RFC 5681

TCP Congestion Control

문서 최종 수정일	2020-06-27
원문 복사일	2020-06-15
번역 및 정리	이병록(roka88)
이메일	roka88.dev@gmail.com

DRAFT STANDARD
Errata Exist

Network Working Group
Request for Comments: 5681
Obsoletes: 2581
Category: Standards Track

M. Allman
V. Paxson
ICSI
E. Blanton
Purdue University
September 2009

TCP Congestion Control

Abstract

This document defines TCP's four intertwined congestion control algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery. In addition, the document specifies how TCP should begin transmission after a relatively long idle period, as well as discussing various acknowledgment generation methods. This document obsoletes RFC 2581.

이 문서는 TCP의 네 가지의 밀접하게 관련된 혼잡 제어 알고리즘, 즉 느린 시작, 혼잡 회피, 빠른 재전송, 빠른 회복 등을 정의한다. 또한, 이 문서는 TCP가 비교적 긴 유휴 기간 후에 전송을 어떻게 시작해야 하는지를 명시하고 있으며, 다양한 확인 응답 생성 방법에 대해서도 논의한다. 이 문서는 RFC 2581을 폐기한다.

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the “Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

이 문서는 인터넷 커뮤니티를 위한 인터넷 표준 트랙 프로토콜을 지정하고, 개선을 위한 논의와 제안을 요청한다. 이 프로토콜의 표준화 형태 및 상태는 "Internet Official Protocol Standards” (STD 1) 최신판을 참조한다. 이 메모의 배포는 제한되어 있지 않다.

Copyright Notice

수정할 수있는 권한을 IETF 신뢰에 부여하지 않았을 수 있다. IETF 표준 프로세스 외부. 해당 자료의 저작권을 관리하는 개인으로부터 적절한 라이센스를 취득하지 않으면 이 문서는 IETF 표준 프로세스 외부에서 수정 될 수 없으며, 문서의 파생물은 RFC로 출판하기 위해 포맷하거나 영어 이외의 언어로 번역하는 것을 제외하고는 IETF 표준 프로세스 외부에서 생성되지 않을 수 있다.

Table Of Contents

1. Introduction

This document specifies four TCP [RFC793] congestion control algorithms: slow start, congestion avoidance, fast retransmit and fast recovery. These algorithms were devised in [Jac88] and [Jac90]. Their use with TCP is standardized in [RFC1122]. Additional early work in additive-increase, multiplicative-decrease congestion control is given in [CJ89].

이 문서는 느린 시작, 혼잡 회피, 빠른 재전송 및 빠른 회복의 네 가지 TCP [RFC793] 혼잡 제어 알고리즘을 지정한다. 이러한 알고리즘은 [Jac88]과 [Jac90]에서 고안되었다. 그것들의 TCP에서의 사용은 [RFC1122]에서 표준화되었다. 가산-증가, 지수-감소 혼잡 제어의 초기 추가 작업은 [CJ89]에 제시되어 있다.

Note that [Ste94] provides examples of these algorithms in action and [WS95] provides an explanation of the source code for the BSD implementation of these algorithms.

[Ste94]는 실제로 이러한 알고리즘의 예를 제공하며 [WS95]는 이러한 알고리즘의 BSD 구현을 위한 소스 코드에 대한 설명을 제공한다.

In addition to specifying these congestion control algorithms, this document specifies what TCP connections should do after a relatively long idle period, as well as specifying and clarifying some of the issues pertaining to TCP ACK generation.

이 문서는 이러한 혼잡 제어 알고리즘을 명시하는 것 외에도, TCP ACK 생성과 관련된 몇 가지 이슈를 명시하고 명확히 할 뿐만 아니라, 비교적 긴 유휴 기간 후에 TCP 커넥션이 무엇을 해야 하는지를 명시한다.

This document obsoletes [RFC2581], which in turn obsoleted [RFC2001].

이 문서는 [RFC2001]를 폐기한 [RFC2581]을 폐기한다.

This document is organized as follows. Section 2 provides various definitions that will be used throughout the document. Section 3 provides a specification of the congestion control algorithms. Section 4 outlines concerns related to the congestion control algorithms and finally, section 5 outlines security considerations.

이 문서는 다음과 같이 정리되어 있다. Section 2는 문서 전체에 사용될 다양한 정의를 제공한다. Section 3은 혼잡 제어 알고리즘의 사양을 제공한다. Section 4에서는 혼잡 제어 알고리즘과 관련된 우려를 간략하게 설명하고 마지막으로 Section 5에서는 보안 고려사항을 간략하게 설명한다.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

본 문서의 핵심 단어 "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", “SHOULD NOT", “RECOMMENDED", "MAY", “OPTIONAL"은 [RFC2119]에 기술된 바와 같이 해석되어야 한다.

2. Definitions

This section provides the definition of several terms that will be used throughout the remainder of this document.

이 섹션은 이 문서의 내에 사용될 몇 가지 용어의 정의를 제공한다.

SEGMENT: A segment is ANY TCP/IP data or acknowledgment packet (or both).

세그먼트: 세그먼트는 TCP/IP 데이터 또는 확인 응답 패킷(또는 둘 다)이다.

SENDER MAXIMUM SEGMENT SIZE (SMSS): The SMSS is the size of the largest segment that the sender can transmit. This value can be based on the maximum transmission unit of the network, the path MTU discovery [RFC1191, RFC4821] algorithm, RMSS (see next item), or other factors. The size does not include the TCP/IP headers and options.

송신자 최대 세그먼트 크기 (SMSS): SMSS는 송신자가 전송할 수 있는 가장 큰 세그먼트의 크기다. 이 값은 네트워크의 최대 전송 단위, Path MTU Discovery [RFC1191, RFC4821] 알고리즘, RMSS(다음 항목 참조) 또는 기타 요인에 기반할 수 있다. 세그먼트 크기에는 TCP/IP 헤더와 옵션이 포함되지 않는다.

RECEIVER MAXIMUM SEGMENT SIZE (RMSS): The RMSS is the size of the largest segment the receiver is willing to accept. This is the value specified in the MSS option sent by the receiver during connection startup. Or, if the MSS option is not used, it is 536 bytes [RFC1122]. The size does not include the TCP/IP headers and options.

수신자 최대 세그먼트 크기 (RMSS): RMSS는 수신자가 받아들이려고 하는 가장 큰 세그먼트의 크기다. 커넥션 시작 시 수신자가 송신한 MSS 옵션에 지정한 값이다. 또는 MSS 옵션을 사용하지 않을 경우 536 바이트 [RFC1122]가 된다. 세그먼트 크기에는 TCP/IP 헤더와 옵션이 포함되지 않는다.

FULL-SIZED SEGMENT: A segment that contains the maximum number of data bytes permitted (i.e., a segment containing SMSS bytes of data).

풀 사이즈 세그먼트: 허용되는 최대 데이터 바이트 수를 포함하는 세그먼트이다.(즉, 데이터의 SMSS 바이트를 포함하는 세그먼트)

RECEIVER WINDOW (rwnd): The most recently advertised receiver window.

수신자 윈도우 (rwnd): 가장 최근에 알려진 수신자 윈도우이다.

CONGESTION WINDOW (cwnd): A TCP state variable that limits the amount of data a TCP can send. At any given time, a TCP MUST NOT send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of cwnd and rwnd.

혼잡 윈도우 (cwnd): TCP가 전송할 수 있는 데이터 양을 제한하는 TCP 상태 변수. 지정된 시간에, TCP는 확인된 가장 높은 시퀀스 번호의 합과 cwnd 및 rwnd의 최소값 보다 높은 시퀀스 번호를 가진 데이터를 전송해서는 안 된다.(MUST NOT)

INITIAL WINDOW (IW): The initial window is the size of the sender's congestion window after the three-way handshake is completed.

초기 윈도우 (IW): 초기 윈도우는 세-방향 핸드셰이크가 완료된 후의 송신자의 혼잡 윈도우 크기다.

LOSS WINDOW (LW): The loss window is the size of the congestion window after a TCP sender detects loss using its retransmission timer.

손실 윈도우 (LW): 손실 윈도우는 TCP 송신자가 재전송 타이머를 사용하여 손실을 감지한 후의 혼잡 윈도우 크기다.

RESTART WINDOW (RW): The restart window is the size of the congestion window after a TCP restarts transmission after an idle period (if the slow start algorithm is used; see section 4.1 for more discussion).

재시작 윈도우 (RW): 재시작 윈도우는 유휴 기간 후에 TCP가 전송을 재시작한 후 혼잡 윈도우의 크기를 말한다(느린 시작 알고리즘을 사용하는 경우, 자세한 내용은 Section 4.1 참조).

