RFC 8312

CUBIC for Fast Long-Distance Networks

문서 최종 수정일	2020-06-29
원문 복사일	2020-06-12
번역 및 정리	이병록(roka88)
이메일	roka88.dev@gmail.com

INFOMATIONAL
Errta Exist

Internet Engineering Task Force (IETF)
Request for Comments: 8312
Category: Informational
ISSN: 2070-1721

I. Rhee
NCSU
L. Xu
UNL
S. Ha
Colorado
A. Zimmermann
L. Eggert
R. Scheffenegger
NetApp
February 2018

CUBIC for Fast Long-Distance Networks

Abstract

CUBIC is an extension to the current TCP standards. It differs from the current TCP standards only in the congestion control algorithm on the sender side. In particular, it uses a cubic function instead of a linear window increase function of the current TCP standards to improve scalability and stability under fast and long-distance networks. CUBIC and its predecessor algorithm have been adopted as defaults by Linux and have been used for many years. This document provides a specification of CUBIC to enable third-party implementations and to solicit community feedback through experimentation on the performance of CUBIC.

CUBIC은 현재의 TCP 표준에 대한 확장이다. 송신자 측의 혼잡 제어 알고리즘에서만 현재의 TCP 표준과 다르다. 특히 기존 TCP 표준의 선형 윈도우 증가 함수 대신 CUBIC 함수를 사용해 빠르고 장거리 네트워크에서의 확장성 및 안정성을 향상시켰다. CUBIC과 그 전신 알고리즘은 리눅스에 의해 기본으로 채택되어 여러 해 동안 사용되어 왔다. 이 문서는 제3자 구현을 가능하게 하고, CUBIC의 성능에 대한 실험을 통해 커뮤니티의 피드백을 요청하기 위한 CUBIC의 사양을 제공한다.

Status of This Memo

Copyright Notice

2018 IETF 트러스트 및 문서 작성자로 식별된 사람.

Table of Contents

1. Introduction

The low utilization problem of TCP in fast long-distance networks is well documented in [K03] and [RFC3649]. This problem arises from a slow increase of the congestion window following a congestion event in a network with a large bandwidth-delay product (BDP). [HKLRX06] indicates that this problem is frequently observed even in the range of congestion window sizes over several hundreds of packets. This problem is equally applicable to all Reno-style TCP standards and their variants, including TCP-RENO [RFC5681], TCP-NewReno [RFC6582] [RFC6675], SCTP [RFC4960], and TFRC [RFC5348], which use the same linear increase function for window growth, which we refer to collectively as "Standard TCP" below.

고속 장거리 네트워크에서 TCP의 낮은 이용률 문제는 [K03]과 [RFC3649]에 잘 설명되어 있다. 이 문제는 대규모 대역폭-지연 곱(BDP)이 있는 네트워크에서 혼잡 이벤트에 이어 혼잡 윈도우가 느리게 증가하면서 발생한다. [HKLRX06]은 이 문제가 수백 개의 패킷에 걸친 혼잡 윈도우 크기 범위에서도 자주 발견된다는 것을 나타낸다. 이 문제는 TCP-RENO [RFC5681], TCP-NewReno [RFC6582] [RFC6675], SCTP [RFC4960], TFRC [RFC5348] 등 모든 Reno 스타일의 TCP 표준과 그 변형에 동일하게 적용되며, 이러한 표준은 동일한 선형 증가 함수를 사용한다.

CUBIC, originally proposed in [HRX08], is a modification to the congestion control algorithm of Standard TCP to remedy this problem. This document describes the most recent specification of CUBIC. Specifically, CUBIC uses a cubic function instead of a linear window increase function of Standard TCP to improve scalability and stability under fast and long-distance networks.

CUBIC은 원래 [HRX08]에서 제안된 것으로, 이 문제를 해결하기 위해 표준 TCP의 혼잡 제어 알고리즘을 개조한 것이다. 이 문서는 가장 최근의 CUBIC 명세를 설명한다. 구체적으로 CUBIC은 표준 TCP의 선형 윈도우 증가 기능 대신 CUBIC 함수를 사용하여 빠르고 장거리 네트워크에서의 확장성 및 안정성을 향상시킨다.

Binary Increase Congestion Control (BIC-TCP) [XHR04], a predecessor of CUBIC, was selected as the default TCP congestion control algorithm by Linux in the year 2005 and has been used for several years by the Internet community at large. CUBIC uses a similar window increase function as BIC-TCP and is designed to be less aggressive and fairer to Standard TCP in bandwidth usage than BIC-TCP while maintaining the strengths of BIC-TCP such as stability, window scalability, and RTT fairness. CUBIC has already replaced BIC-TCP as the default TCP congestion control algorithm in Linux and has been deployed globally by Linux. Through extensive testing in various Internet scenarios, we believe that CUBIC is safe for testing and deployment in the global Internet.

CUBIC의 전신인 바이너리 증가 혼잡 제어(BIC-TCP) [XHR04]는 2005년 리눅스에 의해 기본 TCP 혼잡 제어 알고리즘으로 선정되어 인터넷 커뮤니티에서 수년째 사용하고 있다. CUBIC은 BIC-TCP와 유사한 윈도우 증가 함수를 사용하며 BIC-TCP보다 대역폭 사용량이 덜 공격적이고 공정하면서도 안정성, 윈도우 확장성, RTT 공정성 등 BIC-TCP의 강점을 유지하도록 설계됐다. CUBIC은 이미 Linux의 기본 TCP 정체 제어 알고리즘으로 BIC-TCP를 대체했으며 리눅스에 의해 전 세계에 배포되었다. 다양한 인터넷 시나리오에서의 광범위한 테스트를 통해, 우리는 CUBIC이 글로벌 인터넷에서의 테스트와 배포에 안전하다고 믿는다.

In the following sections, we first briefly explain the design principles of CUBIC, then provide the exact specification of CUBIC, and finally discuss the safety features of CUBIC following the guidelines specified in [RFC5033].

다음 섹션에서는 먼저 CUBIC의 설계 원리를 간략하게 설명한 다음, 정확한 CUBIC 명세를 제공하고, 마지막으로 [RFC5033]에 명시된 지침에 따라 CUBIC의 안전 기능에 대해 논의한다.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

본 문서의 핵심 단어 "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", “SHOULD NOT", "RECOMMENDED", “NOT RECOMMENDED", "MAY", "OPTIONAL"은 BCP 14 [RFC2119] [RFC8174]에 기술된 바와 같이 해석되어야 한다.

3. Design Principles of CUBIC

CUBIC is designed according to the following design principles:

CUBIC은 다음과 같은 설계 원리에 따라 설계된다.

Principle 1: For better network utilization and stability, CUBIC uses both the concave and convex profiles of a cubic function to increase the congestion window size, instead of using just a convex function.

원칙 1: 네트워크 이용률과 안정성을 높이기 위해 CUBIC 함수의 오목한 프로파일과 볼록한 프로파일을 모두 사용해 볼록함수만을 사용하는 대신 혼잡 윈도우 크기를 늘린다.

Principle 2: To be TCP-friendly, CUBIC is designed to behave like Standard TCP in networks with short RTTs and small bandwidth where Standard TCP performs well.

