(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2014-0130212
(43) 공개일자 2014년11월07일

(51) 국제특허분류(Int. Cl.)
H04L 29/06 (2006.01) *H04L 12/861* (2013.01)
(21) 출원번호 10-2014-7027022
(22) 출원일자(국제) 2013년02월26일
심사청구일자 없음
(85) 번역문제출일자 2014년09월25일
(86) 국제출원번호 PCT/US2013/027807
(87) 국제공개번호 WO 2013/130473
국제공개일자 2013년09월06일
(30) 우선권주장
13/745,799 2013년01월19일 미국(US)
61/603,569 2012년02월27일 미국(US)

(71) 출원인
퀄컴 인코포레이티드
미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
(72) 발명자
가오, 퀴앙
미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
루비, 마이클 조지
미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
(뒷면에 계속)
(74) 대리인
특허법인 남앤드남

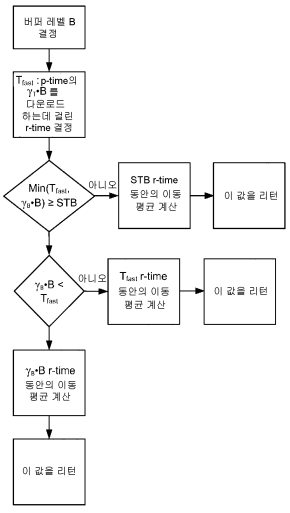
전체 청구항 수 : 총 17 항

(54) 발명의 명칭 **다운로드 레이트 평가기를 갖는 개선된 DASH 클라이언트 및 수신기**

(57) 요 약

클라이언트 디바이스는 스트리밍 미디어를 제공하고 스트림들을 제어하기 위한 스트림 매니저, 콘텐츠에 대한 네트워크 리퀘스트들을 하기 위한 리퀘스트 액셀러레이터, 스트림 매니저 및 리퀘스트 액셀러레이터에 커플링되고 어떤 리퀘스트들을 할지 결정하기 위한 소스 컴포넌트, 네트워크 접속, 및 미디어 플레이어를 포함한다. 수신 레이트 변화에 빠르게 반응하는 레이트 추정을 위한 프로세스가 제공된다. 윈도윙 너비를 크게(따라서 특정 편차를 크게) 유지하면서, 필요가 있다면 레이트가 충분히 빠르게 적응되는 것을 보증하는 방식으로 레이트 추정기는 적응성 윈도우드 평균을 이용하고 비디오 버퍼 레벨 및 비디오 버퍼 레벨의 변화를 고려할 수 있다. 보증은 레이트가 하락하거나 상승할 때, 추정기가 버퍼가 비워지는 레이트 또는 버퍼가 채워진 레벨에 비례하는 시간 내에 그 추정을 조정하는 것일 수 있다.

대 표 도 - 도14



(72) 발명자

마오, 인니안

미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775

마인더, 로렌즈 크리스토프

미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775

특허청구의 범위

청구항 1

유한 대역폭을 가지는 네트워크 경로에 의해 데이터 소스들로 커플링되는 수신기에서 다운로드 레이트를 추정하는 방법으로서,

상기 다운로드 레이트는 상기 네트워크 경로를 통해 상기 수신기에서 데이터가 수신되는 레이트이고, 상기 방법은,

상기 수신기의 프리젠테이션 버퍼를 모니터하는 단계 -상기 프리젠테이션 버퍼는 미디어 데이터를, 적어도 상기 미디어 데이터가 수신되는 시간과 상기 수신기와 연관된 프리젠테이션 엘리먼트에 의해 상기 미디어 데이터가 소비되는 시간 사이에 저장함-;

상기 다운로드 레이트의 추정이 기초할 0 이 아닌 추정 주기를 결정하는 단계;

상기 추정 주기 동안 버퍼 레벨들의 표시들을 저장하는 단계 -주어진 시간의 버퍼 레벨은, 수신되지만 상기 프리젠테이션 엘리먼트에 의해 아직 소비되지 않은 미디어 데이터에 의해, 상기 주어진 시간에 상기 프리젠테이션 버퍼가 적어도 대략적으로 얼마나 많이 차지되는지에 대응함-; 및

저장된 표시들을 상기 추정된 다운로드 레이트의 측정의 부분으로서 사용하는 단계를 포함하는, 다운로드 레이트를 추정하는 방법.

청구항 2

제 1 항에 있어서,

상기 프리젠테이션 엘리먼트는 디스플레이 및 오디오 출력을 포함하는, 다운로드 레이트를 추정하는 방법.

청구항 3

제 1 항에 있어서,

상기 다운로드 레이트가 측정되는 상기 추정 주기는 측정된 버퍼 레벨에, 미리결정된 비례 인수로 비례하는 지속 기간을 갖는, 다운로드 레이트를 추정하는 방법.

청구항 4

제 1 항에 있어서,

상기 추정 주기의 지속 기간은 측정 시간에 프리젠테이션 버퍼의 소비되지 않은 미디어 데이터의 바이트들의 수에 비례하도록 취해지는, 다운로드 레이트를 추정하는 방법.

청구항 5

제 1 항에 있어서,

상기 추정 주기는, 적어도 부분적으로, 미디어가 상기 프리젠테이션 버퍼에 추가되는 추가 레이트의 함수인, 다운로드 레이트를 추정하는 방법.

청구항 6

제 1 항에 있어서,

상기 추정 주기는 상기 프리젠테이션 버퍼의 미리 결정된 부분을 다운로드하는데 사용되는 시간에 비례하는, 다운로드 레이트를 추정하는 방법.

청구항 7

제 6 항에 있어서,

상기 추정된 다운로드 레이트는 상기 추정 주기와 같은 시간의 이전 지속 기간에 걸친 평균 다운로드 레이트인, 다운로드 레이트를 추정하는 방법.

청구항 8

제 1 항에 있어서,

상기 추정 주기는 상기 프리젠테이션 버퍼의 콘텐츠의 미리 결정된 부분이 다운로드되는 시간 및 상기 프리젠테이션 버퍼에 있는 미디어 데이터의 프리젠테이션 시간 중 더 작은 것인, 다운로드 레이트를 추정하는 방법.

청구항 9

제 8 항에 있어서,

상기 추정된 다운로드 레이트는 상기 추정 주기와 같은 시간의 이전 지속 기간에 걸친 평균 다운로드 레이트인, 다운로드 레이트를 추정하는 방법.

청구항 10

유한 대역폭을 가지는 네트워크 경로에 의해 데이터 소스들로 커플링되는 수신기에서 다운로드 레이트를 추정하기 위해 프로세서에 의해 실행가능한 비-일시적 컴퓨터 판독가능 매체로서,

상기 다운로드 레이트는 상기 네트워크 경로를 통해 상기 수신기에서 데이터가 수신되는 레이트이고,

상기 수신기의 프리젠테이션 버퍼를 모니터하기 위한 프로그램 코드 -상기 프리젠테이션 버퍼는 미디어 데이터를, 적어도 상기 미디어 데이터가 수신되는 시간과 상기 수신기와 연관된 프리젠테이션 엘리먼트에 의해 상기 미디어 데이터가 소비되는 시간 사이에 저장함-;

상기 다운로드 레이트의 추정이 기초할 0 이 아닌 추정 주기를 결정하기 위한 프로그램 코드;

상기 추정 주기 동안 버퍼 레벨들의 표시들을 저장하기 위한 프로그램 코드 -주어진 시간의 버퍼 레벨은, 수신되지만 상기 프리젠테이션 엘리먼트에 의해 아직 소비되지 않은 미디어 데이터에 의해, 상기 주어진 시간에 상기 프리젠테이션 버퍼가 적어도 대략적으로 얼마나 많이 차지되는지에 대응함-; 및

저장된 표시들을 상기 추정된 다운로드 레이트의 측정의 부분으로서 사용하기 위한 프로그램 코드를 포함하는, 비-일시적 컴퓨터 판독가능 매체.

청구항 11

제 10 항에 있어서,

디스플레이 및 오디오 출력에 인터페이싱하기 위한 프로그램 코드를 더 포함하는, 비-일시적 컴퓨터 판독가능 매체.

청구항 12

제 10 항에 있어서,

측정된 버퍼 레벨에, 미리결정된 비례 인수로 비례하는 지속 기간을 사용하여 상기 다운로드 레이트가 측정되는 상기 추정 주기를 계산하기 위한 프로그램 코드를 더 포함하는, 비-일시적 컴퓨터 판독가능 매체.

청구항 13

제 10 항에 있어서,

측정 시간에 프리젠테이션 버퍼의 소비되지 않은 미디어 데이터의 바이트들의 수에 비례하는 상기 추정 주기의 지속 시간을 계산하기 위한 프로그램 코드를 더 포함하는, 비-일시적 컴퓨터 판독가능 매체.

청구항 14

제 10 항에 있어서,

적어도 부분적으로, 미디어가 상기 프리젠테이션 버퍼에 추가되는 추가 레이트의 함수로서 및/또는 상기 프리젠

테이션 버퍼의 미리 결정된 부분을 다운로드하는데 사용되는 시간에 비례하는 상기 추정 주기를 계산하기 위한 프로그램 코드를 더 포함하는, 비-일시적 컴퓨터 판독가능 매체.

청구항 15

제 14 항에 있어서,

상기 추정된 다운로드 레이트는 상기 추정 주기와 같은 시간의 이전 지속 기간에 걸친 평균 다운로드 레이트인, 비-일시적 컴퓨터 판독가능 매체.

청구항 16

제 10 항에 있어서,

상기 추정 주기는 상기 프리젠테이션 버퍼의 콘텐츠의 미리 결정된 부분이 다운로드되는 시간 및 상기 프리젠테이션 버퍼에 있는 미디어 데이터의 프리젠테이션 시간 중 더 작은 것인, 비-일시적 컴퓨터 판독가능 매체.

청구항 17

제 16 항에 있어서,

상기 추정된 다운로드 레이트는 상기 추정 주기와 같은 시간의 이전 지속 기간에 걸친 평균 다운로드 레이트인, 비-일시적 컴퓨터 판독가능 매체.

명 세 서

기 술 분 야

[0001] 관련 출원들에 대한 상호-참조

[0002] [1] 본 출원은, 발명의 명칭이 “Improved DASH Client and Receiver with Rate Adaptation and Downloading for Adaptive Video" 로 2012년 2월 27일자로 출원된 미국 가출원 제 61/603,569호에 대한 우선권을 주장하며, 그 내용들은, 사실상 그 전체가 인용에 의해 본원에 포함된다.

배 경 기 술

[0003] [2] DASH는 “Dynamic Adaptive Streaming over HTTP”를 지칭한다. DASH를 이용하여, 콘텐츠 제공자는 콘텐츠를 MPD 파일들과 같은 연관 메타데이터와 함께 세그먼트들, 프래그먼트들, 리프리젠테이션들, 어뎁테이션들 및 기타 등등으로 포맷하고, 그들 모두를 표준 HTTP 서버 또는 특수 HTTP 서버를 통해 이용 가능한 파일들로 저장한다. DASH 클라이언트는 DASH 클라이언트의 사용자에게 프리젠테이션을 제공하기 위해 필요한 이들 파일들을 획득하는 수신기이다.

[0004] [3] 사용자들은 일반적으로 네트워크가 제한된 환경에서 예고 없이 고품질의 스트리밍을 원하기 때문에, DASH 클라이언트들은 까다로운 제약을 갖는다. 따라서, 개선된 DASH 클라이언트들이 바람직하다.

발명의 내용

[0005] [4] 클라이언트 디바이스는 스트리밍 미디어를 제공하고, 스트림들을 제어하기 위한 스트림 매니저, 콘텐츠에 대한 네트워크 리퀘스트들을 하기 위한 리퀘스트 액셀러레이터, 스트림 매니저 및 리퀘스트 액셀러레이터에 커플링되고 어떤 리퀘스트들을 할지 결정하기 위한 소스 컴포넌트, 네트워크 접속, 및 미디어 플레이어를 포함한다. 리퀘스트 액셀러레이터는 리퀘스트들을 버퍼링하기 위한 리퀘스트 데이터 버퍼 및 응답할 수 있는 각 리퀘스트에 완전한 응답들을 되돌려주기 위한 로직을 포함한다. 스트림 매니저, 리퀘스트 액셀러레이터, 및 소스 컴포넌트는 프로세서 명령들 또는 프로그램 코드로서 구현될 수 있고, 클라이언트 디바이스는 프로그램 메모리, 워킹 메모리, 프로세서, 및 전력 소스를 더 포함할 수 있다. 클라이언트 디바이스는 디스플레이 및 사용자 입력 디바이스 또한 포함할 수 있다. 클라이언트 태스크들은 데이터를 효율적으로 스트리밍 하기 위해 소스 컴포넌트, 스트림 매니저, 및 리퀘스트 액셀러레이터 사이에 파싱될 수 있다.

[0006] [5] 본원에 기술된 바와 같이, 다양한 측면에서, 클라이언트는 언제 리프리젠테이션을 유지할지 또는 다른 리프리젠테이션으로 스위칭 할지를 결정하는 것과 같은 동작을 수행하고, 어느 프래그먼트들을 요청할지를 결정

하며 미디어 플레이어가, 대부분의 상황에서, 스톨링 없이 스트림을 계속할 수 있는 충분한 데이터를 획득할 수 있음을 보증할 수 있다.

[0007] [6] 수신 레이트 변화에 빠르게 반응할 레이트 평가를 위한 프로세스가 제공된다. 다운로드 레이트 평가기는 특히 미디어 스트리밍에서의 사용에 적응된 적응성 윈도윙된 평균을 사용할 수 있다. 다운로드 레이트 추정기는 윈도윙 너비를 크게(따라서 특정 편차를 크게) 유지하면서, 필요가 있다면 레이트가 충분히 빠르게 적응되는 것을 보증하는 방식으로 비디오 버퍼 레벨 및 비디오 버퍼 레벨의 변화를 고려할 수 있다. 프로세스에 의해 제공되는 보증은 (a) B가 레이트 하락이 발생하는 때 버퍼의 비디오 데이터의 양(플레이백 시간의 초 단위)이면, 추정기는 버퍼가 B/2까지 비워지는데(drain) 걸린 시간 내에 그 레이트 추정을 조정할 것이고, (b) B가 레이트 증가가 일어나는 동안 버퍼의 데이터 양이면, 레이트 추정기는 새 레이트로 충분히 빠르게 조정하여 그것이 원칙적으로 많아도 3*B 내에 나타날 수 있도록 한다(스마트 레이트 변경 프로세스라고 가정하면).

[0008] [7] 예시적인 방법 또는 장치에서, 수신기는 수신기의 프리젠테이션 버퍼를 모니터링하고, 프리젠테이션 버퍼는 적어도 미디어 데이터가 수신된 시간 및 미디어 데이터가 수신기에 연관된 프리젠테이션 엘리먼트에 의해 소비되는 시간 사이에 미디어 데이터를 저장하고, 다운로드 레이트의 추정이 근거할 0이 아닌 추정 기간을 결정하고, 추정 기간 동안의 버퍼 레벨들의 표시자들을 저장하고, 소정 시간에서의 버퍼 레벨은 적어도 대략적으로, 얼마나 많은 프리젠테이션이 그 시간에, 수신되었지만 아직 프리젠테이션 엘리먼트에 의해 소비되지 않은 미디어 데이터에 의해 차지되었는지에 대응하고, 저장된 표시자를 추정된 다운로드 레이트의 측정의 부분으로서 사용한다.

[0009] [8] 프리젠테이션 엘리먼트는 디스플레이 및 오디오 출력을 포함할 수 있다. 다운로드 레이트가 측정되는 추정 주기는, 추정된 다운로드 레이트가 추정 주기와 같은 시간의 이전 지속 시간 동안의 평균 다운로드 레이트일 때와 같이, 측정 시간에 프리젠테이션 버퍼의 소비되지 않은 미디어 데이터의 바이트 수에 비례하고, 및/또는 미디어가 프리젠테이션 버퍼에 추가되는 추가 레이트의 함수이고, 및/또는 프리젠테이션 버퍼의 미리 결정된 부분을 다운로드하는데 사용되는 시간에 비례하는 지속 시간과 같이, 미리 결정된 비례 인수로, 측정된 버퍼 레벨에, 비례하는 지속시간을 가질 수 있다.

[0010] [9] 추정 주기는 프리젠테이션 버퍼의 콘텐츠의 미리결정된 부분이 다운로드된 시간 및 프리젠테이션 버퍼에 있는 미디어 데이터의 프리젠테이션 시간 중 더 작은 것일 수 있다. 추정된 다운로드 레이트는 추정 주기와 같은 시간의 이전 지속 시간 동안의 평균 다운로드 레이트일 수 있다.

[0011] [10] 다양한 엘리먼트들이 네트워크 경로에 의해 커플링된 소스 및 수신기 사이의 네트워크 경로를 통해 데이터 다운로드를 제어하기 위한 프로세서에 의한 실행을 위한 컴퓨터 판독가능 매체를 이용하여 구현될 수 있다. 컴퓨터 판독가능 매체는 비-일시적 컴퓨터 판독가능 매체일 수 있다.

[0012] [11] 발명의 다른 측면들이 이 설명들로부터 분명해 질 것이다.

도면의 간단한 설명

[0013] [12] 도 1은 DASH 배치에서 DASH 클라이언트를 포함하는 다양한 엘리먼트들을 도시하고, 기록, 콘텐츠 준비 및 콘텐츠 전달 단계들을 수반하여 어떻게 미디어 기록이 최종 사용자에게 도착하는지를 보여준다.

[13] 도 2는 다른 컴포넌트들을 갖는 DASH 클라이언트의 예시적 아키텍처를 나타내고, 이는 스트림 매니저, 리퀘스트 액셀러레이터, 소스 컴포넌트, 네트워크 접속, 및 미디어 플레이어를 포함한다.

[14] 도 3은 리프리젠테이션 스위칭 프로세스들의 타이밍차트이고, 백워드 루킹 프로세스(backward looking process)에 대한 도 3A 및 포워드 루킹 프로세스(forward looking process)에 대한 도 3B를 포함한다.

[15] 도 4는 스위치 포인트들이 정렬된 경우에 대한 리프리젠테이션 스위칭 프로세스를 도시하는 타이밍 차트이다.

[16] 도 5는 레이트 추정기, 특히 버퍼 레벨이 적응적인(*pker*-유형 레이트 추정기)에 의해 관리되는 시간 동안의 레이트들을 도시하는 도표이다..

[17] 도 6은 비 적응성 지수 가중 이동 평균(“EWMA”) 필터가 사용되는 때의 레이트 증가 대 다운로드 시간(r-time)을 도시하는 도표이다.

[18] 도 7은 비-적응성 EWMA 필터가 사용되는 때의 레이트 증가 대 플레이백 시간(p-time)을 도시하는 도표

이다.

[19] 도 8은 가변 윈도우 크기 가중 이동(“WMA”) 필터가 사용되는 때의 레이트 증가 대 다운로드 시간(r-time)을 도시하는 도표이다.

[20] 도 9는 *pker*-유형 프로세스가 사용되는 때의 레이트 증가 대 플레이백 시간(p-time)을 도시하는 도표이다.

[21] 도 10은 섹션 2.1의 *pker* 프로세스가 사용되는 때의 레이트 감소 대 다운로드 시간을 도시하는 도표이다.

[22] 도 11은 레이트의 갑작스런 증가들에 대한 *pker* 프로세스의 동작을 도시한다.

[23] 도 12은 레이트의 갑작스런 하락들에 대한 *pker* 프로세스의 동작을 도시한다.

[24] 도 13은 간단한 (고정된-너비) 이동 윈도우 평균과 지수 가중된 이동 평균의 비교를 도시한다.

[25] 도 14는 *pker* 레이트 추정 프로세스의 플로우차트이다.

[26] 도 15는, 도 16과 함께, *pker* 프로세스에 의해 사용되는 B 및 T_{fast} 값이 어떻게 기록된 (Tp, Tr) 값들로부터 결정될 수 있는지를 도시한다.

[27] 도 16은 값들을 결정하는 것의 측면들을 도시한다.

[28] 도 17은 “워터마크” 페칭 프로세스의 동작을 도시한다.

[29] 도 18은 플레이백 레이트를 선택하는데 사용될 수 있는 lambda 및 mu 함수들의 예들을 도시한다.

[30] 도 19는 “보수적인” 설정을 사용하는 (lambda, mu)-함수들의 예시적 선택을 나타낸다.

[31] 도 20는 “중도적인” 설정을 사용하는 (lambda, mu)-함수들의 예시적 선택을 나타낸다.

[32] 도 21는 “공격적인” 설정을 사용하는 (lambda, mu)-함수들의 예시적 선택을 나타낸다.

[33] 도 22는 어느 정도까지, MLB 프로세스를 에뮬레이팅하기 위한 프로세스를 사용하는 (lambda, mu)-함수들의 예시적 선택을 나타낸다.

[34] 도 23은 lambda 설정들에 대한 나란한 값들의 예를 도시한다.

[35] 도 24은 mu 설정들에 대한 나란한 값들의 예를 도시한다.

[36] 도 25는 레이트 추정, 다음에 레이트-기반 레이트 선택, 다음에 버퍼 관리-기반 레이트 선택을 위한 프로세스를 도시한다.

[37] 도26은 리퀘스트 취소 없는 레이트 하락을 도시한다.

[38] 도 27은 리퀘스트 취소가 있는 레이트 하락을 도시한다.

[39] 도 28은 예시적 리퀘스트 취소 프로세스를 도시하는 플로우차트이다.

[40] 도 29는 리퀘스트 취소 검출을 위한 프로세스를 도시한다.

[41] 도 30은 복수의 TCP 접속들로 페칭하지만 수신 버퍼 튜닝이 없는 동작들의 도표이다.

[42] 도 31은 복수의 TCP 접속들로 페칭하고 수신 버퍼 튜닝이 있는 다른 동작들의 도표이다.

[43] 도 32는 예시적 리퀘스트 액셀러레이터 프로세스의 플로우차트이다.

[44] 도 33은 소정의 프래그먼트 리퀘스트에 대해 만들 다수의 서브리퀘스트들을 찾기 위한 프로세스를 도시한다.

[45] 도 34는 계산된 크기들을 갖는 소스 리퀘스트들의 해체 구간들로 선택되는 개개의 리퀘스트들을 선택하기 위한 프로세스를 도시한다.

[46] 도 35는 시간 오프셋들 및 시간 오프셋들에 의해 결정된 수선 세그먼트에 대한 프래그먼트 구조의 예를 나타낸다.

[47] 도 36은 레이트 선택에서 lambda 및 mu에 대해 사용될 수 있는 값들의 테이블들을 포함한다.

발명을 실시하기 위한 구체적인 내용

[0014] [48] 도 2에 도시된 바와 같이, 본원에 설명된 DASH 클라이언트는 스트림 매니저(SM), 리퀘스트 액셀러레이터(RA), 소스 컴포넌트(SC), 네트워크 접속, 및 미디어 플레이어를 포함한다. DASH 클라이언트는 또한 하나 이상의 미디어 데이터 버퍼들을 포함할 수도 있다. 몇몇 실시예들에서, RA, SC 및 미디어 플레이어는 모두 그들 자신의 데이터 버퍼들, 또는 하나의 큰 데이터 버퍼의 논리적 파티션들을 가질 수도 있다. 다른 실시예들에서, RA가 응답할 수 있는 모든 리퀘스트에 대해 완전한 응답을 되돌려줄 수 있도록 RA 만이 리퀘스트들을 버퍼링하기 위한 데이터 버퍼를 가지고 미디어 플레이어는 SC가 설정한 모든 데이터 버퍼를 사용할 수도 있다. SM은 그 결정들을 하는데 필요한 메타데이터를 저장하기 위한 그 자신의 (물리적 또는 논리적) 국부 스토리지를 가질 수 있다.

[0015] [49] 도 1은 DASH 클라이언트를 갖는, DASH 배치를 도시한다.

[0016] [50] 도 2는 다른 컴포넌트들을 갖는 DASH 클라이언트의 예시적 아키텍처를 도시한다. SM, RA, SC 및 미디어 플레이어는 하드웨어, 소프트웨어 또는 어떤 조합으로 구현될 수도 있음이 이해되어야 한다. 따라서, 어떤 기능이 어떤 컴포넌트에 할당된 경우에, 그것은 프로세서 명령들, 프로그램 코드, 또는 기타 등등으로 구현될 수도 있고, 그러한 경우 (프로그램 메모리, ROM, RAM, 프로세서, 전원, 커넥터들, 회로 보드들, 등등) 그러한 명령들을 실행하는데 필요한 하드웨어가 내포되어 있다. 네트워크 기능들이 기술된 경우, 네트워크 접속이 존재하는 것으로 이해되어야 하고 유선, 광학, 무선, 기타 등등일 수 있고, 사용자 상호작용이 내포된 경우, (디스플레이, 키보드, 터치패드, 스피커들, 마이크로폰들, 등등) 사용자 인터페이스 기능들 또한 내포된다.

[0017] [51] DASH 클라이언트는 두 클록들, 또는 그들의 논리적 등가물을 유지한다. 한 클록은 클라이언트에서 구동하는 로컬 클록의 시간을 표시하는 소프트회로 또는 실시간 클록 회로이고, 다른 클록은 미디어 콘텐츠의 프리젠테이션 시간을 그 시작과 관련하여 나타내는, 프리젠테이션 시간이다. 여기서, 실시간 클록 시간은 “r-time”으로 지칭되고, “p-time”은 프리젠테이션 시간을 나타내는 기술어이다.

[0018] [52] 리프리젠테이션들은 동일한 콘텐츠에 대해 다른 비트-레이트들 또는 다른 차이점들에서 인코딩된 미디어 스트림들이다. 따라서, 사용자는 일반적으로 오로지 하나의 리프리젠테이션을 필요로 할 것이나, 클라이언트는 상황들 및/또는 조건들 변화에 따라 한 리프리젠테이션에서 다른 것으로 스위칭 할 수도 있다. 예를 들어, 대역폭이 높은 경우, 스트리밍 클라이언트는 높은 품질, 높은 비트레이트의 리프리젠테이션을 선택할 수 있다. 대역폭이 감소되면, 클라이언트는 낮은 품질, 낮은 비트레이트의 리프리젠테이션으로 스위칭 함으로써 이들 상황들에 적응할 수 있다.

[0019] [53] 스위치 포인트들(또는 랜덤 액세스 포인트들)은 스트림에 선행하는 데이터의 정보를 요구함 없이, 그로부터 미디어 샘플들의 디코딩이 시작될 수 있는 리프리젠테이션의 샘플들이다. 보다 구체적으로 비디오 리프리젠테이션들의 경우, 샘플들(프레임들)이 일반적으로 앞선 프레임들에 의존하기 때문에, 모든 샘플이 랜덤 액세스 포인트인 것은 아니다. 스트리밍 클라이언트가 리프리젠테이션들을 스위칭 하고자 한다면, 노력 낭비를 피하기 위해 꼭 스위치 포인트에서 새로운 리프리젠테이션을 디코딩하기 시작해야 한다. 몇몇 경우에는, 스위치 포인트들은 스트리밍 클라이언트로 세그먼트 인덱스(sidx)에 시그널링된다.

[0020] [54] 리프리젠테이션 그룹(종종 단순히 그룹으로 축약된다)은 스위칭 가능한 리프리젠테이션들의 세트이다. 미디어 프리젠테이션은 하나보다 많은 리프리젠테이션 그룹들을 포함할 수 있다. 예를 들어, 다른 비트레이트들의 비디오 리프리젠테이션들에 대한 하나의 리프리젠테이션 그룹과, 오디오 비트레이트들을 위한 다른 리프리젠테이션 그룹을 가질 수 있다. DASH 표준에서, 리프리젠테이션 그룹은 종종 어댑테이션 세트(adaptation set)로도 불린다.

[0021] [55] 세그먼트는 하나의 리프리젠테이션의 적어도 일부분에 대한 미디어 데이터를 포함하는 파일이다. 프래그먼트는 프래그먼트의 시작 p-time으로부터 세그먼트 내의 프래그먼트의 바이트 범위까지의 매핑이 이용 가능한 세그먼트의 일부이다. 때때로, 서브세그먼트라는 용어가 프래그먼트 대신에 사용되고, 이들은 동등한 것으로 여겨질 수 있다. 몇몇 미디어 콘텐트는 프래그먼트들로 분할되지 않고, 이러한 경우, “프래그먼트들”은 세그먼트들 그 자신으로 지칭될 수 있다.