FLIGHT SIZE: The amount of data that has been sent but not yet cumulatively acknowledged.

FLIGHT SIZE: 전송되었지만 아직 확인 응답되지 않은 누적된 데이터의 양.

DUPLICATE ACKNOWLEDGMENT: An acknowledgment is considered a "duplicate" in the following algorithms when (a) the receiver of the ACK has outstanding data, (b) the incoming acknowledgment carries no data, (c) the SYN and FIN bits are both off, (d) the acknowledgment number is equal to the greatest acknowledgment received on the given connection (TCP.UNA from [RFC793]) and (e) the advertised window in the incoming acknowledgment equals the advertised window in the last incoming acknowledgment.

중복 확인 응답: 확인 응답은 다음의 알고리즘을 통해 "중복"으로 간주한다. (a) 수신자의 ACK에 처리되지 않은 데이터가 있고, (b) 들어오는 ACK 데이터가 없고, (c) SYN과 FIN의 플래그 비트가 모두 꺼져 있고, (d) 확인 응답 번호는 해당 커넥션에 대해 수신된 최대 수신 확인 번호와 동일하며 (TCP.UNA from [RFC793]) (e) 들어오는 확인 응답을 통해 알려진 윈도우와 마지막 확인 응답을 통해 알려진 윈도우와 동일하다.

Alternatively, a TCP that utilizes selective acknowledgments (SACKs) [RFC2018, RFC2883] can leverage the SACK information to determine when an incoming ACK is a "duplicate" (e.g., if the ACK contains previously unknown SACK information).

대안으로, 선택적 확인 (SACKs)[RFC2018, RFC2883]을 활용하는 TCP는 SACK 정보를 활용하여 들어오는 ACK가 "중복"인 경우(e.g., ACK에 이전에 알려지지 않은 SACK 정보가 포함되어 있는 경우)를 결정할 수 있다.

3. Congestion Control Algorithms

This section defines the four congestion control algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery, developed in [Jac88] and [Jac90]. In some situations, it may be beneficial for a TCP sender to be more conservative than the algorithms allow; however, a TCP MUST NOT be more aggressive than the following algorithms allow (that is, MUST NOT send data when the value of cwnd computed by the following algorithms would not allow the data to be sent).

이 섹션에서는 [Jac88] 및 [Jac90]에서 개발된 느린 시작, 혼잡 회피, 빠른 재전송 및 빠른 회복의 네 가지 혼잡 제어 알고리즘을 정의한다. 어떤 상황에서는, TCP 송신자의 알고리즘이 허용하는 것보다 더 보수적인 것이 유익할 수 있지만, TCP는 다음 알고리즘이 허용하는 것보다

더 위협적이지 않아야 한다.(MUST NOT) (즉, 다음 알고리즘에 의해 계산된 cwnd 값이 데이터 전송을 허용하지 않을 때는 데이터를 전송해서는 안 된다.(MUST NOT))

Also, note that the algorithms specified in this document work in terms of using loss as the signal of congestion. Explicit Congestion Notification (ECN) could also be used as specified in [RFC3168].

또한 이 문서에 명시된 알고리즘은 혼잡의 신호로 손실을 사용하는 측면으로 작동한다는 점에 유의한다. 명시적 혼잡 알림(ECN)도 [RFC3168]에 명시된 대로 사용할 수 있다.

3.1. Slow Start and Congestion Avoidance

The slow start and congestion avoidance algorithms MUST be used by a TCP sender to control the amount of outstanding data being injected into the network. To implement these algorithms, two variables are added to the TCP per-connection state. The congestion window (cwnd) is a sender-side limit on the amount of data the sender can transmit into the network before receiving an acknowledgment (ACK), while the receiver's advertised window (rwnd) is a receiver-side limit on the amount of outstanding data. The minimum of cwnd and rwnd governs data transmission.

느린 시작과 혼잡 회피 알고리즘은 네트워크에 주입되는 미처리 데이터의 양을 제어하기 위해 TCP 송신자에 의해 사용되어야 한다.(MUST) 이러한 알고리즘을 구현하기 위해 두 개의 변수를 커넥션당 TCP 상태에 추가한다. 혼잡 윈도우(cwnd)는 수신자가 확인 응답(ACK)을 받기 전에 송신자가 네트워크로 전송할 수 있는 데이터 양에 대한 송신자 측 제한이며, 수신자의 알려진 윈도우(rwnd)는 처리되지 않은 데이터의 양에 대한 수신자 측 제한이다. cwnd 및 rwnd 최소값이 데이터 전송을 제어한다.

Another state variable, the slow start threshold (ssthresh), is used to determine whether the slow start or congestion avoidance algorithm is used to control data transmission, as discussed below.

또 다른 상태 변수인 느린 시작 임계값(ssthresh)은 아래에서 설명한 바와 같이 느린 시작 또는 혼잡 회피 알고리즘을 사용하여 데이터 전송을 제어하는지 여부를 결정하는 데 사용된다.

Beginning transmission into a network with unknown conditions requires TCP to slowly probe the network to determine the available capacity, in order to avoid

congesting the network with an inappropriately large burst of data. The slow start algorithm is used for this purpose at the beginning of a transfer, or after repairing loss detected by the retransmission timer. Slow start additionally serves to start the "ACK clock" used by the TCP sender to release data into the network in the slow start, congestion avoidance, and loss recovery algorithms.

알려지지 않은 조건을 가진 네트워크로 전송을 시작하려면, TCP가 부적절하게 큰 데이터가 차 있는것으로 네트워크가 혼잡해지는 것을 피하기 위한 이용 가능한 용량을 결정하기 위해 네트워크를 천천히 탐색해야 한다. 느린 시작 알고리즘은 전송을 시작할 때 또는 재전송 타이머에 의해 감지된 손실을 회복한 후에 이 목적으로 사용된다. 느린 시작은 추가적으로 TCP 송신자가 사용하는 "ACK 클럭"을 시작하여 느린 시작, 혼잡 회피 및 손실 회복 알고리즘에서 네트워크로 데이터를 방출하는 역할을 한다.

IW, the initial value of cwnd, MUST be set using the following guidelines as an upper bound.

cwnd의 초기 값인 IW는 반드시 다음 지침을 상한으로 사용하여 설정해야 한다.(MUST)

If SMSS > 2190 bytes:
IW = 2 * SMSS bytes and MUST NOT be more than 2 segments
If (SMSS > 1095 bytes) and (SMSS <= 2190 bytes):
IW = 3 * SMSS bytes and MUST NOT be more than 3 segments
if SMSS <= 1095 bytes:
IW = 4 * SMSS bytes and MUST NOT be more than 4 segments

As specified in [RFC3390], the SYN/ACK and the acknowledgment of the SYN/ACK MUST NOT increase the size of the congestion window. Further, if the SYN or SYN/ACK is lost, the initial window used by a sender after a correctly transmitted SYN MUST be one segment consisting of at most SMSS bytes.

[RFC3390]에 명시된 바와 같이 SYN/ACK와 SYN/ACK의 확인 응답은 혼잡 윈도우 크기를 증가시키지 않아야 한다.(MUST NOT) 또한 SYN 또는 SYN/ACK가 손실된 경우, SYN이 올바르게 전송된 후 송신자가 사용하는 초기 윈도우는 최대 SMSS 바이트로 구성된 하나의 세그먼트여야 한다.(MUST)

A detailed rationale and discussion of the IW setting is provided in [RFC3390].

IW 설정에 대한 상세한 근거와 논의는 [RFC3390]에 제시되어 있다.

When initial congestion windows of more than one segment are implemented along with Path MTU Discovery [RFC1191], and the MSS being used is found to be too large, the congestion window cwnd SHOULD be reduced to prevent large bursts of smaller segments. Specifically, cwnd SHOULD be reduced by the ratio of the old segment size to the new segment size.

Path MTU Discovery [RFC1191]에 따라 둘 이상의 세그먼트의 초기 혼잡 윈도우가 구현되고 사용 중인 MSS가 너무 큰 것으로 확인되면, 더 작은 세그먼트의 큰 버스트를 방지하기 위해 혼잡 윈도우를 줄여야 한다.(SHOULD) 특히, cwnd는 이전 세그먼트 크기 대 새 세그먼트 크기의 비율로 감소해야 한다.(SHOULD)

The initial value of ssthresh SHOULD be set arbitrarily high (e.g., to the size of the largest possible advertised window), but ssthresh MUST be reduced in response to congestion. Setting ssthresh as high as possible allows the network conditions, rather than some arbitrary host limit, to dictate the sending rate. In cases where the end systems have a solid understanding of the network path, more carefully setting the initial ssthresh value may have merit (e.g., such that the end host does not create congestion along the path).

ssthresh의 초기값은 임의로 높게 설정해야 하지만(SHOULD) (e.g., 가능한 가장 큰 알려진 윈도우 크기에 맞춰), ssthresh는 혼잡에 대응하여 감소해야 한다.(MUST) ssthresh를 가능한 높게 설정하면 일부 임의의 호스트의 제한이 아닌 네트워크 조건이 전송 비율을 지시할 수 있다. 엔드 시스템이 네트워크 경로를 확실하게 이해하는 경우, 초기 값을 보다 신중하게 설정하는 것이 장점을 가질 수 있다(e.g., 엔드 호스트가 경로를 따라 혼잡을 일으키지 않도록).