원칙 2: TCP에 친화적이 되기 위해, CUBIC은 표준 TCP가 잘 작동하는 짧은 RTT와 작은 대역폭을 가진 네트워크에서 표준 TCP처럼 동작하도록 설계되었다.

Principle 3: For RTT-fairness, CUBIC is designed to achieve linear bandwidth sharing among flows with different RTTs.

원칙 3: RTT-공정성을 위해, CUBIC은 다른 RTT를 가진 흐름들 사이에서 공유하는 선형 대역폭을 달성하도록 설계되었다.

Principle 4: CUBIC appropriately sets its multiplicative window decrease factor in order to balance between the scalability and convergence speed.

원칙 4: CUBIC은 확장성과 수렴 속도 사이의 균형을 맞추기 위해 윈도우 지수 감소 계수를 적절하게 설정한다.

Principle 1: For better network utilization and stability, CUBIC [HRX08] uses a cubic window increase function in terms of the elapsed time from the last congestion event. While most alternative congestion control algorithms to Standard TCP increase the congestion window using convex functions, CUBIC uses both the concave and convex profiles of a cubic function for window growth. After a window reduction in response to a congestion event is detected by duplicate ACKs or Explicit Congestion Notification-Echo (ECN-Echo) ACKs [RFC3168], CUBIC registers the congestion window size where it got the congestion event as W_max and performs a multiplicative decrease of congestion window. After it enters into congestion avoidance, it starts to increase the congestion window using the concave profile of the cubic function. The cubic function is set to have its plateau at W_max so that the concave window increase continues until the window size becomes W_max. After that, the cubic function turns into a convex profile and the convex window increase begins. This style of window adjustment (concave and then convex) improves the algorithm stability while maintaining high network utilization [CEHRX07]. This is because the window size remains almost constant, forming a plateau around W_max where network utilization is deemed highest. Under steady state, most window size samples of CUBIC are close to W_max, thus promoting high network utilization and stability. Note that those congestion control algorithms using only convex functions to increase the congestion window size have the maximum increments around W_max, and thus introduce a large number of packet bursts around the saturation point of the network, likely causing frequent global loss synchronizations.

원칙 1: 네트워크 이용률과 안정성 향상을 위해 CUBIC [HRX08]은 마지막 혼잡 이벤트로부터 시간이 경과되면 CUBIC 윈도우 증가 함수를 사용한다. 표준 TCP에 대한 대부분의 대안 혼잡 제어 알고리즘은 볼록 함수를 사용하여 혼잡 윈도우를 증가시키는 반면, CUBIC은 윈도우 증가에 대해 CUBIC 함수의 오목한 프로파일과 볼록한 프로파일을 모두 사용한다. 중복 ACKs 또는 Explicit Congestion Notification-Echo(ECN-Echo) ACKs [RFC3168]에 의해 혼잡 이벤트에 대한 윈도우 감소가 감지된 후, CUBIC은 혼잡 이벤트가 발생한 혼잡 윈도우 크기를 W_max로 등록하고 혼잡 윈도우의 지수 감소를 수행한다. 혼잡 회피 상태에 들어간 후 CUBIC 함수의 오목 프로파일을 이용하여 혼잡 윈도우를 늘리기 시작한다. CUBIC 함수는 W_max에 플래토 현상으로 윈도우 크기가 W_max가 될 때까지 오목한 윈도우 증가가 계속되도록 설정한다. 이후 CUBIC 함수가 볼록한 프로파일로 바뀌고 볼록한 윈도우 증가가 시작된다. 이러한 윈도우 조정 스타일(오목 다음으로 볼록)은 높은 네트워크 활용도를 유지하면서 알고리즘 안정성을 향상시킨다 [CEHRX07]. 이는 윈도우 크기가 거의 일정하게 유지되어 네트워크 활용도가 가장 높다고 판단되는 W_max를 중심으로 플래토 현상을 형성하기 때문이다. 안정된 상태에서는 대부분의 CUBIC의 윈도우 크기 샘플이 W_max에 가깝기 때문에 높은 네트워크 활용도와 안정성을 촉진한다. 혼잡 창 크기를 증가시키기 위해 볼록함수만 사용하는 혼잡 제어 알고리즘은 W_max 전후의 최대 증가분을 가지며, 따라서 네트워크의 포화점

을 중심으로 다수의 패킷 버스트를 도입하여 글로벌 손실 동기화가 빈번하게 발생할 가능성이 높다는 점에 유의한다.

Principle 2: CUBIC promotes per-flow fairness to Standard TCP. Note that Standard TCP performs well under short RTT and small bandwidth (or small BDP) networks. There is only a scalability problem in networks with long RTTs and large bandwidth (or large BDP). An alternative congestion control algorithm to Standard TCP designed to be friendly to Standard TCP on a per-flow basis must operate to increase its congestion window less aggressively in small BDP networks than in large BDP networks. The aggressiveness of CUBIC mainly depends on the maximum window size before a window reduction, which is smaller in small BDP networks than in large BDP networks. Thus, CUBIC increases its congestion window less aggressively in small BDP networks than in large BDP networks. Furthermore, in cases when the cubic function of CUBIC increases its congestion window less aggressively than Standard TCP, CUBIC simply follows the window size of Standard TCP to ensure that CUBIC achieves at least the same throughput as Standard TCP in small BDP networks. We call this region where CUBIC behaves like Standard TCP, the “TCP-friendly region".

원칙 2: CUBIC은 표준 TCP에 대한 per-flow 공정성을 촉진한다. 표준 TCP는 짧은 RTT 및 작은 대역폭(또는 작은 BDP) 네트워크에서도 잘 작동한다는 점에 유의한다. 긴 RTT와 큰 대역폭(또는 큰 BDP)을 가진 네트워크에는 확장성 문제만 있을 뿐이다. per-flow 기반에 둔 표준 TCP에 친숙하기 위해 설계된 표준 TCP에 대안적인 혼잡 제어 알고리즘은 큰 BDP 네트워크보다 소규모 BDP 네트워크에서 덜 위협적으로 혼잡 제어 윈도우를 증가시키도록 작동해야 한다. CUBIC의 위협성은 주로 윈도우 축소 전 최대 윈도우 크기에 따라 달라지는데, 이는 대형 BDP 네트워크보다 소규모 BDP 네트워크에서 더 작다. 따라서, CUBIC은 대형 BDP 네트워크보다 소규모 BDP 네트워크에서의 혼잡 윈도우를 덜 위협적으로 증가시킨다. 더욱이, CUBIC의 CUBIC 함수가 표준 TCP보다 덜 위협적으로 혼잡 윈도우를 증가시키는 경우에, CUBIC은 작은 BDP 네트워크에서 적어도 표준 TCP와 같은 처리량을 달성하도록 하기 위해서 단순하게 표준 TCP의 윈도우 크기를 따라간다. 우리는 표준 TCP와 같이 동작하는 영역을 "TCP 친화적인 영역"이라고 부른다.

