다른 컨트롤러를 사용하는 경우 선택한 컨트롤러가
RFC8085] 섹션 3.1에 명시된 혼잡 제어 가이드라인을 준수해야 합니다.

TCP와 마찬가지로, ACK 프레임만 포함된 패킷은 전송 중 바이트 수에 포함되지 않으며 혼잡 제어가 되지 않습니다. TCP와 달리 QUIC은 이러한 패킷의 손실을 감지할 수 있으며 해당 정보를 사용하여 혼잡 컨트롤러 또는 전송되는 ACK 전용 패킷의 속도를 조정할 **수** 있지만 이 문서에서는 이를 위한 메커니즘에 대해 설명하지 않습니다.

혼잡 컨트롤러는 경로별이므로 다른 경로로 전송된 패킷은 [QUIC-TRANSPORT] 섹션 9.4에 설명된 대로 현재 경로의 혼잡 컨트롤러를 변경하지 않습니다.

이 문서의 알고리즘은 컨트롤러의 혼잡 구간을 바이트 단위로 지정하고 사용합니다.

엔드포인트는 패킷이 PTO 타이머 만료(섹션 6.2 참조)에 전송되거나 복구에 들어갈 때(섹션 7.3.2 참조)가 아닌 한, bytes_in_flight(부록 B.2 참조)가 혼잡 윈도우보다 커질 경우 패킷을 전송해서는 **안 됩니다**.

7.1. 명시적 혼잡 알림

경로가 명시적 혼잡 알림(ECN) [RFC3168] [RFC8311]을 지원하는 것으로 확인된 경우, QUIC은 IP 헤더의 혼잡 경험(CE) 코드 포인트를 혼잡 신호로 취급합니다. 이 문서는 피어 보고 ECN-CE 카운트가 증가할 때 엔드포인트의 응답을 명시합니다; [QUIC-TRANSPORT] 13.4.2 절을 참조하세요.

7.2. 초기 및 최소 혼잡 기간

QUIC은 모든 연결을 초기값으로 설정한 혼잡 윈도우로 슬로우 스타트에서 시작합니다. 엔드포인트는 초기 혼잡 윈도우를 최대 데이터그램 크기(max_datagram_size)의 10배로 사용해야 하며, 윈도우를 14,720바이트 또는 최대 데이터그램 크기의 2배 중 더 큰 값으로 제한해야 합니다. 이는 [RFC6928]의 분석 및 권장 사항을 따르며, TCP의 20바이트 오버헤드에 비해 UDP의 8바이트 오버헤드가 더 작다는 점을 고려하여 바이트 제한을 늘립니다.

연결 중에 최대 데이터그램 크기가 변경되면, 초기 혼잡 윈도우는 새로운 크기로 다시 계산되어야 합니다. 핸드셰이크를 완료하기 위해 최대 데이터그램 크기를 줄인 경우, 혼잡 윈도우를 새로운 초기 혼잡 윈도우로 설정해야 합니다.

클라이언트의 주소를 확인하기 전에 서버는 [QUIC-TRANSPORT] 섹션 8.1에 명시된 증폭 방지 제한에 의해 추가로 제한될 수 있습니다. 증폭 방지 제한은 혼잡 창이 완전히 활용되는 것을 방지하여 혼잡 창 증가 속도를 늦출 수 있지만, 혼잡 창에 직접적인 영향을 미치지는 않습니다.

최소 혼잡 창은 손실, 피어 보고 ECN-CE 수의 증가 또는 지속적인 혼잡에 대응하여 혼잡 창이 도달할 수 있는 가장 작은 값입니다. **권장** 값은 2 * max_datagram_size입니다.

7.3. 혼잡 제어 상태

이 문서에서 설명하는 뉴레노 혼잡 컨트롤러는 그림 1과 같이 세 가지 상태가 있습니다.

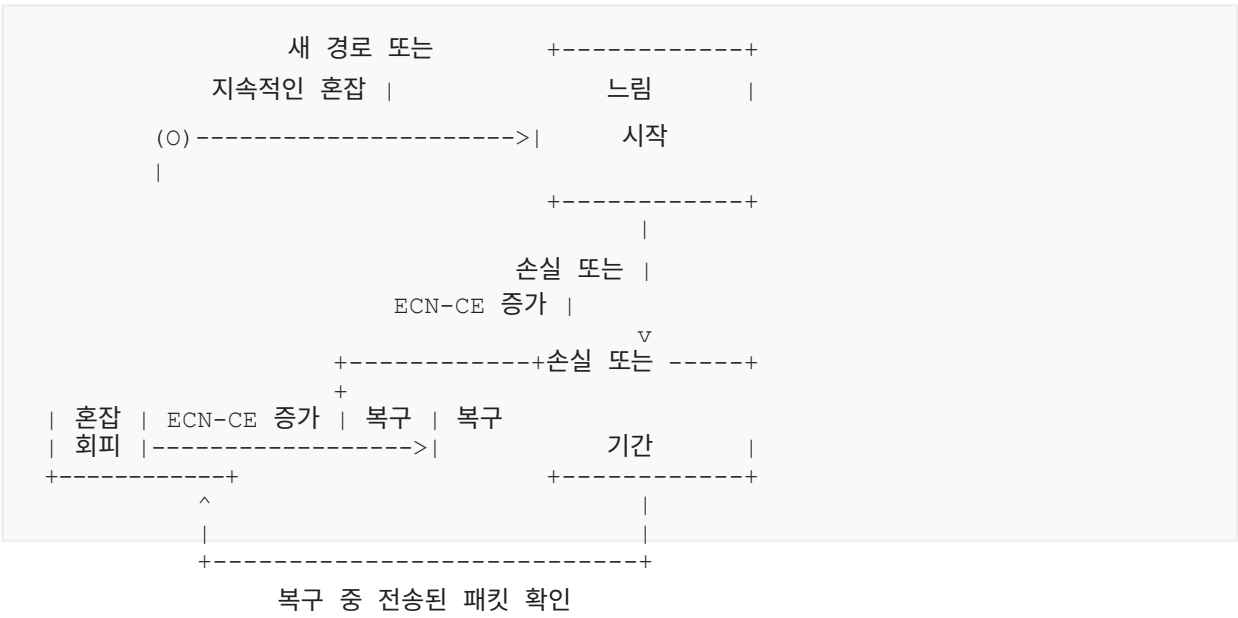

그림 1: 혼잡 제어 상태 및 전환

이러한 상태와 상태 간의 전환은 다음 섹션에서 설명합니다.

7.3.1. 슬로우 스타트

NewReno 발신자는 혼잡 창이 느린 시작 임계값보다 낮을 때마다 느린 시작 상태가 됩니다. 발신자는 슬로우 스타트 임계값이 무한대 값으로 초기화되므로 슬로우 스타트 상태로 시작됩니다.

발신자가 슬로우 스타트 상태인 동안에는 각 승인이 처리될 때마다 승인되는 바이트 수만큼 혼잡 창이 증가합니다. 그 결과 혼잡 윈도우가 기하급수적으로 증가합니다.

발신자는 패킷이 손실되거나 상대방이 보고한 ECN-CE 수가 증가하면 **반드시** 슬로우 스타트를 종료하고 복구 기간에 들어가야 합니다.