[0022] [56] 도 3은 가능한 두 리프리젠테이션 스위칭 프로세스들을 도시하는 타이밍 차트이다. 스위치는 역방향일 수 있고(backward looking, 제 1 프로세스; 도 3A), 이 경우 스위치-투 리프리젠테이션(switch-to

representation)에서 스위치 포인트는 스위치-프롬 리프리젠테이션(switch-from representation)에서 이미 리퀘스트 된 p-time 스트래치에서 루킹하고 이 스트래치의 끝에 가장 가까운 스위치-투 리프리젠테이션으로부터 p-time 역방향의 이전 스위치 포인트를 선택함으로써 발견된다. 제 2 프로세스(도 3B)는 순방향이고: d는 스위치-프롬 리프리젠테이션에서 마지막 리퀘스트된 p-time으로부터 시작하는 스위치-투 리프리젠테이션에서 p-time 순방향의 다음 스위치 포인트를 찾는다.

[0023] [57] 도 4는 스위치 포인트가 얼라인되는 때 그리고 스위치 포인트가 마지막 리퀘스트된 프래그먼트를 바로 팔로우하는 때의 스위칭 프로세스들을 도시하는 타이밍 차트이다. 두 프로세스는 그러한 설정에서 동일하게 동작하므로, 다이어그램은 역방향 및 순방향 방법들 모두의 동작들을 도시한다. 따라서, 스위치 포인트들이 얼라인되는 때, 어떤 프로세스도 오버래핑 데이터를 다운로드 할 수 없다.

[0024] [58] 프리젠테이션 타임은 미디어가 일반적으로 보통의 속도로, 플레이 아웃 또는 플레이 백 될 것으로 예상되는 시간 기간이다. 예를 들어, 30분 비디오 프리젠테이션은 30분동안 플레이 할 것이다. 사용자는 빨리 감기 또는 되감기를 할 수도 있고, 이는 걸리는 실제 시간을 변경시킬 것이나, 프리젠테이션은 여전치 30분 비디오 프리젠테이션임이 이해되어야 한다. 프리젠테이션 엘리먼트는 프리젠테이션 시간에 걸쳐 사용자에게 프리젠테이션을 제공한다. 프리젠테이션 엘리먼트들의 예들은 시각 디스플레이 및 청각 디스플레이, 또는 그것을 프리젠팅할 수 있는 디바이스로 보내지는 비디오/오디오 스트림을 포함한다. “플레이백”은 미디어의 소비를 기술하는데 사용되는 용어이다. 예를 들어, 스마트폰은 프리젠테이션의 프리젠테이션 타임(p-time) 동안 프리젠테이션을 리프리젠팅하는 미디어를 다운로드 또는 획득하고, 그것을 버퍼링하며, 미디어 플레이어는 그 미디어를 "소비한다", 바람직하게는 리시버가 데이터를 더 획득하기 위해 기다리는 동안 프리젠테이션에서 스톨을 경험하지 않도록 적어도 프리젠테이션 타임의 끝까지 버퍼가 완전히 비지 않게 소비한다고 말해진다. 물론, “플레이백” 또는 “플레이 아웃”은 미디어가 한번보다 많이 플레이 된다는 것을 내포하는 것은 아니다. 많은 경우에서, 미디어가 한번 소비되면, 절대 다시 사용되지 않는다.

[0025] [59] 프리젠테이션 버퍼는 리시버, 미디어 플레이어의 또는 어느 하나 또는 둘 모두에 액세스할 수 있는 메모리 엘리먼트이다. 설명의 편의를 위해, 우리는 “프리젠테이션 버퍼”, “버퍼”, “미디어 버퍼” 및 “플레이백 버퍼” 용어들을 상호 교환 가능하게 사용하며, 이는 다운로드 되었지만 아직 플레이 아웃 되거나 소비되지 않은 데이터, 일반적으로 미디어 데이터를 포함하는 논리적 버퍼임이 분명히 이해되어야 한다. 프리젠테이션 버퍼를 포함하는 데이터는 디바이스 내에서 상이한 컴포넌트들 사이에 분할될 수 있으며, 즉 다운로드 된 데이터의 어떤 부분들은 제 1 프로세스, 예를 들어 디바이스 내의 수신 프로세스에 의해 홀딩되고, 반면에 다른 부분들은 다른 프로세스, 예를 들어 디바이스 내의 플레이 아웃 프로세스로 이미 전달되었을 수 있다. 또한 프리젠테이션 버퍼를 포함하는 데이터의 적어도 일부는 다른 버퍼 또는 다른 프로세스들에 걸쳐 적어도 부분적으로 복제될 수도 있다. 어떤 경우들에서는 다운로드 되었지만 아직 플레이 아웃 되지 않은 데이터 모두가 아직 프리젠테이션 버퍼 내에 있는 것으로 여겨지지는 않으며, 예를 들어, 어떤 경우들에서 미디어 콘텐츠가 일단 미디어 플레이로 넘겨지면 이는 더 이상 프리젠테이션 버퍼에 있지 않는 것으로 여겨질 수도 있다. 일반적으로, 다운로드 되었지만 아직 플레이 아웃 되지 않고 아직 프리젠테이션 버퍼 내에 있는 것으로 여겨지지 않는, 미디어 데이터의 양은, 있더라도, 아주 적다.

[0026] [60] A presentation buffer accommodates unevenness is receiving and playing back media, storing received media data until it is consumed. 미디어 데이터가 소비된 뒤에, 그것은 구성에 따라, 삭제될 수 있거나 계속 저장될 것이다. 어떤 구현에서는, 프리젠테이션 버퍼의 크기(프리젠테이션 버퍼에 저장될 수 있는 데이터의 바이트 수에 의해 측정될 수도 있는)는 시간에 따라 변할 수도 있다. 예를 들어, 프리젠테이션 버퍼는 공유된 메모리로부터 필요한 만큼 동적으로 할당될 수도 있다.

[0027] [61] 본원에 상세히 설명된 많은 예들에서, 프리젠테이션 버퍼는 크기에 의해 특징지어지는 것으로 간주될 수 있다. 프리젠테이션 버퍼 전용의 고정 메모리의 경우, 그 크기는 가용 메모리에 저장될 수 있는 바이트 수에 의해 측정될 수도 있다. 프리젠테이션 버퍼가 동적으로 할당되는 경우에, 프리젠테이션 버퍼의 것으로 보는 “크기”는 프리젠테이션 버퍼에 현재 할당된 바이트 수, 프리젠테이션 버퍼에 할당될 수 있는 최대 바이트 수, 또는 어떤 다른 적절한 측정값일 수 있다. 프리젠테이션 버퍼 크기는 또한 종종 프리젠테이션 버퍼에서 현재 이용 가능한 미디어의 프리젠테이션 타임 플레이 아웃 지속 기간으로 측정될 수도 있다.

[0028] [62] 프리젠테이션 버퍼는 또한 다른 특징, 그 “레벨” 또는 “필 레벨”을 가진다. 프리젠테이션 버퍼의 레벨은, 예를 들어 바이트 또는 프리젠테이션 타임 지속기간으로 측정되는, 얼마나 많은 소비되지 않은 데이터가 프리젠테이션 버퍼에 존재하는지를 나타낸다. 레벨은 미디어 데이터가 수신되면서 높아지고 그것이 소비되

면서 낮아질 것으로 예상된다. 레벨은 오로지 논리적이다 - 예를 들어, 프리젠테이션 버퍼는 계속 미디어 데이터로 가득 차 있지만, 예를 들어, 이미 소비된 미디어 데이터와 같이 어떤 미디어는 새로운 미디어 데이터가 수신되면서 오버라이팅을 위해 마킹된다. 어떤 수신기들은 "빈 버퍼"는 0인 소비되지 않은 미디어 데이터가 있을 때의 상태이고, "꽉 찬 버퍼"는 프리젠테이션 버퍼의 100%가 소비되지 않은 미디어 데이터로 차 있을 때의 상태 이도록 프로그램 될 수도 있다. 다른 수신기들은 프리젠테이션 버퍼 크기의 0%에서 100%보다 작은 범위에 걸쳐 변동하도록 다른 경계들을 가질 수도 있다. 공유 메모리가 사용되고 소비되지 않은 미디어 데이터가 그곳에 저장될 때 유일하게 할당되는 프리젠테이션 버퍼인 경우, 프리젠테이션 버퍼는, 정의상, 항상 가득 차 있으므로, 레벨 비율을 나타낼 때 동적으로 할당된 프리젠테이션 버퍼의 메모리의 크기를 분모로 사용하는 것은 의미가 없을 수 있다. 대신에, 프리젠테이션 버퍼의 레벨은 프리젠테이션 버퍼에 대해 허용되는 최대 크기에 의해 나누어진 프리젠테이션 버퍼의 소모되지 않은 미디어 데이터의 양의 비율로 측정될 수도 있다.

[0029] 1. 클라이언트 컴포넌트들의 개관

[0030] [63] 다시 도 1-2를 참조하면, 예시적인 클라이언트의 다양한 컴포넌트들이 도시된다.

[0031] [64] SC는 어떤 리프리젠테이션들이 이용가능한지, 및 그들의 프래그먼트들이 무엇인지에 관한 정보와 같은, 메타데이터를 계속 파악한다. SC는 또한 네트워크를 통해 수신된 미디어 데이터를 버퍼링하고 미디어 플레이어에 그것을 넘겨줄 책임이 있다. SM은 어떤 리프리젠테이션들이 시간상 어떤 포인트에서 다운로드 되어야 하고 레이트 스위치 결정을 할 책임이 있다. 마지막으로, RA는 SC에 의해 제공된 바와 같이 미디어 프래그먼트들, 주어진 정확한 URL 및 바이트-범위 정보를 다운로드 할 책임이 있다.

[0032] [65] SM은 레이트 스위칭 결정을 책임지는 소프트웨어 컴포넌트이다. SM의 목표들 중 하나는 주어진 상황에서 최적의 콘텐츠를 고르는 것이다. 예를 들어, 많은 대역폭이 가능하다면, 높은 다운로드 레이트가 획득될 수 있고, 따라서 SM은 높은 레이트 리프리젠테이션을 골라야 한다. 다운로드 레이트가 심각하게 떨어지면, 선택된 높은 리프리젠테이션은 더 이상 유지될 수 없을 수 있고, 따라서 SM은 상황에 보다 적절한, 낮은 리프리젠테이션 레이트로 스위칭 해야 한다. SM은 플레이백 버퍼를 완전히 비워지는 것을 피하고(이는 플레이백 스톨을 유발할 수 있으므로), 그러나 동시에 너무 서두르거나 너무 자주 스위치 하지 않을 만큼 충분히 빠르게 스위치 해야 한다. 더욱이, 네트워크를 통해 다운로드 되고 스톨링 없이 플레이백 될 수 있는 가장 높은 품질의 콘텐츠를 리퀘스트하는 것을 목표로 한다. SM은 결정 프로세스에서 다운로드 속도 외 다른 인자들을 고려하는 것까지 확장될 수 있다. 그것은 배터리 수명, 디스플레이 크기, 및 다른 인자들과 같은 것들을 리프리젠테이션 결정 때에 잠재적으로 고려할 수 있다. 그러한 추가 제약들은 SM에 필터로서 추가될 수 있고, 본원의 기본적인 레이트 결정 계산에 영향을 미치지 않는다.

[0033] [66] 대표적이고, 고-레벨의, 클라이언트의 동작이 이하에 기술될 것이다. 사용자는 비디오 및 오디오와 다른 타입의 미디어를 수반할 수 있는, 라이브 스포츠 브로드캐스터, 사전-녹화된 영화, 오디오 스트림, 또는 다른 오디오-비주얼 또는 다른 콘텐츠와 같은 특정 미디어 콘텐츠를 리퀘스트한다고 가정한다. 클라이언트는 그 리퀘스트를, 아마 사용자 인터페이스 또는 컴퓨터 인터페이스를 통해, SM으로 공급할 것이다. SM은 SC에 리퀘스트하고 어떤 리프리젠테이션들이 이용가능한지, 어떤 p-time span이 어떤 프래그먼트들에 의해 커버되는지, 리프리젠테이션들의 스위치 포인트들이 어디에 위치하는지에 대한 표시들을 수신할 것이다. 그에 추가로, SM은 단기 다운로드 레이트에 관한 그 마음대로 이용할 수 있는 어떤 정보를 가질 수 있다 - 이하에 설명된 바와 같이, RA는 이 데이터를 SC에 보고하고 SC는 이를 SM에 보고하거나 제공한다.

[0034] [67] SM은 그 정보를 지난 히스토리와 함께 사용하여 지속 가능한 레이트를 추정하고 리프리젠테이션 내의 적절한 스위치 포인트 및 그 스위치 포인트로부터 시작하는 그 리프리젠테이션으로부터 다운로드할 미디어 콘텐츠의 양을 결정한다. 다운로드들이 진행되고 미디어 콘텐츠가 플레이백 됨에 따라, SM은 레이트 스위치가 적절한지 아닌지를 결정하기 위해 공급된 정보를 사용한다. 레이트 스위치가 적절하지 않으면, SM은 SC에 현재 리프리젠테이션으로부터 프래그먼트들을 계속하여 페치하라고 알린다. 레이트 스위치가 절절하면, SM은 잠재적인 스위치 포인트들을 찾고 원하는 스위치를 하기 위해 어떤 리프리젠테이션들로부터 어떤 프래그먼트들이 페치될 필요가 있는지를 결정한다. SM은 그 다음에 그 정보를 SC로 전달한다. 다운로드 될 비디오의 다음 섹션에 대한 결정이 되어야 할 때는 언제든지, SC와 SM 사이의 이 교환은 주기적으로 수행된다. 좋은 결정을 하기 위해, SM은 버퍼 레벨을 모니터하고, SM은 버퍼가 충분히 찼는지, 그리고 얼마간의 시간 주기 동안 프래그먼트들이 다운로드 될 필요가 없는지를 결정할 수 있다.

[0035] [68] SM이 다운로드할 프래그먼트를 결정했으면, SC는 RA가 프래그먼트를 실제로 다운로드하고, 미디어 버퍼에 다운로드 된 프래그먼트를 유지하며, 마지막으로 그것을 플레이 아웃할 시간이 왔을 때 미디어 버퍼의 미디어 데이터를 미디어 플레이어로 전달하도록 할 책임을 진다.

[0036] [69] SM은 그것이 SC에 다운로드하라고 한 그 프래그먼트들에 더 이상 능동적으로 관련되지 않는다. 그러나, SM은, 소정의 프래그먼트의 다운로드가 이미 시작된 이후라 할지라도, 그 마음을 바꾸고 그것이 이전에 발한 프래그먼트 리퀘스트를 취소할 수 있다. 이러한 기능은 다운로드 레이트가 극적으로 떨어지고 다운로드되고 있는 프래그먼트가 미디어 버퍼가 완전히 비워질 때까지 이용 가능하지 않을 것 같다고 드러나는 경우에 유용하다. 그러한 상황이 발생하면, SM은 그것을 감지하고 리퀘스트를 취소하며 대신에 보다 적절한 레이트로 스위칭하여야 한다.

[0037] [70] 일단 SC가 SM으로부터 페치할 프래그먼트 핸들을 수신하면, 그것은 그 데이터 구조에서 대응하는 프래그먼트의 URL 및 바이트 레인지를 찾고, 그것을 사용하여 그것이 RA로 넘길 리퀘스트를 생성한다. 그것은 또한 RA로부터 응답 데이터를 리트리빙하고, 수신된 미디어 프래그먼트들을 플레이 가능한 스트림으로 변형할 책임이 있다. 마지막으로, SC는 MPD로부터 획득된 데이터, 세그먼트 인덱스(sidx) 박스들, 또는 Apple의 HTTP Live Streaming(HLS)의 경우에, 플레이 리스트들과 같은, 메타데이터를 파싱하고 계속 파악할 책임을 진다.

[0038] [71] RA는 SC로부터 수신된 프래그먼트 및 메타데이터 리퀘스트들을 취하고, 대응하는 HTTP 리퀘스트들을 생성하고, 이들을 네트워크 접속을 통해 전송하고, 대응하는 응답들을 리트리빙하여 이들을 SC로 다시 전달하는 컴포넌트이다. 네트워크 접속은 인터넷 접속, 셀룰러-기반 접속, WiFi 접속 또는 HTTP 리퀘스트들 및 응답들을 다룰 수 있는 다른 네트워크 접속들일 수 있다. 네트워크 접속은 단일 디바이스 내에 내제할 수 있으며, 즉 그것은 디바이스 내에 이미 캐싱된 미디어 데이터로의 내부 인터페이스일 수 있다. 많은 조합들이 있을 수 있으며, 즉, 어떤 미디어 콘텐츠는 유선 인터넷 접속으로부터 다운로드 될 수 있고, 어떤 것들은 셀룰러 기반 접속을 통해, 어떤 것들은 WiFi 접속을 통해, 어떤 것들은 로컬 캐시로부터 다운로드 될 수 있다. 어떤 경우에는 미디어 데이터가 다운로드 되는 접속은 혼합되어 있을 수 있으며, 즉 일부는 셀룰러를 통해, 일부는 WiFi를 통해, 일부는 유선 등등을 통할 수 있다. 특정 리퀘스트들은 어떤 경우에는 HTTP와 다를 수 있으나, 미디어 콘텐츠를 서빙하는 서버들이 HTTP 서버들일 경우에 HTTP가 바람직하다.

[0039] [72] 가장 간단한 형태에서, RA는 HTTP 클라이언트이다. 그러나 RA가 일반적인 HTTP 클라이언트보다 더 능률적인 것이 바람직할 수도 있다. RA의 한 목표는 충분히 높은 다운로드 속도를 달성하는 것이고; 그것은 선택된 플레이백 미디어 레이트보다 상당히 빠르게 다운로드하는 것을 목표로 해야 한다. 반면에, 로우 쓰로우풋(raw throughput)에 대한 적시성을 해치지 않도록 하는 것 또한 주의해야 한다: 곧 플레이 아웃될 프래그먼트들은 더 뒤의 것들보다 더 급하고, RA는 그것들을 적시에 수신하도록 시도해야 한다. 따라서, 적시성을 위해 다소간의 쓰로우풋을 희생해야 할 필요가 있을 수 있다. RA는 모든 합당한 네트워크 상황에서 양호하게 동작하도록 설계되어야 한다.

[0040] [73] RA의 기본적 설계는 최적의 결과를 얻기 위해 몇 개의 접속들과 아마도 FEC(forward error correction) 또한 사용하는 것이다. 따라서, RA는 일반적으로 하나보다 많은 개방 HTTP 접속을 다룰 필요가 있을 것이다. RA는 그 접속들 상으로 리퀘스트를 발송할 것이다. RA는, 어떤 상황들에서는, 보다 작은 리퀘스트들의 세트로 리퀘스트들을 분할할 수 있다. 대응하는 응답들을 수신하면, RA는 데이터를 코헤런트 응답으로 다시 모은다. 다시 말하면, RA는 보낼 HTTP 리퀘스트들의 그래뉼러리티(granularity)를 결정하고, 어떤 접속들로 리퀘스트들을 디스패칭할지, 및 어느 부분의 소스 프래그먼트들 또는 수선 세그먼트들을 리퀘스트할지를 결정할 책임을 진다. 그 리퀘스트들의 그래뉼러리티는 버퍼 레벨, 리퀘스트의 긴급성, 이용 가능한 접속들의 수 등과 같은, 많은 것들에 의존할 수 있다.

[0041] [74] RA에 의해 발신된 각 리퀘스트는 메타데이터에 대한, 또는 SC에 의해 RA로 전달된 프래그먼트 리퀘스트의 부분 또는 전체에 대한 HTTP 리퀘스트이다. 그것은 소스 미디어 데이터 또는 소스 미디어 데이터로부터 생성된 수리(repair) 데이터일 수 있다. SC 프래그먼트 리퀘스트로부터 생성된 RA 리퀘스트들에 대한 응답들은, 대부분의 경우에, RA가 프래그먼트 리퀘스트의 모든 미디어 데이터를 재건하기에 충분해야 하고, RA는 다음에 이를 SC에 다시 전달할 수 있다. 따라서, RA는 미디어 프래그먼트 리퀘스트와 연관된 RA 리퀘스트들로부터의 응답들을 다시 SC에 제공된 프래그먼트 리퀘스트에 대한 응답으로 어셈블링할 책임을 진다. 예를 들어 FEC 수선 데이터에 대한 몇몇의 RA 리퀘스트들이 있다면, RA에 의한 어셈블링은 FEC 디코딩을 포함할 수 있다.

[0042] [75] HTTP 리퀘스트들을 다루는 것에 추가하여, RA는 어떤 샘플링 레이트의 타임 슬라이스들 동안, 단기 기간들 동안, 다운로드 속도를 측정한다. 예시적 샘플링 레이트는 100 ms이고, 즉, RA는 100 ms 기간들 동안 다

운로드 속도들을 측정한다. 이 데이터는 SM에 의해 그 다운로드 속도 추정을 계산하고, 결국 레이트 결정들을 하는데 사용된다. 다른 샘플링 레이트들도 물론 가능하다.

[0043] [76] RA는 DASH 미디어 프리젠테이션 디스크립션(MPD)과 같은 메타데이터 또는 세그먼트 구조들에 대해 알아야 할 필요가 없다. 특정 구현에서, RA는 HTTP 스택 구현의 몇몇 동시의 예들을 사용하여 몇 개의 접속들에 걸쳐, 심지어 비슷하거나 상이한 서버들로의 상이한 유형의 접속들에 걸친 몇몇의 경우에도 HTTP 리트리벌을 구현한다.

[0044] [77] RA는 SC가 언제 새로운 리퀘스트가 받아들여질 수 있는지를 알게 할 책임을 진다. SC는 요청할 다음 프래그먼트를 결정하기 위해 SM을 호출하고 RA에 적절한 리퀘스트를 제공한다. RA는 또한 몇몇 상태 정보를 제공한다. RA는 SC를 통해 SM에게, 단기 다운로드 속도 및 다운로드에 소요된 전체 시간을 정규적으로 제공할 수 있다. SM은 또한, SC를 통해 간접적으로, 이 정보를 위해 RA를 폴링할 수 있다. 그 외에, RA는 또한 SM에 각 개별 리퀘스트의 몇 퍼센트가 이미 완결되었는지에 대해 알린다. 이 정보는 SM이 그것을 리트리브하기 위해 호출하는 API에 유사하게 제공된다.

[0045] [78] RA 또는 SC 내에 버퍼링된 데이터가 가능한 한 적고(고의적인 미디어 버퍼는 제외) RA, SC 및 실제 미디어 파이프라인 사이에는 아주 타이트한 데이터 흐름이 있어야 한다. 다양한 형태의 HTTP 리퀘스트들에 대해서도 마찬가지이다; 실제 대응하는 HTTP 리퀘스트들이 네트워크를 통해 전송되는 때보다 단지 근소한 양의 시간만큼 이르게 리퀘스트할 프래그먼트에 대해 결정해야 한다. 하나의 이유는 SM이 더 미리 리퀘스트에 대해 결정해야 하면, 그 정보가 덜 정확하고 최근이며, 결과적으로 그 결정이 더 낮은 품질이 될 것이기 때문이다.

[0046] [79] SM은 차례로 이슈될 리퀘스트들을 제출한다. 그러나, SM은 모든 이전 리퀘스트들이 완결되지 않은 경우에도 또한 새로운 리퀘스트들을 이슈할 수 있고; 공존 리퀘스트들이 허용된다. SC는 SM이 그것들을 이슈하기 위해 RA에 리퀘스트들을 전달한다. 그 다음 RA가 공존 프로세싱을 처리하고, 그것이 수신된 데이터를 SC로 다시 전달하는 것을 확인한다.

[0047] [80] 공존 리퀘스트들은 RA가 HTTP 파이프라이닝을 구현할 수 있도록 한다. 실제로, 다중 접속들을 사용하는 RA 조차도 이 스킴(scheme)에 맞다.

[0048] 1.1 스트림 매니저(SM)

[0049] [81] SM은 언제 프래그먼트들을 요청하고, 사용자 액션들, 네트워크 상황 및 다른 인자들의 결합에 응답하여 어떤 프래그먼트들을 리퀘스트할지를 결정한다. 사용자가 콘텐츠를 시청하기 시작할 것을 결정하면, SM은 사용자에 의해 또는 제공되는 서비스에 의해 특정되는 p-time으로부터 시작하는 그 콘텐츠에 대해 리퀘스트하는 제 1 프래그먼트를 결정할 책임을 진다. 예를 들어, 어떤 라이브 스트리밍은 모든 사용자가 같은 r-time에 미디어 콘텐츠의 같은 p-time 부분을 보고 있을 것을 요구할 수 있고, 반면에 다른 라이브 스트리밍 및 온-디맨드 서비스들은 어떤 r-time에 어떤 p-time을 플래이백하는지에 대해 최종 사용자 또는 어플리케이션에 유연성을 허용할 수 있다. 미디어 버퍼가 차면, SM은 추가 프래그먼트 리퀘스트들을 제공하는 것을 일시적으로 정지시킨다. SM은 네트워크 상황 및 디스플레이 크기, 잔여 배터리 수명 등과 같은, 다른 인자들에 따라, p-time의 각 포인트에서 콘텐츠를 어떤 품질로 플레이백할지를 결정할 책임이 있다.

[0050] [82] SM이 프래그먼트 리퀘스트를 제공하는 것이 적절하다고 판단하면, SM은 RA가 프래그먼트 리퀘스트들을 수신하고 처리할 준비가 된 경우에만 오로지 리퀘스트를 제공할 수 있다. SC는 RA를 폴링함으로써 이것이 사실인 때를 결정하고, 이 정보를 SM에 포워딩한다.

[0051] [83] RA가 다음 리퀘스트를 수신할 준비가 되어 있으면, SM은 새로운 리퀘스트가 이슈되어야 하는지를 결정하고 리퀘스트할 다음 프래그먼트를 선택한다. SM은 미디어 데이터에 대해 한번에 프래그먼트 하나씩 리퀘스트한다. SM은 콘텐츠의 적시의 심리스 플레이백을 허용하도록 프래그먼트들을 리퀘스트할 책임을 진다. 리프리젠테이션들의 플레이백 변경은 일반적으로 스위치 포인트들에서만 발생할 수 있고, 두 연속하는 스위치 포인트들 사이에 복수의 프래그먼트들이 있을 수 있다; SM은 그 제한을 고려한다.

[0052] [84] 일반적으로, SM은 오로지 그것들이 부드러운 플레이백을 위해 적시에 수신될 것으로 믿는 것이 합리적인 프래그먼트들을 리퀘스트할 것을 시도한다. 그러나 네트워크 상태가 종종 아주 빠르게 격렬히 변화할 수 있음을 고려하면, 이는 모든 상황에서 보증될 수 없다. 따라서, SM은 리퀘스트들을 취소할 수 있는 능력 또한 가진다. SM은 정체가 감지되고 아무런 조치가 취해지지 않는다면 심각한 스톨링 위험이 있는 경우 리퀘스트들을

취소할 것이다. 예를 들어 프래그먼트 리퀘스트가 이슈된 직후에 네트워크 상황 악화로 인해 다운로드 레이트가 갑자기 급격하게 떨어지는 경우, 아무런 조치가 취해지지 않으면 스톨링이 가능성이 있다.

[0053] [85] SM은 리프리젠테이션, R, 및 이전에 선택된 가장 최근의 프래그먼트의 p-time 끝, E를 계속 파악한다. SM은 일반적으로 $E'=E$ 의 시작 p-time을 갖는 다음 프래그먼트를 요청할 것을 선택한다. 몇몇 변화들은 시작 시간이 버퍼 레벨 및 현재 플레이백 시간으로부터 결정되도록 할 수도 있다.

[0054] [86] SM은 스위치 포인트들에서 잠재적인 오버랩이 폐기되면 원활하게 플레이백될 수 있는 스트림을 생성하도록 의도되는 리퀘스트들의 시퀀스를 생성할 수 있다. SM이 리퀘스트들을 생성하는 순서는 RA가 그것들을 (꼭 필수적으로 이슈하지 않지만) 우선순위를 매겨야 하는 순서와 같다. 이는 또한 RA가 수신된 데이터를 SC로 다시 전달하고, SC가 그것을 플레이 아웃해야 하는 순서와 같다.

[0055] [87] SM이 레이트를 스위치해야 할 필요가 있다고 결정하면, 일반적인 경우 이를 하기 위한 두 프로세스가 있다. 한 프로세스에서는, SM은 E 이하의 p-time을 갖는 새로운 ("switch-to") 리프리젠테이션에서 스위치 포인트 (또한 종종 "랜덤 액세스 포인트" 또는 "RAP"로 지칭된다) P를 찾고, 일단 그러한 포인트가 식별되면, SM은 새로운 리프리젠테이션의 프래그먼트들을 리퀘스트하기 시작한다. 제 2 프로세스는 E의 그것보다 늦거나 같은 p-time을 갖는 스위치 포인트, P를 찾는 것이고 P를 넘는 end-time을 갖는 프래그먼트가 리퀘스트되기까지 이전 ("switch-from") 리프리젠테이션의 프래그먼트들을 계속 요청한다. 어느 경우에도, SC로 스위칭을 시그널링하는 것이 유용할 수 있다.