Principle 3: Two CUBIC flows with different RTTs have their throughput ratio linearly proportional to the inverse of their RTT ratio, where the throughput of a flow is approximately the size of its congestion window divided by its RTT. Specifically, CUBIC maintains a window increase rate independent of RTTs outside of the TCP-friendly region, and thus flows with different RTTs have similar congestion window sizes under steady state when they operate outside the TCP-friendly region.
This notion of a linear throughput ratio is similar to that of Standard TCP under high statistical multiplexing environments where packet losses are independent of individual flow rates. However, under low statistical multiplexing environments, the

throughput ratio of Standard TCP flows with different RTTs is quadratically proportional to the inverse of their RTT ratio [XHR04]. CUBIC always ensures the linear throughput ratio independent of the levels of statistical multiplexing. This is an improvement over Standard TCP. While there is no consensus on particular throughput ratios of different RTT flows,
we believe that under wired Internet, use of a linear throughput ratio seems more reasonable than equal throughputs (i.e., the same throughput for flows with different RTTs) or a higher-order throughput ratio (e.g., a quadratical throughput ratio of Standard TCP under low statistical multiplexing environments).

원칙 3: 서로 다른 RTTs를 가진 두 개의 CUBIC 흐름은 처리량 비율이 RTT 비율의 역에 비례하며, 여기서 흐름의 처리량은 RTT로 나눈 혼잡 윈도우의 대략적인 크기이다. 구체적으로, CUBIC은 TCP 친화적인 영역 외부의 RTTs와 독립적으로 윈도우 증가율을 유지하며, 따라서 다른 RTTs와의 흐름은 TCP 친화적인 영역 밖에서 작동할 때 일정한 상태에서 유사한 혼잡 윈도우 크기를 가진다. 이러한 선형 처리량 비율의 개념은 패킷 손실이 개별 흐름 비율과 독립적인 높은 통계적 멀티플렉싱 환경에서 표준 TCP와 유사하다. 그러나, 낮은 통계적 멀티플렉싱 환경에서, 다른 RTTs를 가진 표준 TCP 흐름의 처리량 비율은 RTT 비율의 역에 이차식으로 비례한다[XHR04]. CUBIC은 항상 통계적 멀티플렉싱 수준과 독립적인 선형 처리량 비율을 보장한다. 이것은 표준 TCP보다 개선된 것이다. 서로 다른 RTT 흐름의 특정 처리량 비율에 대해서는 합의가 이루어지지 않지만, 유선 인터넷에서는 선형 처리량 비율을 사용하는 것이 동일한 처리량(i.e., 서로 다른 RTT를 가진 흐름의 경우 동일한 처리량) 또는 고차 처리량 비율보다 더 합리적이라고 생각한다.(e.g., 낮은 통계적 멀티플렉싱 환경에서 표준 TCP의 이차식 처리량 비율)

Principle 4: To balance between the scalability and convergence speed, CUBIC sets the multiplicative window decrease factor to 0.7 while Standard TCP uses 0.5. While this improves the scalability of CUBIC, a side effect of this decision is slower convergence, especially under low statistical multiplexing environments. This design choice is following the observation that the author of HighSpeed TCP (HSTCP) [RFC3649] has made along with other researchers (e.g., [GV02]): the current Internet becomes more asynchronous with less frequent loss synchronizations with high statistical multiplexing. Under this environment, even strict Multiplicative-Increase Multiplicative-Decrease (MIMD) can converge. CUBIC flows with the same RTT always converge to the same throughput independent of statistical multiplexing, thus achieving intra-algorithm fairness. We also find that under the environments with sufficient statistical multiplexing, the convergence speed of CUBIC flows is reasonable.

원칙 4: 확장성과 수렴속도의 균형을 맞추기 위해, CUBIC은 윈도우 지수 감소 계수를 0.7로 설정하고, 표준 TCP는 0.5를 사용한다. 이것은 CUBIC의 확장성을 향상시키지만, 이 결정의 부작용은 특히 낮은 통계적 멀티플렉싱 환경에서 더 느린 수렴이다. 이러한 설계 선택은 고속 TCP (HSTCP) [RFC3649]의 저자가 다른 연구자들과 함께 한 관찰에 따른 것이다.(e.g.,

[GV02]): 현재의 인터넷은 높은 통계적 멀티플렉싱과의 빈번한 손실 동기화와 함께 더 비동기화 된다. 이런 환경에서는 엄격한 Multiplicative-Increase Multiplicative-Decrease (MIMD)도 수렴할 수 있다. 동일한 RTT를 가진 CUBIC 흐름은 항상 통계적 멀티플렉싱과 독립적으로 동일한 처리량으로 수렴되므로 알고리즘 내 공정성을 달성한다. 또한 충분한 통계적 멀티플렉싱이 있는 환경에서는 CUBIC 흐름의 수렴 속도가 합리적이라는 것을 알게 되었다.

4. CUBIC Congestion Control

The unit of all window sizes in this document is segments of the maximum segment size (MSS), and the unit of all times is seconds. Let cwnd denote the congestion window size of a flow, and ssthresh denote the slow-start threshold.

이 문서에서 모든 윈도우 크기의 단위는 최대 세그먼트 크기(MSS)의 세그먼트이며, 모든 시간의 단위는 초이다. cwnd는 흐름의 혼잡 윈도우 크기를 나타내며, ssthresh는 느린 시작 상태 임계값을 나타낸다.

4.1. Window Increase Function

CUBIC maintains the acknowledgment (ACK) clocking of Standard TCP by increasing the congestion window only at the reception of an ACK. It does not make any change to the fast recovery and retransmit of TCP, such as TCP-NewReno [RFC6582] [RFC6675]. During congestion avoidance after a congestion event where a packet loss is detected by duplicate ACKs or a network congestion is detected by ACKs with ECN-Echo flags [RFC3168], CUBIC changes the window increase function of Standard TCP. Suppose that W_max is the window size just before the window is reduced in the last congestion event.

CUBIC은 ACK의 수신에서만 혼잡 윈도우를 증가시켜 표준 TCP의 수신 확인(ACK) 클로킹(패킷을 수신 할 때까지 일정하게 만드는것)을 유지한다. TCP-NewReno [RFC6582] [RFC6675]와 같은 TCP의 빠른 회복과 재전송에는 아무런 변화가 없다. 중복 ACK에 의해 패킷 손실이 감지되거나 ECN-Echo 플래그 [RFC3168]가 있는 ACK에 의해 네트워크 정체가 감지되는 혼잡 이벤트 후 혼잡 회피 상태 동안, CUBIC은 표준 TCP의 윈도우 증가 함수를 변경한다. W_max가 마지막 혼잡 이벤트에서 윈도우가 축소되기 바로 전의 윈도우 크기라고 가정한다.

CUBIC uses the following window increase function:

CUBIC은 다음의 윈도우 증가 함수를 사용한다:

W_cubic(t) = C*(t-K)^3 + W_max (Eq. 1)

where C is a constant fixed to determine the aggressiveness of window increase in high BDP networks, t is the elapsed time from the beginning of the current congestion avoidance, and K is the time period that the above function takes to increase the current window size to W_max if there are no further congestion events and is calculated using the following equation:

여기서 C는 높은 BDP 네트워크에서 윈도우 증가의 강도를 결정하기 위해 고정된 상수이며, t는 현재 혼잡 회피 상태의 시작부터 경과된 시간이며, K는 더 이상의 혼잡 이벤트가 없을 경우, 현재 윈도우 크기를 W_max로 증가시키기 위한 함수가 걸리는 시간이며 다음의 방정식을 사용하여 계산된다.