The slow start algorithm is used when cwnd < ssthresh, while the congestion avoidance algorithm is used when cwnd > ssthresh. When cwnd and ssthresh are equal, the sender may use either slow start or congestion avoidance.

느린 시작 알고리즘은 cwnd < ssthresh 일 때, 반면에 혼잡 회피 알고리즘은 cwnd > ssthresh 일 때 사용된다. cwnd와 ssthresh가 동일할 경우, 송신자는 느린 시작 또는 혼잡 회피를 사용할 수 있다.

During slow start, a TCP increments cwnd by at most SMSS bytes for each ACK received that cumulatively acknowledges new data. Slow start ends when cwnd exceeds ssthresh (or, optionally, when it reaches it, as noted above) or when congestion is observed. While traditionally TCP implementations have increased cwnd by precisely SMSS bytes upon receipt of an ACK covering new data, we RECOMMEND that TCP implementations increase cwnd, per:

느린 시작 동안, TCP는 새로운 데이터를 누적으로 확인 응답하는 각 ACK에 대해 최대 SMSS 바이트까지 증가시킨다. 느린 시작은 cwnd가 ssthresh(또는 위에 언급된 대로 선택적으로 도달했을 때)를 초과하거나 혼잡 상태가 관찰될 때 종료된다. 전통적으로 TCP 구현은 새로운 데이터를 포함하는 ACK를 수신한 후 정확히 SMSS 바이트만큼 cwnd를 증가시켰지만, TCP 구현은 다음당 cwnd를 증가시킬 것을 권장한다.(RECOMMEND)

cwnd += min (N, SMSS) (2)

where N is the number of previously unacknowledged bytes acknowledged in the incoming ACK. This adjustment is part of Appropriate Byte Counting [RFC3465] and provides robustness against misbehaving receivers that may attempt to induce a sender to artificially inflate cwnd using a mechanism known as "ACK Division" [SCWA99]. ACK Division consists of a receiver sending multiple ACKs for a single TCP data segment, each acknowledging only a portion of its data. A TCP that increments cwnd by SMSS for each such ACK will inappropriately inflate the amount of data injected into the network.

여기서 N은 들어오는 ACK에서 이전에 확인 응답되지 않은 바이트 수이다. 이 조정은 적절한 바이트 카운팅 [RFC3465]의 일부로서, 송신자가 "ACK 분할" [SCWA99]로 알려진 메커니즘을 사용하여 인위적으로 cwnd를 증가시키도록 유도할 수 있는 잘못된 수신자에 대한 엄격함을 제공한다. ACK 분할은 단일 TCP 데이터 세그먼트에 대해 여러 ACK를 전송하는 수신자로 구성되며, 데이터의 일부만 각각 확인 응답한다. 그러한 각 ACK의 SMSS에 의해 cwnd를 증가시키는 TCP는, 네트워크에 주입되는 데이터의 양을 부적절하게 부풀릴 것이다.

During congestion avoidance, cwnd is incremented by roughly 1 full-sized segment per round-trip time (RTT). Congestion avoidance continues until congestion is detected. The basic guidelines for incrementing cwnd during congestion avoidance are:

혼잡 회피 동안, cwnd는 왕복 시간(RTT)당 약 1개의 풀 사이즈 세그먼트만큼 증가한다. 혼잡 회피는 혼잡이 감지될 때까지 계속된다. 혼잡 회피 시 cwnd 증가를 위한 기본 지침은 다음과 같다.

* MAY increment cwnd by SMSS bytes

* SMSS 바이트로 cwnd가 증가할 수 있다.

* SHOULD increment cwnd per equation (2) once per RTT

* RTT 한 번당 등식 (2)으로 cwnd를 증가해야 한다.

* MUST NOT increment cwnd by more than SMSS bytes

* SMSS 바이트 이상으로 cwnd를 증가시키면 안 된다(MUST NOT)

We note that [RFC3465] allows for cwnd increases of more than SMSS bytes for incoming acknowledgments during slow start on an experimental basis; however, such behavior is not allowed as part of the standard.

우리는 [RFC3465]가 실험적으로 느린 시작 동안 들어오는 확인 응답에 대해 SMSS 바이트 이상의 cwnd 증가를 허용하지만, 그러한 행동은 표준의 일부로 허용되지 않는다는 점에 주목한다.

The RECOMMENDED way to increase cwnd during congestion avoidance is to count the number of bytes that have been acknowledged by ACKs for new data. (A drawback of this implementation is that it requires maintaining an additional state variable.) When the number of bytes acknowledged reaches cwnd, then cwnd can be incremented by up to SMSS bytes. Note that during congestion avoidance, cwnd MUST NOT be increased by more than SMSS bytes per RTT. This method both allows TCPs to increase cwnd by one segment per RTT in the face of delayed ACKs and provides robustness against ACK Division attacks.

혼잡 회피 시 cwnd를 증가시키는 권장하는 방법(RECOMMENDED)은 새로운 데이터의 ACK에 의해 확인 응답받은 바이트 수를 계산하는 것이다. (이 구현의 단점은 추가적인 상태 변수를 유지해야 한다는 것이다.) 확인 응답 바이트 수가 cwnd에 도달하면, cwnd는 최대 SMSS 바이트까지 증가될 수 있다. 혼잡 회피 중에 cwnd는 RTT당 SMSS 바이트 이상 증가해서는 안 된다(MUST NOT)는 점에 유의한다. 이 메서드는 둘 다 지연된 ACK에 직면하여 TCP가 RTT당 하나의 세그먼트씩 cwnd를 증가시킬 수 있도록 하며 ACK 분할 공격에 대한 튼튼함을 제공한다.

Another common formula that a TCP MAY use to update cwnd during congestion avoidance is given in equation (3):

혼잡 회피 동안 TCP가 cwnd를 업데이트하기 위해 사용할 수 있는,(MAY) 또 다른 일반적인 공식은 등식 (3)에 제시되어 있다.

cwnd += SMSS*SMSS/cwnd (3)

This adjustment is executed on every incoming ACK that acknowledges new data. Equation (3) provides an acceptable approximation to the underlying principle of increasing cwnd by 1 full-sized segment per RTT. (Note that for a connection in which the receiver is acknowledging every-other packet, (3) is less aggressive than allowed -- roughly increasing cwnd every second RTT.)

이 조정은 새로운 데이터를 확인 응답 하는 모든 들어오는 ACK에 대해 실행된다. 등식 (3)은 RTT당 하나의 풀 사이즈 세그먼트에 의해 cwnd를 증가시키는 기본 원리에 대한 허용 가능한 근사치를 제공한다(참고, 수신자가 모든 다른 패킷을 확인 응답하는 연결의 경우, (3)은 허용된 것보다 덜 위협적이다 - 대략 매초 RTT당 cwnd를 증가시킨다).

Implementation Note: Since integer arithmetic is usually used in TCP implementations, the formula given in equation (3) can fail to increase cwnd when the congestion window is larger than SMSS*SMSS. If the above formula yields 0, the result SHOULD be rounded up to 1 byte.

구현 참고: 정수 계산은 보통 TCP 구현에 사용되므로, 혼잡 윈도우가 SMSS*SMSS보다 크면 등식 (3)에 주어진 공식은 cwnd를 증가시키지 못할 수 있다. 위의 공식에서 0이 나오면 결과는 1 바이트까지 반올림해야 한다.(SHOULD)

Implementation Note: Older implementations have an additional additive constant on the right-hand side of equation (3). This is incorrect and can actually lead to diminished performance [RFC2525].

구현 참고: 오래된 구현에는 등식 (3)의 우측에 추가적인 가산 상수가 있다. 이는 부정확하며 실제로 성능 저하를 초래할 수 있다 [RFC2525].

Implementation Note: Some implementations maintain cwnd in units of bytes, while others in units of full-sized segments. The latter will find equation (3) difficult to use, and may prefer to use the counting approach discussed in the previous paragraph.