발신자는 정체 기간이 슬로우 스타트 임계값( )보다 작을 때마다 슬로우 스타트로 재진입하며, 이는 지속적인 정체가 선언된 후에만 발생합니다.

7.3.2. 복구

뉴레노 발신자는 패킷 손실이 감지되거나 상대방이 보고한 ECN- CE 수가 증가하면 복구 기간에 들어갑니다. 이미 복구 기간에 있는 발신자는 복구 기간에 머무르며 재진입하지 않습니다.

복구 기간에 들어가면 발신자는 손실이 감지될 때 느린 시작 임계값을 혼잡 기간 값의 절반으로 설정해야 합니다. 복구 기간을 종료하기 전에 혼잡 구간을 느린 시작 임계값의 감소된 값으로 설정해야 합니다.

구현은 복구 기간에 들어가면 즉시 혼잡 구간을 줄이거나 비례 비율 감소[PRR]와 같은 다른 메커니즘을 사용하여 혼잡 구간을 더 점진적으로 줄일 **수 있습니다.** 혼잡 구간이 즉시 감소하는 경우, 감소 전에 단일 패킷을 전송할 수 있습니다. 이렇게 하면 손실된 패킷의 데이터가 재전송되는 경우 손실 복구 속도가 빨라지며, [RFC6675] 섹션 5에 설명된 대로 TCP와 유사합니다.

복구 기간은 혼잡 구간 감소를 왕복 1회로 제한하는 것을 목표로 합니다. 따라서 복구 기간 동안에는 새로운 손실이 발생하거나 ECN-CE 수가 증가하더라도 혼잡 구간은 변경되지 않습니다.

복구 기간 동안 전송된 패킷이 승인되면 복구 기간이 종료되고 발신자는 혼잡 회피에 들어갑니다. 이는 복구가 시작된 손실된 세그먼트가 확인되면 종료되는 TCP의 복구 정의( )와는 약간 다릅니다[RFC5681].

7.3.3. 혼잡 회피

뉴레노 발신자는 복구 기간이 아닌 혼잡 구간이 느린 시작 임계값 이상일 때마다 혼잡 회피 상태에 있습니다.

혼잡 회피의 발신자는 혼잡 창에 대한 증가를 인정되는 각 혼잡 창에 대해 최대 하나의 최대 데이터그램 크기로 제한해야 하는 추가 증가 곱셈 감소(AIMD) 방식을 사용합니다.

발신자는 패킷이 손실되면 혼잡 회피를 종료하고 복구 기간에 들어가거나, 상대방이 보고한 ECN-CE 카운트가 증가하면 복구 기간에 들어갑니다.

7.4. 해독 불가능한 패킷 손실 무시하기

핸드셰이크 중에 패킷이 도착할 때 일부 패킷 보호 키를 사용할 수 없을 수 있으며, 수신자는 패킷을 삭제하도록 선택할 수 있습니다. 특히 핸드셰이크와 0-RTT 패킷은 초기 패킷이 도착할 때까지 처리할 수 없으며, 1-RTT 패킷은 핸드셰이크가 완료될 때까지 처리할 수 없습니다. 엔드포인트는 상대방이 해당 패킷을 처리할 패킷 보호 키를 갖기 전에 도착했을 수 있는 핸드셰이크, 0-RTT 및 1-RTT 패킷의 손실을 무시할 **수 있습니다.** 엔드포인트는 주어진 패킷 번호 공간에서 가장 먼저 승인된 패킷 이후에 전송된 패킷의 손실을 무시해서는 **안 됩니다.**

7.5. 프로브 시간 초과

프로브 패킷은 혼잡 컨트롤러에 의해 차단되어서는 **안 됩니다.** 그러나 이러한 패킷은 패킷 손실 없이 네트워크 부하를 추가하므로 발신자는 이러한 패킷이 추가로 전송 중인 것으로 계산해야 합니다. 프로브 패킷을 전송하면 패킷 손실 또는 패킷 전송을 확인하는 확인이 수신될 때까지 발신자의 전송 중 바이트가 혼잡 구간을 초과할 수 있다는 점에 유의하세요.

7.6. 지속적인 혼잡

발신자가 충분히 긴 시간 동안 전송된 모든 패킷의 손실이 확인되면 네트워크에 지속적인 정체가 발생하고 있는 것

으로 간주합니다.

7.6.1. 기간

지속적인 혼잡 시간은 다음과 같이 계산됩니다:

```
(smoothed_rtt + max(4*rttvar, kGranularity) + max_ack_delay) *
    kPersistentCongestionThreshold
```

섹션 6.2의 PTO 계산과 달리 이 기간에는 손실이 설정된 패킷 수 공간에 관계없이 max_ack_delay가 포함됩니다.

이 기간을 통해 발신자는 테일 손실 프로브[RFC8985] 및 RTO[RFC5681]를 사용하는 TCP처럼 PTO 만료에 대한 응답을 포함하여 지속적인 정체가 발생하기 전에 많은 패킷을 전송할 수 있습니다.

kPersistentCongestionThreshold 값이 크면 발신자가 네트워크의 지속적인 혼잡에 덜 반응하게 되어 혼잡한 네트워크에 공격적으로 전송할 수 있습니다. 값이 너무 작으면 발신자가 불필요하게 지속적인 정체를 선언하여 발신자의 처리량이 감소할 수 있습니다.

kPersistentCongestionThreshold의 **권장** 값은 3이며, 이는 두 개의 TLP 후에 TCP 발신자가 RTO를 선언하는 것과 거의 동일한 동작을 초래합니다.

이 설계에서는 애플리케이션 패턴이 PTO 만료에 영향을 미치기 때문에 연속적인 PTO 이벤트를 사용하여 지속적인 정체를 설정하지 않습니다. 예를 들어, 소량의 데이터를 전송하는 발신자가 그 사이에 침묵 기간을 두고 전송할 때마다 PTO 타이머를 다시 시작하면 수신 확인이 없는 경우에도 PTO 타이머가 장기간 만료되지 않을 수 있습니다. 기간을 사용하면 발신자가 PTO 만료에 의존하지 않고 지속적인 정체를 설정할 수 있습니다.

7.6.2. 지속적인 혼잡 설정

발신자가 확인을 수신한 후 확인을 유도하는 패킷 두 개가 분실되었다고 선언되면 지속적인 혼잡이 발생하게 됩니다:

- 이 두 패킷의 전송 시간 사이에 전송된 패킷은 모두 인정되지 않습니다;
- 이 두 패킷의 전송 시간 사이의 지속 시간이 지속적 혼잡 지속 시간을 초과하는 경우(섹션 7.6.1); 그리고
- 이 두 패킷이 전송될 때 이전 RTT 샘플이 존재했습니다.

수신자는 최대 확인 지연 시간 내에 확인을 유도하는 패킷만 확인해야 하므로 이 두 패킷은 **반드시** 확인을 유도해야 합니다([QUIC- 전송] 섹션 13.2 참조).

지속적 혼잡 기간은 적어도 하나의 RTT 샘플이 있을 때까지 시작되지 **않아야 합니다**. 첫 번째 RTT 샘플 전에 발신자는 초기 RTT(섹션 6.2.2)를 기준으로 PTO 타이머를 작동시키며, 이는 실제 RTT보다 훨씬 더 클 수 있습니다. 사전 RTT 샘플을 요구하면 발신자가 잠재적으로 너무 적은 수의 프로브로 지속적인 정체를 설정하는 것을 방지할 수 있습니다.