K = cubic_root(W_max*(1-beta_cubic)/C) (Eq. 2)

where beta_cubic is the CUBIC multiplication decrease factor, that is, when a congestion event is detected, CUBIC reduces its cwnd to W_cubic(0)=W_max*beta_cubic. We discuss how we set beta_cubic in Section 4.5 and how we set C in Section 5.

여기서 beta_cubic은 CUBIC 지수 감소 인자이며, 즉, 혼잡 이벤트가 감지되면 CUBIC은 cwnd를 W_cubic(0)=W_max*beta_cubic로 감소시킨다. Section 4.5에서 beta_cubic을 설정하는 방법과 Section 5에서 C를 설정하는 방법에 대해 논의한다.

Upon receiving an ACK during congestion avoidance, CUBIC computes the window increase rate during the next RTT period using Eq. 1. It sets W_cubic(t+RTT) as the candidate target value of the congestion window, where RTT is the weighted average RTT calculated by Standard TCP.

혼잡 회피 상태에서 ACK를 받으면, CUBIC은 Eq. 1을 사용하여 다음 RTT 기간 동안의 윈도우 증가율을 계산한다. W_cubic(t+RT)을 혼잡 윈도우의 후보 목표값으로 설정하며, 여기서 RTT는 표준 TCP가 계산한 가중평균 RTT이다.

Depending on the value of the current congestion window size cwnd, CUBIC runs in three different modes.

현재 혼잡 윈도우 크기 cwnd의 값에 따라, CUBIC은 세 가지 다른 모드로 실행된다.

1. The TCP-friendly region, which ensures that CUBIC achieves at least the same throughput as Standard TCP.

1. CUBIC이 최소한 표준 TCP와 같은 처리량을 달성하도록 하는, TCP 친화적인 영역.

2. The concave region, if CUBIC is not in the TCP-friendly region and cwnd is less than W_max.

2. CUBIC이 TCP 친화적인 영역에 있지 않고 cwnd가 W_max보다 작을 경우, 오목한 영역.

3. The convex region, if CUBIC is not in the TCP-friendly region and cwnd is greater than W_max.

3. 만약 CUBIC이 TCP 친화적인 영역에 있지 않고 cwnd가 W_max보다 크다면, 볼록한 영역.

Below, we describe the exact actions taken by CUBIC in each region.

아래에서는 각 영역에서 CUBIC이 취한 정확한 조치를 설명한다.

4.2. TCP-Friendly Region

Standard TCP performs well in certain types of networks, for example, under short RTT and small bandwidth (or small BDP) networks. In these networks, we use the TCP-friendly region to ensure that CUBIC achieves at least the same throughput as Standard TCP.

예를 들어, 표준 TCP는 짧은 RTT 네트워크와 작은 대역폭(또는 작은 BDP) 네트워크에서는 특정 유형의 네트워크에서 좋은 성능을 발휘한다. 이러한 네트워크에서는, 우리는 CUBIC이 적어도 표준 TCP와 같은 처리량을 달성하도록 하기 위해 TCP 친화적인 영역을 사용한다.

The TCP-friendly region is designed according to the analysis described in [FHP00]. The analysis studies the performance of an Additive Increase and Multiplicative Decrease (AIMD) algorithm with an additive factor of alpha_aimd (segments per RTT) and a multiplicative factor of beta_aimd, denoted by AIMD(alpha_aimd, beta_aimd). Specifically, the average congestion window size of AIMD(alpha_aimd, beta_aimd) can be calculated using Eq. 3. The analysis shows that AIMD(alpha_aimd, beta_aimd) with alpha_aimd=3*(1-beta_aimd)/(1+beta_aimd) achieves the same average window size as Standard TCP that uses AIMD(1, 0.5).

TCP 친화적인 영역은 [FHP00]에 기술된 분석에 따라 설계된다. 이 분석은 AIMD(alpha_aimd, beta_aimd)로 표시된 alpha_aimd(RTT당 세그먼트 수)의 가산 인자와 beta_aimd의 지수 인자를 갖는 가산 증가 및 지수 감소(AIMD) 알고리즘의 성능을 연구한다. 구체적으로 AIMD(alpha_aimd, beta_aimd)의 평균 혼잡 윈도우 크기는 Eq. 3을 사용하여 계산할 수 있다. 분석 결과 alpha_aimd=3*(1-beta_aimd)/(1+beta_aimd)를 가진 AIMD(alpha_aimd, beta_aimd)는 AIMD(1, 0.5)를 사용하는 표준 TCP와 동일한 평균 혼잡 윈도우 크기를 달성하는 것으로 나타났다.

AVG_W_aimd = [alpha_aimd * (1+beta_aimd) / (2*(1-beta_aimd)*p)]^0.5 (Eq. 3)

Based on the above analysis, CUBIC uses Eq. 4 to estimate the window size W_est of AIMD(alpha_aimd, beta_aimd) with alpha_aimd=3*(1-beta_cubic)/(1+beta_cubic) and beta_aimd=beta_cubic, which achieves the same average window size as Standard TCP. When receiving an ACK in congestion avoidance (cwnd could be greater than or less than W_max), CUBIC checks whether W_cubic(t) is less than W_est(t). If so, CUBIC is in the TCP-friendly region and cwnd SHOULD be set to W_est(t) at each reception of an ACK.

위의 분석에 기초하여, CUBIC은 Eq. 4를 사용하여 alpha_aimd=3*(1-beta_cubic)/(1+beta_cubic)와 beta_aimd=beta_cubic로 표준 TCP와 동일한 평균 윈도우 크기를 달성하는 AIMD(alpha_aimd, beta_aimd)의 W_est 윈도우 크기를 추정한다. 혼잡 회피 상태에서 ACK를 수신할 때 (cwnd가 W_max보다 크거나 작을 수 있음), CUBIC은 W_cubic(t)이 W_est(t)보다 작은지 여부를 확인한다. 만약에 그렇다면, CUBIC은 TCP 친화적인 영역에 있고, ACK의 각 수신에서 cwnd를 W_est(t)로 설정해야 한다.(SHOULD)

W_est(t) = W_max*beta_cubic + [3*(1-beta_cubic)/(1+beta_cubic)] * (t/RTT) (Eq. 4)

4.3. Concave Region

When receiving an ACK in congestion avoidance, if CUBIC is not in the TCP-friendly region and cwnd is less than W_max, then CUBIC is in the concave region. In this region, cwnd MUST be incremented by (W_cubic(t+RTT) - cwnd)/cwnd for each received ACK, where W_cubic(t+RTT) is calculated using Eq. 1.

혼잡 회피 상태에서 ACK를 수신할 때, CUBIC이 TCP 친화적인 영역에 있지 않고 cwnd가 W_max보다 작으면 CUBIC이 오목한 영역에 있다. 이 영역에서 cwnd는 반드시 (W_cubic(t+RTT) - cwnd) / cwnd씩 증가해야 하며 여기서 W_cubic(t+RTT)은 Eq. 1을 사용하여 계산된다.