[0056] [88] 이들 프로세스들은 모두 얼마간의 오버래핑 데이터가 다운로드되어야 할 수도 있다는 특징을 갖는데 주목하라. 스위치-프롬 리프리젠테이션 및 스위치-투 리프리젠테이션 모두에 대해 데이터가 다운로드되어야 할 필요가 있을 수 있는 p-time의 스트래치가 있다.

[0057] [89] 이들 스위칭 프로세스들 중 어느 것이 적합한지는 상황에 달려있다. 예를 들어, 특정 상황에서는, 어느 하나의 프로세스에 대해 오버랩이 과도하게 크고, 반면에 다른 하나에 대해 그것은 매우 짧다. 모든 프래그먼트들이 리프리젠테이션들에 걸쳐 얼라인되어 있고 모든 프래그먼트들이 RAP로 시작하는 간단한 경우에, 이들 스위칭 프로세스들은 더 간단한 방법으로 축소되고, 여기서 SM은 단지 스위치-프롬 리프리젠테이션 대신 스위치-투 리프리젠테이션으로부터 다음 프래그먼트를 리퀘스트함으로써 스위치한다. 이 경우, 어떤 오버래핑 데이터도 다운로드 될 필요가 없음에 또한 주목하라.

[0058] 1.1.1 SM 프래그먼트 결정 프로세스

[0059] [90] 이 섹션은 SC에게 어느 프래그먼트들을 요청할지를 결정하기 위한 SM 프래그먼트 결정 프로세스를 기술한다. 이 예들에서, 단일 리프리젠테이션 그룹이 가정되나, 예들은 복수의 리프리젠테이션 그룹들을 사용하는, 예를 들어, 비디오 리프리젠테이션 그룹에서 비디오 리프리젠테이션을 그리고 오디오 리프리젠테이션 그룹에서 오디오 리프리젠테이션을 고르는, 프로세스들을 설명하는데 확장될 수 있다.

[0060] [91] SM에 의해 선택되는 다음 프래그먼트는 일반적으로 이전 프래그먼트 리퀘스트의 end p-time인 시작 p-time을 갖는다. 이하에 요청할 다음 프래그먼트를 선택하기 위해 SM에 구현될 수도 있는 몇몇의 상세한 로직이 기술된다.

[0061] [92] 다음의 예들에서, 프래그먼트들은 RAP들로 시작하고 리프리젠테이션들 간에 얼라인되는 것으로 가정한다. 그것이 사실이 아니라면, 이 설명의 변형들이 가능하다. 그 조건이 있으면, SM의 프래그먼트 결정은 레이트 결정으로 축소되고, 즉 SM은 현재 리프리젠테이션에 머무를지 아니면 다른 것으로 스위치할지를 결정한다. 프래그먼트들이 리프리젠테이션들에 걸쳐 반드시 얼라인되지는 않고 RAP들로 시작하지 않을 수도 있는, 보다 일반적인 경우에, 결정은 유사하지만, 스위칭 비용은 더 높으며, 이는 고려될 수도 있다.

[0062] [93] SM 리프리젠테이션 프로세스는 논리적으로 분리된 두 프로세스들을 포함한다. 제 1 프로세스는 레이트 추정기이고, 이는 RA가 제공하는 단기 샘플들로부터 유지되는 대략적인 다운로드 레이트를 계산하고, 제 2 프로세스는 이 추정을 사용하여 스위치 결정들을 하는 결정 프로세스이다.

[0063] 2. 레이트 추정 프로세스

[0064] [94] 적응성 비트레이트 스트리밍 클라이언트는 일반적으로 올바른 비트레이트 미디어를 선택하기 위해 레이

트 결정 모듈에 의해 이후에 사용되는 다운로드 레이트 추정기를 사용한다. 이 접근법에서, 다운로드 레이트가 크면, 고품질의 미디어가 스트리밍 될 수 있다. 다운로드 레이트에서의 변화는 리프리젠테이션 스위치들을 트리거할 수 있다. 레이트 추정의 품질은 스트리밍 클라이언트의 품질에 큰 영향을 준다.

[0065] [95] 적응성 비디오 스트리밍 디바이스를 위한 양질의 레이트 추정기는 많은 특성들을 가져야 한다. 첫 번째로, 그것은 단기 다운로드 레이트가 많이 변하더라도 변화를 거의 갖지 않아야 한다. 두 번째로 그것은 아래의 채널 상의 레이트 변화에 빠르게 적응해야 한다. 채널 레이트가 상당히 떨어지는 경우, 추정은 그 사실을 빠르게 반영하여 디바이스가 스톨링 없이 품질을 그에 알맞게 조절할 수 있도록 해야 한다. 상응하여, 비디오 품질에서의 상승은 빠르게 관찰되어 더 높은 품질의 콘텐츠가 페치될 수 있도록 해야 한다.

[0066] [96] 이들 두 조건들을 만족시키는 것은 트레이드-오프들을 요구할 수 있다. 일반적으로, 작은 변화를 갖는 추정기는 큰 반응 시간을 가질 것이고 그 반대도 그러하다. 예를 들어, 디바이스에 사용될 수 있는 간단한 추정기를 고려하라. 그 추정기는 어떤 고정된 X에 대해, 다운로드의 마지막 X 초 동안 이동 평균을 취할 것이다. 예를 들어, X=30초(s)의, 큰 X를 고르는 것은 변화가 거의 없는 상대적으로 평탄한 추정을 낳지만, 다만 다운로드 레이트 변화들에 느리게 반응할 것이다. 그러한 추정기가 레이트 결정에 사용된 경우, 결과적인 플레이어는 대역폭 하락에 자주 스톨하거나 더 높은 비트레이트로 적시에 스위칭하는 것을 그렇게 하는 것이 안전하게 가능한 때에 실패할 수도 있다. 이들 이유로, 구현은 더 작은 X, 말하자면 X=3s를 고를 수도 있다. 그러한 선택은 안정성을 훼손하지만 보다 빠른 레이트 조정을 낳을 것이다. 레이트 추정은 많이 변할 것이고, 따라서 플레이어는 비디오 플레이백 레이트를 아주 자주 변경하여 나쁜 사용자 체험을 초래할 수도 있다.

[0067] [97] 도 5에서, 울퉁불퉁한 커브는 많은 단기 변동을 갖는, 로우(raw) 다운로드 레이트이다. 레이트 추정기는 울퉁불퉁한 다운로드 레이트의 평편화된 버전이다. 레이트 변화에서, 그것은 지속되는 새로운 레이트로 수렴하고, 레이트가 변하지 않는 한 그에 비슷하게 유지된다.

[0068] [98] 바람직한 특성들 중 하나는 작은 버퍼 레벨이 있는 경우, 조정이 빠르다는 것이고, 이는 레이트의 빠른 적응을 초래하여, 다운로드 레이트가 하락하고 있을 때 조정 전에 프리젠테이션 버퍼가 비지 않도록 한다. 한편으로, 미디어 버퍼 내에 많은 미디어 데이터가 있는 경우, 레이트 추정은 보다 느린 조정으로 보다 평탄해야 한다. 미디어 버퍼에 보다 많은 미디어 데이터가 있는 경우, 플레이 아웃 레이트는 미디어 버퍼에 보다 적은 미디어 데이터가 있을 때보다 다운로드 레이트가 하락하는 보다 긴 시간 동안 더 높게 유지되는 경향이 있어야 한다.

[0069] [99] *pker*, *pker* 프로세스, 또는 *pker*-타입 프로세스로 지칭하는, 다음에 소개되는 레이트 추정 프로세스는 레이트 변화들에 빠르게 반응할 뿐만 아니라, 안정적이어서 낮은 변동성 및 높은 반응성에 대한 조건들 모두를 만족시킨다.

[0070] 2.1. *pker* 프로세스

[0071] [100] 이 섹션은 본원에서 *pker*, *pker*-타입 프로세스, 또는 단순히 "*pker* 프로세스"로 지칭되는 레이트 추정 프로세스를 기술한다. 기본적 레이트 추정기는 그 추정들을 오로지 단기 레이트 측정들에 기초하고, 한 방법 또는 다른 것을 사용하여 그로부터 더 긴 이동 평균(running average)을 계산한다. 전술한 기본적 이동 윈도우 평균(moving window average; "MWA")은 그러한 프로세스의 일 예이다.

[0072] [101] 도 6-7은 레이트 선택용으로 비-적응성(고정 계수) 지수 가중 평균을 사용한 것의 효과들을 도시한다. 이들 플롯들은, 간략화를 위해, 새 레이트 추정은 새 다운로드 선택을 즉시 트리거하고(즉, 프래그먼트들은 상대적으로 작다), 새 레이트 선택은 단지 레이트 추정이다.

[0073] [102] 도 6은 r-time 측면을 도시한다. 여기 도시된 바와 같이, x-축은 다운로드 시간(실 시간)이다. 극적인 레이트 증가가 시간 T1에서 발생하면, 비디오 데이터가 플레이 아웃 되고 있는 것보다 훨씬 빨리 다운로드 되고 있기 때문에, 버퍼가 매우 빨리 커지기 시작한다. EWMA 추정은 점차 실 레이트로 수렴한다.

[0074] [103] 도 7은 같은 이벤트의 p-time 측면을 도시한다. 도면에서, 선(702)은 스크린에 디스플레이되는 비트레이트를 나타낸다. 레이트는 도 6의 r-time 도면보다 훨씬 느리게 조정된다. r-time 에 비해 p-time에 대한 수렴 속도가 시작에서 NR/OR 인자에 의해 느려졌다(왜냐하면 플레이어는 그 포인트에서 다운로드 초 당 약 비디오 NR/OR 초를 수신했기 때문이다). 따라서, 순 효과는 이 타입의 레이트 추정기를 사용할 때 상당한 양의 p-time 동안 다운로드 레이트보다 훨씬 낮은 레이트로 미디어가 플레이 아웃할 수 있다는 것이다.

[0075] [104] 레이트가 미디어 스트리밍의 목적으로 추정되는 경우, 추정기는 다른 적절한 정보를 이용할 수 있다. 보다 구체적으로, 미디어 플레이어의 버퍼가, 또는 일반적으로, 버퍼링 되었거나, 또는 이미 플레이 아웃된, 각 미디어 세그먼트를 다운로드 하는데 얼마나 걸렸는지에 대한 정보를 포함하는, 미디어 플레이어의 다운로드 히스토리(현재 버퍼에 있는 것보다 더 과거로)가 관심의 대상이다.

[0076] [105] 일 구현은 예를 들어 MWA를 사용하나, 미디어 버퍼의 함수로서 윈도우 크기를 선택할 수 있다.

[0077] [106] 미디어 플레이어의 버퍼 레벨이 높으면, 플레이어는 스톨링의 위험에 당면하지 않으며, 큰 윈도우를 사용하여 장기 추정이 행해질 수 있고, 이는 보다 안정적인 추정을 낳을 것이다. 반대로, 버퍼 레벨이 낮으면 플레이어는 빠르게 반응해야 하고, 이는 보다 짧은 평균 윈도우가 이 경우에 더 나은 선택임을 암시한다.

[0078] [107] 따라서 레이트 추정 프로세스의 일 구현은, 현재 미디어 버퍼의 p-time 양(즉, 다운로드되었지만 아직 플레이아웃되지 않은 p-time 의 현재 양)에 비례하는 r-time 윈도우를 사용하여, 변하는 윈도우 너비를 사용할 수도 있다.

[0079] [108] 다른 구현은 미디어 버퍼에 현재 보관된 바이트의 수에 비례하여 윈도우 너비를 선택할 수도 있다.

[0080] [109] 일 구현은 또한 단지 그 레벨이 아니라 버퍼 그 자체의 콘텐츠를 검사할 수 있다. 예를 들어, 그것이 버퍼의 큰 부분이 그 동일한 콘텐츠의 플레이백 지속시간인 것 보다 짧은 시간에 다운로드되었다고 결정하면, 이는 다운로드 버퍼가 빠르게 커지고 있음을 암시하고, 따라서 레이트 추정기는 추정이 조정될 필요가 있다고 결정할 수도 있다.

[0081] [110] 유사하게, 레이트 추정기는 버퍼 레벨의 레이트 변화를 추적하고, 레이트 추정이 빠르게 조정될 필요가 있다는 표시들에 따라 버퍼 레벨을 빠르게 변화시킬 수도 있다.

[0082] [111] 도 8-9는 가변 윈도우 크기 가중 이동 평균 필터가 사용되는 경우의 도 6-7과 동일한 시나리오에서의 동작을 도시한다. 예들에서, "*pker*" 프로세스가 가변 윈도우 크기 WMA 필터와 같은 프로그래밍 코드로 설명된다. *pker* 프로세스는 프로세서에 의해 실행되는 프로그램 명령들로 구현될 수도 있다.

[0083] [112] 도 8에서, 선(802)은 기저 채널이 레이트 OR(구 레이트)에서 레이트 NR(신 레이트)로 급격한 레이트 증가를 가지는 경우의 *pker* 레이트 추정이다. 새로운 레이트로 조정하는 레이트 선택에 걸리는 r-time의 양은 OR/NR에 비례한다. 증가가 클수록, 실시간에서 조정이 더 빠르게 일어난다. 도시한 바와 같이, 시간 T2에서, Buff@T2=2*Buff@T1 및 T_{fast}=OR/NR*Buff@T1 이다.

[0084] [113] 도 9는 p-time에서의 플레이백 동작을 도시한다. *pker* 추정기가 새 레이트로 조정하는데 약 일 버퍼 지속기간(레이트 증가가 일어났을 때 버퍼에 있던 p-time의 양)이 걸리고, 즉 미디어 버퍼가 미디어 버퍼에 추가된 p-time 지속기간 B를 갖는 미디어 콘텐츠의 양을 갖는 시간까지 *pker* 추정기가 새 레이트로 조정했고, 여기서 B는 새 레이트로의 레이트 증가 시에 미디어 버퍼의 미디 콘텐츠의 p-time 지속기간이다.

[0085] [114] 이를 하는 구체적인 프로세스가 이제 기술될 것이다. 프로세스는 플레이백 버퍼의 마지막 γ_T-부분을 다운로드 하는데 얼마나 많은 r-time이 걸렸는지를 결정하고, 여기서 γ_T는 절절히 선택된 상수이다. 예를 들어, 이는 전체 현재 플레이백 버퍼(γ_T=1)를 다운로드 하는데 걸린 완전한 시간이거나, 플레이백 버퍼의 마지막 절반(γ_T=0.5)를 다운로드 하는데 걸린 시간일 수도 있다. γ_T>1도 또한 가능하다. T_{fast}를 플레이백의 버퍼의 마지막 γ_T-부분을 다운로드 하는데 걸린 r-time의 양이라고 두자. 추정된 다운로드레이트는 다운로드 시간의 이전 T_{fast} 초 동안 다운로드 레이트를 추정함으로써 계산될 수 있다. 다른 값의 γ_T가 가능함에 주의하라. 여기에 설명된 바와 같이 다른 값들은 다른 목표에 공헌한다.

[0086] [115] T_{fast} 너비 윈도우에 걸친 이 종류의 윈도윙된 평균은 레이트 증가를 빠르게 검출하는 주목할 만한 특성을 갖는다. 사실, γ_T<1인 값이 T_{fast}를 결정하는데 사용되면, 추정기는 미디어 버퍼의 미디어 콘텐츠의 p-time 지속기간이 B일 때 시간의 어느 순간에 임의의 인자만큼 레이트가 증가하면, 레이트 추정기가 증가된 레이트로 수렴하기 전에 버퍼가 많아야 한정된 B 배로 커질 것이라는 특성을 갖는다.

[0087] [116] 보다 정교한 레이트 추정 방법은 전술한 두 접근법을 결합할 수 있다. 그것은 특히 버퍼 레벨 *B* 및 T_{fast} 의 최소값을 평균 윈도우 너비, 즉, 다운로드 레이트를 평균하는 r-time의 양으로 사용할 수 있다. 보다 일반적으로, 다운로드 레이트는 γ_B 및 T_{fast} 의 최소값의 이전 r-time 동안 평균될 수 있고, 여기서 γ_B 는 적절

히 선택된 상수이다. 그러한 선택은 스톨링 위험이 있는 레이트 하락이 있을 때 그것이 빠르게 반응할 것이라는 특성을 가질 것이고, 그러한 경우에, B가 최소이고 평균은 미디어 버퍼의 미디어 콘텐츠의 p-time 지속기간에 비례하는 r-time동안 이루어질 것이고, 따라서 미디어 버퍼가 반쯤 비는 때까지 레이트 추정이 새 레이트가 될 것이다. 예를 들어, 레이트가 감소하는 때 미디어 버퍼의 미디어 콘텐츠 지속기간이 B이고, 다운로드 레이트가 다운로드 레이트가 감소하기 전에 선택된 프리젠테이션의 플레이백 레이트의 a<1 인 부분 이도록 다운로드 레이트가 감소하고, 비관적으로 레이트 추정이 새 다운로드 레이트로 감소할 때까지 선택된 리프리젠테이션의 플레이백 레이트가 감소하지 않는다고 가정하자. 그러면, 레이트 감소가 일어나는 시간을 넘는 x의 r-time 동안 계속됨에 따라, 버퍼 레벨은 B'=B - x + α·x 이고, 즉, x p-time 은 미디어 버퍼로부터 비워지고 α·x 는 미디어 버퍼로 다운로드 된다. x=B'인 시간의 포인트에서, 즉 p-time의 미디어 버퍼 레벨이 다운로드가 새 레이트에서 된 r-time과 동일한 시간의 포인트에서, 레이트 추정이 새 레이트가 될 것이며, 이는 이 전체 시간 동안 다운로드가 새 레이트에서 되었기 때문에 이 시간의 포인트에서 다운로드의 이전 r-time 동안 추정이 새 레이트가 될 것이기 때문이다. 방정식 x=B'=B - x + α·x 을 x에 대해 풀면 x=B'=B/(2-α)가 산출되고, 즉, 레이트 추정은 버퍼 B'가 여전히 적어도 B/2일 때 새 레이트에 도달할 것이다. 대신에 레이트가 시간의 어떤 포인트에서 상당히 증가하면 T_{fast} 가 최소가 될 것이고 이전 T_{fast} r-time 동안 평균 다운로드 레이트가 이전 B r-time 동안의 평균보다 상당히 높을 것이다.

[0088] [117] 우리는 이제 이 구성에 기초하여 *pker* 레이트 추정 프로세스의 예를 상세히 설명한다. 그것은 단기 레이트 측정들을 사용하고, 이는 리퀘스트 액셀러레이터 (RA)와 같은 다운로드 모듈, 및 추정을 계산하기 위한 버퍼 정보로부터 획득될 수 있다. 버퍼 정보는 유용한 추정을 얻기 위한 단기 레이트 측정들의 윈도우 너비를 결정하는데 사용된다.

[0089] [118] 도 10은 다운로드 레이트가 급격히 하락하는 때 어떻게 *pker* 레이트 추정기가 결론을 이끌어 내는지를 도시한다. 레이트가 하락하자 마자 버퍼 레벨이 하락하기 시작한다. 레이트 추정 역시 조정을 시작한다. 늦어도 버퍼 레벨이 2 인수만큼 하락한 때까지 레이트 추정이 새 레이트(NR)에 도달한다. 예를 들어, 중간에 레이트 결정들이 없어서 Buff가 선형적으로 하락한다. 중간 결정이 있으면, Buff의 하락이 점차 느려질 것이다.

[0090] [119] *pker* 프로세스의 설계 목표는 노이즈가 있는 수들을 갖지 않도록 충분히 큰 평균 윈도우, 그러나 그것이 반응하기에 충분히 짧은 수들을 사용하는 것이다. *pker* 프로세스는 이 목표를 동적으로 변하는 윈도우 크기를 갖는 윈도윙된 평균을 사용함으로써 달성한다. RA는 B, (p-time에서) 플레이백 버퍼의 레벨, 프로세스 파라미터들 γ_B 및 γ_T, T_{fast}, 버퍼의 (p-time에서) 마지막 γ_T-부분을 다운로드 하는데 걸린 r-time에 대한 저장된 값, R, r-time에서 다운로드의 마지막 *C* 지속기간 동안의 평균 다운로드 속도를 포함하는, *pker* 프로세스에 의한 사용을 위해 메모리에 몇몇 변수들을 유지하고, 여기서, C=max(STP, min($\gamma_B \cdot B$, T_{fast},))이고, STP는 수용가능한 최소의 윈도우 크기이고, 이는 (예를 들어, 100 ms와 같은) 샘플 시간 주기를 초과해야 한다. 몇몇 실시예들에서, γ_B =1 및 γ_T =0.5이고, 그러나 다른 값들이 가능하고, 둘 다 양수이고 γ_T >1인 한, 질적으로 유사한 동작을 초래한다. 작은 γ_B는 *pker* 프로세스가 레이트 감소들에 빠르게 반응하도록 하고, 반면에 작은 γ_T는 레이트 증가에 빠르게 반응하도록 한다.

[0091] [120] 여기에 설명된 바와 같이, *C* 지속기간 동안 다운로드 속도를 계산하기 위해, SM은 RA에 의해 주기적으로 제공되는 다운로드 속도 정보를 사용한다. 그 목적을 위해, SM은 RA에 의해 제공되는 다운로드 속도 정보의 히스토리를 보관할 수 있다. 평균이 취해지는 지속기간은 많아야 γ_B 버퍼 지속기간이고, 미디어 버퍼 레벨에 상한이 있는 때 얼마나 많은 히스토리가 보관될 필요가 있는지를 효과적으로 제한한다.

[0092] [121] 스트림을 다운로드 하는데 걸린 양의 시간이 그것을 플레이 아웃 하는데 걸린 것과 동일하다면, 우리는 $T_{fast} = \gamma_T \cdot B$ 을 가지므로, 선택된 플레이아웃 레이트가 대략 다운로드 레이트와 동일하다면, 버퍼링 값, C는 대략 버퍼 지속기간임을 주목하라. r-time에서 대략 버퍼 레벨 가량을 선택하는 것이 다운로드 레이트 추정을 위한 평판화 구간(smoothing interval)에 대한 자연스러운 선택이고, 그것이 스톨링을 피하기를 원한다면 스트리밍 클라이언트가 가져야 하는 전시(foresight)의 양이기 때문이다.

[0093] [122] 간단한 한 구현에서, 평균 윈도우 너비는 B, 비디오 버퍼에 포함된 p-time의 양에 비례한다. 그러한 선택이 스톨링으로부터 잘 보호하지만, 단점을 갖는다: 다운로드 레이트가 선택된 미디어의 레이트의 배수이면, 다운로드의 매 초가 다운로드 되고 있는 미디어의 p-time의 k 를 초래하고, 이는 레이트 추정이 아주 느리게 조정하도록 한다. 예를 들어, k=10이고, 10초의 버퍼가 있으면, 레이트 추정기가 조정 전에 약 k·10s=100s의 p-

time을 다운로드 할 것이며, 이는 매우 긴 시간이다. 이는 *pker* 방법들에 T_{fast} 를 도입하는 동기를 부여한다. 사실, 지수 가중 이동 평균이 평판화를 위해 사용되면 사태가 심지어 다소 나빠질 수 있고, 이는 그러한 필터들이 무한 임펄스 응답을 갖기 때문이다. 이 이유로, *pker* 프로세스는 대신에 유한 임펄스 응답을 사용한다. 보통의 이동 평균이 유효하게 동작하고; 구현은 또한 보다 정교한 가중 이동 평균들도 사용할 수 있다.

[0094] [123] 도 13은 이 마지막 포인트를 도시한다. 그것은 단일 (고정-너비) 이동 윈도우 평균과 지수 가중 이동 평균의 비교를 보여준다. 그래프는 레이트 변화가 보이는 때, 고정 윈도우 이동 평균은 처음에 새 레이트로 보다 느리게 수렴하지만, 그것은 일 윈도우 지속시간 내에 수렴할 것이다. 지수 가중 이동 평균은 초기에 빠르게 움직이는 경향이 있으나, 더 뒤의 단계들에서는 단지 느리게 수렴한다. 윈도윙된 이동 평균과 달리, 그것은 고정된 윈도우 내에 수렴하지 않지만, 대신에 수렴하는데 레이트 변화의 크기에 대수적인 시간이 걸린다.

[0095] [124] γ_B =1 및 γ_T =0.5 인 경우, *pker* 프로세스는 다양한 보증들을 제공한다. 하나는, 다운로드 속도가 임의의 인자에 의해 하락하면, 추정은 버퍼가 그 원래 지속시간의 반으로 줄어드는데 걸리는 시간 내에 새 다운로드 속도로 조정된다. 다른 하나는, 다운로드 속도가 임의의 인자에 의해 증가하면, 많아도 하나의 버퍼 워스(buffer worth)의 추가 p-time이 *pker* 프로세스가 새 레이트에 수렴하기 전에 다운로드 될 것이다. 수월한 계산들이 유사한 상수-부분(constant- fraction) 보장들이 $0<\gamma_B$ 및 $0<\gamma_T<1$의 임의의 선택에 대해 유지된다는 것을 보여줄 것이다.

[0096] [125] 버퍼 레벨, B를 계산하기 위한 하나의 접근법은 다음과 같다. T를 미디어 플레이어의 현재 플레이백 p-time이라고 하고, $F_{i,1}$, …, $F_{i,n}$ 을 다운로드 되었거나 되고 있고 시작 시간 오름차순으로 정렬된, 리프리젠테이션 그룹 i 에서 아직 플레이 아웃되지 않은 프래그먼트들이라고 하자. 아직 다운로드 되고 있는 그룹 i 의 임의의 프래그먼트는 $F_{i,1}$, …, $F_{i,n}$ 중에 있다. $a(F_{i,j})$를 바이트 단위의 프래그먼트 $F_{i,j}$ 의 크기로 나누어진 이미 다운로드 된 프래그먼트 $F_{i,j}$ 의 바이트 수와 같은, 다운로드 된 프래그먼트 $F_{i,j}$ 의 비율이라고 하자. 다양한 i 및 j 에 대한 에 대한 값은 RA에 의해 계산되고 SM 에 전달될 수 있다. 주어진 그룹 i에 대해, 우리는 다운로드 된 p-time의 현재 전체 양을 식 1로 정의한다.

$$T_{\mathrm{p},i} := \mathrm{starttime}(F_{i,1}) + \sum_{j=1}^{N_i} \mathrm{duration}(F_{i,j}) \cdot \alpha(F_{i,j})$$

[0097] (식 1)

[0098] [126] 식 1의 결과들로부터 전체 T_P-값을 계산하기 위해, DASH 클라이언트는 MPD(Media Presentation Description metadata) 및 리프리젠테이션 그룹들, G의 수로부터 결정되는, 각 그룹의 가중 인자들, w'를 고려하고, 식 2의 계산을 수행한다. 버퍼 레벨 B는 다음에 $B:=T_p-T$ 인 것으로 정의된다.

$$T_{\mathrm{p}} := \sum_{i=1}^{G} w_i' \cdot T_{\mathrm{p},i}$$

[0099] (식 2)

[0100] [127] 식 2는 현재 플레이 아웃 되고 있는 프래그먼트들에 속하는 버퍼의 부분을 또한 캡쳐한다. 이 정의는 또한 몇몇 프래그먼트들이 동시 다운로드 되는 경우에도 유효하게 작용한다.

[0101] [128] T_{fast}를 계산하기 위해, SM은 일반적인 경우에 얼마간의 히스토리를 보관한다. T_r을 RA가 미디어를 다운로드 하는데 (시도하는데) 소비한 r-time의 총 양이라고 하고, Z를 RA에 의해 다운로드 된 바이트의 총 양이라고 두자. Tr의 값은 RA에 의해 계산된다. SM은 i=1, 2, …, K에 대해, 보통의 간격에서 (예를 들어, 100 ms 마다)샘플링 된, 투플 $(T_{\mathrm{r}}^i, Z^i, T_{\mathrm{p}}^i)$의 히스토리, H를 보관하고, 여기서 K-번째 관측이 마지막이다. 우리는 히스토리가 관측 순서로 저장된다고 가정하고; 따라서 우리는 $T_{\mathrm{p},j}^1 \le T_{\mathrm{p},j}^2 \le \cdots \le T_{\mathrm{p},j}^K$ 와 $T_{\mathrm{r}}^1 \le T_{\mathrm{r}}^2 \le \cdots \le T_{\mathrm{r}}^K$ 및 $Z^1 \le Z^2 \le \cdots \le Z^K$ 또한 갖는다.