네트워크 혼잡은 패킷 번호 공간의 영향을 받지 않으므로 지속적인 혼잡은 패킷 번호 공간을 가로질러 전송된 패킷을 고려해야 합니다. 모든 패킷 번호 공간에 대한 상태가 없는 발신자 또는 패킷 번호 공간에 걸쳐 전송 시간을 비교할 수 없는 구현은 승인된 패킷 번호 공간에 대해서만 상태를 사용할 **수 있습니다**. 이 경우 지속적 정체를 잘못 선언할 수 있지만 지속적 정체를 감지하지 못하는 결과로 이어지지는 않습니다.

지속적인 혼잡이 선언되면 발신자의 혼잡 창은 RTO [RFC5681]에 대한 TCP 발신자의 응답과 유사하게 최소 혼잡 창(kMinimumWindow)으로 축소되어야 합니다.

7.6.3. 예

다음 예는 발신자가 지속적인 정체를 설정하는 방법을 설명합니다. 가정합니다:

```
smoothed_rtt + max(4*rttvar, kGranularity) + max_ack_delay = 2
kPersistentCongestionThreshold = 3
```

다음 이벤트 순서를 고려하세요:

시간	액션
t=0	패킷 #1 전송(애플리케이션 데이터)
t=1	패킷 #2 전송(애플리케이션 데이터)
t=1.21	번 승인 받기
t=2	패킷 #3 전송(애플리케이션 데이터)
t=3	패킷 #4 전송(애플리케이션 데이터)
t=4	패킷 #5 전송(애플리케이션 데이터)
t=5	패킷 #6 전송(애플리케이션 데이터)
t=6	패킷 #7 전송(애플리케이션 데이터)
t=8	패킷 #8 전송(PTO 1)
t=12	패킷 #9 전송(PTO 2)

시간	액션
t=12.2 #9의	승인 받기

표 1

패킷 2~8은 패킷 9에 대한 승인이 `t = 12.2`에 수신되면 손실로 선언됩니다.

혼잡 기간은 가장 오래된 패킷과 가장 최근에 손실된 패킷 사이의 시간인 `8 - 1`로 계산됩니다.
= 지속적 정체 지속 시간은 `2 * 3 = 6`입니다. 임계값에 도달했고 가장 오래된 패킷과 가장 최근에 손실된 패킷 사이에 어떤 패킷도 인식되지 않았기 때문에 네트워크는 지속적인 정체를 경험한 것으로 간주됩니다.

이 예는 PTO 만료를 보여 주지만, 지속적인 혼잡이 설정되기 위해 반드시 필요한 것은 아닙니다.

7.7. 페이싱

발신자는 혼잡 컨트롤러의 입력에 따라 모든 기내 패킷의 전송 속도를 **조절해야 합니다.**

여러 개의 패킷을 지연 없이 네트워크에 전송하면 패킷 버스트가 발생하여 단기적인 정체 및 손실이 발생할 수 있습니다. 발신자는 **반드시** 페이싱을 사용하거나 이러한 버스트를 제한해야 합니다. 발신자는 버스트를 초기 혼잡 시간으로 제한해야 합니다(7.2항 참조). 수신자에 대한 네트워크 경로가 더 큰 버스트를 흡수할 수 있다는 것을 알고 있는 발신자는 더 높은 제한을 사용할 **수 있습니다.**

구현은 혼잡 컨트롤러가 페이서와 잘 작동하도록 설계하는 데 주의를 기울여야 합니다. 예를 들어, 페이서가 혼잡 컨트롤러를 감싸고 혼잡 윈도우의 가용성을 제어하거나, 페이서가 혼잡 컨트롤러가 전달한 패킷의 속도를 조절할 수 있습니다.

ACK 프레임의 적시 전송은 신속한 손실 복구를 위해 중요합니다. 따라서 피어에게 전달이 지연되지 않도록 ACK 프레임만 포함된 패킷은 속도를 늦추지 않아야 합니다.

엔드포인트는 원하는 대로 페이싱을 구현할 수 있습니다. 완벽하게 페이싱된 발신자는 시간이 지남에 따라 패킷을 정확히 균등하게 분산시킵니다. 이 문서에 설명된 것과 같은 윈도우 기반 혼잡 컨트롤러의 경우, 이 속도는 RTT에 대한 혼잡 윈도우의 평균을 계산하여 계산할 수 있습니다. 시간당 바이트 단위의 비율로 표현되며, 여기서 congestion_window는 바이트 단위입니다:

```
속도 = N * 혼잡도_창 / smoothed_rtt
```

또는 시간 단위의 패킷 간 간격으로 표현할 수도 있습니다:

```
간격 = ( smoothed_rtt * packet_size / congestion_window ) / N
```

N의 값은 작지만 최소 1(예: 1.25)을 사용하면 RTT의 변화로 인해 혼잡 시간대의 활용도가 낮아지지 않도록 보장할 수 있습니다.

패킷화, 스케줄링 지연, 계산 효율성과 같은 실용적인 고려 사항으로 인해 발신자가 RTT보다 훨씬 짧은 기간 동안 이 전송률에서 벗어날 수 있습니다.

페이싱을 위한 한 가지 가능한 구현 전략은 누수 버킷 알고리즘을 사용하는데, 여기서 '버킷'의 용량은 최대 버스트 크기로 제한되고 '버킷'이 채워지는 속도는 위의 함수에 의해 결정되는 입니다.

7.8. 혼잡 시간대 활용 부족

전송 중인 바이트가 혼잡 창보다 작고 전송 속도가 제한되지 않는 경우 혼잡 창이 제대로 활용되지 않는 것입니다. 이는 애플리케이션 데이터가 충분하지 않거나 흐름 제어 제한으로 인해 발생할 수 있습니다. 이 경우 느린 시작 또는 혼잡 회피에서 혼잡 윈도우를 늘려서는 **안** 됩니다.

패킷을 페이싱하는 발신자(섹션 7.7 참조)는 패킷 전송이 지연되고 이 지연으로 인해 혼잡 구간을 충분히 활용하지 못할 수 있습니다. 발신자가 페이싱 지연 없이 혼잡 구간을 충분히 활용했다면 애플리케이션이 제한되었다고 간주해서는 **안 됩니다**.

발신자는 [RFC7661]에서 TCP에 대해 제안된 것과 같이 사용률이 낮은 기간( ) 후에 혼잡 창을 업데이트하는 대체 메커니즘을 구현할 **수 있습니다**.

8. 보안 고려 사항

8.1. 손실 및 혼잡 신호

손실 탐지 및 혼잡 제어는 기본적으로 인증되지 않은 엔티티의 지연, 손실, ECN 표시와 같은 신호를 사용합니다. 공격자는 패킷을 삭제하거나 지연 경로를 전략적으로 변경하거나 ECN 코드 포인트를 변경하는 등 이러한 신호를 조작하여 엔드포인트가 전송 속도를 낮추도록 만들 수 있습니다.