4.4. Convex Region

When receiving an ACK in congestion avoidance, if CUBIC is not in the TCP-friendly region and cwnd is larger than or equal to W_max, then CUBIC is in the convex region. The convex region indicates that the network conditions might have been perturbed since the last congestion event, possibly implying more available bandwidth after some flow departures. Since the Internet is highly asynchronous, some amount of perturbation is always possible without causing a major change in available bandwidth. In this region, CUBIC is being very careful by very slowly increasing its window size. The convex profile ensures that the window increases very slowly at the beginning and gradually increases its increase rate. We also call this region the "maximum probing phase" since CUBIC is searching for a new W_max. In this region, cwnd MUST be incremented by (W_cubic(t+RTT) - cwnd)/ cwnd for each received ACK, where W_cubic(t+RTT) is calculated using Eq. 1.

혼잡 회피 상태에서 ACK를 수신할 때, CUBIC이 TCP 친화적인 영역에 있지 않고 cwnd가 W_max보다 크거나 같으면, CUBIC은 볼록한 영역에 있다. 볼록한 영역은 마지막 혼잡 이벤트 이후 네트워크 상태가 혼란스러웠을 수 있음을 나타내며, 흐름 이탈 후 더 많은 가용 대역폭을 의미일 수 있다. 인터넷은 매우 비동기적이기 때문에, 이용 가능한 대역폭에 큰 변화를 일으키지 않고도 항상 어느 정도의 혼란이 발생할 수 있다. 이 영역에서 CUBIC은 윈도우 크기를 매우 천천히 늘려 매우 조심하고 있다. 볼록한 프로파일은 윈도우가 처음에 매우 느리게 증가하도록 보장하고 점차 증가율을 증가시킨다. 우리는 또한 CUBIC이 새로운 W_max를 찾고 있기 때문에 이 영역을 "최대 탐지 단계"라고 부른다. 이 영역에서 cwnd는 반드시 (W_cubic(t+RTT) - cwnd)/cwnd씩 증가해야 하며 여기서 W_cubic(t+RTT)은 Eq. 1을 사용하여 계산된다.(MUST)

4.5. Multiplicative Decrease

When a packet loss is detected by duplicate ACKs or a network congestion is detected by ECN-Echo ACKs, CUBIC updates its W_max, cwnd, and ssthresh as follows. Parameter beta_cubic SHOULD be set to 0.7.

중복 ACK에 의해 패킷 손실이 감지되거나 ECN-Echo ACKs에 의해 네트워크 혼잡이 감지되면, CUBIC은 다음과 같이 W_max, cwnd 및 ssthresh를 업데이트한다. 매개변수 beta_cubic은 0.7로 설정되어야 한다.(SHOULD)

```
W_max = cwnd;                 // save window size before reduction
ssthresh = cwnd * beta_cubic; // new slow-start threshold
ssthresh = max(ssthresh, 2);  // threshold is at least 2 MSS
cwnd = cwnd * beta_cubic;     // window reduction
```

A side effect of setting beta_cubic to a value bigger than 0.5 is slower convergence. We believe that while a more adaptive setting of beta_cubic could result in faster convergence, it will make the analysis of CUBIC much harder. This adaptive adjustment of beta_cubic is an item for the next version of CUBIC.

beta_cubic을 0.5보다 큰 값으로 설정하면 느린 수렴이 된다. 우리는 beta_cubic의 좀 더 적응적인 설정은 더 빠른 수렴을 초래할 수 있지만, 그것은 CUBIC의 분석을 훨씬 어렵게 만들 것이라고 믿는다. beta_cubic의 이러한 적응적 조정은 다음 버전의 CUBIC을 위한 항목이다.

4.6. Fast Convergence

To improve the convergence speed of CUBIC, we add a heuristic in CUBIC. When a new flow joins the network, existing flows in the network need to give up some of their bandwidth to allow the new flow some room for growth if the existing flows have been using all the bandwidth of the network. To speed up this bandwidth release by existing flows, the following mechanism called "fast convergence" SHOULD be implemented.

CUBIC의 수렴 속도를 향상시키기 위해 CUBIC에 휴리스틱을 추가한다. 새로운 흐름이 네트워크에 결합할 때, 기존 흐름이 네트워크의 모든 대역폭을 사용하고 있었다면, 새로운 흐름이 어느 정도 증가할 수 있는 여지를 갖도록 하기 위해 네트워크의 기존 흐름은 대역폭의 일부를

포기할 필요가 있다. 기존 흐름에 의한 대역폭 방출 속도를 높이기 위해서는 "빠른 수렴"이라고 하는 다음과 같은 메커니즘이 구현되어야 한다.(SHOULD)

With fast convergence, when a congestion event occurs, before the window reduction of the congestion window, a flow remembers the last value of W_max before it updates W_max for the current congestion event. Let us call the last value of W_max to be W_last_max.

빠른 수렴으로 혼잡 이벤트가 발생할 때 혼잡 윈도우의 윈도우 감소 전에 흐름은 현재 혼잡 이벤트에 대해 W_max를 업데이트하기 전에 W_max의 마지막 값을 기억한다. W_max의 마지막 값을 W_last_max라고 부른다.

```
if (W_max < W_last_max){ // should we make room for others
    W_last_max = W_max;            // remember the last W_max
    W_max = W_max*(1.0+beta_cubic)/2.0; // further reduce W_max
} else {
    W_last_max = W_max             // remember the last W_max
}
```

At a congestion event, if the current value of W_max is less than W_last_max, this indicates that the saturation point experienced by this flow is getting reduced because of the change in available bandwidth. Then we allow this flow to release more bandwidth by reducing W_max further. This action effectively lengthens the time for this flow to increase its congestion window because the reduced W_max forces the flow to have the plateau earlier. This allows more time for the new flow to catch up to its congestion window size.

혼잡 이벤트에서 현재 값인 W_max가 W_last_max보다 작을 경우 사용 가능한 대역폭의 변화로 인해 이 흐름에 의해 경험되는 포화점이 감소하고 있음을 나타낸다. 그리고 W_max를 더 줄임으로써 이 흐름이 더 많은 대역폭을 방출하도록 허용한다. 이 작용은 감소된 W_max가 흐름을 더 일찍 정체 현상을 가지도록 하기 때문에, 이 흐름이 혼잡 윈도우를 증가시키는 시간을 효과적으로 연장시킨다. 이것은 새로운 흐름이 혼잡 윈도우 크기에 따라잡을 수 있는 더 많은 시간을 허용한다.

The fast convergence is designed for network environments with multiple CUBIC flows. In network environments with only a single CUBIC flow and without any other traffic, the fast convergence SHOULD be disabled.

빠른 수렴은 다중 CUBIC 흐름이 있는 네트워크 환경을 위해 설계되었다. 단일 CUBIC 흐름만 있고 다른 트래픽이 없는 네트워크 환경에서는 빠른 수렴을 비활성화해야 한다.(SHOULD)

4.7. Timeout

In case of timeout, CUBIC follows Standard TCP to reduce cwnd [RFC5681], but sets ssthresh using beta_cubic (same as in Section 4.5) that is different from Standard TCP [RFC5681].