[0102] [129] 이제, T_{fast}를 계산하기 위해, B가 전술한 방법으로 이미 계산되었다고 가정하자. 다음에, RA는 예를

들어 이진 검색으로 히스토리를 검색함으로써, 식 3의 부등식이 만족되도록 하는 j를 결정한다.

[0103] $$T_{\mathrm{p}}^{K} - T_{\mathrm{p}}^{j+1} < \gamma_T \cdot B \leq T_{\mathrm{p}}^{K} - T_{\mathrm{p}}^{j} \quad \text{(식 3)}$$

[0104] [130] 다음에 $T_{\mathrm{fast}} := T_{\mathrm{r}}^{K} - T_{\mathrm{r}}^{j}$ 이다. 무한 히스토리를 보관할 필요는 없고, T_i 값들이 최대 버퍼 지속 시간의 γ_B 보다 많이 스팬(span)하는 것으로 충분함을 주의해야 한다.

[0105] [131] 도 16의 확대 변형과 함께, 도 15는 *pker* 프로세스에 의해 사용되는 B 및 T_{fast} 값들이 기록된 (T_p, T_r) 값들의 히스토리로부터 어떻게 결정될 수 있는지를 도시한다. 도면은 r-time과 p-time이 똑같이 빠르게 진행하고(다운로드 중단이 없다), 따라서 플레이백 시간(p-time)이 다운로드 시간(r-time)의 45도 기울기 선이다. (T_p, T_r)-값들의 히스토리는 그래프에서 플레이백 스톨이 발생하지 않았다면, 플레이백 시간 선 바로 위에 있는 커브로 그려질 수 있다. 그러면 버퍼 레벨 B는 플레이아웃 시간에 대한 마지막으로 기록된 T_P-값의 차이이다. T_{fast}의 값은 이 그래프에서 현재 (마지막) T_P-값 아래의 $\gamma \cdot B$의 레벨에서 (T_p, T_r)-커브로의 수평 거리를 측정함으로써 나타낼 수 있다.

[0106] [132] 도 11은 갑작스런 레이트 증가에 대한 *pker* 프로세스의 응답들을 도시하기 위해 도 15-16과 같은 종류의 프리젠테이션을 사용한다. 수신 레이트가 플레이어가 아직 반응하지 않은 갑작스런 증가를 겪을 때 T_{fast}가 상대적으로 작다. 그것은 높은 수신 레이트에 빠른 응답을 나타낸다. 평균 윈도우 전체가 그래프의 높은 레이트 부분 내에 있음에 주목하고, 이는 그것이 상대적으로 좁기 때문이다. 따라서, 이 포인트에서, *pker* 추정은 이미 보다 긴 레이트에 수렴했다.

[0107] [133] 도 12는 레이트 하락에 대한 가변 윈도우 크기 WMA 필터(예를 들어, *pker*) 응답을 도시하기 위해 다시 도 15의 프리젠테이션을 사용한다. 이 경우에, T_{fast}는 상대적으로 크게 되나, 버퍼는 비워지고, 따라서 *B*는 작아져서, 평균 윈도우 전체가 얼마간의 비우는 시간 뒤에 낮은-레이트 영역 내에 들어가게 한다. 도시한 바와 같이, 평균 윈도우의 너비, *B*는 *B*가 T_{fast} 보다 작지만, 버퍼가 완전히 비기 전에 추정은 여전히 더 낮은 새 레이트로 수렴하는 것이다.

[0108] [134] 도 14는 *pker* 레이트 추정 프로세스의 플로우차트이다.

[0109] *[135]* T_{fast} 및 B 값들이 계산되면, *C* 값이 쉽게 나오고 마지막 단계는 지속시간 *C*의 지난 윈도우에 걸쳐 레이트 *R*을 계산하는 것이다. 그 목적을 위해, 히스토리의 Z^i 및 T^i 값들이 사용된다.

[0110] [136] 구간 C에 걸친 레이트를 계산하기 위해, SM 또는 RA는 다음을 행한다: (1) $T_{\mathrm{r}}^{K} - T_{\mathrm{r}}^{j} \geq C$ 인 가장 큰 j를 찾고, 다음에 (2) 식 4에서와 같이 평균 다운로드 레이트를 계산한다. 제 1 단계에서 그러한 j가 존재하지 않으면, SM 또는 RA는 j:=0, 즉 가장 오래된 공지의 관측으로 설정한다. j 값은 이전 검색에 의해 효과적으로 결정될 수 있다.

[0111] $$R := \frac{Z^K - Z^j}{T_{\mathrm{r}}^K - T_{\mathrm{r}}^K} \quad \text{(식 4)}$$

[0112] [137] 각 그룹은 그 그룹이 소비할 것으로 예상되는 전체 대역폭의 비율에 대응하는 연관된 가중치, w를 갖는다. 그것은 바람직하게는 사용할 수 없는 리프리젠테이션들이 제거된 뒤에, MPD에 의해 제공되는 정보의 함수이다. 여기에서, 그룹 g의 가중치 w의 제안된 정의는 w(g) := maxrate(g) + minrate(g)이고, 여기서 maxrate()는 그룹 g에서 최대의 플레이백 레이트이고 minrate()는 최소의 그것이다.

[0113] [138] 가중치들 w로부터, SM 또는 RA는 정규화된 가중치들 w'를 다음과 같이 계산할 수 있다. 클라이언트는 그룹 1, …, G를 스트림하기를 원한다고 가정하면, 정규화된 가중치들은 식 5에서와 같이, 모든 가중치들의 합으로 나누어진 가중치들이다.

$$w_i' = \frac{w_i}{\sum_{j=1}^{G} w_j}$$

[0114] (식 5)

[0115] [139] 정규화는 실제 스트림되는 가중치들에 대해 행해지는 것이 의도된다. 예를 들어, 스트리밍되고 있지 않은 그룹이 있다면, 그것은 계산에 들어가지 않아야 한다.

[0116] *[140]* *pker* 프로세스의 동작에서 몇몇 가정들이 된다. 예를 들어, 별개의 리프리젠테이션 그룹들의 버퍼 레벨들은 상대적으로 서로 가깝게 유지되어야 한다. *pker* 프로세스는 그러한 방식으로 더 잘 동작한다. 예를 들어, 한 그룹이 아주 큰 버퍼를 갖고, 다른 하나가 아주 작은 버퍼를 가지며, 둘 다 비슷한 가중치를 갖는다고 가정하자. 그러한 경우, 빠르게 레이트 추정을 조정해야 할 필요가 있을 수 있고, 이는 작은 버퍼의 경우 상황 변화 시에 스톨링을 피하기 위해 그것이 필요하기 때문이다. 그러나 *pker* 프로세스는 훨씬 큰 버퍼에 대해 행동하는 것처럼 그 추정을 여전히 잘 평탄화할 것이다. 반대로, 큰 버퍼의 경우, 측정들은 버퍼레벨이 허용하는, 다소 높은 변동을 가질 것이고, 따라서 과민한 레이트 결정들을 초래할 것이다.

[0117] [141] 어떤 경우에는, 리프리젠테이션 그룹들이 버퍼 레벨에 큰 차이를 갖는 것이 피할 수 없다. 이 이유로, 다른 구현은 어떤 버퍼들이 아주 작을 때 레이트들을 더 빨리 조정하는, 따라서, 그러한 경우들에 스톨들에 대해 비트를 더 잘 보호할 수 있는 변형된 *pker* 방법을 사용할 수 있다. 그러한 구현은 T_{fast}를 이전과 동일한 방법으로 계산하지만, 윈도우 크기를 C=max(STP, min(T_{fast}, $T_{p,1}$-T, $T_{p,2}$-T, …, $T_{p,N}$-T))로 설정할 수 있다.

[0118] [142] 이들 다운로드 레이트들 추정의 다른 변형은 각 리프리젠테이션 그룹에 대해 독립적인 *pker* 추정을 사용하여 그 그룹에 대한 결정들을 하는 것을 포함한다.

[0119] 3. 페칭 전략(Fetching Strategy)

[0120] [143] 스트리밍 비디오 플레이어들은 일반적으로 제한된 미디어 퍼퍼를 갖는다. 따라서 보통의 동작에서, 버퍼가 가득 찬 상태는 언젠가는 도달하게 될 것으로 예상된다. 버퍼가 가득 찬 상태에 도달하는 때에, 스트리밍 모듈은 버퍼를 과도하게 채우는 것을 피하기 위해 미디어 입력을 막아야 한다. 이를 위한 쉬운 방법이 버퍼가 찰 때는 언제나 버퍼가 다음 프래그먼트를 보유할 수 있을 정도로 충분히 비워질 때까지 기다리고, 그 다음에 페칭을 다시 시작하는 것이다.

[0121] [144] 이 방법의 효과는 각 프래그먼트가 개별적으로 페칭될 것이고, 각 프래그먼트 리퀘스트 사이의 시간 간격, 즉 다음 프래그먼트가 적합하고 리퀘스트될 수 있도록 충분한 버퍼를 비우는데 걸리는 시간의 양이 있다는 것이다.

[0122] [145] TCP 프로토콜은 현재 네트워크 상태에 기초하여 그 다운로드 레이트를 자동적으로 조정한다. 다운로드가 TCP 접속을 통해 시작되는 때에, 초기 다운로드 레이트는 일반적으로 아주 느리고, TCP 프로토콜이 더 높은 다운로드 레이트가 달성될 수 있는지를 알기 위해 검사함에 따라 증가한다. TCP가 얼마나 빨리 다운로드 레이트를 정가시키는지, 일반적으로 TCP가 양단간 TCP 접속의 특성들에 반응하는지는, 매우 복잡하고 고유한 양단간 네트워크 레이턴시들, TCP 송달 및 애크널리지먼트 경로들을 따르는 네트워크 엘리먼트들의 버퍼 용량, 이러한 경로들을 따르는 경쟁 트래픽, TCP의 어떤 변형이 사용되는지, 등을 포함하는, 많은 인자들에 의존한다. 일반적으로, TCP는 느린 다운로드 레이트로 시작하고 시간이 지남에 따라 그 다운로드 레이트를 증가시키고, 따라서 전체 다운로드 시간에 걸쳐 TCP 접속의 평균 다운로드 레이트는 단지 전체 다운로드 시간이 상당한 때 지속 가능한 TCP 다운로드 레이트로 접근한다. 예를 들어, 지속 가능한 TCP 다운로드 레이트가 1 megabit/second 이고 TCP 접속이 본질적으로 0 인 다운로드 레이트에서 시작하고 1 초에 걸쳐 1 megabit/second로 시간이 지남에 따라 선형적으로 증가한다면, 첫 초 동안에 평균 다운로드 레이트는 500 kilobits/second이고, 평균 다운로드 레이트가 지속 가능한 다운로드 레이트의 95%를 달성하는데 10 초의 다운로드가 걸린다. 이 이유로, 많은 다운로드 갭들을 갖는 페칭 전략은 이상적이지 않고, 여기서 다운로드 갭들은 하나의 다운로드 리퀘스트의 완결과 다음 다운로드 리퀘스트의 시작 사이의 시간 주기들이다. 다운로드 리퀘스트들 간의 갭이 0인 때조차도 비-이상적인데, 일반적으로 TCP는 이전 리퀘스트의 완결 이후 다음 리퀘스트에 대한 다운로드 레이트를 늘리는데 얼마간의 시간 주기가 걸리기 때문이다. 각 갭 이후에, 지속 가능한 쓰로우풋이 다시 달성되어야 할 수 있고, 이는

전체 달성된 평균 다운로드 레이트를 줄인다.

[0123] [146] 그러한 감소된 레이트는 더 작은 레이트 추정을, 따라서 더 작은 미디어 레이트의 선택에 이르게 할 수 있다. 이것은 차례로 더 작은(바이트 크기에서) 미디어 프래그먼트들이 다운로드 되고 있도록 하고, 이는 갭들의 상대적 크기를 더 증가시키며, 잠재적으로 훨씬 작은 플레이백 레이트가 선택되도록 한다. 다시 말하면, 효과는 자기-증폭한다.

[0124] [147] 그러므로, DASH 클라이언트 구현이 이 이슈의 영향을 최소화하는 프로세스를 사용하는 것이 유리하다.

[0125] [148] 다음과 같이 일 구현은 미디어 데이터를 계속하여 다운로드하고 그런 다음 다음과 같이 버퍼 레벨을 주기적으로 비울 수 있다. 리퀘스트된 그러나 아직 플레이 아웃 되지 않은 p-time의 양이, 미리 설정된 높은 워터마크, M_h를 초과할 때마다, 버퍼 레벨이 낮은 워터마크 M_l 아래로 떨어질 때까지 SM은 더 이상 어떤 리퀘스트도 발하지 않는다. 보다 구체적인 구현에서, M_h=20초이고 M_l=10초이나, 다른 구현에서 그러한 값들은 더 낮거나 높을 수 있다. 낮은 워터마크 아래로의 하락 이후, 평상적인 동작이 재개하고, SM은 프래그먼트 결정들을 다시 발하기 시작한다.

[0126] [149] 다른 구현은 유사한 효과를 달성하기 위해 프리젠테이션 시간보다 바이트로 특정되는 워터마크들을 사용할 수 있다.

[0127] [150] 버퍼가 주기적으로 비워지고 있다는 사실은 시스템의 다른 부분들에 의해 그들에게 유리하게 사용될 수 있다. 예를 들어, 섹션 6.1.2에서 설명된 바와 같이, RTT의 신선한 추정들을 획득하는데 사용될 수 있다.

[0128] [151] 도 17은 “워터마크” 페칭 프로세스의 동작을 도시한다. 상단 그래프는 비움 주기들과 페칭 주기들의 교대 패턴들이 보이는 버퍼 레벨 그래프이다. 다운로드 레이트는 하단 그래프에 표시된다. 각 페칭 주기의 시작에서, TCP는 지속 가능한 최대 속도에 도달하기 위해 얼마간의 시간이 걸리고, 따라서 평균 다운로드 레이트(페칭 주기 동안)는 최대 달성가능 다운로드 레이트보다 작다. 낮은 워터마크 및 높은 워터마크 사이의 차이가 클수록, 페칭 주기들이 더 길어지고, 평균 레이트가 더 높아진다.

[0129] 4. 레이트 선택 프로세스

[0130] [152] 미디어 데이터를 리퀘스트 시작할 때, 스트리밍 모듈(SM)은 첫 플레이 아웃 레이트 선택을 하기 위해 몇몇 방법들을 사용한다. 그것은 가장 낮은 가용 레이트를 취할 수도 있고, 예를 들어 네트워크 상황의 히스토리를 보관하고 그 다음에 이 히스토리에 기초하여 어떤 플레이 아웃 레이트가 스톨들 없이 유지될 수 있을 것 같은지 선택하는 추정을 결정할 수도 있다. SM이 이미 데이터를 수신하고 있고 따라서 그것이 사용할 수 있는(예를 들어 섹션 2로부터의 방법들로 계산된 레이트 추정들 중 하나와 같은) 레이트 추정 R을 가지면, 그것은 그 레이트에 머무를지 또는 리프리젠테이션들을 변경할지 결정한다.

[0131] [153] 간단한 레이트 결정 프로세스가 이제 기술될 것이다. 수신기는 추정된 다운로드 레이트 R 보다 낮은 플레이백 레이트를 갖는 가장 높은 대역폭 리프리젠테이션을 결정하고, 그것을 데이터를 플레이 아웃 (플레이백) 할 리프리젠테이션으로 선택한다. 수월하긴 하지만, 이 접근법은 다수의 문제점들을 갖는다. 첫째로, 그것은 자연적으로 작은 미디어 버퍼가 늘어나도록 하지 않고, 따라서 다운로드 레이트가 단지 조금 변할 때조차도 스톨들을 의심한다. 둘째로, 변하는 추정 R은 빠르게 변하는 레이트 결정들에 이를 것이고, 이는 필수적이지 않을 수도 있고 시각적으로 불안할 수 있다. 셋째로, 그것은 적어도 대략적으로 프래그먼트의 지속시간인 개시 시간, 따라서 일반적으로 몇 초에 이른다.

[0132] [154] 그러므로 DASH 클라이언트는 그 레이트 결정들을 다운로드 추정 R 뿐만 아니라 버퍼 레벨 B(즉, 버퍼링 되었지만 아직 플레이 아웃되지 않은 p-time의 양) 및 일반적으로 두 연속한 스위치 포인트들 사이의 p-time 지속시간의 추정인 변화 레이트 D와 같은, 콘텐츠에 의존하는 변수들에 기초하는 레이트 결정 프로세스를 구현할 수 있다.

[0133] [155] 따라서, 일 구현은 R에 비례하는 가장 큰 플레이백 레이트를 결정 레이트로 선택할 수 있고, 여기서 비례 인수는 버퍼 레벨의 함수이다.

[0134] [156] 일반적으로, 비례 인수 λ는 버퍼 레벨의 증가 함수이다. 예를 들어, 일 구현은 λ를 버퍼 레벨의 아핀(affine) 함수로 만들 수 있다.

[0135] [157] λ가 버퍼 레벨의 함수이면, 일 구현은 버퍼가 비었거나 작은 때 λ가 작은 것으로 선택할 수 있다. 그러한 선택은 유익한데, 그것은 작은 버퍼가 커지도록 하고, 또한 다운로드 레이트가 정확히 예측되지 않은 때 스톨링에 대한 다소간의 안전성을 제공할 것이기 때문이다.

[0136] [158] 보다 큰 버퍼 레벨에 대해, 일 구현은 λ의 값을 1에 가깝게, 같게 또는 심지어 초과하게 선택할 수 있다. 그것은 스톨링의 급박한 위험이 없을 때 높은 플레이 아웃 레이트가 다운로드 되는 것으로 선택되어, 안정적인 상태에서 높은 품질의 미디어가 스트리밍되는 것에 이른다는 것을 보증할 것이다.

[0137] [159] 레이트 결정 프로세스는 간단한 아핀 함수보다는 B의 구분적(piecewise) 아핀 함수인 λ를 구현할 수 있다. 구분적 아핀 함수들은 임의의 연속 함수들을 임의의 원하는 정도의 정확도로 근접시킬 수 있고, 이는 그것들을 적절한 선택으로 만든다. 같은 특성을 갖는 임의의 다른 파라미터화 할 수 있는 종류의 함수들이 대신 선택될 수 있다.

[0138] [160] 다른 구현은 λ를 p-time 단위의 버퍼 레벨보다는 바이트 단위의 버퍼 레벨의 함수로 할 수 있다.

[0139] [161] 또 다른 구현은 λ를, 버퍼 레벨 B 뿐만 아니라, 버퍼 레벨 B 및 스위치 기회들의 빈도 모두의 함수로 할 수 있다. 그렇게 하는 이유는 레이트를 변경할 기회를 적게 갖는 플레이어가 변경할 기회를 보다 자주 갖는 것들보다 각각의 결정에 대해 보다 먼 장래에 관련될 것이기 때문이다. 따라서, 전자의 경우, 각 결정은 보다 큰 시간 스팬, 및 또한 보다 높은 위험에 관련된다. 이는 스톨링의 위험을 낮게 유지하기 위해, 버퍼 레벨 B 및 추정된 다운로드 레이트 R이 동일한 때 후자보다 전자의 경우에 더 낮은 레이트를 선택하는 것이 나을 것임을 암시한다.

[0140] [162] 레이트 스위치 기회들의 빈도를 고려하는 레이트 선택 프로세스의 구체적 방법은 다음과 같다. D를 스트림에 두 연속하는 스위치 포인트 간의 p-time 의 일반적인 양이라고 하자. D의 값은 인코딩된 비디오에 의존하고, 예를 들어, 두 연속하는 스위치 포인트들 간의 p-time 에서의 최대거리, 또는 두 연속하는 스위치 포인트의 평균 거리, 또는 두 연속하는 스위치 포인트들의 90번째 백분위수, 또는 미디어의 두 연속하는 스위치 포인트들의 p-time 거리의 임의의 다른 적절한 측정인 것으로 될 수 있다. 그러한 D를 고려하면, 방법은 λ가 B/D 구분적 아핀 함수, 또는 예를 들어, $B/max(u,D)$ 또는 $B/(D+u)$와 같은, 그 변형인 것으로 선택하는 것을 포함할 수 있고, 여기서 값 u는 리퀘스트들을 발할 때 초래되는 오버해드(overhead)를 고려하기 위해 추가된다. u의 값은 (예를 들어, 100 ms와 같은) 작은 일정한 양의 시간일 수 있다. 보다 세밀하게는, 일 구현은 u를 추정된 RTT의 작은 배수로 할 수 있다.

[0141] [163] 전술한 방법과 같이, 그 레이트 결정을 단지 $\lambda \cdot R$에 기초하는 프로세스는, R에서의 상대적으로 작은 변동성조차도 많은 레이트 스위치들을 초래한다는 단점을 갖는다. 이는 바람직하지 않을 수 있다. 충분한 버퍼가 있을 때에는, R에서의 작은 변화에 즉시 반응하지 않고, 대신에 버퍼 레벨을 따라서 변하게 두는 것이 나을 수 있다.

[0142] [164] 그러한 동작을 얻기 위해, 프로세스는 둘 다 같은 양(예를 들어, 전술한 바와 같이, B, B/D, $B/max(100\ ms, D)$)의 함수들인 λ 및 μ 값들을 사용할 수 있고, 이는 현재 레이트와 함께, 새 레이트 결정을 선택하기 위한 것이다. 함수들은 $\lambda \cdot R$이 낮은 수용가능 레이트 선택이고, $\mu \cdot R$이 높은 수용가능 레이트 선택 이도록 선택되어야 한다. 그러면 프로세스는 그러한 두 값들을 좋은 레이트 결정을 위한 가이드로서 사용하도록 설계될 수 있다.

[0143] [165] 그러한 설정에서, 함수들인 일반적으로 $\lambda \leq \mu$ 이도록 선택되어야 한다.

[0144] [166] 레이트 결정 프로세스는 이전선택이 이미 $\lambda \cdot R$ 에서 $\mu \cdot R$ 까지의 범위에 있었다면 레이트를 동일하게 유지할 것을 결정할 수 있다. 이전선택이 $\lambda \cdot R$ 보다 작다면, $\lambda \cdot R$ 와 같거나 그보다 작은 가장 큰 가용 플레이백 레이트가 선택된다. 이전 선택이 $\mu \cdot R$ 보다 크다면 $\mu \cdot R$ 와 같거나 작은 가장 큰 가용 플레이백 레이트가 선택된다.

[0145] [167] 일 구현은 함수들 λ 및 μ 가 하드코딩되도록(hardcoded) 선택할 수 있다. 대안적으로, 그것은 환경에 의존하는 보다 정교한 방식의 함수들을 선택할 수 있다. 특히, 일 구현은 적절한 λ 및 μ 함수들을 클라이언트가 최대한으로 할 버퍼링의 양의 함수로서 선택할 수 있다. 온 디멘드 콘텐츠에 대해, 클라이언트는 많은 데이터를, 잠재적으로 몇 분의 미디어 데이터를 프리버퍼링(prebuffer)하는 것을 선택할 수 있다. 낮은 레이턴시 라이브 콘텐츠에 대해, 클라이언트는 단지 많아 봐야 양단간 레이턴시에 의해 규정되는 미디어의 양만을 버퍼링할 수 있고, 이는 아마 단지 몇 초에 불과하다. 작은 버퍼링을 갖는 콘텐츠에 대해, 클라이언트는 보다 보

수적인, 즉 보다 작은 값을 갖는 λ 및 μ 함수들을 선택하도록 결정할 수 있다.

[0146] [168] 구체적인 구현은 예를 들어 두 극(extremal) 함수들 λ_1 및 λ_2 사이에 선형적으로 함수를 보간할 수 있고, 여기서 선택된 보간 포인트는 낮은 버퍼 워터마크 M_l이다(섹션 3 참조). 그래서 그것은 두 하드코딩된 함수들, λ_1 및 λ_2를 갖고, 어떤 값들 m_1, m_2에 대해 λ_1은 m_1보다 작은, 작은 값들의 M_l에 대해 사용되고, λ_2는 $M_l \geq m_2$일 때 사용되며, 여기서 $m_1 < m_2$이다. m_1 에서 m_2 까지의 범위에 있는 값들에 대해, 함수 $\lambda(x) := \lambda_1(x)(m_2-M_l)/(m_2-m_1) + \lambda_2(x)(M_l-m_1)/(m_2-m_1)$ 이 사용된다.

[0147] [169] 우리는 이제 전술한 설명을 따르는 레이트 결정 프로세스의 구체적인 예를 설명한다. 이를 위해, 우리는 몇몇 표기법을 도입한다.

[0148] [170] 1) S_1, S_2, …, S_L 을 리프리젠테이션 그룹의 L 가용 리프리젠테이션들(오름 차순으로 주어짐)의 스트림 레이트들이라고 하자.

[0149] [171] 2) $\lambda(x)$ 를 입력으로 음이 아닌 스칼라를 취하고 음이 아닌 실 스케일일 계수를 돌려주는 구분적 선영(piece-wise linear) 함수라고 하자. 함수 $\lambda(x)$ 는 컴파일 시간에 또는 구성 파일을 통해 설정될 수 있어야 한다. 큰 x에 대해, 예를 들어, M_l 보다 큰 x에 대해, $\lambda(x)$ 는 변하지 않는 것이어야 한다.

[0150] [172] 어떻게 그러한 함수가 구현될 수 있는지에 대한 일 예가 있다. 코너 포인트들 $(0, \lambda_0)$, (x_1, λ_1), …, (x_N, λ_N)이 주어지고, 여기서 x_i는 오름차순이다. $\lambda(x)$ 을 계산하기 위해, $x_i \leq x$이도록하는 가장 큰 i를 찾아라. 다음에, 식 6을 이용하여, 수신기는 함수를 계산할 수 있다.

$$\lambda(x) = \begin{cases} \lambda_i + (\lambda_{i+1} - \lambda_i) \cdot \dfrac{x - x_i}{x_{i+1} - x_i}, & \text{if } i<\mathrm{N} \\ \lambda_N, & \text{if } i=\mathrm{N} \end{cases}$$

[0151] (식 6)

[0152] [173] 그러한 $\lambda(x)$ 함수에 대한 적절한 예는 예시적 파라미터들 N=1,[(0,0.5),(3,1)]에 의해 정의되는 것, 즉, x=0에서 0.5와 같고 x가 3에 이를 때까지 선형적으로 증가하는 함수일 수 있고, 그 포인트에서 함수는 1과 같고 그 이후에 1로 유지된다.

[0153] [174] 3) $\mu(x)$를 다른 그러한 구분적 선형 함수라고 하자. 그러한 함수의 일 예는 x=0에서 0으로 계산하고 x=3에서 15에 도달하며, 그 뒤에 일정하게 유지하는 것이다.

[0154] [175] 4) D를 (이전에 특정된 바와 같이) 일 스위치 포인트에서 다음 것까지의 p-time에서의 지속시간의 추정이라고 하자.

[0155] [176] 5) x := min{(T_d - T), M_l}/ max{D , 1 second)라고 하고, T는 현재 플레이백 p-time, T_d는 레이트 결정이 되는 p-time, D는 전술한 바와 같고, M_l은 버퍼 레벨 낮은 마크이다(섹션 3 참조).

[0156] [177] 6) *CURR*을 현재 선택된 리프리젠테이션(즉, 마지막 프래그먼트 리퀘스트에서 사용된 것)이라고 하자. *UP*를 많아도 $\lambda(x) \cdot R$의 레이트를 갖는 가장 높은 비트레이트 리프리젠테이션의 플레이아웃 레이트라고 하고, 그러한 리프리젠테이션이 없으면 *UP*는 가장 낮은 비트레이트 리프리젠테이션의 플레이 아웃 레이트이다. *DOWN*은 많아도 $\mu(x) \cdot R$의 레이트의 가장 높은 비트레이트 리프리젠테이션의 플레이아웃 레이트라고 하고, 그러한 리프리젠테이션이 없으면, *DOWN*은 가장 낮은 비트레이트 리프리젠테이션의 플레이 아웃 레이트이다. 일반적으로 $\lambda(x) \leq \mu(x)$이므로, 일반적으로 *DOWN* $\geq$ *UP*이다.

[0157] [178] 다음에, 레이트 결정 프로세스는 다음 프래그먼트의 레이트 *NEXT*를 다음과 같이 선택한다: (1) *UP* < *CURR* 이면, *NEXT* := min(*DOWN, CURR*); (2) 아니면 NEXT := UP.