8.2. 트래픽 분석

ACK 프레임만 전달하는 패킷은 패킷 크기를 관찰하여 휴리스틱하게 식별할 수 있습니다. 승인 패턴은 링크 특성이나 애플리케이션 동작에 대한 정보를 노출할 수 있습니다. 유출된 정보를 줄이기 위해 엔드포인트는 승인을 다른 프레임과 번들로 묶거나 성능에 잠재적인 비용을 지불하고 PADDING 프레임을 사용할 수 있습니다.

8.3. 잘못된 ECN 표시 보고

수신자가 ECN 표시를 잘못 보고하여 발신자의 혼잡 응답을 변경할 수 있습니다. ECN-CE 표시의 보고를 억제하면 발신자가 전송 속도를 높일 수 있습니다. 이러한 증가는 혼잡과 손실로 이어질 수 있습니다.

발신자는 가끔씩 전송하는 패킷에 ECN-CE 마킹을 표시하여 보고 억제를 감지할 수 있습니다. ECN-CE 마킹으로 전송된 패킷이 승인될 때 CE 마킹이 된 것으로 보고되지 않으면 발신자는 해당 경로로 전송되는 후속 패킷에서 ECN 가능 전송(ECT) 코드 포인트를 설정하지 않음으로써 해당 경로에 대한 ECN을 비활성화할 수 있습니다 [RFC3168].

추가 ECN-CE 표시를 보고하면 발신자가 전송 속도를 낮추게 되는데, 이는 연결 흐름 제어 제한을 광고하는 것과 효과가 비슷하므로 그렇게 해서 얻을 수 있는 이점은 없습니다.

엔드포인트는 사용하는 혼잡 컨트롤러를 선택합니다. 혼잡 컨트롤러는 전송률을 낮추는 방식으로 ECN-CE 보고에 응답하지만, 응답은 다를 수 있습니다. 표시를 손실과 동등한 것으로 처리할 수 있지만[RFC3168], [RFC8511] 또는 [RFC8311]과 같은 다른 응답을 지정할 수 있습니다.

9. 참조

9.1. 규범 참조

[QUIC-TLS] Thomson, M., Ed. 및 S. Turner, Ed., "TLS를 사용하여 QUIC 보안", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/info/rfc9001>.

[QUIC-TRANSPORT] Iyengar, J., Ed. 및 M. Thomson, Ed., "QUIC: A UDP 기반 멀티플렉스 및 보안 전송", RFC 9000, DOI 10.17487/RFC9000, 2021년 5월, <https://www.rfc-editor.org/info/rfc9000>.

[RFC2119] Bradner, S., "요구 수준을 나타내기 위해 RFC에서 사용하는 키워드", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/ rfc2119>.

[RFC3168] Ramakrishnan, K., Floyd, S. 및 D. Black, "명시적 혼잡 알림(ECN)을 IP에 추가", RFC 3168, DOI 10.17487/RFC3168, 2001년 9월, <https://www.rfc-editor.org/info/rfc3168>.

[RFC8085] 에거트, L., 페어허스트, G., 셰퍼드, G., "UDP 사용 가이드라인", BCP 145, RFC 8085, DOI 10.17487/RFC8085, 2017년 3월, <https://www.rfc-editor.org/info/ rfc8085>.

[RFC8174] Leiba, B., "RFC 2119 키워드에서 대문자와 소문자의 모호성", BCP 14, RFC 8174, DOI 10.17487/RFC8174, 2017년 5월, <https://www.rfc-editor.org/info/ rfc8174>.

9.2. 유익한 참고 자료

[FACK] Mathis, M. and J. Mahdavi, "포워드 승인: TCP 혼잡 제어 개선", ACM SIGCOMM 컴퓨터 통신 검토, DOI 10.1145/248157.248181, 1996년 8월,

<https://doi.org/10.1145/248157.248181>.

[PRR] Mathis, M., Dukkipati, N. 및 Y. Cheng, "비례적 TCP 속도 감소", RFC 6937, DOI 10.17487/RFC6937, 2013년 5월, <https://www.rfc-editor.org/info/ rfc6937>.

[재전송] Karn, P. 및 C. Partridge, "왕복 시간 추정치 향상 신뢰할 수 있는 전송 프로토콜", ACM 트랜잭션 온 컴퓨터 시스템, DOI 10.1145/118544.118549, 1991년 11월, <https:// doi.org/10.1145/118544.118549>.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S. 및 A. Romanow, "TCP 선택적 승인 옵션", RFC 2018, DOI 10.17487/RFC2018, 1996년 10월, <https://www.rfc-editor.org/info/rfc2018>.

[RFC3465] Allman, M., "적절한 바이트 카운팅(ABC)을 이용한 TCP 혼잡 제어", RFC 3465, DOI 10.17487/RFC3465, 2003년 2월, <https://www.rfc-editor.org/ info/rfc3465>.

[RFC5681] Allman, M., Paxson, V. 및 E. Blanton, "TCP 혼잡 제어", RFC 5681, DOI 10.17487/RFC5681, 2009년 9월, <https://www.rfc-editor.org/info/rfc5681>.

[RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): TCP에서 허위 재전송 시간 초과를 탐지하는 알고리즘", RFC 5682, DOI 10.17487/RFC5682, 2009년 9월, <https://www.rfc-editor.org/ info/rfc5682>.

[RFC5827] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., P. Hurtig, "TCP 및 스트림 제어 전송 프로토콜(SCTP)을 위한 조기 재전송", RFC 5827, DOI 10.17487/RFC5827, 2010년 5월, <https://www.rfc-editor.org/info/rfc5827>.

[RFC6298] Paxson, V., Allman, M., Chu, J., M. Sargent, "계산하는 TCP의 재전송 타이머", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/ info/rfc6298>.

[RFC6582] 헨더슨, T., 플로이드, S., 구르토프, A., 니시다, Y., "TCP의 빠른 복구 알고리즘에 대한 뉴레노 수정", RFC 6582, DOI 10.17487/RFC6582, 2012년 4월, <https://www.rfc-editor.org/info/rfc6582>.

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., Y. Nishida, "TCP를 위한 선택적 승인(SACK) 기반의 보수적 손실 복구 알고리즘", RFC 6675, DOI 10.17487/RFC6675, August 2012, <https://>, <https:// www.rfc-editor.org/info/rfc6675>.

[RFC6928] 추, J., 두키파티, N., 쳉, Y., M. 매티스, "TCP의 초기 윈도우 늘리기", RFC 6928, DOI 10.17487/RFC6928, 2013년 4월, <https://www.rfc- editor.org/info/rfc6928>.

[RFC7661] 페어허스트, G., 사티아실란, A., 및 R. 세치, "속도 제한 트래픽을 지원하기 위한 TCP 업

데이트", RFC 7661, DOI 10.17487/RFC7661, 2015년 10월, <https://www.rfc-editor.org/info/rfc7661>.

[RFC8311] Black, D., "명시적 혼잡 알림(ECN) 실험에 대한 제한 완화", RFC 8311, DOI 10.17487/RFC8311, 2018년 1월, <https:// www.rfc-editor.org/info/rfc8311>.

[RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "고속 장거리 네트워크를 위한 CUBIC", RFC 8312, DOI 10.17487/RFC8312, February 2018, <https://www.rfc-editor.org/info/rfc8312>, <https://www.rfc-editor.org/info/rfc8312>.

[RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP 대체 백오프와 ECN(ABE)", RFC 8511, DOI 10.17487/RFC8511, 2018년 12월, <https://>, <https://www.rfc-editor.org/info/rfc8511>.

[RFC8985] Cheng, Y., Cardwell, N., Dukkipati, N. 및 P. Jha, "The RACK-TLP 손실 감지 알고리즘 for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <https://www.rfc-editor.org/info/rfc8985>.

부록 A. 손실 복구 의사 코드

이제 섹션 6에서 설명한 손실 감지 메커니즘의 구현 예시를 설명하겠습니다.

이 섹션의 의사 코드 세그먼트는 코드 구성 요소로 라이선스가 부여되어 있습니다(저작권 공지를 참조하세요).

A.1. 전송된 패킷 추적

혼잡 제어를 올바르게 구현하기 위해 QUIC 발신자는 패킷이 승인되거나 손실될 때까지 모든 수신 확인을 유발하는 패킷을 추적합니다. 구현은 패킷 번호와 암호화 컨텍스트별로 이 정보에 액세스하고 손실 복구 및 혼잡 제어를 위해 패킷별 필드(부록 A.1.1)를 저장할 수 있을 것으로 예상됩니다.

패킷이 손실된 것으로 선언된 후에도 엔드포인트는 패킷 재정렬을 위해 일정 시간 동안 해당 상태를 유지할 수 있습니다([QUIC-TRANSPORT] 섹션 13.3 참조). 이를 통해 발신자는 허위 재전송을 감지할 수 있습니다.

전송된 패킷은 각 패킷 번호 공간에 대해 추적되며, ACK 처리는 단일 공간에만 적용됩니다.

A.1.1. 전송된 패킷 필드

패킷 번호: 전송된 패킷의 패킷 번호입니다.

ack_eliciting: 패킷이 응답 유도 패킷인지 여부를 나타내는 부울입니다. true이면
를 설정하면 확인을 받을 수 있지만, 피어는 이를 포함한 ACK 프레임 전송을 최대 최대_ack_delay까지 지연시

킬 수 있습니다.

in_flight: 패킷이 비행 중 바이트 수에 포함되는지 여부를 나타내는 부울입니다.

보낸_바이트: 패킷에 전송된 바이트 수(UDP 또는 IP 오버헤드는 포함되지 않음).
QUIC 프레이밍 오버헤드를 포함합니다.

time_sent: 패킷이 전송된 시간입니다.

A.2. 관심 상수

손실 복구에 사용되는 상수는 RFC, 논문 및 일반적인 관행의 조합을 기반으로 합니다.

k패킷 임계값: 패킷 임계값 손실 감지 전 패킷의 최대 재정렬 횟수
는 패킷이 손실된 것으로 간주합니다. 섹션 6.1.1에서 권장하는 값은 3입니다.

kTimeThreshold: 시간 임계값 손실 감지 전 최대 재정렬 시간: 시간 임계값을 고려합니다.
패킷 손실. RTT 승수로 지정됩니다. 섹션 6.1.2에서 권장하는 값은 9/8입니다.

k세분성: 타이머 세분성. 이는 시스템에 따라 달라지는 값이며, 섹션 6.1.2.
값은 1ms를 권장합니다.

kInitialRtt: RTT 샘플을 채취하기 전에 사용되는 RTT입니다. 섹션에서 권장하는 값입니다.
6.2.2는 333ms입니다

세 개의 패킷 번호 공백을 열거하는 열거형입니다:
.

kPacketNumberSpace:

```
열거형 케이패킷번호공간 { 이니셜,
  핸드셰이크, 애플리케이션데이터,
}
```

A.3. 관심 변수

이 섹션에서는 혼잡 제어 메커니즘을 구현하는 데 필요한 변수에 대해 설명합니다.

latest_rtt: 인증에 대한 확인을 받을 때 가장 최근에 측정한 RTT입니다.
이전에 승인되지 않은 패킷.

smoothed_rtt: 5.3절에 설명된 대로 계산된 연결의 평활화된 RTT입니다.

rttvar: 섹션 5.3에 설명된 대로 계산된 RTT 변동입니다.

min_rtt: 승인 지연을 무시하고 일정 기간 동안 표시되는 최소 RTT는 다음과 같습니다.

섹션 5.2에 설명되어 있습니다.

first_rtt_sample: 첫 번째 RTT 샘플을 얻은 시간입니다.

max_ack_delay: 수신자가 지연시킬 최대 시간입니다.
시조 전송 파라미터에 정의된 대로 애플리케이션 데이터 패킷 번호 공간의 패킷에 대한 승인입니다([QUIC-TRANSPORT] 18.2절). 타이머 지연, 재주문 또는 손실로 인해 수신된 ACK 프레임의 실제 ack_delay는 더 커질 수 있다는 점에 유의하세요.

손실 감지 타이머: 손실 감지에 사용되는 멀티모달 타이머입니다.

pto_count: 확인을 받지 않고 PTO를 전송한 횟수입니다.

time_of_last_ack_eliciting_packet[kPacketNumberSpace]: 가장 최근의 응답을 유도한 시간 패킷이 전송되었습니다.

largest_acked_packet[kPacketNumberSpace]: 가장 큰 패킷 번호는 지금까지의 패킷 번호 공간입니다.

손실 시간[k패킷번호공간]: 해당 패킷 번호의 다음 패킷이 손실되는 시간입니다.
공간은 시간 내에 재주문 기간을 초과하면 손실된 것으로 간주될 수 있습니다.

보낸_패킷[k패킷번호공간]: 패킷 번호에서 패킷 번호의 연관성
공간에 대한 정보를 제공합니다. 위의 부록 A.1에 자세히 설명되어 있습니다.

A.4. 초기화

연결이 시작될 때 다음과 같이 손실 감지 변수를 초기화합니다:

```
손실 감지 타이머 재설정() pto_count = 0
latest_rtt = 0
smoothed_rtt = kInitialRtt
rttvar = kInitialRtt / 2
min_rtt = 0
first_rtt_sample = 0
초기, 핸드셰이크, 애플리케이션데이터 ]의 pn_space에 대해: largest_acked_packet[pn_space]
  = 무한 시간_of_last_ack_eliciting_packet[pn_space] = 0
  손실 시간[PN_공간] = 0
```

A.5. 패킷 전송 시

패킷이 전송된 후에는 패킷에 대한 정보가 저장됩니다. OnPacketSent의 파라미터는 위의 부록 A.1.1에 자세히 설명되어 있습니다.

OnPacketSent의 의사 코드는 다음과 같습니다:

```
OnPacketSent(packet_number, pn_space, ack_eliciting,
             in_flight, sent_bytes):
  보낸_패킷[pn_공간][패킷_번호].패킷_번호 =
                                              packet_number
  sent_packets[pn_space][packet_number].time_sent = now()
  sent_packets[pn_space][packet_number].ack_eliciting =
                                              ack_eliciting
  sent_packets[pn_space][packet_umber].in_flight = in_flight
  sent_packets[pn_space][packet_umber].sent_bytes = sent_bytes
  if (in_flight):
    if (ack_eliciting):
      time_of_last_ack_eliciting_packet[pn_space] = now()
    OnPacketSentCC(sent_bytes)
    SetLossDetectionTimer()
```

A.6. 데이터그램 수신 시

서버가 증폭 방지 제한에 의해 차단된 경우, 데이터그램의 패킷이 성공적으로 처리되지 않더라도 데이터그램을 수신하면 차단이 해제됩니다. 이 경우 PTO 타이머를 재장전해야 합니다.