시간 초과의 경우, CUBIC은 cwnd [RFC5681]를 줄이기 위해 표준 TCP를 따르지만, 표준 TCP [RFC5681]와 다른 beta_cubic(Section 4.5와 동일)을 사용하여 ssthresh를 설정한다.

During the first congestion avoidance after a timeout, CUBIC increases its congestion window size using Eq. 1, where t is the elapsed time since the beginning of the current congestion avoidance, K is set to 0, and W_max is set to the congestion window size at the beginning of the current congestion avoidance.

시간 초과 후 첫 번째 혼잡 회피 상태 시, CUBIC은 Eq. 1을 사용하여 혼잡 윈도우 크기를 증가시킨다. 여기서 t는 현재 혼잡 회피 상태 시작 이후 경과된 시간이며, K는 0으로, W_max는 현재 혼잡 회피 상태 시작 시 혼잡 윈도우 크기로 설정된다.

4.8. Slow Start

CUBIC MUST employ a slow-start algorithm, when the cwnd is no more than ssthresh. Among the slow-start algorithms, CUBIC MAY choose the standard TCP slow start [RFC5681] in general networks, or the limited slow start [RFC3742] or hybrid slow start [HR08] for fast and long-distance networks.

CUBIC은 반드시 느린 시작 알고리즘을 사용해야 하며,(MUST) cwnd가 ssthresh에 지나지 않을 경우. 느린 시작 알고리즘 중에서 CUBIC은 일반 네트워크의 표준 TCP 느린 시작 [RFC5681] 또는 고속 및 장거리 네트워크의 경우 제한된 느린 시작 [RFC3742] 또는 하이브리드 느린 시작을 [HR08] 선택할 수 있다.(MAY)

In the case when CUBIC runs the hybrid slow start [HR08], it may exit the first slow start without incurring any packet loss and thus W_max is undefined. In this special

case, CUBIC switches to congestion avoidance and increases its congestion window size using Eq. 1, where t is the elapsed time since the beginning of the current congestion avoidance, K is set to 0, and W_max is set to the congestion window size at the beginning of the current congestion avoidance.

CUBIC이 하이브리드 느린 시작을 [HR08] 실행할 때, 패킷 손실이 발생하지 않고 첫 번째 느린 시작을[HR08] 종료할 수 있으므로 W_max가 정의되지 않는다. 이 특별한 경우, CUBIC은 혼잡 회피 상태로 전환하며 Eq. 1을 이용하여 혼잡 윈도우 크기를 증가시킨다. 여기서 t는 현재 혼잡 회피 상태가 시작된 이후 경과된 시간이며, K는 0으로, W_max는 현재 혼잡 회피 상태가 시작될 때 혼잡 윈도우 크기로 설정된다.

5. Discussion

In this section, we further discuss the safety features of CUBIC following the guidelines specified in [RFC5033].

이 섹션에서는 [RFC5033]에 명시된 지침에 따라 CUBIC의 안전 기능에 대해 자세히 논의한다.

With a deterministic loss model where the number of packets between two successive packet losses is always 1/p, CUBIC always operates with the concave window profile, which greatly simplifies the performance analysis of CUBIC. The average window size of CUBIC can be obtained by the following function:

2개의 연속적인 패킷 손실 사이의 패킷 수가 항상 1/p인 결정론적 손실 모델에서, CUBIC은 항상 오목한 윈도우 프로파일과 함께 작동하여 CUBIC의 성능 분석을 크게 단순화한다. CUBIC의 평균 윈도우 크기는 다음과 같은 함수로 얻을 수 있다.

AVG_W_cubic = [C*(3+beta_cubic)/(4*(1-beta_cubic))]^0.25 * (RTT^0.75) / (p^0.75) (Eq. 5)

With beta_cubic set to 0.7, the above formula is reduced to:

beta_cubic을 0.7로 설정하면 위의 공식은 다음과 같이 감소한다.

AVG_W_cubic = (C*3.7/1.2)^0.25 * (RTT^0.75) / (p^0.75) (Eq. 6)

We will determine the value of C in the following subsection using Eq. 6.

우리는 Eq. 6을 사용하여 다음 서브 섹션에서 C의 값을 결정할 것이다.

5.1. Fairness to Standard TCP

In environments where Standard TCP is able to make reasonable use of the available bandwidth, CUBIC does not significantly change this state.

표준 TCP가 가용 대역폭을 합리적으로 사용할 수 있는 환경에서, CUBIC은 이 상태를 유의미하게 변경하지 않는다.

Standard TCP performs well in the following two types of networks:

표준 TCP는 다음의 두 가지 유형의 네트워크에서 좋은 성능을 발휘한다.

1. networks with a small bandwidth-delay product (BDP)

1. 대역폭-지연 곱(BDP)이 작은 네트워크

2. networks with a short RTTs, but not necessarily a small BDP

2. RTT가 짧지만 반드시 작은 BDP는 아닌 네트워크

CUBIC is designed to behave very similarly to Standard TCP in the above two types of networks. The following two tables show the average window sizes of Standard TCP, HSTCP, and CUBIC. The average window sizes of Standard TCP and HSTCP are from [RFC3649]. The average window size of CUBIC is calculated using Eq. 6 and the CUBIC TCP-friendly region for three different values of C.

CUBIC은 위의 두 가지 유형의 네트워크에서 표준 TCP와 매우 유사하게 동작하도록 설계되었다. 다음 두 표는 표준 TCP, HSTCP, CUBIC의 평균 윈도우 크기를 보여준다. Standard TCP와 HSTCP의 평균 윈도우 크기는 [RFC3649]이다. 평균 윈도우 크기는 C의 세 가지 다른 값에 대해 Eq. 6과 CUBIC TCP 친화적인 영역을 사용하여 계산한다.

Loss Rate P	Average TCP W	Average HSTCP W	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
10^-2	12	12	12	12	12
10^-3	38	38	38	38	59
10^-4	120	263	120	187	333
10^-5	379	1795	593	1054	1874
10^-6	1200	12279	3332	5926	10538
10^-7	3795	83981	18740	33325	59261
10^-8	12000	574356	105383	187400	333250

Table 1

Table 1 describes the response function of Standard TCP, HSTCP, and CUBIC in networks with RTT = 0.1 seconds. The average window size is in MSS-sized segments.

표 1은 RTT = 0.1초인 네트워크에서 표준 TCP, HSTCP, CUBIC의 응답 함수를 설명한다. 평균 윈도우 크기는 MSS 크기 세그먼트이다.

Loss Rate P	Average TCP W	Average HSTCP W	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
10^-2	12	12	12	12	12
10^-3	38	38	38	38	38
10^-4	120	263	120	120	120
10^-5	379	1795	379	379	379
10^-6	1200	12279	1200	1200	1874
10^-7	3795	83981	3795	5926	10538
10^-8	12000	574356	18740	33325	59261

Table 2

Table 2 describes the response function of Standard TCP, HSTCP, and CUBIC in networks with RTT = 0.01 seconds. The average window size is in MSS-sized segments.

표 2는 RTT = 0.01초인 네트워크에서 표준 TCP, HSTCP, CUBIC의 응답 함수를 설명한다. 평균 윈도우 크기는 MSS 크기 세그먼트이다.