[0158] [179] 위의 단계 5에서, 단순히 D 대신에 max{D, 1 second}를 사용하는 이유는 RTT 때문이고; 1의 역할은 RTT의 상한으로서 역할 하는 것이다.

[0159] [180] 함수 $\lambda(x)$ 및 $\mu(x)$는 x의 함수로서 증가하는 것이 바람직하다. 작은 x에 대해 λ 및 μ 함수들은 <1인 것이 바람직하고, 이는 선택된 플레이 아웃 레이트가 R 보다 작아 작은 버퍼 레벨들에 대한 버퍼 성장을 유발하는 것을 보증한다. 선택된 플레이백 레이트가 많아도 max(λ(B/max{D,1}), μ(B/max{D,1}))·R 과 같아,

$\lambda(B/\max\{D,1\})$ 및 $\mu(B/\max\{D,1\})$ 모두가 1 보다 작은 모든 버퍼 레벨들 B 에 대해 버퍼 성장을 보증하는 것에 주목하라.

[0160] [181] 유사한 프로세스가 새 리프리젠테이션을 플레이백 레이트가 $\lambda(B)\cdot R$ 보다 작은 가장 좋은 리프리젠테이션인 것으로 직접 선택할 수 있다. 이는 여전히 버퍼가 거의 비었을 때 버퍼가 채우려고 하는 경향을 갖는 특성을 가질 것이다. 그러나 그것은 또한 많은 리프리젠테이션 스위치들을 유발하고, 이는 R이 아주 많이 요동칠 수 있기 때문이다. 여기에 설명된 보다 복잡한 레이트 선택 프로세스가 스위치들을 피하려고 시도하고, 대신에 더 낮은 플레이백 레이트로 스위칭 다운 전에 버퍼가 어느 정도까지 비는 것을 허용한다. 이것이 동작하기 위해, 함수들 μ 및 λ 는 큰 버퍼 레벨에 알맞도록 μ 가 λ 를 초과하도록 선택되어야 한다: 선택된 플레이백 레이트가 $CURR$이고, 측정된 레이트가 R이면, 식 7이 만족되는 한 레이트 변경은 일어나지 않고, 수신 레이트가 레이트 스위치들 없이 다소 요동치는 것을 허용함을 주의하라.

[0161]
$$\frac{CURR}{\mu(B/\max\{D,1\})} \le R \le \frac{CURR}{\lambda(B/\max\{D,1\})} \quad \text{(식 7)}$$

[0162] [182] 어떤 버전들에서는, λ 및 μ 는 단지 $B/\max\{D,1\}$의 비 대신에 버퍼 레벨 B의 함수일 것이다. 후자를 도입하는 동기는 다음과 같다.

[0163] [183] a는 다운로드 레이트에 대한 선택된 리프리젠테이션의 플레이백 레이트의 비율을 나타낸다고 하자. 우리는 양호한 a를 결정하기를 원한다. 다음 스위치 포인트까지 다운로드 하는데 대략 $a\cdot D$ 의 r-time이 소요된다. 수신된 데이터가 버퍼에 추가되기 직전에, 버퍼는 $B - a\cdot D$ 로 비워질 것이다. 스톨링을 피하기 위해, 우리는 그 양이 양수이기를 원한다; 안전 쿠션으로서 그것은 일단 그것이 다운로드되면 버퍼에 추가되는 프래그먼트의 플레이백 지속시간 D 에도 비례하여야 하고, 그래서 그것은 어떤 $\beta>0$ 에 대해 적어도 $\beta\cdot D$이어야 한다. 요약하면, 우리는 $B - aD \ge \beta\cdot D$ 을 원한다.

[0164] [184] a 에 대해 풀면 $B/D\beta - \ge a$ 가 나온다. 이는 리프리젠테이션 선택 프로세스는 $B/D - \beta$ 를 넘지 않는 다운로드 레이트 대비 플레이백 비를 선택해야 한다. 함수들 $\lambda(x)$ 및 $\mu(x)$ 는 수용 가능한 그러한 비율들 상의 한계이다; 따라서 그것들은 $x-\beta$ 를 넘지 않는 $x = B/D$의 함수이어야 한다.

[0165] [185] 하나의 프래그먼트를 송신하기 위한 RTT의 추가 비용을 실제로 고려하기 위해, 우리는 B/D를 $B/\max\{D,1\}$으로 대체한다. 보다 일반적으로, 1은 RTT의 추정의 어떤 배수, 또는 서버로부터 미디어 데이터의 다운로드를 시작하는 프로세스들의 반응 시간을 고려하는 다른 파라미터들로 대체될 수 있다.

[0166] [186] 도 18은 플레이백 레이트를 선택하는데 사용될 수도 있는 λ 및 μ 함수들의 예들을 도시한다. x-축은 D 단위의 버퍼 레벨이고, y-축은 수신 비율, 즉 현재 수신 또는 다운로드 레이트로 나누어진 플레이백 리프리젠테이션이다. 선(1802)로 도시된 바와 같이, 수신 비율이 1보다 작으면, 버퍼가 커질 것이고, 그것이 1보다 크면, 그것은 줄어들 것이다. 3 영역들이 식별된다. 첫째로, 플레이어가 결정 포인트에서 λ-커브(1804) 아래에 있으면, 그것은 레이트에서 위로 스위칭할 것이다. 그것이 λ-커브(1804) 및 μ-커브(1806) 사이에 있으면, 그것은 선택된 레이트에 머무를 것이다. 그것이 μ-커브(1806) 위에 있으면, 그것은 아래로 스위칭할 것이다.

[0167] [187] 도 19는 "보수적"인 설정을 사용하는 (λ,, μ)-함수들의 예시적 선택을 나타낸다. 이 설정은 가용 대역폭을 모두 사용하지 않고 그 대신 매우 드물게 스톨할 것이라는 점에서 "보수적"이다.

[0168] [188] 도 20은 "중도적"인 설정을 사용하는 (λ,, μ)-함수들의 예시적 선택을 나타낸다. 이 설정은 보수적인 것보다는 많은 대역폭을 사용하나, 조금 더 스톨들을 하기 쉽다는 점에서 "중도적"이다.

[0169] [189] 도 21은 "공격적"인 설정을 사용하는 (λ,, μ)-함수들의 예시적 선택을 나타낸다. 이 설정은 가용 대역폭을 모두 공격적으로 사용하려 한다는 점에서 "공격적"이다. 그것은 제시된 다른 두 예시적 설정들보다 더 자주 스톨할 것이다.

[0170] [190] 도 22는 MLB 프로세스, 즉, Major League Baseball (MLB)와 일하는 몇몇 연구원들에 의해 제안된 것과 어느 정도까지 유사한 프로세스를 에뮬레이팅하는 프로세스를 사용하는 (λ, μ)-함수들의 예시적 선택을 나타낸다. (λ, μ)-함수들은 미디어 버퍼 참(fullness)에 기초하여 변하지 않는다는 점을 주의하라.

[0171] [191] 도 23은 λ 및 μ 설정들에 대한 나란한 값들의 일 예를 도시한다.

[0172] [192] 도 24는 λ 및 μ 설정들에 대한 나란한 값들의 일 예를 도시한다.

[0173] [193] 도 36은 레이트 선택에서 λ 및 μ 에 대해 사용될 수 있는 값들의 표들을 포함한다.

[0174] [194] 도 25는 레이트 추정, 다음에 레이트-기반 레이트 선택, 다음에 버퍼 관리-기반 레이트 선택을 위한 프로세스를 도시한다. 이 예시적 프로세스에서, 여기에 기술된 하나 이상의 접근법들이 레이트 추정을 행하는데 사용된다. 그 추정에 기초하여, 새 플레이백 레이트가 선택되고 버퍼 관리 룰들에 기초하여 조정될 수 있다.

[0175] 5. 리퀘스트 취소

[0176] [195] 어떤 경우에는, 심지어 양호한 레이트 선택 프로세스도 단독으로는 비디오 플레이백 스톨들을 방지할 수 없다. 예를 들어, 다운로드 레이트가 리퀘스트가 생성되었으나 완결되기 전에 급격히 하락하면, 선택된 비트레이트는 너무 컸을 수 있고, 느린 다운로드 레이트는 플레이백 레이트를 변경할 다음 기회가 오기도 전에 플레이백 스톨에 이를 수 있다.

[0177] [196] 다른 예에서, 예를 들어, 셀룰러 접속에서 WiFi 접속으로의 전이로 인해 가용 대역폭이 극적으로 증가하는 때에 미디어 버퍼는 상대적으로 낮은 플레이백 레이트 미디어로 찰 수 있다. 이 경우, 이미 다운로드 되었지만 아직 플레이 아웃 되지 않은 미디어의 큰 부분을 폐기하고, 폐기된 p-time 부분들을 다시 그러나 이번에는 다운로드할 보다 높은 플레이백 레이트 리프리젠테이션을 선택하여 다운로드하는 것이 유리할 수 있다. 따라서 이미 다운로드 된 낮은 플레이백 레이트 미디어는 취소하고, 다른 리프리젠테이션으로부터의 보다 높은 플레이백 레이트 미디어가 플레이 아웃될 그 곳에 다운로드 되고, 따라서 보다 높은 품질의 사용자 경험을 유도한다.

[0178] [197] 이 이유로, 스트리밍 모듈 구현은 다운로드 레이트를 모니터링하는 모듈을 구현하고 어떤 상황에서 이전의 결정들을 취소할 수 있다. 리퀘스트가 취소되면, 스트리밍 모듈은 보다 새롭고, 보다 적절한 다운로드 레이트의 추정에 기초하여 새 리퀘스트를 발해야 한다. 우리는 여기서 이 모니터링 모듈을 리퀘스트 취소 프로세스라고 부른다.

[0179] [198] 리퀘스트 취소 프로세스는 다른 이유들에서 리퀘스트들을 취소할 수 있다. 예를 들어, 그것은 다운로드 레이트가 급격히 하락하였고, 플레이백이 충분히 빠르게 수신되지 않고 있는 데이터로 인한 스톨링의 위험이 있어 리퀘스트들을 취소할 수 있다. 취소하는 다른 이유는 보다 높은 품질의 미디어가 플레이백을 위해 적시에 선택되고 리트리빙될 수 있다고 결정되는 여부이다. 취소하는 또 다른 이유는 리시버가 리시버가 하는 것과 무관하게 스톨이 발생할 것이라고 결정하고 미완의 리퀘스트의 완결을 허용하는 것에 비해 취소가 스톨 기간을 단축시킬지를 추정하는데 있다. 리시버는 추정된 보다 짧은 스톨이 동반하는 동작을 선택하고, 또한 플레이 백될 미디어 리프리젠테이션의 품질도 잠재적으로 고려한다. 물론, 스톨이 있는지 없는지 및 스톨이 있다면 그 지속시간은 추정과 다를 수 있다.

[0180] [199] 실제 취소는, 일단 그것이 결정되었다면, 리퀘스트가 이슈된 TCP 접속을 닫음으로써 달성될 수 있다. 닫는 것은 서버에게 취소된 프래그먼트에 대한 데이터를 계속 보내지 말라고 하는 효과를 가질 것이고, 따라서, 닫힌 접속에 의해 사용되는 대역폭이 대체 데이터를 페칭하는데 사용될 수 있게 된다.

[0181] [200] 그 다음 스트리밍 모듈은 취소된 조각을 대체할 리퀘스트를 이슈할 수 있다. 이 목적을 위해 새로운 TCP 접속을 여는 것이 필요할 수 있다.

[0182] [201] 일 구현은 대체 리퀘스트를 선택하는 몇몇의 옵션들을 갖는다. 어느 것이 가장 적합한 것인가는 플레이 아웃 되고 있는 콘텐츠의 유형에 의존할 수 있다.

[0183] [202] 그것은 스트림의 끊김 없는 플레이 백을 허용하는 대체 리퀘스트를 선택하려 할 것이다. 일반적인 경우에 이는 대체 리퀘스트는 이전 다운로드 된 프래그먼트의 종료 시간에 또는 그 전에 스위치 포인트를 가져야 한다는 것을 의미한다.

[0184] [203] 그 경우에, 취소 없이 다운로드를 계속할 때 스톨이 예견되고, 취소 및 대체 세그먼트의 선택으로, 스톨이 회피되거나 적어도 지속시간에서 줄어든다면, 플레이어는 취소해야 한다. 그 다음에 그것은 대체 리퀘스트를 위해 그 특성을 갖는 최고 품질의 미디어 리퀘스트를 선택할 수 있다.

[0185] [204] 레이트 취소 프로세스는 스톨들을 다음과 같이 예견할 수 있다: 그것은 프래그먼트의 아웃스탠딩(outstanding) 바이트들의 수를 다운로드 레이트의 추정으로 나눔으로써 이슈된 리퀘스트의 추정된 완결 시간을 계산할 수 있다. 그 시간이 프래그먼트가 평활한 플레이백을 위해 필요한 마감시간보다 뒤이면, 스톨이 예견된

다.

[0186] [205] 긴박한 스톨이 예견되는 때, 리퀘스트 취소 프로세스는 레이트에서의 스위치가 상황을 개선할지 아닐지를 결정하고; 오로지 개선이 있을 것 같을 때만 취소 결정이 된다.

[0187] [206] 일 구현은 레이트 추정 및 후보 대체 프래그먼트의 크기에만 기초하여 대체 프래그먼트를 로드하는데 걸리는 시간을 추정할 수 있다.

[0188] [207] 다른 구현은 취소로 인한 추가 오버헤드 또한 고려할 수 있다: 그것은 존재하는 리퀘스트를 취소하고 새로운 리퀘스트를 이슈하는데 필요한 시간을 계산하기 위해 추정된 RTT의 배수를 추가할 수 있다. 취소된 리퀘스트로부터 네트워크 상으로의 전달을 위해 큐잉되나(queued), 아직 목적지데 도착하지 않은 데이터가 추가 지연에 기여할 수 있다. 클라이언트는 TCP 수신 윈도우 크기를 추정된 레이트로 나눔으로써 이 지연을 추정할 수 있다. 지연의 다른 추정은 추정된 대역폭-지연 곱에 기초할 수 있다. 클라이언트는 둘 중의 최대와 같이, 두 추정들의 조합을 취할 수 있다.

[0189] [208] 요약하면, 클라이언트는 큐잉 지연의 추정에 더하여 전체 대체 프래그먼트를 다운로드 하는데 필요한 시간, RTT에 일반적으로 비례하는 양의 합을 계산한다. 스톨이 예견되고 그 시간이 현재 프래그먼트를 다운로드 하는데 추정되는 잔여 시간보다 작으면, 취소가 이슈된다.

[0190] [209] 초기 레이트 선택이 정확하지 않았기 때문에, 첫 번째 프래그먼트를 다운로드 하는데 원했던 것보다 더 오래 걸리는 것을 플레이어가 인식하면, 리퀘스트 취소 프로세스는, 개시 시에도 또한 취소할 수 있다.

[0191] [210] 다른 레이트 취소 구현은 또한 끊김 없는 플레이백을 허용하지 않지만, 대신에 다수의 프레임들을 스킵(skip)하는 것을 수반하는 대체 리퀘스트를 선택할 수 있다. 이는 양단간 지연이 작게 유지되어야 하는 것을 요구하는 라이브 콘텐츠를 재생하는 때에 필요할 수 있다.

[0192] [211] 프레임 스킵으로 취소들을 하는 구현은 프레임 스킵이 가능한 한한 작도록 대체 프래그먼트를 선택할 수 있다.

[0193] [212] 그 구현은, 대체 리퀘스트로서, 특정 스톨 지속시간 또는 스킵 프레임 지속시간을 초과함 없이 지속 가능하게 다운로드 될 수 있는 최고 품질의 리퀘스트를 선택할 수 있다.

[0194] [213] 다른 종류의 취소는 이미 다운로드 된 프래그먼트에 대해 구현될 수 있다: 플레이어가 플레이 아웃 될 어떤 미디어를 이미 버퍼링했다면, 그것은 이전에 버퍼링된 저-품질 버전을 폐기함과 동시에, 네트워크를 통해 더 고품질의 리프리젠테이션을 페치하고 그것을 스트림하기 위해 그렇게 결정할 수 있다.

[0195] [214] 그 취소 프로세스는 그것이 그것이 더 양호한 품질의 비디오를 적시에 수신하여 그것이 스톨링 없이 플레이 아웃될 수 있다고 결정하면 이들 대체 동작들을 수행하는 것으로 결정할 수 있다

[0196] [215] 도 26은 T1 시각의 새 프래그먼트 리퀘스트 직후에 발생하는 다운로드 레이트에서의 강한 하락을 도시한다. 리퀘스트 시까지 수신 레이트는 OR이었고, 그 다음에 그것은 NR로 떨어진다. 버퍼 레벨은 그때 떨어진다. 새로 리퀘스트된 프래그먼트는 약 T2 = T 1 + OR/NR * 프래그먼트 지속시간에 완전히 다운로드 될 것이다. OR/NR이 크면, 이는 T1 시각의 버퍼의 미디어 콘텐츠의 p-time 지속시간보다 많고, 이는 리퀘스트된 프래그먼트가 스톨 없이 플레이 백 될 수 없다는 것을 의미한다. *pker* 추정기는 레이트 NR로 훨씬 빨리 수렴할 것이나, 리퀘스트가 T1 이전에 되었으므로 추정이 새 레이트 NR로 수렴할 기회를 가지기 전에 프래그먼트의 다운로드가 이루어진다. 스톨을 회피하고, 수정된 추정으로 새 리퀘스트를 이슈하기 위해, 리퀘스트를 취소하고 보다 적절한 비트레이트의 리퀘스트를 다시 이슈하는 것이 필요하다.

[0197] [216] 도 27은 리퀘스트 취소를 갖는 경우를 도시한다. 다운로드 레이트(선 2702)의 급격한 하락 이후에, 버퍼는 비워지기 시작하고,, 추정된 다운로드 레이트(예를 들어, *pker* 프로세스)가 새 다운로드 레이트로 수렴하기 시작한다. 어떤 포인트에서, 스트림 매니저가 프래그먼트가 스톨링 없는 플레이백을 위해 적시에 수신되지 않을 것임을 인식한다. 그 포인트는 도 27의 도표에서 “취소 포인트”(2704)로 표시된다. 그 포인트에서, 부분적으로 수신된 프래그먼트가 취소될 것이고, 그것은 (버퍼 레벨에서의 추가 하락 때문에) 버퍼로부터 쫓겨난다. 그러나 그 이후에, 올바른 레이트의 프래그먼트가 리퀘스트 될 수 있고, 따라서 버퍼 레벨은 더 하락하지 않는다. 사실, 비자명(nontrivial) 레이트-선택 프로세스가 사용되면, 그것은 다시 증가할 것이다.

[0198] [217] 도 28은 예시적 리퀘스트 취소 프로세스를 도시하는 플로우차트이다.

[0199] [218] 도 29는 리퀘스트 취소 검출을 위한 프로세스를 도시한다.

[0200] [219] 우리는 이제 위의 상세한 설명에 기초하여 리퀘스트 취소 구현을 기술한다.

[0201] [220] 이 섹션에서, N_i는 리퀘스트되었지만 아직 완전히 수신되지 않은 리프리젠테이션 그룹 i의 프래그먼트들의 수를 나타낸다. 그것들은 $F_{i,1}$, …, $F_{i,Ni}$으로 참조된다. $F_{i,j}$는 시작-p-time의 오름차순으로 정렬되고, α($F_{i,j}$)는 리퀘스트된 프래그먼트 $F_{i,j}$에 대해 바이트 단위에서 그 크기로 나누어진 이미 다운로드 된 바이트의 양임을 더 가정하라. 변수 T는 현재 플레이백 p-time을 나타낸다. 리퀘스트 취소 검출 프로세스는 도 29의 의사코드(pseudocode)에 의해 도시된 바와 같이 진행할 수 있다.

[0202] [221] 리퀘스트 취소 검출 프로세스가 구동중인 때에, 그것은 어떤 동작도 취해질 필요가 없는 경우, 0(nil)을 리턴하고, 또는 그것은 취소할 그룹의 프래그먼트를 식별할 것이다. 그러한 프래그먼트가 식별되면, 그것은 이 프래그먼트, 및 (p-time 순서에서) 그것 뒤에 오는 동일한 그룹의 모든 것이, 취소되고, 버퍼로부터 내보내질 것임을 의미한다. 다음에 SM은 그 레이트 결정 프로세스를 다시 발동하고, 그 섹션에 대한 새 대안적 리퀘스트를 이슈하여야 한다.

[0203] [222] 프로세스를 설명하기 위해, 당분간 오로지 하나의 리퀘스트만이 여전히 아웃스탠딩(outstanding) 이라고 가정하라. 그 경우에, R이 다운로드 레이트의 정확한 추정이라고 하고, d_{avail}을 문제의 프래그먼트가 플레이아웃될 때까지 여전히 수신될 수 있는 바이트의 수라고 하자. 양 d_{need}는 그 프래그먼트에서 여전히 빠진 바이트의 수이다. 따라서, $d_{avail} < d_{need}$ 이면, 우리는 플레이어가 프래그먼트 $F_{i,j}$ 를 재생하기 전에 스톨될 것임을 예견한다. 이는 위 프로세스에서 첫 번째 “if” 조건을 설명한다.

[0204] [223] 스톨이 예견된다 하더라도, 취소가 스톨을 회피할 수 있거나, 적어도 그 지속시간을 줄일 수 있을 때만 취소하는 것이 타당하다. 취소 이후, 새 프래그먼트가 선택되고 스크래치(scratch)로부터 다운로드되어야 할 것이다. 오로지 하나의 리프리젠테이션 그룹이 있고, 레이트 결정 프로세스가 올바른 레이트를 선택한다면, 이는 대략 λ 곱하기 지속시간 ($F_{i,j}$) 의 시간이 걸릴 것이고, 여기서, λ는 현재 관련된 람다(lambda) 인수이다. 반면에, SM이 스위치 하지 않기로 결정하면, 현재 프래그먼트 다운로드를 완료하는데 $d_{need} \cdot R^{-1}$의 시간이 걸질 것이다. 간략화를 위해, $\lambda = 1$ 을 가정하면, 우리는, 아마도 다른 인자들과 함께, 두 번째 조건을 갖는다.

[0205] 6. 리퀘스트 액셀러레이터

[0206] [224] 스트리밍 미디어 클라이언트를 위한 수월한 방법은 단일 HTTP 접속으로 미디어를 페칭하는 것이다. 그러한 클라이언트는 프래그먼트 리퀘스트들을 순차적으로 처리할 것이다. 그러한 접근법은 비디오 스트리밍에 있어 다소간의 단점을 갖는다. 첫째로, 일반적인 네트워킹 소프트웨어는 긴 다운로드에 걸쳐 최대 쓰로우풋만을 위해 종종 튜닝된다. 이것은 많은 파일들을 받는데 좋지만, 그것은 안정된 수신 레이트와 같은, 중요한 다른 스트리밍 목표들과 상충된다. 둘째로, TCP의 특성상, 링크의 전체 용량은 필수적으로 그러한 HTTP 접속으로 사용될 수는 없다. 채널이 다소간의 지연과 패킷 손실을 경험한다면, TCP는 달성될 수 있는 실제 쓰로우풋을 제한하고, 이는 스트리밍 클라이언트가 양호한 품질의 미디어를 스트리밍하는 것을 잠재적으로 방해한다.

[0207] [225] 이러한 문제점들을 회피하기 위해, 특수한 HTTP 클라이언트가 구현될 수 있고, 이를 우리는 여기서 리퀘스트 액셀러레이터(RA)라고 부른다. 리퀘스트 액셀러레이터는 전술한 문제점들을 회피하거나 줄일 수 있는 특수한 프로세스들을 갖는다. 리퀘스트 액셀러레이터의 구현은 몇몇 키 구성요소를 이용하여 그 목표를 달성할 수 있다. 그것은 몇몇 TCP 접속들을 이용하여 데이터를 수신할 수 있다. 그러한 접속들은 병렬로 활성화될 수 있다. 그것은 데이터 리퀘스트들을 보다 작은 청크(chunk) 리퀘스트들로 분할할 수 있고, 이는 상이한 접속들상으로 개별적으로 다운로드 되고 리퀘스트 액셀러레이터에서 하나의 큰 조각으로 재집합될 수 있다. 그것은 접속들이 서로에게 적정하고, 상대적으로 안정적인 데이터 수신을 갖도록, (특히 TCP 수신 윈도우 크기와 같은) TCP 접속 파라미터들을 튜닝할 수 있다. 그것은 측정된 네트워크 상태와 목표 플레이백 레이트들에 기초하여 사용할 TCP 접속들의 수를 동적으로 조절할 수 있다.

[0208] [226] 사용할 TCP 접속들의 이상적인 수는 네트워크 상태, 및 특히 라운드 트립 시간(RTT) 및 패킷 손실 동작에 의존한다. 따라서 RA는 이들 양들을 추정하는 방법들을 사용할 수 있다.

[0209] [227] RA는 HTTP 리퀘스트를 이슈하는 것으로부터 응답이 들어오기 시작하는 때까지 걸리는 시간을 샘플링함으로써 RTT를 추정할 수 있다. 일 구현은 고정된 시간의 주기, 말하자면 마지막 몇 초 동안 획득된 그러한 모

든 샘플들의 최소를 취함으로써 획득된 RTT의 추정을 사용할 수 있다. 다른 구현은 마지막으로 획득된 N 샘플들의 최소를 추정으로서 사용할 수 있고, 여기서 N은 어떤 정수이다.

[0210] [228] TCP 레이어 상에서 패킷 손실의 측정을 획득하는 것이 대개 어렵고, 이는 TCP 프로토콜이 패킷 손실을 다루고 데이터의 연속하는 프리픽스(prefix)들을 어플리케이션으로 전달하기 때문이다. 따라서, 대신에 패킷 손실에 대한 합리적인 값을 RA 프로세스로의 입력으로 고정하는 것이 때로는 유용하다. 일 구현은 손실이 일정하다고 추정한다. 임의의 패킷 손실 측정이 없으면, RA는 손실을 1%로 추정할 수 있고, 또는 RA는 손실을 0.1%로 추정할 수 있다. 추정은 접속의 유형에 의해 결정될 수 있고, 예를 들어, 추정은 WiFi 접속에 대해 0.1%로 설정될 수 있고 셀룰러 접속에 대해 1%로 설정될 수 있다. RTT들에서의 변형과 같은 다른 방법들이 RA가 간접적으로 패킷 손실을 추론하는데 사용될 수 있다. 대안적으로, 일 구현은 그에 대한 정보를 위해 운영 시스템에 질의함으로써 패킷 손실 추정을 획득할 수 있다.

[0211] [229] 다른 구현은 어플리케이션 그 자체에서 손실을 추정할 수 있다. 이를 하기 위해, 그것은 네트워크 소켓으로부터의 데이터가 일반적으로 최대 세그먼트 크기(maximum segment sized; MSS) 청크들로 수신된다는 점, 그러나 패킷 손실은 훨씬 큰 청크, 대략 전체 TCP 수신 윈도우의 크기의 버스트(burst)의 수신을 유발한다는 점의 인지에 기초하는 다음 절차를 사용할 수 있다. M을 바이트 단위의 MSS(잘 맞는 추측은 $M = 1500$)라고 하자; 그러면 n 바이트가 수신되면, 전송된 패킷들의 수는 약 n/M이다. z를 $k \cdot M$ 보다 많은 바이트 리드(read)를 야기하는 소켓 리드들의 수라고 하고, 여기서 k는 얼마간의 작은 정수이다. k는 어플리케이션의 두 네트워크 리드들 사이에 k 이상의 패킷이 도착했을 것 같지 않을 만큼 충분히 크게 선택된다고 가정하라. 소켓 상에 계속 대기하는 어플리케이션에 대해, $k = 3$ 이 좋을 것이다. 그 다음에, $p = z \cdot M/n$ 이 패킷 손실 확률의 추정이다. 원하는 시작 포인트로부터 z 및 n을 계산함으로써, 이 절차는 임의의 원하는 범위의 시간에 걸쳐 패킷 손실 레이트를 추정할 수 있다.

[0212] [230] RTT의 추정 및 패킷 손실 확률이 주어지면, 어플리케이션은 필요한 양호한 수의 접속을 계산할 수 있다. 프로세스는 특히 목표 다운로드 레이트가 그 접속 수로 달성될 수 있도록 충분히 큰 접속의 수를 선택할 수 있다. 단일 레이트의 달성 가능한 레이트는 일반적으로 달성 가능한 쓰로우풋에 대한 TCP 식에 의해 제한되고, 이는 대략 단일 TCP 접속은 $T = MSS/(RTT\sqrt{p})$의 평균 다운로드 레이트를 달성할 수 있다고 한다. 따라서, 프로세스는 T로 나누어진 목표 다운로드 레이트에 비례하는 접속들의 수를 선택할 수 있다.