구현 참고: 어떤 구현은 바이트 단위로 cwnd를 유지하고, 다른 구현은 풀 사이즈 세그먼트의 단위로 유지한다. 후자는 등식 (3)을 사용하기 어려울 것이며, 앞 단락에서 논의한 계수 접근법을 사용하는 것을 선호할 수 있다.

When a TCP sender detects segment loss using the retransmission timer and the given segment has not yet been resent by way of the retransmission timer, the value of ssthresh MUST be set to no more than the value given in equation (4):

TCP 송신자가 재전송 타이머를 사용하여 세그먼트 손실을 감지하고, 주어진 세그먼트가 재전송 타이머를 통해 아직 재전송되지 않은 경우, ssthresh의 값은 반드시 등식 (4)에 주어진 값 이하로 설정되어야 한다.(MUST)

ssthresh = max (FlightSize / 2, 2*SMSS) (4)

where, as discussed above, FlightSize is the amount of outstanding data in the network.

여기서, 위에서 논의한 바와 같이, FlightSize는 네트워크에서 미처리 데이터의 양이다.

On the other hand, when a TCP sender detects segment loss using the retransmission timer and the given segment has already been retransmitted by way of the retransmission timer at least once, the value of ssthresh is held constant.

한편, TCP 송신자가 재전송 타이머를 이용하여 세그먼트 손실을 감지하고 주어진 세그먼트가 이미 재전송 타이머를 통해 적어도 한 번 재전송된 경우, 그 값은 일정하게 유지된다.

Implementation Note: An easy mistake to make is to simply use cwnd, rather than FlightSize, which in some implementations may incidentally increase well beyond rwnd.

구현 참고: 쉽게 저지르는 실수는 FlightSize가 아닌 cwnd를 사용하는 것이다. 일부 구현에서는 실수로 인해 rwnd를 훨씬 넘어설 수 있다.

Furthermore, upon a timeout (as specified in [RFC2988]) cwnd MUST be set to no more than the loss window, LW, which equals 1 full-sized segment (regardless of the value of IW). Therefore, after retransmitting the dropped segment the TCP sender uses the slow start algorithm to increase the window from 1 full-sized segment to the new value of ssthresh, at which point congestion avoidance again takes over.

또한 타임아웃 시 ([RFC2988]에 명시된 대로) cwnd는 손실 윈도우 이하(IW 값에 관계없이) 1개의 풀 사이즈 세그먼트와 동일한 LW로 설정해야 한다.(MUST) 따라서, 드랍된 세그먼트를

재전송한 후 TCP 송신자는 느린 시작 알고리즘을 사용하여 윈도우를 하나의 풀 사이즈 세그먼트로 부터 새로운 값으로 증가시키고, 이때 혼잡 회피로 다시 이어진다.

As shown in [FF96] and [RFC3782], slow-start-based loss recovery after a timeout can cause spurious retransmissions that trigger duplicate acknowledgments. The reaction to the arrival of these duplicate ACKs in TCP implementations varies widely. This document does not specify how to treat such acknowledgments, but does note this as an area that may benefit from additional attention, experimentation and specification.

[FF96] 및 [RFC3782]에 표시된 것처럼, 타임아웃 후 느린 시작 기반 손실 회복은 중복된 확인 응답을 트리거하는 가짜 재전송을 유발할 수 있다. 이러한 중복 ACK가 TCP 구현에서 도착하는 것에 대한 반응은 매우 다양하다. 이 문서에서는 이러한 확인 응답을 취급하는 방법을 명시하지 않지만, 추가적인 주의로 실험 및 명세의 혜택을 받을 수 있는 영역이라는 점에 주목한다.

3.2. Fast Retransmit/Fast Recovery

A TCP receiver SHOULD send an immediate duplicate ACK when an out-of-order segment arrives. The purpose of this ACK is to inform the sender that a segment was received out-of-order and which sequence number is expected. From the sender's perspective, duplicate ACKs can be caused by a number of network problems. First, they can be caused by dropped segments. In this case, all segments after the dropped segment will trigger duplicate ACKs until the loss is repaired. Second, duplicate ACKs can be caused by the re-ordering of data segments by the network (not a rare event along some network paths [Pax97]). Finally, duplicate ACKs can be caused by replication of ACK or data segments by the network. In addition, a TCP receiver SHOULD send an immediate ACK when the incoming segment fills in all or part of a gap in the sequence space. This will generate more timely information for a sender recovering from a loss through a retransmission timeout, a fast retransmit, or an advanced loss recovery algorithm, as outlined in section 4.3.

TCP 수신자는 잘못된 세그먼트가 도착하면 즉시 중복 ACK를 전송해야 한다.(SHOULD) 본 ACK의 목적은 송신자에게 세그먼트가 잘못된 상태로 수신되었다는 것과 예상되는 시퀀스 번호를 알리기 위함이다. 송신자의 관점에서 ACK의 중복은 여러 가지 네트워크 문제로 야기될 수 있다. 첫 번째로, 그것들은 드랍된 세그먼트에 의해 발생할 수 있다. 이 경우, 손실된 세그먼트 이후의 모든 세그먼트는 손실이 복구될 때까지 중복 ACK를 트리거한다. 두 번째로, 중복 ACK는 네트워크에 의한 데이터 세그먼트의 재정렬에 의해 발생할 수 있다.(일부 네트워크 경로를

따르는 드문 사건이 아니다.[Pax97]) 마지막으로, 중복 ACK는 네트워크에 의한 ACK 또는 데이터 세그먼트의 복제에 의해 발생할 수 있다. 또한, TCP 수신자는 수신 세그먼트가 시퀀스 공간의 간격 전체 또는 일부를 채울 때 즉시 ACK를 전송해야 한다.(SHOULD) 이렇게 하면 Secion 4.3에서 설명한 대로 재전송 타임아웃, 빠른 재전송 또는 진보된 손실 회복 알고리즘을 통해 손실로부터 복구하는 송신자에게 적절한 정보가 생성될 것이다.

The TCP sender SHOULD use the "fast retransmit" algorithm to detect and repair loss, based on incoming duplicate ACKs. The fast retransmit algorithm uses the arrival of 3 duplicate ACKs (as defined in section 2, without any intervening ACKs which move SND.UNA) as an indication that a segment has been lost. After receiving 3 duplicate ACKs, TCP performs a retransmission of what appears to be the missing segment, without waiting for the retransmission timer to expire.

TCP 송신자는 수신 중복 ACK에 근거하여 손실을 감지하고 복구하기 위해 "빠른 재전송" 알고리즘을 사용해야 한다.(SHOULD) 빠른 재전송 알고리즘은 (SND.UNA를 이동하는 ACK를 방해하지 않고 Section 2에서 정의한 바와 같이) 세그먼트가 손실되었음을 나타내는 것으로 세 개의 중복 ACK의 도착을 사용한다. 세 개의 중복 ACK 받은 TCP는 재전송 타이머가 만료되는 것을 기다리지 않고 누락된 세그먼트로 보이는 것을 재전송 한다.

After the fast retransmit algorithm sends what appears to be the missing segment, the "fast recovery" algorithm governs the transmission of new data until a non-duplicate ACK arrives. The reason for not performing slow start is that the receipt of the duplicate ACKs not only indicates that a segment has been lost, but also that segments are most likely leaving the network (although a massive segment duplication by the network can invalidate this conclusion). In other words, since the receiver can only generate a duplicate ACK when a segment has arrived, that segment has left the network and is in the receiver's buffer, so we know it is no longer consuming network resources. Furthermore, since the ACK “clock" [Jac88] is preserved, the TCP sender can continue to transmit new segments (although transmission must continue using a reduced cwnd, since loss is an indication of congestion).

빠른 재전송 알고리즘이 누락된 세그먼트로 보이는 것을 재전송한 후, "빠른 회복" 알고리즘은 중복되지 않은 ACK가 도착할 때까지 새로운 데이터의 전송을 통제한다. 느린 시작을 수행하지 않는 이유는 중복 ACK의 수신은 세그먼트가 손실되었음을 나타낼 뿐만 아니라 세그먼트가 네트워크를 이탈할 가능성이 가장 높기 때문이다(네트워크에 의한 대규모 세그먼트 중복은 이러한 결과를 무효화할 수 있지만). 즉, 세그먼트가 도착했을 때만 수신자가 중복 ACK를 생성할 수 있기 때문에, 그 세그먼트는 네트워크를 떠나 수신자의 버퍼에 있으므로, 우리는 해당 세그먼트가 더 이상 네트워크 자원을 소비하지 않는다는 것을 알고 있다. 또한 ACK "클럭" [Jac88]이 보존되기 때문에 TCP 송신자는 새로운 세그먼트를 계속 전송할 수 있다.(손실이 혼잡의 표시이기 때문에 전송은 감소된 cwnd를 사용하여 반드시 계속 전송해야 한다.)