Both tables show that CUBIC with any of these three C values is more friendly to TCP than HSTCP, especially in networks with a short RTT where TCP performs reasonably well. For example, in a network with RTT = 0.01 seconds and p=10^-6, TCP has an average window of 1200 packets. If the packet size is 1500 bytes, then TCP can achieve an average rate of 1.44 Gbps. In this case, CUBIC with C=0.04 or C=0.4 achieves exactly the same rate as Standard TCP, whereas HSTCP is about ten times more aggressive than Standard TCP.

두 표 모두 이러한 세 가지 C 값 중 어느 하나를 가진 CUBIC이 HSTCP보다 TCP에 더 친화적이라는 것을 보여주는데, 특히 TCP가 합리적으로 잘 작동하는 RTT가 짧은 네트워크에서 그러하다. 예를 들어, RTT = 0.01초와 p=10^-6이 있는 네트워크에서, TCP의 평균 윈도우는 1200 패킷이다. 패킷 크기가 1500바이트라면 TCP는 평균 1.44Gbps의 속도를 달성할 수 있다. 이 경우 C=0.04 또는 C=0.4를 가진 CUBIC은 표준 TCP와 정확히 같은 비율을 달성하는 반면 HSTCP는 표준 TCP보다 약 10배 더 위협적이다.

We can see that C determines the aggressiveness of CUBIC in competing with other congestion control algorithms for bandwidth. CUBIC is more friendly to Standard TCP, if the value of C is lower. However, we do not recommend setting C to a very low value like 0.04, since CUBIC with a low C cannot efficiently use the bandwidth in long RTT and high-bandwidth networks. Based on these observations and our experiments, we find C=0.4 gives a good balance between TCP-friendliness and aggressiveness of window increase. Therefore, C SHOULD be set to 0.4. With C set to 0.4, Eq. 6 is reduced to:

우리는 C가 대역폭에 대한 다른 혼잡 제어 알고리즘과 경쟁할 때 CUBIC의 강도를 결정한다는 것을 알 수 있다. CUBIC은 C의 값이 더 낮을 경우 표준 TCP에 더 친화적이다. 단, C가 낮은 CUBIC은 긴 RTT와 고대역폭 네트워크에서 대역폭을 효율적으로 사용할 수 없기 때문에 C를 0.04와 같이 매우 낮은 값으로 설정하는 것을 권장하지 않는다. 이러한 관찰과 우리의 실험을 바탕으로, 우리는 C=0.4가 TCP 친화성과 윈도우 상승의 위협성 사이에 좋은 균형을 제공한다는 것을 발견한다. 따라서 C는 0.4로 설정되어야 한다. C를 0.4로 설정하면 Eq. 6은 다음과 같이 감소한다.(SHOULD)

AVG_W_cubic = 1.054 * (RTT^0.75) / (p^0.75) (Eq. 7)

Eq. 7 is then used in the next subsection to show the scalability of CUBIC.

다음 서브 섹션에서 Eq. 7을 사용하여 CUBIC의 확장성을 보여준다.

5.2. Using Spare Capacity

CUBIC uses a more aggressive window increase function than Standard TCP under long RTT and high-bandwidth networks.

CUBIC은 긴 RTT 및 고대역폭 네트워크 하에서 표준 TCP보다 더 위협적인 윈도우 증가 기능을 사용한다.

The following table shows that to achieve the 10 Gbps rate, Standard TCP requires a packet loss rate of 2.0e-10, while CUBIC requires a packet loss rate of 2.9e-8.

다음 표는 10Gbps 속도를 달성하려면 표준 TCP는 2.0e-10의 패킷 손실률을 요구하는 반면, CUBIC은 2.9e-8의 패킷 손실률을 요구하는 것을 보여준다.

Throughput(Mbps)	Average W	TCP P	HSTCP P	CUBIC P
1	8.3	2.0e-2	2.0e-2	2.0e-2
10	83.3	2.0e-4	3.9e-4	2.9e-4
100	833.3	2.0e-6	2.5e-5	1.4e-5
1000	8333.3	2.0e-8	1.5e-6	6.3e-7
10000	83333.3	2.0e-10	1.0e-7	2.9e-8

Table 3

Table 3 describes the required packet loss rate for Standard TCP, HSTCP, and CUBIC to achieve a certain throughput. We use 1500-byte packets and an RTT of 0.1 seconds.

표 3은 일정한 처리량을 달성하기 위해 표준 TCP, HSTCP 및 CUBIC에 필요한 패킷 손실률을 설명한다. 우리는 1500 바이트 패킷과 0.1초의 RTT를 사용한다.

Our test results in [HKLRX06] indicate that CUBIC uses the spare bandwidth left unused by existing Standard TCP flows in the same bottleneck link without taking away much bandwidth from the existing flows.

[HKLRX06]의 테스트 결과에 따르면, 기존 흐름에서 많은 대역폭을 빼앗지 않고 동일한 병목 현상 링크에서 기존 Standard TCP 흐름에 의해 사용되지 않은 예비 대역폭을 사용하는 것으로 나타났다.

5.3. Difficult Environments

CUBIC is designed to remedy the poor performance of TCP in fast and long-distance networks.

CUBIC은 고속 및 장거리 네트워크에서 TCP의 저조한 성능을 교정하기 위해 설계되었다.

5.4. Investigating a Range of Environments

CUBIC has been extensively studied by using both NS-2 simulation and test-bed experiments covering a wide range of network environments. More information can be found in [HKLRX06].

CUBIC 은 광범위한 네트워크 환경을 포괄하는 NS-2 시뮬레이션과 테스트베드 실험을 모두 사용하여 광범위하게 연구되어 왔다. 자세한 내용은 [HKLRX06]에서 확인할 수 있다.

Same as Standard TCP, CUBIC is a loss-based congestion control algorithm. Because CUBIC is designed to be more aggressive (due to a faster window increase function and bigger multiplicative decrease factor) than Standard TCP in fast and long-distance networks, it can fill large drop-tail buffers more quickly than Standard TCP and increase the risk of a standing queue [KWAF17]. In this case, proper queue sizing and management [RFC7567] could be used to reduce the packet queuing delay.

표준 TCP와 마찬가지로, CUBIC은 손실 기반의 혼잡 제어 알고리즘이다. CUBIC은 빠르고 장거리 네트워크에서 표준 TCP보다 더 위협적이도록 설계되었기 때문에(윈도우 증가 함수가 빠

르고 지수 감소 요인이 크기 때문에), 표준 TCP보다 더 빠르게 대형 drop-tail 버퍼를 채울 수 있으며, 스탠딩 대기열의 위험을 증가시킬 수 있다[KWAF17]. 이 경우 적절한 대기열 크기 조정 및 관리 [RFC7567]를 사용하여 패킷 대기열 지연을 줄일 수 있다.

5.5. Protection against Congestion Collapse

With regard to the potential of causing congestion collapse, CUBIC behaves like Standard TCP since CUBIC modifies only the window adjustment algorithm of TCP. Thus, it does not modify the ACK clocking and Timeout behaviors of Standard TCP.