[0213] [231] RA는 또한 실제적인 이유들에서, 사용할 TCP 접속들의 수에 하한 및 상한을 둘 수 있다. 예를 들어, RA는 그것이 여는 접속들의 최대 수를 8로, 접속들의 최소 수를 2로 한정할 수 있다.

[0214] [232] 대역폭, 손실 확률, 및 RTT는 변하기 쉽다. 리퀘스트 액셀러레이터는 그러한 양들을 모니터링하고 접속들의 수를 동적으로 변경한다.

[0215] [233] 리퀘스트 액셀러레이터는 HTTP 리퀘스트들을 더 작은 서브리퀘스트들로 분할하고 모든 서브리퀘스트에 대해 되돌아온 데이터 응답들을 원래 리퀘스트에 대응하는 코헤런트 응답으로 재집합할 수 있다. 리퀘스트들을 서버리퀘스트들로 분할하는 데는 많은 이점들이 있다. 첫째로, 가용 TCP 접속들을 이용하기 위해서는, 그들 모두에 리퀘스트들을 이슈할 필요가 있다. 미디어 스트리밍 플레이어는 모든 접속들을 사용하기에 충분한 리퀘스트들을 제공하지 않을 수 있다. 리퀘스트 분할은 이러한 문제점들을 완화하는데, 그것은 더 큰 수의 서브리퀘스트들을 낳고, 이는 다음에 다른 접속들 상에 이슈될 수 있기 때문이다. 둘째로, 리퀘스트 분할은 더 짧은 리퀘스트들을 낳고, 이는 부적절한 시기의 데이터 전달의 위험을 줄인다: 어떤 TCP 접속들이 다른 것들보다 일시적으로 더 느리면, 그것들은 여전히 짧은 리퀘스트로 사용될 수 있다. 그것들은 빠른 접속보다 느린 응답을 전달할 것이나, 전체 리퀘스트를 완결하기 위한 추가된 상대적 지연은 일반적으로 그렇게 크지 않을 수 있고, 이는 리퀘스트들이 작기 때문이다.

[0216] [234] 일반적으로, 더 많은 접속들이 사용되면, 리퀘스트 당 더 많은 서브리퀘스트들을 생성하는 것이 바람직하다. 이를 달성하기 위해, n 접속들이 있을 때 리퀘스트 액셀러레이터는 각 리퀘스트를 n 서브리퀘스트로 분할할 수 있다.

[0217] [235] 다른 구현은 리퀘스트 크기에 의존하여 리퀘스트당 서브리퀘스트들의 수를 선택한다. 서브리퀘스트 크기가 다운로드 하는데 고정된 양의 시간(말하자면, 2초)이 걸린다고 예견된 크기인 것으로 선택되면, 원하는 효과를 달성하기 위해, 보다 많은 접속들이 있다면 리퀘스트들은 보다 많은 서브리퀘스트들로 분할될 것이다.

[0218] [236] 분할 룰은 불필요하게 작은 서브리퀘스트들이 없음을 확실히 해야 한다. 예를 들어, RA 구현은 분할 프로세스들에서 최소 서브리퀘스트 크기를 강제하고, 최소가 만족되지 않는다면 더 적은 서브리퀘스트들로 분할

할 수 있다.

[0219] [237] 복수의 TCP 접속들이 사용되는 때에는 그것들은 대역폭에 대해 아마도 경쟁한다. 큰 시간 스케일 상에서, 각 접속은 다른 것들과 같은 양을 수신할 것이지만, 몇 초 동안과 같은, 보다 작은 스케일에서는, 어떤 TCP 접속들은 다른 것들보다 상당히 느릴 수 있다. 이는 스트리밍에 문제를 일으키는데, 그것은 몇몇 서브리퀘스트들은 다른 것들보다 훨씬 오래 걸릴 수 있다는 것을 암시하고, 이는 플레이백 스톨들을 야기할 수 있기 때문이다.

[0220] [238] 이를 회피하기 위해, RA는 접속들을 "길들이는(tame)" TCP 흐름 제어를 사용할 수 있다. 그것은 각 TCP 접속들의 최대 수신 윈도우를 충분히 제한 할 수 있어, 어떤 접속도 쓰로우풋의 그 정당한 몫보다 상당히 많이 사용할 수 없도록 한다. TCP 접속을 통한 비행 중의(전송되었지만 아직 승인되지(acknowledged) 않은) 데이터의 양은 대략 RTT로 나누어진 다운로드 레이트이다. 따라서, TCP 수신 윈도우가 대략 추정된 RTT에 의해 나누어진 접속에 대한 목표 다운로드 레이트로, 또는 그보다 조금 크게 설정되면, 다운로드 레이트는 대략 목표 다운로드 레이트로 또는 그보다 조금 더 크게 제한될 것이다. 따라서, TCP 수신 윈도우 크기를 설정하는 것은 관리자로서 행할 수 있어, 주어진 TCP 접속이 다른 TCP 접속들이 훨씬 낮은 레이트에서 다운로드하도록 강요한 정도의 높은 레이트에서 다운로드 하지 않는 것을 보증할 수 있도록 한다. 그러한 메커니즘이 가동중인 상태에서, 접속들은 대략 동일한 속도로 페칭하는 경향이 있는데, 느린 접속들은 그 경우에 그들의 공정한 몫까지 속도를 올리기 위해 이용 가능한 대역폭을 갖고, 그러나 동시에 접속들은 적어도 집합 목표 수신 레이트인, 또는 그보다 조금 높은, 집합 다운로드 레이트를 달성할 수 있기 때문이다.

[0221] [239] RA는 수신 버퍼를 조정함으로써 수신 클라이언트의 수신 윈도우를 조정할 수 있다. 그것은 연속한 리퀘스트들 사이에서 항상 이 설정들을 재조정한다.

[0222] [240] 일 구현은 각 접속의 TCP 수신 윈도우를 추정된 RTT 및 접속들의 수로 나누어진 목표 다운로드 레이트의 곱 보다 조금 크게 설정할 수 있다.

[0223] [241] 목표 다운로드 레이트는 예를 들어 플레이 백 할 미디어 레이트로부터 결정될 수 있다. 다른 구현은 현재 플레이백 레이트(예를 들어, 현재 다운로드 레이트의 두배)에 기초하여 목표 레이트를 설정할 수 있다.

[0224] 6.1 RA 실시예

[0225] [242] 우리는 이제 전술한 엘리먼트들을 포함하는 리퀘스트 액셀러레이터의 실시예를 설명한다.

[0226] [243] 도 30은 복수의 TCP 접속들로 페칭하는 동작의 도표이다. 도 30-31은 다른 조건 하의 동작을 나타낸다. 예를 들어, 웹 서버로의 접속은 초당 2 megabit("mbps")로 대역폭 제한되었고, 라운드 트립 시간은 150ms였으며, 0.1%의 패킷 손실이 있었다. 프래그먼트들을 패칭하는 4개의 활성 접속들이 있었다. 도 30-31의 도표는 집합 레이트들, 클라이언트에 서 획득된 RTT 추정뿐만 아니라, 4 접속들의 순간 레이트를 나타낸다.

[0227] [244] 도 30에서, 접속들의 수신 버퍼들은 제한되지 않는다. 도 31에서, 그들은 대략 두 배의 대역폭-지연-곱으로 제한된다.

[0228] [245] 도 30 및 도 31의 예에서, 두 방법은 모두 2mbps 전체 쓰로우풋을 안정적으로 달성한다. 접속들이 수신 윈도우들을 제한한 경우(도 31), 접속들 간의 전달은 훨씬 더 고르다: 대부분 그들은 대략 동일한 레이트로 수신한다. 제한되지 않은 윈도우들을 갖는 접속들(도 30)에 대해 그것은 전혀 사실이 아니며, 이 경우 긴 시간의 스트래치에 걸쳐 어떤 접속들은 다른 것들보다 더 느리다.

[0229] [246] 고르지 않은 접속 속도들은 스트리밍 어플리케이션에 문제가 되는데, 이는 어떤 긴급한 데이터가 단지 아주 느리게 (느린 접속 상으로) 수신되고 있고 반면에 대역폭은 긴급하게 필요하지 않은 데이터를 페치할 지도 모르는 더 빠른 접속들로 전용된다는 것을 의미할 수도 있기 때문이다.

[0230] [247] 제한되지 않은 수신 윈도우와 제한된 수신 윈도우 간의 다른 차이는 클라이언트가 동작하는 RTT이다. 제한이 있는 경우, RTT는 전파 지연에 가깝게, 느리게 유지한다. 수신 윈도우 제한이 없는 경우, 비행중인 데이터의 양이 기저 전파 지연 및 접속 용량의 곱을 초과하여 큐잉 지연이 아주 현저하게 되고, 높은 RTT를 유발한다. 높은 RTT는 미디어 스트리밍 클라이언트에 대해 바람직하지 않은데, 많은 이벤트들에 대한 클라이언트의 반응 시간은 일반적으로 RTT의 배수이기 때문이다. 예를 들어, 새 미디어 콘텐츠가 다운로드되도록 하는 사용

자 찾기 이벤트, 또는 리퀘스트 취소 또는 리프리젠테이션들의 스위치를 유발하는 다운로드 속도의 감소에 대한 클라이언트 반응 시간은, 일반적으로 현재 RTT의 큰 배수이고, 따라서 그러한 이벤트에 대한 클라이언트의 일반적 민감도는 RTT가 클 때 저하될 것이다.

[0231] [248] 도 32는 리퀘스트 액셀러레이터 프로세스의 플로우차트이다.

[0232] [249] 도 33은 소정의 프래그먼트 리퀘스트에 대해 만들 서브리퀘스트들의 수를 찾기 위한 프로세스를 도시한다.

[0233] [250] 도 34는 계산된 크기들을 갖는 소스 리퀘스트들의 해체 구간들인 것으로 선택된 개개의 리퀘스트들을 선택하기 위한 프로세스를 도시한다. 이 프로세스에서, 서브리퀘스트 크기는 의도적으로 랜덤화되어, 접속들이 아이들(idle) 상태인 시간이 접속 마다 다르게 한다. 이는 모든 접속들이 동시에 아이들 상태인 것을 방지하여, 더 양호한 채널 이용을 가능하게 한다. 리퀘스트 크기들은 또한 정렬되어, 더 큰 리퀘스트들이 먼저 나가 아이들 시간에서의 차이들을 제한되도록 돕는다.

[0234] [251] 도 35는 시간 오프셋들 및 시간 오프셋들에 의해 결정된 수선 세그먼트(repair segment)에 대한 프래그먼트 구조를 나타낸다.

[0235] [252] 동작에서, 리퀘스트 액셀러레이터는 SC로부터 HTTP 리퀘스트들(각 리퀘스트들은 URL 및 바이트 범위임)을 수신한다.

[0236] [253] 리퀘스트 액셀러레이터는 HTTP를 통해 리퀘스트된 바이트 범위들을 다운로드하고 데이터를, 그것이 완전히 수신되면, SC로 다시 넘긴다. RA는 충분히 큰 다운로드 속도를 달성하지만, 동시에 각 프래그먼트가 그 마감시간 전에 수신된다는 것을 확실히 하는 것을 목표로 한다. 높은 다운로드 속도는 높은 품질의 비디오 리프리젠테이션을 선택하는 것을 가능하게 하고, 한편 마감시간을 지키는 것은 플레이백이 스톨 없이 진행한다는 것을 확실히 한다.

[0237] [254] 높은 다운로드 속도의 목표를 달성하기 위해, RA는 개방된 TCP 접속들의 바뀌는 수를 관리하고, 이들 모두는 HTTP를 통해 데이터를 수신하는데 사용된다. RA는 얼마나 많은 접속들을 사용할지, 필요하면 그것들을 개방 또는 재개방할지, 및 접속들에 리퀘스트들을 어떻게 내보낼지에 대한 세부사항을 관리한다.

[0238] [255] RA는 몇몇 경우에 소스 리퀘스트들을 다음에 상이한 접속들로 보내지는 더 작은 소위 *RA 리퀘스트들* 및 도착 시 RA에 의해 투과적으로(transparently) 재조립되는 응답 데이터로 분할할 것을 결정할 것이다. 예를 들어, 어떤 파일의 처음 64 kilobytes를 포함하는 소스 리퀘스트에 대해, RA는 두 RA 리퀘스트들을 생성할 수 있고, 하나는 32 kilobytes 청크이고 다른 하나는 그 파일의 제 2의 32 kilobytes 청크이다. RA는 그 다음에 그 두 청크들을 두 다른 접속들 상에 병렬로 리퀘스트하고 두 32 kilobyte 청크들이 수신되었으면 원래 리퀘스트를 위해 코헤런트 64kilobyte 응답을 생성할 수 있다.

[0239] [256] RA는 소스 리퀘스트들의 단지 보통의 서브범위들 보다 많은 RA 리퀘스트들을 이슈할 수 있다. 예를 들어, 그것은 평이한 비디오데이터에 더하여 프래그먼트의 FEC 데이터에 대한 리퀘스트를 이슈할 수 있다. 그 경우, RA는 그것이 수신되었으면 FEC 정보를 투과적으로 디코딩하고, 오로지 최종의, 디코딩된 프래그먼트를 소스로 제공할 것이다.

[0240] [257] RA는 네트워크 상태에 따라 그 자신을 자동튜닝한다. 예를 들어, RTT가 크면, RA는 리퀘스트들 간에 많은 아이들 시간을 피할 수 있도록 보다 큰 청크 리퀘스트들을 이슈할 것을 결정할 수 있다. 자동튜닝의 다른 예는 RA가 그 리퀘스트들의 적시성을 보장할 수 있도록 개개의 접속들의 속도들을 보다 작게 유지하려 한다는 것이다. 그러한 것들을 할 수 있기 위해, 바람직하게 RA는 그 접속들의 소켓들에 직접 액세스한다. 예를 들어, Unix-같은 환경에서, 그것은 setsockopt() 함수를 이용하여 소켓 옵션들을 설정할 수 있을 것이다.

[0241] [258] RA는 네트워크 상태를 관리하고 계속 확인한다; 이는 특히 다운로드 레이트 및 추정된 라운드 트립 시간(RTT)를 측정하는 것을 포함한다. 그것은 이 정보들을 수집하는데, 첫째로 접속 자동튜닝은 그것들의 가용성에 의존하기 때문이고, 둘째로 대역폭 정보는 SM에 전달될 필요가 있고, 이는 그것을 이용하여 그 레이트 추정들을 계산하기 때문이다.

[0242] [259] RA가 (SC를 통해) SM으로 전달하는 정보의 다른 부분은 아웃스탠딩(outstanding) 리퀘스트, 즉 주어진 리퀘스트의 얼마나 많은 데이터가 이미 수신되었는지에 관한 진행 정보이다. SM은 그 정보를 그 레이트 추정들 및 리퀘스트 취소 결정들 모두에 사용한다.

[0243] [260] RA는 SM에 의해 대역폭 추정들을 하는데 필요한 정보를 계속 확인한다. 이 정보는 다운로드에 소비된 r-time의 총 양, T_r, 및 다운로드 된 바이트의 총 양, Z이다. 이 수들 모두는 감소하지 않고 증가하고, SM에 의해 자주 조사된다. T_r 타이머는 적어도 하나의 접속이 활성일 때에만(if and only if) 구동하고 있다. 접속은 HTTP 리퀘스트를 보내고 있거나 응답 데이터가 들어오기를 기다리고 있으면 활성이라고 간주된다. Z 카운터는 들어오는 바이트들을 카운트하고 모든 접속들에 걸쳐 집합된다.

[0244] RA 다운로드 레이트 히스토리

[0245] [261] 리퀘스트 액셀러레이터는 (T_r, Z)-쌍들의 커지는 어레이를 보관함으로써 레이트의 어떤 히스토리를 계속 확인하며, 이는 그들의 역시적인 순서로 저장된다. 우리는 이것을 어레이 mapTrZ라고 한다. mapTrZ의 업데이트들은, 빈번하게, 적어도 고정된 시간 간격으로(예를 들어, 100 ms 마다), 및 아마도 또한 뉴 데이터가 수신될 때, 일어난다.

[0246] [262] RA는 다음과 같이 윈도윙된 대역폭 추정을 계산하는데 mapTrZ를 사용한다. 너비 t의 관심 대상인 윈도우를 고려하고, mapTrz[last]를 mapTrZ의 마지막 엔트리(entry)라고 하자. 그 다음 mapTrZ[i].$T_r \leq$ mapTrZ[last].$T_r - t$ 를 만족하는 가장 큰 인덱스 i를 찾아라. i는 이전 검색으로 효과적으로 찾을 수 있음에 주목하라. 그러면 레이트 평균은 식 8에 표시된 바와 같다.

[0247]
$$R := \frac{\mathrm{mapTrZ[last].}Z - \mathrm{mapTrZ}[i].Z}{\mathrm{mapTrZ[last].}T_r - \mathrm{mapTrZ}[i].T_r} \quad \text{(식 8)}$$

[0248] [263] 식 8은 후속하는 Tr에서의 차이가 t에 비하여 작다고 가정한다. 이는 충분히 자주 샘플링하고 절대 작은 윈도우 너비 t를 선택하지 않음으로써 보증된다.

[0249] [264] 실제로, 임의로 커지는 어레이는 성가시다. 과거가 살펴지는 최대 지속시간은 상한이 있을 수 있고, 따라서 mapTrZ를 대신에 고정된 크기의 링 버퍼로 구현하는 방법이 있다. 이는 다음과 같이 행해질 수 있다. mapTrz 어레이가 업데이트되어야 하고, mapTrz 어레이는 이미 적어도 두 쌍을 포함하고 있을 때에는 언제나, T_r - mapTrZ[last-1].T_r < 100 ms 이면 마지막 엔트리를 대체하고, 아니면 새 엔트리를 추가하라.

[0250] 6.1.2 라운드 트립 시간(“RTT”) 추정

[0251] [265] RA는 대역폭 추정들을 수집한다. 선험적으로, RTT 샘플을 얻기 위한 간단한 방법은, HTTP GET 리퀘스트가 아이들 접속 상으로 전송되고 응답이 들어오기 시작하는 시간에서의 차이를 측정하는 것이다.

[0252] [266] 그러나, 그러한 측정들은 큐잉 지연을 포함한다; 클라이언트가 다른 열린 활성 접속들을 가지는 경우, 클라이언트로의 그 링크가 그것이 데이터를 수신하는 레이트보다 낮은 레이트를 가지면. 클라이언트로 데이터를 보내는 마지막 홉(hop)은 다수의 패킷들을 버퍼링할 수 있다. 그 경우에, 패킷들은 그들이 본래 하는 것보다 더 긴 지연으로 전달될 수 있다.

[0253] [267] 우리의 경우, 클라이언트 그 자체의 활동에 의해 유도된 큐잉 지연에 대한 RTT 디스카운팅(discounting)을 아는 것이 바람직하다. 그 양의 추정을 얻기 위해, 우리는 다음과 같이 진행한다.

[0254] [268] 활동의 각 주기 동안, 우리는 후술하는 타이밍 방법으로 RTT 샘플들을 수집한다; 각 GET는 그 결과 샘플이 된다. 현재 추정은 그 다음 그 샘플들 모두의 최소값이다. 샘플들의 리스트는 RA가 비활성이 될 때마다 비워진다. (클라이언트는, 예를 들어, 섹션 3의 높은 워터마크가 초과되고, 시작된 다운로드들이 완료한 때, 비활성이 된다.) 비활성 주기들에서, 또는 임의의 RTT 샘플이 수신되기 전의 활성 주기들에서, RTT 추정은 마지막 공지의 추정이다.

[0255] [269] RTT 추정기는 또한 “어떤 RTT 추정도 알려지지 않음”의 표시를 리턴할 수 있고, 이는 예를 들어 클라이언트 개시 시 사용될 수 있다.

[0256] 6.1.3 TCP 접속들의 수의 조정

[0257] [270] TCP 흐름 제어를 튜닝하는 것은 RA가 다른 접속들에서의 대역폭을 대략 동일하게 유지하게 한다. 많은 설정 가능한 튜닝 상수들은 k_R(RTT들에서 측정된 레이트 측정 윈도우; 제안 값:30), k_N(비례 인수; 제안 값:8192 bytes), N_{min}(N_{target} 낮은 캡); 제안 값: 1), 및 N_{max}(N_{target} 높은 캡; 제안 값: 8)을 포함할 수 있다.

[0258] [271] 추정된 대역폭-지연-곱(*bandwidth-delay-product; BDP*)은 $BDP := RTT \cdot R$ 으로 정의되고, 여기서, *RTT*는 (위에서와 같이) 추정된 *RTT*이고, *R*은 (윈도우 방법으로 추정된) 마지막 $k_R \cdot RTT$ 시간동안의 평균 수신 레이트이다.

[0259] [272] 접속들의 목표 수는 다음에 식 9에서와 같이 정의되고, 여기서 kN은 설정 가능한 상수이다.

[0260] $$N_{target} := \max(N_{min}, \min(N_{max}, \lceil BDP/k_N \rceil)) \quad \text{(식 9)}$$

[0261] *[273]* N_{target} 의 값은 주기적으로 재계산된다. 현재 열린 접속들의 수가 N_{target}보다 작으면, 새 접속들은 N_{target} 에 매칭하기 위해 즉시 열린다. 반면에, N_{target}이 현재 열린 접속들의 수보다 작으면, 어떤 즉각적인 동작도 취해지지 않는다. 대신에, RA 리퀘스트가 완료되는 때는 언제나, RA는 너무 많은 접속들이 열려있는지를 체크하고, 그렇다면, 방금 아이들 상태가 된 접속을 닫는다.

[0262] 6.1.4 접속들 상의 TCP 수신 윈도우의 조정

[0263] [274] RA는 각 접속의 TCP 수신 윈도우 크기를 $\lceil c_w \cdot \mathrm{BDP}/N_{target} \rceil$으로 설정한다. 여기서, c_w는 설정 가능한 하드코딩된 상수이고, 예를 들어, c_w = 3이다. RA는 접속의 TCP 수신 윈도우의 크기를 그것이 그 접속에 다음 HTTP 리퀘스트를 이슈할 할 때마다 설정한다.

[0264] 6.1.5 리퀘스트 분할 프로세스

[0265] [275] RA로 전달된 각 소스 리퀘스트는 잠재적으로 하나보다 많은 RA 리퀘스트로 분할되고, 그 각각은 리퀘스트된 범위의 다른 부분들에 대응한다. 소정의 소스 리퀘스트에 대응하는 RA 리퀘스트들이 모두 완료되면, 수신된 데이터는 RA에 의해 완전한 프래그먼트로 재집합되고, 이는 다음에 SC로 리턴된다.

[0266] [276] 소정의 HTTP 리퀘스트에 대해, RA는 몇몇의 튜닝가능한 값들에 의존하는 프로세스를 이용하여 RA 리퀘스트들의 수, *n*을 결정한다. *n*의 값은 다음 튜닝가능한 상수들에 의존한다: T_{wn}(레이트 추정 윈도우 너비; 제안 값: 4s), D_{min}(최소 페치 지속시간; 제안 값: 2s), 및 c_s(RTT들에서 최소 페치 지속시간; 제안 값: 6).

[0267] [277] 그리고 소정의 프래그먼트 리퀘스트를 위해 준비할 서브리퀘스트들의 수 *n*을 찾기 위한 프로세스는 도 33의 의사코드에 나타난 바와 같다.

[0268] [278] 그리고 개개의 리퀘스트들은 예를 들어, 계산된 크기들을 갖는, 도 34에 도시된 프로세스를 사용하여 소스 리퀘스트들의 해체 구간으로 선택된다.

[0269] 6.1.6 리퀘스트 발송 프로세스

[0270] [279] 리퀘스트 액셀러레이터는 RA 리퀘스트들의 세트를 유지한다. 접속들이 다음 리퀘스트를 이슈할 준비가 되면 언제나, 리퀘스트는 큐(queue)가 비어있지 않으면 RA 큐로부터 디큐잉되고(dequeued), 아이들 접속 상으로 이슈된다. 큐가 비어있으면, 새 프래그먼트 리퀘스트가 SC로부터 획득된다. 그리고 그 리퀘스트는 RA 리퀘스트들로 분할되고 RA 큐 상에 큐잉된다. 큐잉은 바람직하게는 소정의 프래그먼트 리퀘스트를 준비하기 위한 서브리퀘스트들의 수를 찾기 위한 프로세스에 의해 리턴된 바와 같은 슬라이스들의 순서로 행해진다.

[0271] [280] HTTP 접속들은 다양한 이유로, 예를 들어 웹 서버 타임아웃(timeout)이 발생했거나, 단일 접속 상에 이슈될 수 있는 리퀘스트들의 수가 초과되었기 때문에, 셧다운될 수 있다. RA는 이 상황을 우아하고 투명하게 다

루어야 한다. 접속이 셧다운될 때마다, RA는 접속을 자동으로 다시 연다. 리퀘스트가 닫힌 접속 상에 진행하고 있었다면, 그것은 접속으로부터 디큐잉되고, 아직 수신되지 않은 부분에 대한 새 RA 리퀘스트가 RA 큐의 앞에 위치한다.

[0272] [281] 이 절차는 닫힌 접속들이 성능에 최소한의 영향을 주도록 보증한다.

[0273] 6.1.7 특정 실시예에서의 RA 파라미터 선택

[0274] [282] TCP 접속은 그 흐름 제어에 의해 제한된다: 어드버타이징된(advertised) 수신 윈도우는 임의의 포인트의 시간에 승인되지 않도록(unacknowledged) 허용되는 데이터의 양에 상한을 둔다. 따라서, W가 수신 윈도우의 크기를, bdp는 그 접속의 대역폭-지연-곱을 나타낸다고 하면, 우리는 $bdp \leq W$(조건 1)을 갖는다. 섹션 6.1.4의 방법은 c_w>1이면 이 조건(1)이 만족되도록 수신 윈도우 크기를 선택한다. 이는 개별 접속들이 가용 대역폭의 그들의 정당한 부분보다 실질적으로 더 많이 차지할 수 없도록 보증한다. 레이트 증가들을 허용하기 위해, 그리고 레이트 악순환()을 피하기 위해, c_w 를 1보다 다소 큰 것, 예를 들어, c_w = 2 또는 c_w = 4 를 선택하는 것이 바람직하다. 값이 클수록 레이트는 더 빠르게 커지지만, 접속들은 서로에게 덜 공평하다.

[0275] [283] 다른 제한은 TCP 혼잡 제어 프로세스에 의해 부과된다. p가 패킷 손실 확률을 나타내고, M이 TCP 최대 세그먼트 크기를 나타낸다고 하면, 단일 접속의 레이트 r은 식10에 의해 표시된 바와 같이 제한된다.

$$r \leq \frac{M}{RTT \cdot \sqrt{p}}$$

[0276] (식 10)

[0277] [284] 이제, 이것을 ($bdp = r \cdot RTT$ 및 $BDP = N \cdot bdp$ 를 사용하여) BDP 및 접속들의 수 N 에 관하여 다시 쓰면, 우리는 식 11에서 나타나는 것을 얻는다.

$$BDP\frac{\sqrt{p}}{M} \leq N$$

[0278] (식 11)

[0279] [285] 이는 식 11의 부등식이 유지되는 것을 보증하기 위해 k_N이 식 9의 $M/\sqrt{p}$ 보다 조금 작게 선택되어야 한다는 것을 암시한다. M의 전형적인 값은 1kilobyte이고, 우리가 p = 0.01로 두면, $M/\sqrt{p}$ = 10 kilobytes이다. 따라서, 이 예에서, 식 9의 설정 N에 대해 섹션 6.1.3에서 제시된 바와 같이 k_N =8,192 bytes로 설정하는 것은 식 11의 부등식이 만족한다는 것을 보증한다. 수신기는 적절이 설정되고 프로그램될 수 있다.

[0280] [286] 우리는 이제 소정의 소스 리퀘스트에 대한 RA 리퀘스트들의 수 n 을 계산하기 위해, 위의 섹션 6.1.3의 프로세스를 본다. 선험적으로, 우리는 슬라이스들을 가능한 한 작게 만들고 싶어하는데, 작은 슬라이스들은 많은 이점들을 제공하기 때문이다; 한 접속이 다른 것들에 비해 느리면, 이는 작은 리퀘스트들로 문제들을 야기하지 않을 것인데, 작은 리퀘스트들은 느린 접속 상에서도 빠르게 완료할 것이기 때문이다. 따라서 작은 슬라이스 설정에서, 느린 접속은 결국 본질적으로 단지 더 적은 리퀘스트들을 서비스하게 될 것이다. 작은 슬라이스들의 다른 이점은 그것들은 RA가 버퍼의 시간상 상대적으로 짧은 섹션에 애쓰도록 하고, 따라서, 그것은 그 노력을 가장 긴급한 작업 영역에 통합하려는 경향이 있다는 점이다.