The fast retransmit and fast recovery algorithms are implemented together as follows.

빠른 재전송과 빠른 회복 알고리즘은 다음과 같이 함께 구현된다.

1. On the first and second duplicate ACKs received at a sender, a TCP SHOULD send a segment of previously unsent data per [RFC3042] provided that the receiver's advertised window allows, the total FlightSize would remain less than or equal to cwnd plus 2*SMSS, and that new data is available for transmission. Further, the TCP sender MUST NOT change cwnd to reflect these two segments [RFC3042]. Note that a sender using SACK [RFC2018] MUST NOT send new data unless the incoming duplicate acknowledgment contains new SACK information.

1. 송신자에게 수신된 첫 번째와 두 번째 중복 ACK에서, TCP는 수신자에게서 알려진 윈도우가 허용하고, 총 FlightSize가 cwnd + 2*SMSS보다 작거나 같으며, 전송 가능한 새 데이터가 있는 경우, [RFC3042]에 따라 이전에 전송되지 않았던 데이터의 세그먼트를 전송해야 한다.(SHOULD) 또한, TCP 송신자는 이 두 세그먼트를 반영하기 위해 cwnd를 변경해서는 안 된다.(MUST NOT) [RFC3042]. SACK [RFC2018]를 사용하는 송신자는 중복 확인 응답에 새로운 SACK 정보가 포함되지 않는 한 새 데이터를 전송해서는 안 된다(MUST NOT)는 점에 유의한다.

2. When the third duplicate ACK is received, a TCP MUST set ssthresh to no more than the value given in equation (4). When [RFC3042] is in use, additional data sent in limited transmit MUST NOT be included in this calculation.

2. 세 번째 중복 ACK를 수신했을 때, TCP는 ssthresh를 등식 (4)에 주어진 값 이하로 설정해야 한다.(MUST) [RFC3042]를 사용하는 경우, 제한된 전송내에 전송되는 추가 데이터는 이 계산에 포함되지 않아야 한다.(MUST NOT)

3. The lost segment starting at SND.UNA MUST be retransmitted and cwnd set to ssthresh plus 3*SMSS. This artificially “inflates" the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.

3. SND.UNA에서 시작하는 손실 세그먼트는 재전송되어야 하며 ssthresh에 3*SMSS를 더한 값으로 설정되어야 한다.(MUST) 이는 네트워크를 떠난 세그먼트 수(3개)와 수신자가 버퍼된 세그먼트 수만큼 인위적으로 혼잡 윈도우를 "증가"시킨다.

4. For each additional duplicate ACK received (after the third), cwnd MUST be incremented by SMSS. This artificially inflates the congestion window in order to reflect the additional segment that has left the network.

4. 추가적인 중복 ACK가 수신될 때마다(3번째 이후) cwnd는 반드시 SMSS에 의해 증가되어야 한다.(MUST) 이것은 네트워크를 떠난 추가 세그먼트를 반영하기 위해 인위적으로 혼잡 윈도우를 증가시킨다.

Note: [SCWA99] discusses a receiver-based attack whereby many bogus duplicate ACKs are sent to the data sender in order to artificially inflate cwnd and cause a higher than appropriate sending rate to be used. A TCP MAY therefore limit the number of times cwnd is artificially inflated during loss recovery to the number of outstanding segments (or, an approximation thereof).

참고: [SCWA99]는 인위적으로 cwnd를 팽창시키고 적절한 전송 속도보다 높은 전송률을 사용하기 위해 많은 가짜 중복 ACK가 데이터 송신자로 전송되는 수신자-기반 공격에 대해 논의한다. 따라서 TCP는 손실 회복 중에 인위적으로 cwnd가 증가되는 횟수를 미처리 세그먼트의 수(또는 그 근사치)로 제한할 수 있다.(MAY)

Note: When an advanced loss recovery mechanism (such as outlined in section 4.3) is not in use, this increase in FlightSize can cause equation (4) to slightly inflate cwnd and ssthresh, as some of the segments between SND.UNA and SND.NXT are assumed to have left the network but are still reflected in FlightSize.

참고: (Section 4.3에 설명된 것과 같은) 진보된 손실 회복 메커니즘을 사용하지 않을 때, 이러한 FlightSize의 증가는 SND.UNA와 SND.NXT사이의 일부 세그먼트가 네트워크를 떠났다고 가정하지만 FlightSize에 여전히 반영되기 때문에, cwnd와 ssthresh를 등식 (4)로 인해 약간 증가될 수 있다.

5. When previously unsent data is available and the new value of cwnd and the receiver's advertised window allow, a TCP SHOULD send 1*SMSS bytes of previously unsent data.

5. 이전에 전송되지 않은 데이터를 사용할 수 있고 cwnd의 새로운 값과 수신자의 알려진 윈도우가 허용될 때, TCP는 이전에 전송되지 않았던 데이터의 1*SMSS 바이트를 전송해야 한다.(SHOULD)

6. When the next ACK arrives that acknowledges previously unacknowledged data, a TCP MUST set cwnd to ssthresh (the value set in step 2). This is termed "deflating" the window.

6. 이전에 확인 응답되지 않은 데이터를 확인한 다음 ACK가 도착하면, TCP는 반드시 cwnd를 ssthresh(2단계에서 설정한 값)로 설정해야 한다.(MUST) 이것은 윈도우 "감소"라고 불린다.

This ACK should be the acknowledgment elicited by the retransmission from step 3, one RTT after the retransmission (though it may arrive sooner in the presence of significant out-of-order delivery of data segments at the receiver). Additionally, this ACK should acknowledge all the intermediate segments sent between the lost segment and the receipt of the third duplicate ACK, if none of these were lost.

이 ACK는 3단계에서 재전송 후 1개의 RTT로부터 재전송을 통해 유도된 확인 응답이어야 한다. (수신자에게서 유의미한 데이터 세그먼트의 순서 없는 전달이 있을 경우 더 빨리 도착할 수 있음에도) 또한, 이 ACK는 손실된 세그먼트와 세 번째 중복 ACK 수신 사이에 전송된 모든 중간 세그먼트를 확인해야 한다.

Note: This algorithm is known to generally not recover efficiently from multiple losses in a single flight of packets [FF96]. Section 4.3 below addresses such cases.

참고: 이 알고리즘은 일반적으로 단일 패킷 전송에서 여러 손실로부터 효율적으로 회복되지 않는 것으로 알려져 있다[FF96]. 아래 Section 4.3은 그러한 경우를 다룬다.

4. Additional Considerations

4.1. Restarting Idle Connections

A known problem with the TCP congestion control algorithms described above is that they allow a potentially inappropriate burst of traffic to be transmitted after TCP has been idle for a relatively long period of time. After an idle period, TCP cannot use the ACK clock to strobe new segments into the network, as all the ACKs have drained from the network. Therefore, as specified above, TCP can potentially send a cwnd-size line-rate burst into the network after an idle period. In addition, changing network conditions may have rendered TCP's notion of the available end-to-end network capacity between two endpoints, as estimated by cwnd, inaccurate during the course of a long idle period.

위에서 설명한 TCP 혼잡 제어 알고리즘의 알려진 문제는 TCP가 비교적 오랜 시간 동안 유휴 상태일 때 전송하기 위해 잠재적으로 적절하지 않은 트래픽 폭주를 허용한다는 것이다. 유휴 기간이 지난 후, 모든 ACK가 네트워크에서 빠져나갔기 때문에 TCP는 ACK 클럭을 사용하여 네트워크에 새로운 세그먼트를 스트로브(strobe)할 수 없다. 따라서 위에서 명시된 대로 TCP는 유휴 기간이 지난 후 잠재적으로 네트워크로 cwnd 크기의 손실 없는 최대 속도로 전송할 수 있다. 또한, 네트워크 조건의 변경은 두 종단 사이의 가용 종단 간 종단 네트워크 용량에 대한 TCP의 컨셉을 긴 유휴 기간 동안 부정확하게 만들 수도 있다.

[Jac88] recommends that a TCP use slow start to restart transmission after a relatively long idle period. Slow start serves to restart the ACK clock, just as it does at the beginning of a transfer. This mechanism has been widely deployed in the following manner. When TCP has not received a segment for more than one retransmission timeout, cwnd is reduced to the value of the restart window (RW) before transmission begins.

[Jac88]은 TCP가 비교적 긴 유휴 기간 후에 느린 시작을 사용하여 전송을 재시작할 것을 권장한다. 느린 시작은 전송 시작 시와 마찬가지로 ACK 클럭을 다시 시작하는 역할을 한다. 이 메커니즘은 다음과 같은 방법으로 널리 전개되어 왔다. TCP가 둘 이상의 재전송 타임아웃에 대해 세그먼트를 수신하지 못한 경우, cwnd는 전송이 시작되기 전에 재시작 윈도우(RW) 값으로 감소한다.