온데이터그램수신에 대한 의사 코드는 다음과 같습니다:

```
온데이터그램 수신(데이터그램):
  // 이 데이터 그램이 서버의 차단을 해제하면
  // 교착 상태를 피하기 위한 PTO 타이머.
  (서버가 증폭 방지 한계에 도달한 경우):
    SetLossDetectionTimer()
    loss_detection_timer.timeout < now():
      // 만료되었을 경우 PTO를 실행합니다.
      // 증폭 제한이 적용되는 동안 OnLossDetectionTimeout()
```

A.7. 승인 수신 시

ACK 프레임이 수신되면 원하는 수의 패킷을 새로 승인할 수 있습니다.

OnAckReceived 및 UpdateRtt의 의사 코드는 다음과 같습니다:

```
IncludesAckEliciting(packets):
  패킷의 패킷에 대해:
    if (packet.ack_eliciting):
      return true
  거짓을 반환합니다.

OnAckReceived(ack, pn_space):
  if (largest_acked_packet[pn_space] == 무한): largest_acked_packet[pn_space] =
    ack.largest_acked
  else:
    largest_acked_packet[pn_space] =
        max(largest_acked_packet[pn_space], ack.largest_acked)

  // DetectAndRemoveAckedPackets는 새로 수신된 패킷을 찾습니다.
  // 승인하고 보낸_패킷에서 제거합니다. newly_acked_packets =
      탐지 및 제거된 패킷(ack, pn_space)
  // 새로 수신된 패킷이 없으면 할 일이 없습니다. if
  (newly_acked_packets.empty()):
    반환

  // 가장 크게 승인된 것이 새로 승인된 경우 RTT를 업데이트합니다.
  // 그리고 적어도 하나의 응답이 새로 발생했습니다. if
  (newly_acked_packets.largest().packet_number ==)
          ack.largest_acked &&
      IncludesAckEliciting(newly_acked_packets)):
    최신_rtt =
      now() - newly_acked_packets.largest().time_sent
    UpdateRtt(ack.ack_delay)

  // ECN 정보가 있으면 처리합니다. if (ACK 프레
  임에 ECN 정보가 포함됨):
      ProcessECN(ack, pn_space)

  lost_packets = DetectAndRemoveLostPackets(pn_space)
  if (!lost_packets.empty()):
    OnPacketsLost(lost_packets)
  OnPacketsAcked(newly_acked_packets)

  // 클라이언트가 확실하지 않은 경우 pto_count를 재설정합니다.
  // 서버가 클라이언트 주소의 유효성을 검사했습니다. if
  (PeerCompletedAddressValidation()):
    pto_count = 0
  SetLossDetectionTimer()


UpdateRtt(ack_delay):
  if (first_rtt_sample == 0):
    min_rtt = latest_rtt
    smoothed_rtt = latest_rtt
    rttvar = latest_rtt / 2
    first_rtt_sample = now()
    return
```

```
// min_rtt는 승인 지연을 무시합니다. min_rtt
= min(min_rtt, latest_rtt)
// 핸드셰이크 후 최대_ack_delay만큼 ack_delay를 제한합니다.
```

```
// 확인.
if(핸드셰이크가 확인됨):
  ack_delay = min(ack_delay, max_ack_delay)

// 승인 지연에 대한 조정이 필요한 경우 조정된_rtt = 최신_rtt입니
다.
IF (LATEST_RTT >= MIN_RTT + AK_DELAY):
  ADJUSTED_RTT = LATEST_RTT - AK_DELAY

RTTVAR = 3/4 * RTTVAR + 1/4 * ABS(SMOOTHED_RTT - ADJUSTED_RTT)
SMOOTHED_RTT = 7/8 * SMOOTHED_RTT + 1/8 * ADJUSTED_RTT
```

A.8. 손실 감지 타이머 설정

QUIC 손실 감지는 모든 타임아웃 손실 감지에 단일 타이머를 사용합니다. 타이머의 지속 시간은 아래 패킷 및 타이머 이벤트에 설정된 타이머 모드에 따라 달라집니다. 아래에 정의된 SetLossDetectionTimer 함수는 단일 타이머가 설정되는 방법을 보여줍니다.

이 알고리즘은 특히 타이머가 늦게 깨어나는 경우 타이머가 과거에 설정되는 결과를 초래할 수 있습니다. 과거에 설정된 타이머는 즉시 실행됩니다.

SetLossDetectionTimer의 의사 코드는 다음과 같습니다(여기서 "^" 연산자는 지수화를 나타냅니다):

```
GetLossTimeAndSpace():
  시간 = 손실_시간[초기] 공간 = 초기
  핸드셰이크, 애플리케이션데이터 ]의 pn_space에 대해: if (
    시간 == 0 || 손실_시간[pn_space] < 시간):
      시간 = 손실시간[PN_공간]; 공간 = PN_
      공간
  반환 시간, 공간

GetPtoTimeAndSpace():
  duration = (smoothed_rtt + max(4 * rttvar, kGranularity))
      * (2 ^ pto_count)
  // (비행 중 응답을 유도하는 패킷이 없는 경우) 현재 시간부터
  교착 방지 PTO가 시작됩니다:
    (핸드셰이크 키가 있는 경우)
    assert(!PeerCompletedAddressValidation()):
      반환 (now() + 기간), Handshake else:
      반환 (now() + 기간), 초기 pto_timeout
  = 무한대
  pto_space = 초기
  초기, 핸드셰이크, 애플리케이션데이터 ]의 스페이스에 대해: if (스페
    이스에서 비행 중인 응답을 유도하는 패킷이 없음):
        계속합니다;
    if (공간 == ApplicationData):
      // 핸드셰이크가 확인될 때까지 애플리케이션 데이터 건너뛰기 if (
      핸드셰이크가 확인되지 않음):
        반환 위치_시간 초과, 위치_공간
      // 애플리케이션 데이터에 대한 max_ack_delay 및 백오프를 포함합니다.
      duration += max_ack_delay * (2 ^ pto_count)

    t = time_of_last_ack_eliciting_packet[space] + duration
    if (t < pto_timeout):
      pto_timeout = t
      pto_space = space
  반환 위치_시간 초과, 위치_공간

피어완료주소검증():
  // 클라이언트가 서버의 주소를 암시적으로 검증한다고 가정합니다. if (엔드포
  인트가 서버):
    참을 반환합니다.
  // 서버는 다음과 같은 경우 주소 유효성 검사를 완료합니다.
  // 보호된 패킷이 수신되었습니다. 반환은 핸드셰
  이크 확인을 받았습니다 ||.
       악수 확인

SetLossDetectionTimer():
  earliest_loss_time, _ = GetLossTimeAndSpace()
```

```
if (earliest_loss_time != 0):
  // 시간 임계값 손실 감지.
  loss_detection_timer.update(earliest_loss_time) 반환

경우(서버가 증폭 방지 한계에 도달한 경우):
  // 아무것도 전송할 수 없으면 서버의 타이머가 설정되지 않습니다. 손실 감지 타이머 취소()
  반환
```

```
  if (비행 중 응답을 유도하는 패킷 없음 &&
      PeerCompletedAddressValidation()):
    // 분실을 감지할 수 없으므로 타이머가 설정되지 않습니다.
    // 그러나 클라이언트는 타이머를 활성화해야 합니다.
    // 서버가 증폭 방지 제한에 의해 차단될 수 있습니다. 손실 감지 타이머.취소()
    반환

  시간 초과, _ = GetPtoTimeAndSpace()
  loss_detection_timer.update(timeout)
```