[0281] [287] 그러나, 슬라이스를 작게 만드는 것은 비용을 치른다: 첫째로, 각 리퀘스트는 업링크 상에 및 다운링크 상에 모두 얼마간의 오버헤드를 포함한다. 둘째로, 한 리퀘스트를 완료한 뒤, 접속들은 약 RTT 동안 아이들 상태에 있을 것이다. 따라서, 리퀘스트 분할 프로세스는 이상적으로 업링크 트래픽을 과도하게 유발하지도 않고 각 가용 링크의 용량을 충분히 이용한다는 조건하에 가능한 한 작은 청크들을 선택하도록 해야 한다. 따라서 바람직한 특성들은:

[0282] [288] 1. D_{min} 실시간 당 접속당 많아도 하나의 리퀘스트를 목표로 한다. 이는 업링크 트래픽이 가장 나쁜

경우에 N_{target}에 비례하는 값에 의해 제한되도록 한다.

[0283] [289] 2. $c_s \cdot RTT$ 마다 접속 당 많아도 하나의 리퀘스트를 목표로 한다. 이는 접속의 활성 시간을 적어도 약 c_s /(c_s + 1), 즉 중도적 c_s 에 대해 1에 가깝도록 한다.

[0284] *[290]* D_{min}의 좋은 선택은 사용 실정에 따른다. 그것을 원하는 대략 양단간 지연으로(그러나 그것보다 작은) 선택하는 것은 보통 프래그먼트의 일반적 지속시간이다. 양단간 지연이 크다면, 더 큰 버퍼들이 사용될 수 있고, 더 큰 슬라이스들의 나쁜 영향이 더 작다. 반면에, 짧은 양단간 지연에 대해, 버퍼들이 작고, 따라서 슬라이스들은 스톨들을 유발하는 느린 접속들을 회피하기 위해 작아야 한다. 그 시나리오에서, 더 작은 리퀘스트의 더 높은 비용은 버퍼 레벨에서 얻어진 안정성의 가치가 된다.

[0285] [291] 사용되는 파라미터들은 클라이언트로 스트리밍되는 미디어의 특성들의 요약인, MPD(Media Presentation Description)의 프로파일 표시자에 따라 튜닝될 수 있다. 모든 미디어 세그먼트를 다운로드하고 그것들을 최종 사용자에게 보여주는 대신, 클라이언트는 MPD 내의 프로파일로부터의 상이한 사용 실정들에 기초하여 세그먼트들을 "스킵" 할 것을 선택할 수 있다.

[0286] *[292]* c_s 선택의 하한은 다음과 같이 고안될 수 있다. 열린 N 접속들이 있고, RA가 활성이면, 평균적으로 대략 $N \cdot c_s$ /(c_s + 1)의 활성 접속들이 있을 것이다. 모든 N 접속들의 수신 윈도우들이 총합 목표 레이트를 지속하기에 총합적으로 충분히 크다는 것을 보증하기 위해, $c_w \cdot c_s$ /(c_s + 1) 가 적어도 1 인 것이 바람직하다.

[0287] [293] 이 한도는 보수적이다. 활성 접속들의 추정된 수 $N \cdot c_s$ /(c_s + 1)는 단지 평균이고, 다소간의 편차가 있을 것으로 보임에도, 편차들을 고려하지 않는다. 실제로, c_s 를 위의 한도에 의해 제안된 값들의 2에서 3배로 하는 것이 유리하고, 예를 들어, c_w = 3이고, c_s = 6이면, $c_w \cdot c_s$ /(c_s + 1)는 적어도 2.5이다.

[0288] 6.2 순방향 에러 정정을 갖는 RA

[0289] [294] 데이터가 여러 TCP 접속들을 통해 수신되면, 그들은 때때로 일시적으로 다른 다운로드 레이트들을 갖는다. 프래그먼트의 리퀘스트가 여러 서브리퀘스트들로 분할되면, 전체 프래그먼트는 오로지 마지막 서브리퀘스트 응답(청크)가 수신되는 때 수신된다. 프래그먼트가 긴급하게 수신될 필요가 있는 때에, 이는 문제가 되는데, 서브리퀘스트들 중 하나는 느린 접속 상에 처리되어 프래그먼트가 빠르게 수신되는 것을 방해할 수 있기 때문이다.

[0290] [295] 콘텐츠 제공자는, 비디오 데이터에 더하여, 클라이언트가 원래 프래그먼트를 재건하는 것을 돕기 위해 페칭할 수 있는, 각 프래그먼트에 대하여 추가적인 순방향 에러 정정(forward error correction; "FEC") 수선 데이터를 제공할 수 있다. 예를 들어, 클라이언트가 4 접속들을 갖고 4000 bytes 크기의 프래그먼트를 긴급하게 수신할 필요가 있다고 하자. 그 리퀘스트 액셀러레이터는 프래그먼트를 각각 1000bytes의 4 범위로 분할하고 4 접속들 각각에 하나의 리퀘스트를 이슈할 것이다. 접속 1은 빠르고, 접속 4는 중도적으로 빠르지만, 제 2 및 제 3 접속들은 매우 느릴 지도 모른다. 따라서, 전체 다운로드 레이트가 원칙적으로 전체 프래그먼트를 적시에 다운로드하기에 충분히 높다 하더라도, 접속들 2 및 3이 막혀있으므로 그것은 단지 아주 느리게 도착할 수 있다.

[0291] [296] 이 문제를 피하기 위해, 클라이언트는 그것이 그 자신의 서브리퀘스트를 처리하자 마자, 접속 1을 사용하여 접속 2 또는 3 하려는 것과 동일한 데이터를 페칭하려고 할 수 있다. 이는 도움이 될 수 있지만, RA는 어느 접속이 더 도움이 필요한지; 그것이 접속 2 인지 3인지 결정해야 한다. 그것이 틀린 예견을 하면, 그것은 필요 없이 중복 데이터를 다운로드하는 것이고, 프래그먼트는 여전히 적시에 도착하지 않을 수 있다.

[0292] [297] 대신에 더 양호한 리퀘스트 액셀러레이터는 접속 1 을 사용하여 얼마간의 수선 데이터를 페칭할 수 있다. 수선 (즉, FEC 코딩된) 데이터는, 성공적으로 다운로드되면, 리퀘스트 2 또는 3으로부터의 데이터가 유실되었는지 여부에 부관하게 유실 데이터를 재건하는데 사용될 수 있다. 유일한 제약은 수신된 데이터의 양이 프래그먼트를 재건하는데 충분하다는 것이다. 다른 말로, 우리 예에서, 수선 바이트들의 수 더하기 수신된 프래그먼트 바이트들의 수가 4000 이상이어야 한다.

[0293] [298] 일 구현에서, 콘텐츠 제공자는 코딩된 비디오 세그먼트들에 대한 FEC 수선 데이터로의 액세스를 제공한

다. 그것은 원래 비디오 데이터와 유사한 방법으로 수선 데이터를 이용 가능하게 만든다. 예를 들어, 그것은 각 미디어 세그먼트 파일에 대해, 수선 정보를 포함하는 추가 FEC 파일을 제공할 수 있다. 콘텐츠 제공자는 미디어 프리젠테이션 설명의 FEC를 사용하기 위해 필요한 정보 및 파라미터들을 제공할 수 있다. 다른 구현에서, 미디어 프리젠테이션 설명은 FEC에 관한 어떤 정보도 포함하지 않지만, 클라이언트는 세그먼트 URL로부터 FEC 수선 URL의 이름을 어떻게 추론하는지에 대한 룰과 같은, 공통 협약을 이용하여 그것에 액세스할 수 있다.

[0294] [299] 클라이언트 구현은 어떻게 및 언제 수선 데이터를 리퀘스트하는지에 관한 프로세스들을 구현한다. 리퀘스트된 수선 데이터의 양은 얼마나 많은 데이터가 아웃스탠딩인지에 의존할 수 있다. 그것은 또한 얼마나 빨리 프래그먼트가 이용될 필요가 있는지에 의존할 수 있다. 예를 들어, 충분한 시간이 남아있다면, 모든 소스 데이터를 적시에 수신하기를 바랄 것이고, 임의의 수선을 요청하는 것이 아마 불필요하다. 반면에, 프래그먼트가 긴급하게 되고 있으면, 스톨이 임박하여 클라이언트가 그 프래그먼트에 대해 적시에 충분한 데이터를 얻을 수 없기 때문에, 많은 수선 데이터를 리퀘스트하길 원할 것이다. 따라서, 일 구현은 리퀘스트된 수선 데이터의 양을 $\beta(B)S$로 설정할 수 있고, 여기서, S는 아웃스탠딩 소스데이터의 양이고, $\beta(B)$는 버퍼 레벨의 감소하는 함수이다.

[0295] [300] 다른 구현은 아웃스탠딩 데이터의 양을 전체 아웃스탠딩 양 보다는, 가장 불완전한 리퀘스트의 아웃스탠딩 데이터의 양에 비례하는 아웃스탠딩 데이터의 양으로 정할 수 있다.

[0296] 6.2.1 수선 세그먼트 생성기의 실시예

[0297] [301] 어떻게 DASH 표준이 FEC, 특히 FEC에 대한 RaptorQ 를 사용하는지에 관련될 수 있는 아래의 모든 계산들은 바람직하게는 고정-소수점/정수 연산을 사용하여 수행된다. 이는 리프리젠테이션의 프래그먼트 내에 소스 심볼들의 수 및 위치들을 계산하는 것을 포함하고, 수선 세그먼트 내의 프래그먼트에 대한 수선 심볼들의 수 및 위치들을 계산하는 것은 고정-소수점 연산을 이용하여 수행되어야 한다. 이는 수신된 FEC 수선 프래그먼트들 및 소스 프래그먼트들의 조합들을 사용하여 소스 프래그먼트들을 디코딩하는 RA 프로세스와 정확한 같은 결과가 소스 세그먼트들로부터 FEC 수선 프래그먼트들을 생성하는 인제스쳔(ingestion) 프로세스에 의해 달성될 필요가 있고, 따라서 이 계산들은 정확히 동일한 결과를 가져야 하기 때문이다.

[0298] [302] 고정-소수점 연산 대신에 유동-소수점 계산들을 사용하는 것은, 상이한 플랫폼들 상에 상이한 유동-소수점 구현들의 상이한 코너 케이스 동작으로 인해, 가끔 찾아내기 힘든 미묘한 버기(buggy) 동작을 낳을 수 있고, 양 끝-점들이 정확히 같은 계산 결과를 낳아야 하는 표준에서 받아들일 수 없을 것이다.

[0299] [303] 수선 세그먼트들은 sidx 테이블들을 포함하는 이미 처리된 소스 세그먼트들에 기초하는 별개의 프로세스에서 생성될 수 있다. 소스 세그먼트들 자체에 추가하여, 프로세스로의 두 입력들은 수선 비율 R 및 심볼 크기 S이다. 세그먼트 내의 수선 프래그먼트의 수선 심볼들의 수 및 위치들의 계산을 위한 고정소수점 연산을 사용하는 것을 용이하게 하기 위해, R 값은 mille 단위로 표현될 수 있고, 즉 R = 500은 비율이 1/2임을 의미한다.

[0300] [304] 각 세그먼트 내에, 소스 세그먼트의 시작에, 세그먼트 인덱싱 정보가 있고, 이는 시간/바이트-오프셋 세그먼트 맵을 포함한다. 시간/바이트-오프셋 세그먼트 맵은 시간/바이트-오프셋 쌍들 ($T(0)$, $B(0)$), ($T(1)$, $B(1)$) …,($T(i)$, $B(i)$),…,($T(n)$, $B(n)$)의 리스트이고, 여기서 $T(i-1)$은 모든 미디어 세그먼트들 중에 미디어의 초기시작 시간에 관련된 미디어의 i 번째 프래그먼트의 플레이백을 위한 세그먼트 내의 시작시간을 나타내고, $T(i)$는 i 번째 프래그먼트에 대한 끝 시간(따라서 다음 프래그먼트에 대한 시작 시간)을 나타내고, 바이트-오프셋 $B(i-1)$은 소스 세그먼트의 시작과 관련된 미디어의 i 번째 프래그먼트가 시작하는 이 소스 세그먼트 내의 데이터의 시작의 대응하는 바이트 인덱스이고, $B(i)$는 i 번째 프래그먼트 까지 그리고 그것을 포함하는 세그먼트의 대응하는 바이트 수이다 (따라서 $B(i)$)는 프래그먼트 i+1의 제 1 바이트의 인덱스이다). 세그먼트가 복수의 미디어 컴포넌트들을 포함한다면, $T(i)$ 및 $B(i)$는 절대적 방법으로 세그먼트의 각 컴포넌트에 제공될 수 있고, 또는 그것들은 기준 미디어 컴포넌트를 서빙하는 다른 미디어 컴포넌트에 관련하여 표시될 수 있다. 어떤 경우에든, $B(0)$는 세그먼트의 제 1 프래그먼트의 시작 바이트 인덱스이고, 이는 세그먼트의 제 1 프래그먼트에 선행하는 sidx 정보로 인해 0보다 클 수 있다. $B(0)$이 0이 아니면, sidx에 대응하는 수선 세그먼트의 시작 시에 어떤 수선 심볼들이 있다. 구현에 따라, 이 첫 번째 수선 심볼들은 제 1 프래그먼트의 시작까지 세그먼트의 데이터를 보호할 수 있고, 또는 그것들은 사용되지 않는 페디드-제로(padded-zero) 데이터 바이트들일 수 있다.

[0301] [305] 수선 비율 R은 수선 세그먼트 메타데이터와 함께 MPD에 시그널링되거나, 다른 수단들(TBD)에 의해 획득될 수 있다. R 에 대한 값의 예로, R =500이면 수선 세그먼트 크기는 그로부터 생성되는 소스 세그먼트의 대응 크기의 0.5배로 (아주 밀접하게) 근사하고, 소스 세그먼트 내의 소스 프래그먼트에 대응하는 수선 세그먼트의수선 프래그먼트의 크기는 또한 소스 세그먼트의 크기의 0.5 배에 (아주 헐렁하게) 근사한다. 예를 들어, 소스 세그먼트가 1,000 kilobytes 데이터를 포함하면, 대응하는 수선 세그먼트는 대략 500 kilobytes 수선 데이터를 포함한다.

[0302] [306] S 값은 또한 수선 세그먼트 데이터와 함께 MPD에 시그널링되거나, 다른 수단들에 의해 획득될 수 있다. 예를 들어, S = 64는 소스 데이터 및 수선 데이터가 FEC 인코밍 및 디코딩을 위해 각각 64 바이트 크기의 심볼들을 포함한다는 것을 표시한다. S 값은 연관된 소스 세그먼트의 리프리젠테이션의 스트리밍 레이트에 비례하도록 선택될 수 있다. 예를 들어, 스트리밍 레이트가 100 kbps이면, S=12bytes가 적절할 수 있고, 반면에, 스트리밍 레이트가 1 Mbps 이면 S = 120bytes가 적절할 수 있고, 스트리밍 레이트가 10 Mbps 이면 S = 1,200bytes가 적절할 수 있다. 하나의 목표는 어떻게 그레널러(granular) 프래그먼트들이 심볼들로 분할되는지와 스트리밍 레이트에 대비한 FEC 디코딩에 대한 프로세싱 조건들 간의 양호한 트레이드-오프를 가진다는 것일 수 있다. 예를 들어, 1Mbps의 스트리밍 레이트, 그리고 500ms 크기 근처의 프래그먼트들에서, 각 프래그먼트는 64 KB 근처의 데이터이고, S=120이면 프래그먼트는 대략 500 소스 심볼들로 구성되고, 이는 각 심볼은 0.2% 근처의 데이터가 소스 블록들을 복원하는데 필요하고, 이는 심볼 그래널러리티(granularity)로 인해 필요한 여분 수신은 프래그먼트가 수신되고 있는 HTTP 접속들의 수의 0.2%배로 상한 된다는 것을 의미한다. 예를 들어, HTTP 접속들의 수가 6이면, 심볼 그래널러리티 수신 오버해드는 1.2%로 제한된다.

[0303] [307] 수신 세그먼트는 다음과 같이 소스 세그먼트를 위해 생성될 수 있다. 소스 세그먼트의 각 프래그먼트는 FEC 인코딩 목적을 위한 소스 블록으로 여겨지고, 따라서 각 프래그먼트는 그로부터 수선 심볼들이 생성되는 소스 블록의 소스 심볼들의 시퀀스로 다루어진다. 제 1 i 프래그먼트에 대해 생성된 전체 수선 심볼들의 수는 TNRS(i) = divceil(R*B(i),S*1000)으로 계산되고, 여기서, divceil(I,J)는 적어도 Jfh 나누어진 I인 값을 갖는 가장 작은 정수를 출력하는 함수이고, 즉 divxeil(I, J) = (I+J-1) div J이고, 여기서 div는 결과가 가장 가까운 정수로 내림되는 고정-소수점 나눔이다. 따라서, 프래그먼트 i에 대해 생성된 수선 심볼들의 수는 NRS(i) = TNRS(i) - TNRS(i-1) 이다.

[0304] [308] 수선 세그먼트는 프래그먼트들에 대한 수선 심볼들의 연결(concatenation)을 포함하고, 수선 세그먼트 내의 수선 심볼들의 순서는 그로부터 그것들이 생성되는 프래그먼트들의 순서이고, 프래그먼트 내에 수선 심볼들은 그들의 인코딩 심볼 식별자(ESI”)의 순서이다.

[0305] [309] 전술한 바와 같이 프래그먼트에 대한 수선 심볼들의 수를 정의함으로써 모든 이전 프래그먼트들에 대한 수선 심볼들의 전체 수, 따라서 수선 프래그먼트 i의 심볼들에 대한 바이트 인덱스 및 바이트 범위는 오로지 R, S, B(I-1) 및 B(i)에 의존하고, 소스 세그먼트 내의 프래그먼트들의 이전 또는 후속 구조 어느 것에도 의존하지 않는다. 이것은 그로부터 수선 블록이 생성되는 소스 세그먼트의 대응 프래그먼트의 구조에 관한 로컬 정보만을 이용하여, 그것이 클라이언트가 수선 세그먼트 내의 수선 블록의 시작 포인트를 빠르게 계산하고, 또한 그 수선 블록 내에 수선 심볼들의 수를 빠르게 계산할 수 있도록 하기 때문에 유익하다. 따라서, 클라이언트가 소스 세그먼트의 중간부터 프래그먼트를 다운로드 및 플레이백을 시작한다고 결정하면, 그것은 또한 대응하는 수선 세그먼트 내로부터 프래그먼트에 대응하는 대응 수선 블록을 빠르게 생성하고 액세스할 수 있다.

[0306] [310] 프래그먼트 i에 대응하는 소스 블록의 소스 심볼들의 수는 NSS(i) = divceil(B(i)-B(i -1),S)로 계산된다. B(i)-B(i -1)가 S의 배수가 아니면 FEC 인코딩 및 디코딩의 목적을 위해 마지막 소소 심볼은 i는 0 바이트들로 패딩 아웃되고, 즉, 마지막 심볼이 0 바이트들로 패딩 아웃되어 FEC 인코딩 및 디코딩 목적으로 그 크기가 S 바이트가 되도록 하나, 이 0 패딩 바이트들은 소스 세그먼트의 부분으로 저장되지 않는다. 이 실시예에서, 소스 심볼에 대한 ESI들은 0, 1, …, NSS(i)-1이고, 수선 심볼들에 대한 ESI들은 NSS(i), …, NSS(i)+NRS(i)-1이다.

[0307] [311] 이 실시예에서 수선 세그먼트에 대한 URL은 예를 들어 소스 세그먼트의 URL에 서픽스(suffix) “.repair”를 단순히 추가함으로서 대응 소스 세그먼트에 대한 URL로부터 생성될 수 있다.

[0308] [312] 수선 세그먼트는 또한, 예를 들어 마지막에 첨부된, 대응 소스 세그먼트의 부분일 수 있다. 결합된 세그먼트의 구조는 또한 소스 프래그먼트들 및 수선 프래그먼트들이 결합된 세그먼트 내에 연속한다는 것일 수 있고, 즉 결합된 세그먼트는 제 1 소스 프래그먼트, 뒤따르는 제 1 수선 프래그먼트, 뒤따르는 제 2 소스 프래그먼트, 뒤따르는 제 2 수선 프래그먼트 등을 포함한다. 당업자는, 전술한 방법들 및 프로세스들이 그러한 결합

세그먼트들에 적용하도록 쉽게 채용될 수 있음을 알 것이다.

[0309] 6.2.2 수선 세그먼트들을 이용하는 리퀘스트 액셀러레이터의 실시예

[0310] [313] 수선 세그먼트에 대한 수선 인덱싱 정보 및 FEC 정보는 대응 소스 세그먼트에 대한 인덱싱 정보에 의해, 그리고 R 및 S 값으로부터 내재적으로 정의되고, 여기서 R은 mille 당 표시하는 0 및 1000 사이의 정수로 표현되고, S 는 바이트들로 표현된다. 수선 세그먼트를 포함하는 프래그먼트 구조 및 시간 오프셋들은 대응하는 소스 세그먼트의 시간 오프셋들 및 구조에 의해 정의된다. 프래그먼트 i에 대응하는 수선 세그먼트의 수선 심볼들의 시작 및 끝에 대한 바이트 오프셋은 RB(i-1) = S*divceil(R*B(i-1), S* 1000) 및 RB(i) = S*divceil(R*B(i), S* 1000) 으로 각각 계산된다. 프래그먼트 i에 대응하는 수선 세그먼트의 바이트들의 수는 RB(i) - RB(i-1)이고, 따라서 프래그먼트 i에 대응하는 수선 심볼들의 수는 NRS(i) = (RB(i) - RB(i-1))/S으로 계산된다. (분자가 S의 배수임이 보증되므로 여기서 divceil 연산의 필요가 없으나, divceil가 여기서 사용될 수 있고 결과는 여전히 정확할 것임에 주의하라.) 프래그먼트 i에 대응하는 소스 심볼들의 수는 NSS(i) = divceil(B(i)-B(i-1), S)로 계산될 수 있고, 여기서 마지막 소스 심볼은 인코딩에 대해 설명한 바와 동일하게, 디코딩 목적으로 필요하다면 0들로 패딩된다. 따라서, 수선 세그먼트 내 수선 블록에 대한 수선 인덱싱 정보 및 대응 FEC 정보는 대응 소스 세그먼트의 대응 프래그먼트에 대한 R, S 및 인덱싱 정보로부터 내재적으로 유도될 수 있다.

[0311] [314] 일 예로서, 바이트 오프셋 B(1) = 6,410 에서 시작하고, 바이트 오프셋 B(2) = 6,770 에서 끝나는 프래그먼트 2를 나타내는, 도 35에 도시된 예를 고려하고, 즉 프래그먼트 2 는 6,770-6,410 bytes 크기이고 6,770은 프래그먼트 3의 시작 바이트 인덱스이다. 이 예에서, 심볼 크기는 S = 64 bytes이고, 점선의 수직 선들은 S의 배수들에 대응하는 소스 세그먼트 내의 바이트 오프셋들을 나타낸다. 소스 세그먼트 크기의 비율로서의 전체 수선 세그먼트 크기는 이 예에서 mille 당 R = 500으로 설정된다(수선은 대략 소스의 1/2이다). 프래그먼트 2에 대한 소스 블록의 소스 심볼들의 수는 NSS(2) = divceil(6,770-6,410, 64) = (6,770- 6,410+64-1) div 64 = 6으로 계산되고, 이 6 소스 심볼들은 ESI들 0, 5을 각각 갖고, 여기서 제 1 소스 심볼은 소스 세그먼트 내의 바이트 인덱스 6,410에서 시작하는 프래그먼트 2의 제 1 64 bytes이고, 제 2 소스 심볼은 소스 세그먼트 내의 바이트 인덱스 6,474에서 시작하는 프래그먼트 2의 다음 64bytes 등등이다. 프래그먼트 2에 대응하는 수선 블록의 끝 바이트 오프셋은 RB(2) = 64 * divceil(500*6,770, 64* 1,000) = 64 * (3,385,000 + 64,000 - 1) div 64,000 = 64 * 53 = 3,392으로 계산되고, 프래그먼트 2에 대응하는 수선 블록의 시작 바이트 오프셋은 RB(1) = 64*divceil(500 * 6,410, 64* 1,000) = 64 * (3,205,000 + 64,000 - 1) div 64,000 = 64 * 51 = 3,264으로 계산되고, 따라서, 이 예에서 각각, 수선 세그먼트 내의 바이트 오프셋 3,264에서 시작하고 바이트 오프셋 3,392에서 끝나는, ESI들 6 및 7을 갖는 프래그먼트 2에 대응하는 수선 블록의 두 수선 심볼들이 있다.

[0312] [315] 이것은 도 35에 도시된다. 도 35에 나타난 예에서, R = 500(수선은 대략 소스의 1/2이다)이고 프래그먼트 2에 대응하는 6 소스 심볼들이 있다 하더라도, 수선 심볼들의 수를 계산하기 위해 단순히 소스 심볼들의 수를 사용했다면, 예상하는 바와 같이, 수선 심볼들의 수는 3이 아니고, 그러나 대신에 2인것으로 계산된다. 수선 심볼들의 수를 결정하기 위해 단순히 프래그먼트의 소스 심볼들의 수를 사용하는 것과는 대조적으로, 여기서 행해지는 방법은 단지 대응 소스 세그먼트의 대응 소스 블록과 연관된 인덱스 정보로부터 수선 세그먼트 내의 수선 블록의 위치를 계산할 수 있게 한다. 이것이 인제스천(ingestion) 프로세스에 일치하는 계산이고 RA 프로세스 내에 있기 위해서, 수선 세그먼트 내의 수선 프래그먼트에 대한 수선 심볼들의 수 및 위치들은 고정-소수점 연산을 사용하여 계산된다는 것이 중요하다. 더욱이, 소스 블록의 소스 심볼들의 수, K가 커지면서, 대응 수선 블록의 수선 심볼들의 수, KR은 K * R/1,000에 의해 가깝게 근사하고, 일반적으로 KR은 많아도 divceil(K * R, 1,000) 이고, KR은 적어도 divfloor((K-1) * R, 1000)이고, 여기서 where divfloor(I, J) = I div J 이다.

[0313] 도시된 예들

[0314] [316] 도 25는 레이트 선택 프로세스를 도시한다. λ 및 μ에 대한 설정들이 클수록, 설정들은 더 공격적이다. 도 23은 파라미터 λ에 대한 다른 값들을 도시한다. 도 24는 파라미터 μ에 대한 다른 값들을 도시한다. 하이브리드 설정은 두 주요 메커니즘들에 의해 레이트 동요를 감시시키려 한다. 제 1은 B가 더 클 때 레이트를 증가시키는데 더 주의함에 의하고, 다음은 B가 더 작을 때 현재 레이트에 머무르기 위해 더 애쓰는 것이다.

[0315] *[317]* *pker* x.y: C = x*min(y*Tdl,B) 에 대한 예시적 설정은 x.y가 8.1, 4.2, 2.4, 4.4 또는 다른 x.y 값들로 설정될 수 있다. *pker*의 실제 평균 윈도우는 다운로드 보류 주기의 스킵으로 인해 C보다 길다는 점에 주의하라. EWMA 으로 스킵 없고 다운로드 보류 주기에서의 레이트가 마지막 다운로드 구간의 그것과 동일하다고 가정하라.

[0316] [318] MWA(Moving Window Average)에 대해, $H(z) = (1/D)*((1-z^{D})/(1-z^{-1}))$이고, D는 윈도우 크기이다. $X_i = \min\{R_k : k \geq i\}$이고, 여기서 R_k는 가중치 W_k를 갖는 레이트의 EWMA이RH, $W_1 < W_2 < W_3 <\cdots$이다. EWMA에 대해, $H(z) = ((1-\beta)/(1-\beta z^{-1}))$이고, β는 이전 평균의 가중치이다. MWA 및 EWMA는 어떤 경우들에는 대략 균등하다.