For the purposes of this standard, we define RW = min(IW,cwnd).

이 표준의 목적상, 우리는 RW = min(IW,cwnd)을 정의한다.

Using the last time a segment was received to determine whether or not to decrease cwnd can fail to deflate cwnd in the common case of persistent HTTP connections [HTH98]. In this case, a Web server receives a request before transmitting data to the Web client. The reception of the request makes the test for an idle connection fail, and allows the TCP to begin transmission with a possibly inappropriately large cwnd.

영속적 HTTP 커넥션의 일반적인 경우에 cwnd를 감소시킬 수 있는지 여부를 판별하기 위해 세그먼트가 마지막으로 수신 된 시간을 사용한다. [HTH98]. 이 경우 웹 서버는 데이터를 웹 클라이언트로 전송하기 전에 요청을 받는다. 요청을 수신하면 유휴 커넥션에 대한 테스트가 실패하며, TCP가 부적절하게 큰 cwnd로 전송을 시작할 수 있다.

Therefore, a TCP SHOULD set cwnd to no more than RW before beginning transmission if the TCP has not sent data in an interval exceeding the retransmission timeout.

따라서 TCP가 재전송 타임아웃을 초과하여 데이터를 전송하지 않은 경우 TCP는 전송을 시작하기 전에 cwnd를 RW 이하로 설정해야 한다.(SHOULD)

4.2. Generating Acknowledgments

The delayed ACK algorithm specified in [RFC1122] SHOULD be used by a TCP receiver. When using delayed ACKs, a TCP receiver MUST NOT excessively delay acknowledgments. Specifically, an ACK SHOULD be generated for at least every second full-sized segment, and MUST be generated within 500 ms of the arrival of the first unacknowledged packet.

[RFC1122]에 명시된 지연 ACK 알고리즘은 TCP 수신자에 의해 사용되어야 한다.(SHOULD) 지연된 ACK를 사용할 때, TCP 수신자는 확인 응답을 과도하게 지연시키지 않아야 한다.(MUST NOT) 특히, 적어도 두 번째 풀 사이즈 세그먼트시에는 ACK가 생성되어야 하며,(SHOULD) 첫 번째 확인되지 않은 패킷이 도착한 후 500ms 이내에 생성되어야 한다.(MUST)

The requirement that an ACK "SHOULD" be generated for at least every second full-sized segment is listed in [RFC1122] in one place as a SHOULD and another as a MUST. Here we unambiguously state it is a SHOULD. We also emphasize that this is a SHOULD, meaning that an implementor should indeed only deviate from this requirement after careful consideration of the implications. See the discussion of "Stretch ACK violation" in [RFC2525] and the references therein for a discussion of the possible performance problems with generating ACKs less frequently than every second full-sized segment.

적어도 두 번째 풀 사이즈 세그먼트에 대해 ACK를 "생성해야 한다"는 요건은 [RFC1122]에 필수(SHOULD와 MUST) 항목으로 기재되어 있다. 여기서 우리는 그것이 필수(SHOULD)라고 분명하게 말한다. 우리는 또한 이것이 필수(SHOULD)라는 것을 강조하는데, 이는 구현자가 실제로 시사점을 주의 깊게 고려한 후에만 이 요건에서 벗어나야 한다는 것을 의미한다. [RFC2525]의 "지연 ACK 위반"에 대한 설명과 두 번째 풀 사이즈 세그먼트보다 덜 빈번하게 ACK를 생성하여 발생할 수 있는 성능 문제에 대한 자세한 내용을 참조한다.

In some cases, the sender and receiver may not agree on what constitutes a full-sized segment. An implementation is deemed to comply with this requirement if it

sends at least one acknowledgment every time it receives 2*RMSS bytes of new data from the sender, where RMSS is the Maximum Segment Size specified by the receiver to the sender (or the default value of 536 bytes, per [RFC1122], if the receiver does not specify an MSS option during connection establishment). The sender may be forced to use a segment size less than RMSS due to the maximum transmission unit (MTU), the path MTU discovery algorithm or other factors. For instance, consider the case when the receiver announces an RMSS of X bytes but the sender ends up using a segment size of Y bytes (Y < X) due to path MTU discovery (or the sender's MTU size). The receiver will generate stretch ACKs if it waits for 2*X bytes to arrive before an ACK is sent. Clearly this will take more than 2 segments of size Y bytes. Therefore, while a specific algorithm is not defined, it is desirable for receivers to attempt to prevent this situation, for example, by acknowledging at least every second segment, regardless of size. Finally, we repeat that an ACK MUST NOT be delayed for more than 500 ms waiting on a second full-sized segment to arrive.

송신자와 수신자가 무엇이 풀 사이즈 세그먼트를 구성하는지 합의하지 못하는 경우도 있다. 구현은 송신자로부터 2*RMSS 바이트의 새로운 데이터를 수신할 때마다 최소한 하나의 응답 확인을 전송하는 경우, 여기서 RMSS는 수신자가 송신자에게 지정한 최대 세그먼트 크기(또는 수신자가 커넥션 설립중에 MSS 옵션을 지정하지 않는 경우 [RFC1122]에 따라 기본값 536 바이트)를 전송하는 경우, 이 요건을 준수하는 것으로 간주된다. 송신자는 최대 전송 단위(MTU), Path MTU Discovery 알고리즘 또는 기타 요인으로 인해 RMSS 미만의 세그먼트 크기를 사용하도록 강제할 수 있다. 예를 들어, 수신자가 X 바이트의 RMSS를 알려주지만 Path MTU Discovery (또는 송신자의 MTU 크기)으로 인해 송신자가 Y 바이트 (Y < X)의 세그먼트 크기를 사용하게 되는 경우를 고려한다. ACK가 전송되기 전에 2*X 바이트가 도착하기를 기다리면 수신자는 지연 ACK를 생성한다. 분명히 이것은 Y바이트 크기의 2개 이상의 세그먼트를 필요로 할 것이다. 따라서 특정 알고리즘이 정의되어 있지 않지만, 예를 들어, 크기와 관계없이 최소한 매번 두 번째 세그먼트를 인정함으로써 수신자가 이러한 상황을 방지하려고 시도하는 것이 바람직하다. 마지막으로, 우리는 ACK는 두 번째 풀 사이즈 세그먼트가 도착하기 위해 500ms 이상 지연되어서는 안 된다(MUST NOT)는 것을 반복한다.

Out-of-order data segments SHOULD be acknowledged immediately, in order to accelerate loss recovery. To trigger the fast retransmit algorithm, the receiver SHOULD send an immediate duplicate ACK when it receives a data segment above a gap in the sequence space. To provide feedback to senders recovering from losses, the receiver SHOULD send an immediate ACK when it receives a data segment that fills in all or part of a gap in the sequence space.

손실 회복를 가속화하려면 문제 있는 데이터 세그먼트를 즉시 인정해야 한다.(SHOULD) 빠른 재전송 알고리즘을 트리거하려면 수신자가 시퀀스 공간의 간격보다 높은 데이터 세그먼트를 수신할 때 즉시 중복 ACK를 전송해야 한다.(SHOULD) 손실로부터 회복되는 송신자에게 피드백을

제공하기 위해, 수신자는 시퀀스 공간의 간격의 전부 또는 부분을 채우는 데이터 세그먼트를 수신할 때 즉시 ACK를 전송해야 한다.(SHOULD)

A TCP receiver MUST NOT generate more than one ACK for every incoming segment, other than to update the offered window as the receiving application consumes new data (see [RFC813] and page 42 of [RFC793]).

TCP 수신자는 수신하는 애플리케이션이 새로운 데이터를 소비함에 따라 제공된 윈도우를 업데이트하는 것 외에는 들어오는 세그먼트마다 ACK를 둘 이상 생성해서는 안 된다.(MUST NOT) ([RFC813] 및 [RFC793]의 42페이지 참조)

4.3. Loss Recovery Mechanisms

A number of loss recovery algorithms that augment fast retransmit and fast recovery have been suggested by TCP researchers and specified in the RFC series. While some of these algorithms are based on the TCP selective acknowledgment (SACK) option [RFC2018], such as [FF96], [MM96a], [MM96b], and [RFC3517], others do not require SACKs, such as [Hoe96], [FF96], and [RFC3782]. The non-SACK algorithms use “partial acknowledgments" (ACKs that cover previously unacknowledged data, but not all the data outstanding when loss was detected) to trigger retransmissions. While this document does not standardize any of the specific algorithms that may improve fast retransmit/fast recovery, these enhanced algorithms are implicitly allowed, as long as they follow the general principles of the basic four algorithms outlined above.