A.9. 시간 초과 시

손실 감지 타이머가 만료되면 타이머의 모드에 따라 수행될 작업이 결정됩니다. OnLossDetectionTimeout의 의

사 코드는 다음과 같습니다:

```
온로스감지타임아웃():
  earliest_loss_time, pn_space = GetLossTimeAndSpace()
  if (earliest_loss_time != 0):
    // 시간 임계값 손실 감지
    lost_packets = DetectAndRemoveLostPackets(pn_space)
    assert(!lost_packets.empty())
    OnPacketsLost(lost_packets)
    SetLossDetectionTimer()
    를 반환합니다.

  if (비행 중 응답을 유도하는 패킷이 없음): assert(!PeerCompletedAddressValidation())
    // 클라이언트가 교착 상태 방지 패킷을 전송합니다: 이니셜이 패딩됨
    //를 입력하면 증폭 방지 크레딧을 더 많이 받을 수 있습니다,
    // 핸드셰이크 패킷은 주소 소유권을 증명합니다. if (핸드셰이
    크 키가 있습니다):
      SendOneAckElicitingHandshakePacket()
    else:
      SendOneAckElicitingPaddedInitialPacket()
  else:
    // PTO. 가능한 경우 새 데이터를 보내고, 그렇지 않으면 이전 데이터를 재전송합니다.
    // 둘 다 사용할 수 없는 경우 단일 PING 프레임을 보냅니다.
    _, pn_space = GetPtoTimeAndSpace()
    SendOneOrTwoAckElicitingPackets(pn_space)

  pto_count++
  SetLossDetectionTimer()
```

A.10. 손실된 패킷 감지

DetectAndRemoveLostPackets는 ACK가 수신되거나 시간 임계값 손실 감지 타이머가 만료될 때마다 호출됩니

다. 이 함수는 해당 패킷 번호 공간에 대한 보낸_패킷에서 작동하며 새로 손실된 것으로 감지된 패킷 목록을 반환합니다.

DetectAndRemoveLostPackets의 의사 코드는 다음과 같습니다:

```
DetectAndRemoveLostPackets(pn_space):
  assert(largest_acked_packet[pn_space] != 무한)
  loss_time[pn_space] = 0
  lost_packets = []
  loss_delay = kTimeThreshold * max(latest_rtt, smoothed_rtt)

  // 패킷이 손실된 것으로 간주되기 전 kGranularity의 최소 시간 손실_지연 =
  max(loss_delay, kGranularity)

  // 이 시간 이전에 전송된 패킷은 손실된 것으로 간주됩니다.
  lost_send_time = now() - loss_delay

  보낸_패킷[PN_SPACE]에서 언락 해제합니다:
    if (unacked.packet_number > largest_acked_packet[pn_space]):
      continue

    // 패킷을 손실된 것으로 표시하거나 표시해야 하는 시간을 설정합니다.
    // 참고: 여기서 kPacketThreshold를 사용하려면 다음이 있다고 가정합니다.
    // 패킷 번호 공간에 발신자로 인한 간격이 없는 경우
    (unacked.time_sent <= lost_send_time ||
        largest_acked_packet[pn_space] >=
          unacked.packet_number + kPacketThreshold):
      보낸_패킷[pn_space].remove(unacked.packet_number) 잃어버린_패킷.삽입(unacked)
    else:
      if (loss_time[pn_space] == 0):
        loss_time[pn_space] = unacked.time_sent + loss_delay
      else:
        loss_time[pn_space] = min(loss_time[pn_space]),
                                  unacked.time_sent + loss_delay)
  반환된_패킷
```

A.11. 이니셜 또는 핸드셰이크 키 드롭 시

이니셜 키 또는 핸드셰이크 키가 삭제되면 해당 공간의 패킷이 삭제되고 손실 감지 상태가 업데이트됩니다.

OnPacketNumberSpaceDiscarded의 의사 코드는 다음과 같습니다:

```
OnPacketNumberSpaceDiscarded(pn_space):
  assert(pn_space != ApplicationData)
  RemoveFromBytesInFlight(sent_packets[pn_space])
  sent_packets[pn_space].clear()
  // 손실 감지 및 PTO 타이머 재설정 time_of_last_ack_eliciting_packet[pn_space] = 0
  손실 시간[PN_공간] = 0
  pto_count = 0
  SetLossDetectionTimer()
```

부록 B. 혼잡 제어 의사 코드

이제 섹션 7에서 설명한 혼잡도 컨트롤러의 구현 예시를 설명하겠습니다.

이 섹션의 의사 코드 세그먼트는 코드 구성 요소로 라이선스가 부여되어 있습니다(저작권 공지를 참조하세요).

B.1. 관심 상수

혼잡 제어에 사용되는 상수는 RFC, 논문 및 일반적인 관행의 조합을 기반으로 합니다.

kInitialWindow: 7.2절에 설명된 대로 비행 중 초기 바이트에 대한 기본 제한입니다.

kMinimumWindow: 7.2절에 설명된 바이트 단위의 최소 혼잡 창입니다.

kLossReductionFactor: 새로운 손실이 발생할 때 혼잡 창을 줄이기 위해 적용되는 스케일링 계수입니다.
이벤트가 감지됩니다. 섹션 7에서는 0.5 값을 권장합니다.

kPersistentCongestionThreshold: 지속적인 혼잡이 설정될 때까지의 기간입니다,
를 PTO 승수로 지정합니다. 섹션 7.6에서는 3을 권장합니다.

B.2. 관심 변수

이 섹션에서는 혼잡 제어 메커니즘을 구현하는 데 필요한 변수에 대해 설명합니다.

최대_데이터그램_크기: 발신자의 현재 최대 페이로드 크기입니다. 여기에는 UDP 또는
IP 오버헤드. 최대 데이터그램 크기는 혼잡 구간 계산에 사용됩니다. 엔드포인트는 경로 최대 전송 단위
(PMTU, [QUIC-TRANSPORT] 14.2절 참조)에 따라 이 변수의 값을 설정하며, 최소값은 1200바이트입니다.

ecn_ce_counters[kPacketNumberSpace]: ECN-CE 카운터에 대해 보고된 최고 값입니다.
ACK 프레임에서 피어에 의한 패킷 번호 공간입니다. 이 값은 보고된 ECN-CE 카운터의 증가를 감지하는 데 사용됩니다.

bytes_in_flight: 적어도 하나의 응답을 포함하는 모든 전송 패킷의 바이트 단위 크기 합계입니다.
프레임을 유도하거나 패딩하고 승인되지 않았거나 손실된 것으로 선언되지 않은 경우입니다. 이 크기에는 IP 또는 UDP 오버헤드는 포함되지 않지만 QUIC 헤더와 연결된 데이터로 인증된 암호화(AEAD) 오버헤드는 포함됩니다. 혼잡 제어가 혼잡 피드백을 방해하지 않도록 하기 위해 ACK 프레임만 포함된 패킷은 bytes_in_flight에 포함되지 않습니다.

congestion_window: 비행 중 허용되는 최대 바이트 수입니다.

congestion_recovery_start_time: 현재 복구가 시작된 시간으로
손실 또는 ECN을 감지합니다. 이 시간 이후에 전송된 패킷이 확인되면 QUIC은 혼잡 회복을 종료합니다.

ssthresh: 슬로우 스타트 임계값(바이트 단위)입니다. 혼잡 창이 ssthresh보다 낮으면
모드는 느린 시작이며 창은 승인된 바이트 수만큼 커집니다.