혼잡 붕괴 발생 가능성에 관해서, CUBIC은 TCP의 윈도우 조정 알고리즘만 수정하기 때문에, CUBIC은 표준 TCP와 같이 동작한다. 따라서 표준 TCP의 ACK 클로킹과 타임아웃 동작을 수정하지 않는다.

5.6. Fairness within the Alternative Congestion Control Algorithm

CUBIC ensures convergence of competing CUBIC flows with the same RTT in the same bottleneck links to an equal throughput. When competing flows have different RTTs, their throughput ratio is linearly proportional to the inverse of their RTT ratios. This is true independent of the level of statistical multiplexing in the link.

CUBIC은 동일한 링크의 병목현상에서 동일한 RTT로 경쟁하는 CUBIC 흐름의 수렴을 보장하여 동일한 처리량을 제공한다. 경쟁 흐름의 RTTs가 서로 다를 경우, 처리량 비율은 RTT 비율의 역으로 선형 비례한다. 이것은 링크의 통계적 멀티플렉싱 단계와는 독립적이다.

5.7. Performance with Misbehaving Nodes and Outside Attackers

This is not considered in the current CUBIC.

이것은 현재 CUBIC내에서 고려되지 않는다.

5.8. Behavior for Application-Limited Flows

CUBIC does not raise its congestion window size if the flow is currently limited by the application instead of the congestion window. In case of long periods when cwnd has not been updated due to the application rate limit, such as idle periods, t in Eq. 1 MUST NOT include these periods; otherwise, W_cubic(t) might be very high after restarting from these periods.

CUBIC은 현재 혼잡 윈도우 대신 애플리케이션에 의해 흐름이 제한되어 있다면 혼잡 윈도우 크기를 올리지 않는다. 유휴 기간과 같이 애플리케이션 비율 제한으로 인해 cwnd가 업데이트되지 않은 긴 기간의 경우, Eq. 1의 t는 이 기간을 포함해서는 안 된다.(MUST NOT) 그렇지 않으면 이 기간부터 다시 시작한 후 W_cubic(t)이 매우 높을 수 있다.

5.9. Responses to Sudden or Transient Events

If there is a sudden congestion, a routing change, or a mobility event, CUBIC behaves the same as Standard TCP.

갑작스러운 혼잡, 라우팅 변경 또는 이동성 이벤트가 발생하는 경우, CUBIC은 표준 TCP와 동일하게 동작한다.

5.10. Incremental Deployment

CUBIC requires only the change of TCP senders, and it does not make any changes to TCP receivers. That is, a CUBIC sender works correctly with the Standard TCP receivers. In addition, CUBIC does not require any changes to the routers and does not require any assistance from the routers.

CUBIC은 TCP 송신자의 변경만을 필요로 하며, TCP 수신자의 변경은 하지 않는다. 즉, CUBIC 송신자는 표준 TCP 수신자와 정확하게 작동한다. 또한, CUBIC은 라우터에 대한 어떠한 변경도 요구하지 않으며 라우터의 어떤 도움도 요구하지 않는다.

6. Security Considerations

This proposal makes no changes to the underlying security of TCP. More information about TCP security concerns can be found in [RFC5681].

이 제안은 TCP의 기본 보안을 변경하지 않는다. TCP 보안 문제에 대한 자세한 내용은 [RFC5681]에서 확인할 수 있다.

7. IANA Considerations

This document does not require any IANA actions.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <https://www.rfc-editor.org/info/rfc3649>.

[RFC3742] Floyd, S., "Limited Slow-Start for TCP with Large Congestion Windows", RFC 3742, DOI 10.17487/RFC3742, March 2004, <https://www.rfc-editor.org/info/rfc3742>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <https://www.rfc-editor.org/info/rfc5033>.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, “TCP Friendly Rate Control (TFRC): Protocol Specification”, RFC 5348, DOI 10.17487/RFC5348, September 2008, <https://www.rfc-editor.org/info/rfc5348>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, “The NewReno Modification to TCP's Fast Recovery Algorithm”, RFC 6582, DOI 10.17487/RFC6582, April 2012, <https://www.rfc-editor.org/info/rfc6582>.

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP”, RFC 6675, DOI 10.17487/RFC6675, August 2012, <https://www.rfc-editor.org/info/rfc6675>.

[RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., “IETF Recommendations Regarding Active Queue Management”, BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <https://www.rfc-editor.org/info/rfc7567>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

8.2. Informative References

[CEHRX07] Cai, H., Eun, D., Ha, S., Rhee, I., and L. Xu, “Stochastic Ordering for Internet Congestion Control and its Applications", In Proceedings of IEEE INFOCOM, DOI 10.1109/INFCOM.2007.111, May 2007.

[FHP00] Floyd, S., Handley, M., and J. Padhye, "A Comparison of Equation-Based and AIMD Congestion Control", May 2000.

[GV02] Gorinsky, S. and H. Vin, "Extended Analysis of Binary Adjustment Algorithms", Technical Report TR2002-29, Department of Computer Sciences, The University of Texas at Austin, August 2002.

[HKLRX06] Ha, S., Kim, Y., Le, L., Rhee, I., and L. Xu, "A Step toward Realistic Performance Evaluation of High-Speed TCP Variants", International Workshop on Protocols for Fast Long-Distance Networks.

[HR08] Ha, S. and I. Rhee, "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks", International Workshop on Protocols for Fast Long-Distance Networks.

[HRX08] Ha, S., Rhee, I., and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant", ACM SIGOPS Operating System Review, DOI 10.1145/1400097.1400105, July 2008.

[K03] Kelly, T., "Scalable TCP: Improving Performance in HighSpeed Wide Area Networks", ACM SIGCOMM Computer Communication Review, DOI 10.1145/956981.956989, April 2003.

[KWAF17] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", Work in Progress, draft-ietf-tcpm-alternativebackoff-ecn-05, December 2017.

[XHR04] Xu, L., Harfoush, K., and I. Rhee, "Binary Increase Congestion Control for Fast, Long Distance Networks", In Proceedings of IEEE INFOCOM, DOI 10.1109/INFCOM.2004.1354672, March 2004.

Acknowledgements

Alexander Zimmermann and Lars Eggert have received funding from the European Union's Horizon 2020 research and innovation program 2014-2018 under grant agreement No. 644866 (SSICLOPS). This document reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

The work of Lisong Xu was partially supported by the National Science Foundation (NSF) under Grant No. 1526253. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

Authors' Addresses

Injong Rhee
North Carolina State University
Department of Computer Science
Raleigh, NC 27695-7534
United States of America
Email: rhee@ncsu.edu

Lisong Xu
University of Nebraska-Lincoln
Department of Computer Science and Engineering
Lincoln, NE 68588-0115
United States of America

Email: xu@unl.edu

Sangtae Ha
University of Colorado at Boulder
Department of Computer Science
Boulder, CO 80309-0430
United States of America
Email: sangtae.ha@colorado.edu

Alexander Zimmermann
Phone: +49 175 5766838
Email: alexander.zimmermann@rwth-aachen.de

Lars Eggert
NetApp
Sonnenallee 1
Kirchheim 85551
Germany
Phone: +49 151 12055791
Email: lars@netapp.com

Richard Scheffenegger
NetApp
Am Europlatz 2
Vienna 1120
Austria
Email: rs.ietf@gmx.at