빠른 재전송과 빠른 회복을 증가시키는 많은 손실 회복 알고리즘이 TCP 연구자들에 의해 제안되었고 RFC 시리즈에 명시되었다. 이러한 알고리즘 중 일부는 [FF96], [MM96a], [MM96b], [RFC3517]과 같이 TCP 선택적 확인 응답(SACK) 옵션 [RFC2018]에 기반을 두고 있지만, [Ho96], [FF96], [RFC3782]와 같은 다른 알고리즘은 SACK이 필요하지 않다. 비 SACK 알고리즘은 재전송을 트리거하기 위해 "부분 확인 응답"을 사용한다.(기존에 확인되지 않은 데이터를 다루는 ACK, 손실이 감지되었을 때 모든 미처리 데이터를 다루는 것은 아님) 이 문서는 빠른 재전송/빠른 회복을 개선할 수 있는 특정 알고리즘을 표준화하지는 않지만, 위에서 설명한 기본 4개 알고리즘의 일반 원칙을 따르는 한 암묵적으로 이러한 강화된 알고리즘은 허용된다.

That is, when the first loss in a window of data is detected, ssthresh MUST be set to no more than the value given by equation (4). Second, until all lost segments in the window of data in question are repaired, the number of segments transmitted in each RTT MUST be no more than half the number of outstanding segments when

the loss was detected. Finally, after all loss in the given window of segments has been successfully retransmitted, cwnd MUST be set to no more than ssthresh and congestion avoidance MUST be used to further increase cwnd. Loss in two successive windows of data, or the loss of a retransmission, should be taken as two indications of congestion and, therefore, cwnd (and ssthresh) MUST be lowered twice in this case.

즉, 데이터 윈도우 내의 첫 번째 손실이 감지될 때, ssthresh는 등식 (4)에 의해 주어진 값 이하로 설정되어야 한다.(MUST) 두 번째로, 해당 데이터 윈도우에서 모든 손실 세그먼트가 회복될 때까지 각 RTT에서 전송되는 세그먼트 수는, 손실이 감지되었을 때 미처리 세그먼트 수의 절반 이하이어야 한다.(MUST) 마지막으로, 주어진 세그먼트 윈도우에서의 모든 손실이 성공적으로 재전송된 후, cwnd는 ssthresh 이하로 설정되어야 하며(MUST) 혼잡 회피를 사용하여 cwnd를 더 증가시켜야 한다. 두 개의 데이터 윈도우에서의 손실 또는 재전송의 손실은 혼잡의 두 가지 신호로 간주되어야 하며 따라서 이 경우 cwnd (및 ssthresh)는 두 배 낮아져야 한다.(MUST)

We RECOMMEND that TCP implementors employ some form of advanced loss recovery that can cope with multiple losses in a window of data. The algorithms detailed in [RFC3782] and [RFC3517] conform to the general principles outlined above. We note that while these are not the only two algorithms that conform to the above general principles these two algorithms have been vetted by the community and are currently on the Standards Track.

우리는 TCP 구현자들이 데이터 윈도우에서 복수의 손실에 대처할 수 있는 어떤 형태의 진보된 손실 회복 방식을 채택할 것을 권장한다.(RECOMMECD) [RFC3782]와 [RFC3517]에 자세히 작성된 알고리즘은 위에서 설명한 일반 원리를 준수한다. 상기 일반 원칙을 준수하는 알고리즘은 이들 두 개뿐이 아니지만, 이 두 알고리즘은 커뮤니티에 의해 검토되었으며 현재 표준 트랙에 있다.

5. Security Considerations

This document requires a TCP to diminish its sending rate in the presence of retransmission timeouts and the arrival of duplicate acknowledgments. An attacker can therefore impair the performance of a TCP connection by either causing data packets or their acknowledgments to be lost, or by forging excessive duplicate acknowledgments.

이 문서는 TCP가 재전송 타임아웃 및 중복 확인 도착 시 전송 속도를 감소시킬 것을 요구한다. 따라서 공격자는 데이터 패킷 또는 데이터 패킷의 확인 응답을 손실시키거나 과도한 중복 확인 응답을 위조하여 TCP 커넥션의 성능을 손상시킬 수 있다.

In response to the ACK division attack outlined in [SCWA99], this document RECOMMENDS increasing the congestion window based on the number of bytes newly acknowledged in each arriving ACK rather than by a particular constant on each arriving ACK (as outlined in section 3.1).

[SCWA99]에 설명된 ACK 분할 공격에 대응하여, 본 문서는 도착하는 ACK에 대해 특정 상수(Section 3.1에 설명된 대로)가 아니라 도착하는 ACK에서 새로 확인된 바이트 수를 기준으로 혼잡 윈도우를 늘릴 것을 권장한다.(RECOMMENDED)

The Internet, to a considerable degree, relies on the correct implementation of these algorithms in order to preserve network stability and avoid congestion collapse. An attacker could cause TCP endpoints to respond more aggressively in the face of congestion by forging excessive duplicate acknowledgments or excessive acknowledgments for new data. Conceivably, such an attack could drive a portion of the network into congestion collapse.

인터넷은 상당 부분 네트워크 안정성을 보존하고 혼잡 붕괴를 피하기 위해 이러한 알고리즘의 정확한 구현에 의존한다. 공격자는 과도한 중복 확인이나 새로운 데이터에 대한 과도한 확인 응답 내용을 위조하여 혼잡 상황에서 TCP 종단이 더 공격적으로 반응하도록 할 수 있다. 짐작컨대, 그러한 공격은 네트워크의 일부를 혼잡 붕괴로 몰고 갈 수 있다.

6. Changes between RFC 2001 and RFC 2581

[RFC2001] was extensively rewritten editorially and it is not feasible to itemize the list of changes between [RFC2001] and [RFC2581]. The intention of [RFC2581] was to not change any of the recommendations given in [RFC2001], but to further clarify cases that were not discussed in detail in [RFC2001]. Specifically, [RFC2581] suggested what TCP connections should do after a relatively long idle period, as well as specified and clarified some of the issues pertaining to TCP ACK generation. Finally, the allowable upper bound for the initial congestion window was raised from one to two segments.

[RFC2001]은 편집상 광범위하게 다시 작성되었으며, [RFC2001]과 [RFC2581] 사이의 변경 목록을 항목화 하는 것은 불가능하다. [RFC2581]의 의도는 [RFC2001]에 제시된 권고사

항을 변경하지 않고, [RFC2001]에서 자세히 논의되지 않은 사례를 더욱 명확히 하기 위함이었다. 구체적으로, [RFC2581]은 TCP ACK 생성과 관련된 몇 가지 이슈를 명시하고 명확히 할 뿐만 아니라, 비교적 긴 유휴 기간 후에 TCP 커넥션이 무엇을 해야 하는지를 제안했다. 마지막으로, 초기 혼잡 윈도우의 허용 상한을 1개에서 2개로 높였다.

7. Changes Relative to RFC 2581

A specific definition for "duplicate acknowledgment" has been added, based on the definition used by BSD TCP.

BSD TCP에서 사용하는 정의에 기초하여 "중복 확인 응답"에 대한 구체적인 정의가 추가되었다.

The document now notes that what to do with duplicate ACKs after the retransmission timer has fired is future work and explicitly unspecified in this document.

현재 이 문서는 재전송 타이머가 생긴 후 중복 ACK로 수행할 작업이 향후 작업이며 이 문서에 명시적으로 명시되어 있지 않다는 점에 주목한다.

The initial window requirements were changed to allow Larger Initial Windows as standardized in [RFC3390]. Additionally, the steps to take when an initial window is discovered to be too large due to Path MTU Discovery [RFC1191] are detailed.

초기 윈도우 요건은 [RFC3390]에서 표준화된 큰 초기 윈도우를 허용하도록 변경되었다. 또한 Path MTU Discovery [RFC1191]로 인해 초기 윈도우가 너무 큰 것으로 확인될 때 취해야 할 조치가 자세히 설명되어 있다.

The recommended initial value for ssthresh has been changed to say that it SHOULD be arbitrarily high, where it was previously MAY. This is to provide additional guidance to implementors on the matter.

권장되는 ssthresh 초기 값은 이전에 예상했던 대로 임의로 높여야 한다고 말하도록 변경되었다. 이것은 그 문제에 대한 구현자에게 추가적인 지침을 제공하기 위함이다.

During slow start, the usage of Appropriate Byte Counting [RFC3465] with L=1*SMSS is explicitly recommended. The method of increasing cwnd given in [RFC2581] is still explicitly allowed. Byte counting during congestion avoidance is also recommended, while the method from [RFC2581] and other safe methods are still allowed.