또한 혼잡 제어 의사 코드는 손실 복구 의사 코드의 일부 변수에 액세스합니다.

B.3. 초기화

연결이 시작될 때 다음과 같이 혼잡 제어 변수를 초기화합니다:

```
congestion_window = kInitialWindow
bytes_in_flight = 0
혼잡_복구_시작 시간 = 0 ssthresh = 무한대
초기, 핸드셰이크, 애플리케이션데이터 ]의 pn_space에 대해: ecn_ce_counters[pn_space] = 0
```

B.4. 패킷 전송 시

패킷이 전송될 때마다 패킷에 비ACK 프레임이 포함되면 패킷은 bytes_in_flight를 증가시킵니다.

```
OnPacketSentCC(보낸_바이트):
  bytes_in_flight += 보낸_바이트
```

B.5. 패킷 승인 시

이는 손실 감지의 OnAckReceived에서 호출되며 보낸_패킷의 새로운 acked_packets과 함께 제공됩니다.

혼잡 회피에서 congestion_window에 정수 표현을 사용하는 구현자는 나누기에 주의해야 하며, [RFC3465] 섹션 2.1에서 제안한 대체 접근 방식을 사용할 수 있습니다.

```
InCongestionRecovery(sent_time):
  반환 전송 시간 <= 혼잡 복구 시작 시간

OnPacketsAcked(acked_packets):
  acked_packets의 acked_packet에 대해:
    OnPacketAcked(acked_packet)

OnPacketAcked(acked_packet):
  if (!acked_packet.in_flight): 반환
    합니다;
  // bytes_in_flight에서 제거. bytes_in_flight
  -= acked_packet.sent_bytes
  // 애플리케이션의 경우 congestion_window를 늘리지 마십시오.
  // 제한 또는 흐름 제어 제한. if
  (IsAppOrFlowControlLimited())
    반환
  // 복구 기간에 혼잡 창을 늘리지 마십시오. if
  (InCongestionRecovery(acked_packet.time_sent)):
    반환
  if (congestion_window < ssthresh):
    // 느리게 시작.
    congestion_window += acked_packet.sent_bytes
  else:
    // 혼잡 회피. 혼잡_창 +=
      최대 데이터그램_크기 * acked_packet.sent_bytes
      / 혼잡_창
```

B.6. 새로운 혼잡 이벤트

이것은 새로운 정체 이벤트가 감지될 때 ProcessECN과 OnPacketsLost에서 호출됩니다. 아직 복구 중이 아니라면 복구 기간을 시작하고 느린 시작 임계값과 혼잡 기간을 즉시 줄입니다.

```
온정 이벤트(보낸_시간):
  // 이미 복구 기간 중이라면 반응이 없습니다. if
  (InCongestionRecovery(sent_time)):
    반환

  // 복구 기간 입력. congestion_recovery_start_time =
  now()
  ssthresh = 혼잡도_창 * kLossReductionFactor 혼잡도_창 =
  max(ssthresh, kMinimumWindow)
  // 패킷을 전송하여 손실 복구 속도를 높일 수 있습니다.
  MaybeSendOnePacket()
```

B.7. ECN 정보 처리

ECN 섹션이 포함된 ACK 프레임이 피어에서 수신될 때 호출됩니다.

```
ProcessECN(ack, pn_space):
  // 피어에서 보고한 ECN-CE 카운터가 증가한 경우,
  // 이것은 새로운 혼잡 이벤트일 수 있습니다.
  if (ack.ce_counter > ecn_ce_counters[pn_space]):
    ecn_ce_counters[pn_space] = ack.ce_counter
    sent_time = sent_packets[ack.largest_acked].time_sent
    OnCongestionEvent(sent_time)
```

B.8. 패킷 손실 시

DetectAndRemoveLostPackets가 패킷이 손실된 것으로 간주할 때 호출됩니다.

```
OnPacketsLost(lost_packets):
  sent_time_of_last_loss = 0
  // bytes_in_flight에서 손실된 패킷을 제거합니다.
  잃어버린_패킷의 잃어버린_패킷에 대해:
    if_packet.in_flight:
      bytes_in_flight - = lost_packet.sent_bytes
      sent_time_of_last_loss =
        최대(보낸_시간_오후_손실, 잃어버린_패킷.시간_보내짐)
  // 비행 중 패킷이 손실된 경우 혼잡 이벤트 if
  (sent_time_of_last_loss != 0):
    온정 이벤트(보낸_시간_오후_손실)

  // 이 손실이 발생하면 혼잡 기간을 초기화합니다.
  // 패킷은 지속적인 정체를 나타냅니다.
  // RTT 샘플을 받은 후 전송된 패킷만 고려합니다. if
  (first_rtt_sample == 0):
    반환 pc_lost
  = []
  잃어버린_패킷의 경우:
    if lost.time_sent > first_rtt_sample:
      pc_lost.insert(lost)
  if (InPersistentCongestion(pc_lost)):
    congestion_window = kMinimumWindow
    congestion_recovery_start_time = 0
```

B.9. 비행 중 바이트에서 버려진 패킷 제거하기

초기 키 또는 핸드셰이크 키가 삭제되면 해당 공간에서 전송된 패킷은 더 이상 비행 중 바이트 수에 포함되지 않습니

다.

RemoveFromBytesInFlight의 의사 코드는 다음과 같습니다:

```
RemoveFromBytesInFlight(discarded_packets):
  // 승인되지 않은 패킷을 비행에서 제거합니다. 폐기된_패킷에서 패킷
  을 찾습니다:
    if packet.in_flight
      bytes_in_flight -= size
```

기여자

IETF QUIC 워킹 그룹은 많은 사람들로부터 엄청난 양의 지원을 받았습니다. 다음 사람들이 이 문서에 실질적인 기여를 해주었습니다:

- 알레산드로 게디니
- 벤자민 손더스
- 고리 페어허스트
- 山本和彦 (야마모토 카즈)
- 奥 一穂 (오쿠 카즈호)
- 라스 에거트
- 매그너스 웨스터룬드
- 마르텐 시만
- 마틴 듀크
- 마틴 톰슨
- 미르야 퀼레빈트
- 닉 뱅크스
- 프라빈 발라수브라마니안

작성자 주소

자나 아이엔가(EDIToR)
빠르게
이메일: jri.ietf@gmail.com

이안 스윗(EDIToR)
Google
이메일: ianswett@google.com