느린 시작 중에는 L=1*SMSS와 함께 적절한 바이트 카운팅 [RFC3465]을 사용하는 것이 명시적으로 권장된다. [RFC2581]에서 주어진 cwnd를 증가시키는 방법은 여전히 명시적으로 허용된다. [RFC2581]의 방법 및 기타 안전한 방법이 여전히 허용되는 가운데 혼잡 회피 시 바이트 카운팅도 권장된다.

The treatment of ssthresh on retransmission timeout was clarified. In particular, ssthresh must be set to half the FlightSize on the first retransmission of a given segment and then is held constant on subsequent retransmissions of the same segment.

재전송 타임아웃에 대한 ssthresh 처리가 명확해졌다. 특히, 해당 세그먼트의 첫 번째 재전송 시 FlightSize의 절반으로 설정되어야 하며, 이후 동일한 세그먼트의 재전송 시 일정하게 유지되어야 한다.

The description of fast retransmit and fast recovery has been clarified, and the use of Limited Transmit [RFC3042] is now recommended.

빠른 재전송과 빠른 회복에 대한 설명이 명확해졌고, 이제 제한된 전송 [RFC3042]의 사용이 권장된다.

TCPs now MAY limit the number of duplicate ACKs that artificially inflate cwnd during loss recovery to the number of segments outstanding to avoid the duplicate ACK spoofing attack described in [SCWA99].

이제 TCP는 [SCWA99]에 설명된 중복 ACK 스푸핑 공격을 피하기 위해 손실 회복 중 cwnd를 인위적으로 증가시키는 중복 ACK의 수를 미처리 세그먼트의 수로 제한할 수 있다.

The restart window has been changed to min(IW,cwnd) from IW. This behavior was described as "experimental" in [RFC2581].

재시작 윈도우가 IW에서 min(IW,cwnd)로 변경되었다. 이러한 행동은 [RFC2581]에서 "실험적"으로 설명되었다.

It is now recommended that TCP implementors implement an advanced loss recovery algorithm conforming to the principles outlined in this document.

이제 TCP 구현자는 이 문서에 기술된 원칙을 준수하는 진보된 손실 회복 알고리즘을 구현할 것을 권고한다.

The security considerations have been updated to discuss ACK division and recommend byte counting as a counter to this attack.

보안 고려사항은 ACK 분리에 대해 논의하고 이 공격에 대한 대응책으로 바이트 카운팅을 권장하도록 업데이트되었다.

8. Acknowledgments

The core algorithms we describe were developed by Van Jacobson [Jac88, Jac90]. In addition, Limited Transmit [RFC3042] was developed in conjunction with Hari Balakrishnan and Sally Floyd. The initial congestion window size specified in this document is a result of work with Sally Floyd and Craig Partridge [RFC2414, RFC3390].

우리가 기술한 핵심 알고리즘은 Van Jacobson에 의해 개발되었다. [Jac88, Jac90] 또한, 제한된 전송[RFC3042]은 Hari Balakrishnan, Sally Floyd와 연계하여 개발되었다. 이 문서에 명시된 초기 혼잡 윈도우 크기는 Sally Floyd 및 Craig Partridge와의 작업의 결과물이다.[RFC2414, RFC3390]

W. Richard ("Rich") Stevens wrote the first version of this document [RFC2001] and co-authored the second version [RFC2581]. This present version much benefits from his clarity and thoughtfulness of description, and we are grateful for Rich's contributions in elucidating TCP congestion control, as well as in more broadly helping us understand numerous issues relating to networking.

W. Richard ("Rich") Stevens는 이 문서의 첫 번째 버전[RFC2001]을 썼고 두 번째 버전[RFC2581]을 공동저술했다. 이 현재 버전은 그의 명확성과 사려 깊은 설명에서 많은 이점을 얻으며, 우리는 TCP 혼잡 제어를 해명하는 데 Rich의 기여에 감사하며, 네트워킹과 관련된 수많은 이슈를 우리가 이해하는 데 더 광범위하게 도움을 준다.

We wish to emphasize that the shortcomings and mistakes of this document are solely the responsibility of the current authors.

우리는 이 문서의 단점과 실수는 오로지 현 저자의 책임이라는 점을 강조하고 싶다.

Some of the text from this document is taken from “TCP/IP Illustrated, Volume 1: The Protocols" by W. Richard Stevens (Addison-Wesley, 1994) and "TCP/IP Illustrated, Volume 2: The Implementation" by Gary R. Wright and W. Richard Stevens (Addison-Wesley, 1995). This material is used with the permission of Addison-Wesley.

본 문서의 일부 텍스트는 W. Richard Stevens (Addison-Wesley, 1994)의 "TCP/IP illustrated, Volume 1: The Protocols"와 Gary R. Wright와 W. Richard Stevens (Addison-Wesley, 1995)의 "TCP/IP illustrated, Volume 2: The Implemention"에서 가져온 것이다. 이 재료는 Addison-Wesley의 허락을 받아 사용된다.

Anil Agarwal, Steve Arden, Neal Cardwell, Noritoshi Demizu, Gorry Fairhurst, Kevin Fall, John Heffner, Alfred Hoenes, Sally Floyd, Reiner Ludwig, Matt Mathis, Craig Partridge, and Joe Touch contributed a number of helpful suggestions.

Anil Agarwal, Steve Arden, Neal Cardwell, Noritoshi Demizu, Gorry Fairhurst, Kevin Fall, John Heffner, Alfred Hoenes, Sally Floyd, Reiner Ludwig, Matt Mathis, Craig Partridge, and Joe Touch가 많이 도움이 되는 제안으로 공헌했다.

9. References

9.1. Normative References

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -Communication Layers", STD 3, RFC 1122, October 1989.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

9.2. Informative References

[CJ89] Chiu, D. and R. Jain, "Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks", Journal of Computer Networks and ISDN Systems, vol. 17, no. 1, pp. 1-14, June 1989.

[FF96] Fall, K. and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno and SACK TCP", Computer Communication Review, July 1996, ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z.

[Hoe96] Hoe, J., "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", In ACM SIGCOMM, August 1996.

[HTH98] Hughes, A., Touch, J., and J. Heidemann, "Issues in TCP Slow-Start Restart After Idle", Work in Progress, March 1998.

[Jac88] Jacobson, V., "Congestion Avoidance and Control", Computer Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988. ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

[Jac90] Jacobson, V., "Modified TCP Congestion Avoidance Algorithm", end2end-interest mailing list, April 30, 1990. ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail.

[MM96a] Mathis, M. and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", Proceedings of SIGCOMM'96, August, 1996, Stanford, CA. Available from http://www.psc.edu/networking/papers/papers.html

[MM96b] Mathis, M. and J. Mahdavi, "TCP Rate-Halving with Bounding Parameters", Technical report. Available from http://www.psc.edu/networking/papers/FACKnotes/current.

[Pax97] Paxson, V., "End-to-End Internet Packet Dynamics", Proceedings of SIGCOMM '97, Cannes, France, Sep. 1997.

[RFC813] Clark, D., "Window and Acknowledgement Strategy in TCP", RFC 813, July 1982.

[RFC2001] Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, January 1997.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, “TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC2414] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP’s Initial Window", RFC 2414, September 1998.

[RFC2525] Paxson, V., Allman, M., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., Semke, J., and B. Volz, "Known TCP Implementation Problems", RFC 2525, March 1999.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, “An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.

[RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

[RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, “Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP’s Initial Window", RFC 3390, October 2002.

[RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, February 2003.

[RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, “A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, April 2003.

[RFC3782] Floyd, S., Henderson, T., and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 3782, April 2004.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.

[SCWA99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP Congestion Control With a Misbehaving Receiver", ACM Computer Communication Review, 29(5), October 1999.

[Ste94] Stevens, W., "TCP/IP Illustrated, Volume 1: The Protocols", Addison-Wesley, 1994.

[WS95] Wright, G. and W. Stevens, "TCP/IP Illustrated, Volume 2: The Implementation", Addison-Wesley, 1995.

Authors' Addresses

Mark Allman
International Computer Science Institute (ICSI)
1947 Center Street
Suite 600
Berkeley, CA 94704-1198
Phone: +1 440 235 1792
EMail: mallman@icir.org
http://www.icir.org/mallman/

Vern Paxson
International Computer Science Institute (ICSI)
1947 Center Street
Suite 600
Berkeley, CA 94704-1198
Phone: +1 510/642-4274 x302
EMail: vern@icir.org
http://www.icir.org/vern/

Ethan Blanton
Purdue University Computer Sciences
305 North University Street
West Lafayette, IN 47907
EMail: eblanton@cs.purdue.edu

http://www.cs.purdue.edu/homes/eblanton/