[0317] [319] 적응성 추정기가 긴 평균 윈도우를 갖는다면, 그것은 레이트 스위치 빈도를 감소시키고 동시에 라이브 스트리밍에 대해 대략 동일한 평균 레이트를 유지한다. 다른 시나리오에 대해 다른 설정들이 잘 동작한다. 공격적인 설정들은 보다 안정적인 시나리오들에 대해 잘 동작하고, 반면에 덜 공격적인 설정은 보다 변덕적인 시나리오에 더 적합하다. 대역폭이 상당한 부분의 시간 동안 어떤 마진만큼 가장 높은 리프리젠테이션 레이트보다 더 높다면(예를 들어, 20-sec 평균일 때의 시간의 %가 레이트 캡(cap)보다 더 높다), 보다 공격적인 설정으로 가는 것이 유익하다. 이상적으로, 디바이스는 시나리오 유형들을 검출하고 적절한 설정을 적용할 수 있어야 한다. 시나리오 검출은 라이오 기술 유형, 어던 단위 시간 내의 레이트 변동들의 수, 이동 속도, 등등과 같은 인자들에 기초할 수 있다. 더 간단한 전략은 위 관찰에 기초할 수 있다: “전체” 대역폭이 레이트 캡 보다 높은 때 보다 공격적인 설정을 사용한다.

[0318] 8. 레이트 선택 파라미터들의 설정

[0319] [320] 이 섹션에서는, 레이트 선택 파라미터들을 설정하는 예들이 제공된다.

[0320] [321] MLB에 대해, EFF = 1 - Rv/Rdl이고, Rv는 선택된 리프리젠테이션의 현재 레이트이고 Rdl은 현재 다운로드 레이트이다. 제안된 룰은 다음과 같다:

[0321] [322] EFF < 0이면, 아마 1 레이트보다 더 많이 하락

[0322] [323] 0 <= EFF < 0.1이면, 일 레이트 하락

[0323] [324] 0.1 <= EFF < 0.6이면, 현 레이트에 머묾

[0324] [325] 0.6 <= EFF < 0.8이면, 일 레이트 상승

[0325] [326] 0.8 <= EFF <= 1이면, 아마 일 레이트보다 더 많이 상승

[0326] [327] alpha = Rv/Rdl 이라고 하자. 그러면 이는 대략 다음으로 바뀐다:

[0327] [328] alpha <= 0.4이면, 적어도 일 레이트 상승

[0328] [329] 0.4 < alpha <= 0.9이면, 동일한 레이트에 머묾

[0329] [330] 0.9 < alpha 이면, 적어도 일 레이트 하락

[0330] [331] 이를 DASH클라이언트 레이트 선택 프로세스의 컨텍스트에 넣으면:

[0331] [332] RUP를 UP에 대응하는 리프리젠테이션의 레이트라고 하고, RDOW를 DOWN에 대응하는 리프리젠테이션의 레이트라고 하고, 위와 같이 Rv를 현재 선택된 리프리젠테이션의 레이트라고 하자. RUP는 RUP <= lambda(t)*Rdl이도록 가능한 한 크게 선택하고, RDOWN은 RDOWN <= mu(t)*Rdl이도록 가능한 한 크게 선택된다. 파라미터 t = B/(D+delta)이고, 여기서 B는 미디어 버퍼의 프리젠테이션의 현재 양이고, D는 현재 결정이 되고 있는 포인트를 넘는 다음 가능한 스위치 포인트까지의 시간의 한도이고, delta는 네트워크 레이턴시 및 라운드 트립 시간을 고려하는 작은 파라미터이고, 예를 들어, delta는 근사로서 1 초 또는 2 초로 설정될 수 있고, 또는 delta는 현재 RTT의 측정된 상한에 따라 설정될 수 있다.

[0332] [333] 다음 레이트 RNEXT의 전체 선택은 다음과 같다:

[0333] [334] RUP < Rv이면 RNEXT = min{Rv, RDOWN} 아니면 RNEXT = RUP이다.

[0334] [335] 위 MLB 파라미터들은 모든 t에 대해 lambda(t) = 0.4*R 및 mu(t) = 0.9으로 설정함으로써 근사될수 있고, 여기서 R은 다음 높은 리프리젠테이션의 레이트의 현재 리프리젠테이션의 레이트의 그것에 대한 비율이다. 예를 들어, 현재 레이트가 500 Kbps이고, 다음 높은 레이트는 750 Kbps 이면, R = 1.5이고 따라서 lambda(t) = 0.6이다. 이것은 MLB 프로세스를 다음과 같이 근사한다.

[0335] [336] 결정 포인트에서, EFF >= 0.6, 즉 alpha <= 0.4이면, Rv <= 0.4*Rdl이고, 이 경우에 RUP는 적어도 Rv*R일 것이고(왜냐하면 모든 t에 대해 lambda(t) = 0.4*R이므로) 따라서 RNEXT = RUP, 즉 레이트는 Rv*R의 다음 높은 리프리젠테이션으로 상승할 수 있고, Rdl이 0.4*Rv보다 훨씬 크면 RUP는 Rv*R보다 더 크게 될 것이고(리프리젠테이션 레이트들의 그래널러리티에 기초하여), EFF가 예를 들어 0.8보다 크면 RUP는 Rv*R 넘는 일 레이트보다 클 것이다. EFF < 0.1이면 Rv > 0.9*Rdl이고, 이 경우 RDOWN은 Rv보다 작을 것이고(왜냐하면 RDOWN <= 0.9*Rdl이므로), 그리고 레이트는 하락할 거이고, 즉 RNEXT < Rv이다. EFF가 0.1과 0.6 사이이면 RUP <= Rv*R이고 RDOWN >= Rv이며, 그 경우 RNEXT는 Rv와 동일하게 선택될 것이다.

[0336] 레이트 선택 파라미터 세트들

[0337] [337] 아래 테이블들은 몇몇 가능한 레이트 선택 파라미터 세트들을 특정한다. 아래 테이블에 표시되지 않은 t의 중간 값에 대한 lambda 및 mu의 값들은 주변 값들 사이에 선형적으로 보간함에 의해 계산되어야 한다. 아래 테이블에 표시된 것들을 넘는 t의 값에 대한 lambda 및 mu의 값들은 표시된 t의 최대 값에 대한 lambda 및 mu 값들로 설정되어야 한다.

[0338] [338] 모든 t에 대해 제약들 mu(t) <= t 및 lambda(t) <= t가 만족되면, 이론적으로 플레이백에 스톨이 없을 것이나, 실질적 관점에서 스톨이 전혀 없는 것보다는 플레이백에 작은 스톨을 갖고 그러나 훨신 낮아진 레이트에서 플레이 아웃을 계속하는 것이 바람직할 수 있고, 예를 들어 1 Mbps에서 20 kbps로의 도약은 사이에 1 초 중지를 갖는 1 Mbps에서 250 kbps로의 도약보다 더 나쁜 경험일 수 있다. lambda 및 mu의 최소 값이 도 36의 테이블들에 설정되고, mu(t) > t 및/또는 lambda(t) > t 인 값들에 대해 스톨이 발생할 것임에 주의하라(그럼에도 불구하고 lambda(t) 및 mu(t)의 설정들과 무관하게 버퍼가 이렇게 비는 임의의 경우에 스톨이 발생할 수 있다).

[0339] [339] 이제 설명되는 바와 같이, 클라이언트 디바이스는 HTTP를 통한 적응성 비디오 스트리밍을 위한 레이트 적응 및 다운로드 프로세스들을 제공할 수 있다. 인터넷(및 다른 네트워크들)을 통해 비디오를 스트리밍하는 클라이언트들은 요동치는 대역폭의 문제에 직면한다. 고품질 비디오가 스트리밍되면, 링크는 때때로 충분히 빠르지 않을 수 있고, 이는 플레이어가 중단하고 다시 버퍼링하도록 한다. 다른 경우들에서, 저 품질 비디오는 훨씬 더 작은 대역폭을 하용하나 더 나쁜 사용자 경험이다. 하나의 해결책은 비디오 품질을 적응성 있게 조정하는 것이다: 쓰로우풋이 높을 때 더 높은 품질을 선택하고, 자동적으로 아래로 스위칭한다.

[0340] [340] 그러나, 적응성 비디오 스트리밍은 다수의 도전들을 일으킨다: (1) 비디오 레이트(품질)을 선택하기 위한 프로세스 또는 알고리즘은 레이트 증가들은 물론 레이트 감소들에 적응하도록 충분히 빠르게 행동해야 한다. 동시에 그것은 너무 이르거나 이상한 결정들을 피하고 불필요한 레이트 스위칭 결정들을 피해야 한다. 클라이언트는 높은 비디오 품질이 획득될 수 있도록 충분히 높은 레이트에서 데이터를 페칭하는 것을 목표해야 한다. 동시에 다운로드 프로세스는 데이터가 적시에 수신된다는 것을 보증해야 한다. 각 프레임은 그것이 플레이 아웃되기 전에 전체가 수신되어야 한다. 그것들은 불필요하게 큰 플레이백 버퍼를 요구함 없이 이 목표들을 달성할 수 있어야 한다. 큰 버퍼들의 몇몇 문제들은 라이브 이벤트들에 대해, 버퍼의 비디오 양이 목표 양단간 레이턴시에 의해 제한된다는 것이고, 이는 이들 경우에서 가능한 플레이백 버퍼를 심하게 제한한다. 또한, 큰 버퍼에 의존하는 것은 버퍼가 미리 채워질 필요가 있기 때문에 플레이백 시작들 및 검색들에서 바람직하지 않은 지연들을 유발한다. 또한, 큰 플레이백 버퍼는 많은 메모리를 사용하고, 그것은 모바일 폰들 및 다른 클라이언트 디바이스들에 충분하지 않다.

[0341] [341] 이들 문제점들을 해결하기 위한 신 레이트 변화들에 빠르게 반응할 레이트 추정을 위한 프로세스. 레이트 추정은, 특히 스트리밍 비디오에서의 사용에 잘 맞는, 적응성 윈도우드(windowed) 평균이다. 레이트 추정기는 윈도윙 너비를 크게 유지하면서(따라서 측정 편차를 크게) 필요하면 레이트가 충분히 빠르게 적응함을 보증할 수 있도록 하는 방법으로 비디오 버퍼 레벨 및 비디오 버퍼 레벨에서의 변화를 고려한다. 프로세스에 의해 제공된 보증은 (a) B가 레이트 하락이 발생하는 때 버퍼의 비디오 데이터의 양(플레이백 시간의 초 단위)이면, 추정기는 버퍼가 B/2로 비는데 걸리는 시간 내에 그 레이트 추정을 조정할 것이고, (b) B가 레이트 증가가

일어나는 동안 버퍼의 데이터 양이면, 레이트 추정기는 새 레이트로 충분히 빠르게 조정하여 그것이 원칙적으로 많아도 3*B 내에 나타날 수 있도록 한다(스마트 레이트 변경 프로세스라고 가정하면).

[0342] [342] 레이트 결정 프로세스는 (a) 버퍼가 낮은 레벨에 있을 때, 버퍼가 채워지도록 레이트 결정들을 할 수 있고, 심지어 낮은 다운로드 레이트 추정들이 관측되더라도, 비정상적으로 레이트들을 변경하는 것을 피하도록 버퍼를 이용하고, (c) 안정된 레이트 시나리오에서, 올바른 안정된 레이트를 빠르게 선택한다. (a) 정확한 추정을 고려하고, (b) 네트워크 지연들 및 패킷 손실 레이트들이 높더라도 링크 용량을 달성할 수 있고, (c) 스트림의 적시 전달을 달성하는 HTTP를 위해 멀티미디어 다운로드 전략들이 사용된다. 이를 달성하기 위해, 복수의 HTTP 접속들을 사용하고, 네트워크 상태에 따라 미디어 리퀘스트들을 더 작은 청크 리퀘스트들로 분해하고, TCP 흐름제어 메커니즘들을 사용하여 접속들을 동기화시키고, 버스트들(bursts)의 데이터를 리퀘스트할 수 있다. 우리는 또한 접속을 busy로 유지하기 위해 HTTP 파이프라이닝(pipelining) 프로세스를 사용할 수 있다.

[0343] [343] 많은 특징들, 측면들 및 상세한 설명들이 지금까지 설명되었다. 설명된 바와 같이, 다양한 실시예들에서 방법 단계들은 당업자에게 자명한 바와 같이 대응하는 프로그래밍된 엘리먼트들, 프로세서에 제공되는 명령들, 하드웨어 또는 다른 장치들에 의해 수행될 수 있다. 마찬가지로, 엘리먼트들은 프로세스들 또는 프로그램 엘리먼트들에 의해 기능할 수도 있다. 일 실시예의 엘리먼트들의 구조는 단순히 프로세서에 의해 실행되는 명령들의 세트를 포함할 수 있으나 여기에서 대응하는 방법 단계로서 설명되었다.

[0344] [344] 다양한 실시예들에서, 다운로드 레이트 변속(acceleration)은 사용될 수도 아닐 수도 있다. 다운로드 레이트 변속의 일 예는 TCP 접속들을 통한 HTTP 리퀘스트들을 사용하여 다운로드들을 변속하는 방법 또는 장치이다. TCP 접속은 특정 윈도우 크기를 갖고 TCP 접속의 끝들의 노드들은 윈도우 크기에 대한 설정들을 변경할 수 있다. 그 크기가 목표 다운로드 레이트의 함수인 연속하는 HTTP 리퀘스트들에 대한 윈도우 크기를 설정하는 것은 새롭다(on novelty). 따라서, 목표 다운로드 레이트가 변함에 따라, TCP 윈도우 크기가 변할 수 있다.

[0345] [345] 일 실시예에서, 네트워크 경로에 의해 커플링된 소스 및 수신기 사이의 네트워크 경로를 통한 데이터 다운로드를 제어하기 위한 방법 및/또는 장치 또는 컴퓨터 판독가능 매체가 사용되고, 그 방법은, 소스 및 수신기 사이의 복수의 TCP 접속들 각각에 대해, 그 TCP 접속에 대한 TCP 수신기 윈도우 크기를 결정하고, 소스 및 수신기 사이의 TCP접속은 직접 접속 또는 간접 접속일 수 있고, 미디어 콘텐츠에 대한 목표 다운로드 레이트를 결정하고, 목표 다운로드 레이트는 적어도 두 연속되는 HTTP 리퀘스트들에 대한 적어도 두 값들 사이에서 변하고, 복수의 TCP 접속들의 각 TCP 접속을 사용하여 다운로드될 미디어 콘텐츠의 미디어 데이터 엘리먼트들을 다운로드하는 것을 포함하고, 미디어 콘텐츠는 복수의 HTTP 리퀘스트들의 응답의 부분 또는 전체이고, 소정의 TCP 접속에 대한 결정된 TCP 수신기 윈도우 크기는, 적어도 부분적으로, 목표 다운로드 레이트에 기초하여 결정되고, 결정된 TCP 수신기 윈도우 크기는 적어도 두 연속되는 HTTP 리퀘스트들에 대한 적어도 두 값들 사이에서 변한다.

[0346] [346] 현재 TCP 접속에 대한 결정된 TCP 수신기 윈도우 크기는, 적어도 부분적으로, 곱셈 레이트로 곱해진 현재 TCP 접속에 대한 현재 추정된 라운드-트립 시간(“ERTT”)의 곱에 기초하여 결정되고, 곱셈 레이트는 현재 TCP 접속에 대한 목표 다운로드 레이트 및 목표 다운로드 레이트보다 소정의 양만큼 더 높은 레이트에 의해 한정된 범위 내이다. 현재 ERTT는 1초, 10초, 50초 등과 같이 바로 이전의 측정 주기 동안의 최소 관측 RTT의 측정에 의해 결정될 수 있다. 현재 ERTT는 정지 중의 주기의 끝에서의 측정에 의해 결정될 수 있고, 정지중의 기간은 다운로드 주기 다음에 오고 미리-결정된 지속시간 주기 동안 TCP 접속들을 통해 어떤 활성 HTTP 리퀘스트들도 제시되지 않은 기간이다. 목표 다운로드 레이트는 현재 총합의 다운로드 레이트의 두 배 또는 세 배와 같이, 사용된 TCP 접속들의 수로 나누어진, 사용된 모든 TCP 접속들을 통한 현재 집합한 다운로드 레이트에 비례할 수 있다. 목표 다운로드 레이트는 미디어 콘텐츠의 플레이백 레이트에 비례할 수 있고, 플레이백 레이트는 사용된 TCP 접속들의 수로 나누어진 사용된 모든 TCP 접속들에 걸친 집합을 통한 레이트이다. 각 미디어 데이터 엘리먼트는 소정 범위의 편차 내의 크기들을 갖는 다수의 청크들로 분할 될 수 있고, 그러한 청크들의 수는 사용된 TCP 접속들의 수에 기초한다. 그러한 청크들의 n는 현재 TCP접속들에 대한 평가된 라운드-트립 시간(“ERTT”), 현재 다운로드 레이트, 및/또는 리퀘스트되는 미디어 프래그먼트의 크기 중 적어도 하나에 더 기초할 수 있다. 미리 결정된 범위의 편차는 0일 수 있고 따라서 각 청크는 프래그먼트 리퀘스트 당 같은 크기를 가지며, 청크의 수는 사용된 TCP 접속들의 수 곱하기 미리 결정된 인자이다. 후속하는 미디어 데이터 엘리먼트에 대한 더 나중의 HTTP 리퀘스트는 제 1 가용 TCP 접속으로 할당될 수 있다.

[0347] [347] 제어는 소스 및 수신기 사이에 사용할 TCP 접속들의 수를 결정하고, 그 수는 1 보다 크고, 사용할 TCP 접속들의 수는, 적어도 부분적으로, 결정된 적어도 하나의 네트워크 상태에 기초하여 결정되고, TCP 접속들의

수 각각을 이용하여 다운로드될 미디어 콘텐츠의 복수의 미디어 데이터 엘리먼트들을 다운로드하는 것을 또한 포함할 수 있고, 미디어 콘텐츠는 복수의 HTTP 리퀘스트들에 대한 응답의 부분 또는 전체이다. 사용된 TCP 접속들의 수는 TCP 접속들에 대한 추정된 라운드-트립 시간("ERTT"), 목표 다운로드 레이트 및 손실 레이트의 추정에 기초할 수 있다. 손실 레이트는 1% 또는 0.1%로 추정될 수 있다. 사용할 TCT 접속들의 수는 (a) 목표 다운로드 레이트, (b) ERTT, (c) 추정된 손실 레이트의 제곱근의 곱을 포함하거나, 및/또는 그에 비례하는, 2 및 16 사이일 수 있다. 각각의 TCP 접속들에 대해, TCP 수신기 윈도우 크기는 그 TCP 접속에 대해 목표 다운로드 레이트에 기초하여 결정될 수 있고, 결정된 TCP 수신기 윈도우 크기는 적어도 두 연속하는 HTTP 리퀘스트들에 대한 적어도 두 값들 사이에서 변한다.

[0348] [348] 일 실시예에서, 프리젠테이션 버퍼를 고려하고 버퍼가 얼마나 큰지/찼는지/비었는지, 즉 그 레벨이 어떤가에 기초하여 다운로드 레이트를 추정하는, 다운로드 추정을 위한 방법 및/또는 장치 또는 컴퓨터 판독가능 매체가 사용된다. 예를 들어, 유한대역폭을 갖는 네트워크 경로에 의해 데이터 소스들로 커플링되는 리시버에서 다운로드 레이트를 추정하는 것으로서, 다운로드 레이트는 데이터가 리시버에서 네트워크 경로를 통해 수신될 수 있는 레이트인 것은, 리시버의 프리젠테이션 버퍼를 모니터링하고, 프리젠테이션 버퍼는 적어도 미디어 데이터가 수신되는 시간 및 미디어 데이터가 리시버와 연관된 프리젠테이션 엘리먼트에 의해 소비되는 시간 사이에 미디어 데이터를 저장하고, 다운로드 레이트의 추정이 기초할 0이 아닌 추정 주기를 결정하고, 추정 주기 동안 버퍼 레벨들의 표시들을 저장하고, 소정 시간에서의 버퍼 레벨은 적어도 대략적으로, 얼마나 많은 프리젠테이션이 그 시간에, 수신되었지만 아직 프리젠테이션 엘리먼트에 의해 소비되지 않은 미디어 데이터에 의해 차지되었는지에 대응하고, 저장된 표시자를 추정된 다운로드 레이트의 측정의 부분으로서 사용하는 것을 포함할 수 있다.

[0349] [349] 프리젠테이션 엘리먼트는 디스플레이 및 오디오 출력을 포함할 수 있다. 추정 주기는 측정된 버퍼 레벨에, 미리 결정된 비례 인수로, 비례하는 지속시간을 가질 수 있다. 추정 주기의 지속시간은 측정 시간에 프리젠테이션 버퍼의 소비되지 않은 미디어 데이터의 바이트 수에 비례하고, 및/또는 미디어가 프리젠테이션 버퍼에 추가되는 추가 레이트의 함수이고, 및/또는 프리젠테이션 버퍼의 미리 결정된 부분을 다운로드하는데 사용되는 시간에 비례하도록 할 수 있다. 미리결정된 지속 시간은 프리젠테이션 버퍼의 콘텐츠의 미리 결정된 부분이 다운로드된 지속 시간에 대응할 수 있다. 추정 주기는 프리젠테이션 버퍼의 콘텐츠의 미리결정된 부분이 다운로드된 시간 및 프리젠테이션 버퍼에 있는 미디어 데이터의 프리젠테이션 시간 중 더 작은 것일 수 있다.

[0350] [350] 일 실시예에서, 플레이백 레이트 선택을 위한 방법 및/또는 장치 또는 컴퓨터 판독 가능 매체가 사용되고, 플레이백 레이트는 미디어가 프리젠테이션 버퍼로부터 소비되는 레이트이고, megabits/second 와 같이, 메모리 단위/시간으로 측정된다. 수신기가 어떤 미디어에 대해 리퀘스트하는 경우, 그 미디어에 대한 플레이백 레이트가 있다. 자주, 그러나 아마 항상은 아니고, 더 높은 품질의 미디어는 더 높은 플레이백 레이트를 갖고 따라서 트레이드-오프를 보인다. 어떤 플레이백 레이트를 사용/리퀘스트 할지는, 적어도 가끔, 얼마나 많은 미디어가 프리젠테이션 버퍼에 있는지의 함수이다. 수신기는 수신기의 프리젠테이션 엘리먼트를 이용하여 플레이 아웃 할 미디어를 수신할 수 있고, 플레이 아웃은 미디어가 프리젠테이션 버퍼로부터 플레이백 레이트로 소비되는 것을 초래하고, 수신기는 복수의 플레이백 레이트들로부터 선택하도록 구성되고, 프리젠테이션 버퍼를 모니터링하고, 프리젠테이션 버퍼는 적어도 미디어 데이터가 수신되는 시간 및 미디어 데이터가 수신기에 연관된 프리젠테이션 엘리먼트에 의해 소비되는 시간 사이에 미디어 데이터를 저장하고, 버퍼 레벨의 표시를 저장하고, 버퍼 레벨은 얼마나 많은 프리젠테이션 버퍼가 수신되었지만 아직 프리젠테이션 엘리먼트에 의해 소비되지 않은 미디어 데이터에 의해 차지되었는지에 대응하고, 추정된 다운로드 레이트를 결정하고, 저장된 표시 및 추정된 다운로드 레이트를 사용하여 목표 플레이백 레이트를 계산하고, 목표 플레이백 레이트에 따라 복수의 플레이백 레이트들 중에서 선택하는 것을 포함한다.

[0351] [351] 선택된 플레이백 레이트는 추정된 다운로드 레이트의 미리 결정된 곱셈보다 작거나 같고, 미리 결정된 곱셈은 버퍼 레벨의 증가하는 함수이다. 미리 결정된 곱셈은 프리젠테이션 버퍼의 미디어 데이터의 플레이백 지속 시간의 아핀 선형 함수일 수 있고, 미리 결정된 곱셈은 거기서 프리젠테이션 버퍼의 버퍼 레벨이 임계 양보다 작을 때 1보다 작을 수 있다. 미리 결정된 곱셈은 프리젠테이션 버퍼의 미디어 데이터의 프리젠테이션 지속 시간이 프리젠테이션 시간의 미리 설정된 최대 양보다 많거나 같을 때 1 보다 크거나 같을 수 있다. 미리 결정된 곱셈이 프리젠테이션 버퍼의 미디어의 플레이백 지속 시간의 구분적 선형 함수일 수 있다. 선택된 플레이백 레이트는 추정된 다운로드 레이트의 미리 결정된 곱셈보다 작거나 같을 수 있고, 미리 결정된 곱셈은 프리젠테이션 버퍼의 미디어 데이터의 바이트 수의 증가 함수일 수 있다. 플레이백 레이트는 비례 인수 곱하기 다운로드 레이트 추정보다 작거나 같은 복수의 플레이백 레이트들 중 가장 큰 가용 플레이백 레이트로 선택될 수

있고, 비례 인수는 레이트 변화들에의 반응 시간의 추정으로 나누어진 프리젠테이션 버퍼의 미디어 데이터의 플레이백 지속 시간의 증가 함수이다. 반응 시간은 미디어 데이터의 스위치 포인트들 사이의 프리젠테이션 시간에 상한될 수 있고 및/또는 반응 시간의 추정은 미디어 데이터의 스위치 포인트들 사이의 프리젠테이션 시간의 평균일 수 있다. 반응 시간의 추정은 미리 결정된 상수 곱하기 추정된 라운드-트립 시간(“ERTT”)보다 크거나 같을 수 있다.

[0352] [352] 수신기의 프리젠테이션 엘리먼트를 사용하여 플레이 아웃할 미디어를 수신하는 수신기로서, 플레이 아웃은 미디어가 프리젠테이션 버퍼로부터 플레이백 레이트로 소비되는 것을 초래하고, 수신기는 복수의 플레이백 레이트들로부터 선택하도록 구성되는, 수신기는, 프리젠테이션 버퍼를 모니터링하고, 프리젠테이션 버퍼는 적어도 미디어 데이터가 수신되는 시간 및 미디어 데이터가 수신기에 연관된 프리젠테이션 엘리먼트에 의해 소비되는 시간 사이에 미디어 데이터를 저장하고, 버퍼 레벨의 표시를 저장하고, 버퍼 레벨은 얼마나 많은 프리젠테이션 버퍼가 수신되었지만 아직 프리젠테이션 엘리먼트에 의해 소비되지 않은 미디어 데이터에 의해 차지되었는지에 대응하고, 버퍼 레벨의 저장된 표시 및 버퍼 레벨의 허용된 편차를 사용하여 목표 플레이백 레이트를 계산하고, 목표 플레이백 레이트에 따라 복수의 플레이백 레이트들 중에서 선택하기 위한 방법 또는 장치를 포함한다.

[0353] [353] 플레이백 레이트는 높은 비례 인수, 낮은 비례 인수, 다운로드 레이트 추정, 현재 플레이백 레이트, 버퍼 레벨, 및 레이트 변경들에 대한 반응 시간의 추정에 기초하여 선택될 수 있다. 높은 비례 인수 및 낮은 비례 인수는 모두 레이트 변화에의 반응 시간의 추정에 의해 나누어진 프리젠테이션 버퍼의 미디어 데이터의 플레이백 지속 시간의 증가 함수들 및/또는 구분적 선형 함수들일 수 있고, 높은 비례 인수는 낮은 비례 인수보다 크거나 같을 수 있다. 플레이백 레이트는 이전 플레이백 레이트가 낮은 비례 인수 곱하기 추정된 다운로드 레이트 및 높은 비례인수 곱하기 다운로드 레이트 추정 사이에 있으면 이전 플레이백 레이트와 동일할 수 있다. 이전 플레이백 레이트가 높은 비례 인수 곱하기 다운로드 레이트 추정을 넘으면 플레이백 레이트는 높은 비례 인수 곱하기 추정된 다운로드 레이트보다 크지 않은 가장 큰 가용 플레이백 레이트 이도록 선택될 수 있다. 이전 플레이백 레이트가 낮은 비례 인수 곱하기 다운로드 레이트 추정 아래이면 플레이백 레이트는 낮은 비례 인수 곱하기 다운로드 레이트 추정보다 크지 않은 가장 큰 가용 플레이백 레이트 이도록 선택될 수 있다.

[0354] [354] 일 실시예에서, 리퀘스트들을 하고 또한 프로세스에서 리퀘스트들을 취소할지를 결정하기 위한 방법 및/또는 장치 또는 컴퓨터 판독가능 매체가 사용된다. 수신기가 미디어의 세그먼트들/부분들/프래그먼트들에 대해 리퀘스트들을 하고, 리퀘스트에 대한 응답을 수신하고, 응답으로부터 미디어를 저장하고 아마 다른 리퀘스트를 하면서, 그것은 리퀘스트를 취소하고 다른 리퀘스트를 이슈하는 것이 바람직할 지를 결정할 수 있다. 미디어의 플레이백 레이트는 수신기에 의해 가장 공격적이고 그것이 소비됨에 따라 프리젠테이션 버퍼의 미디어가 없어짐 없이 획득할 것으로 예상하는 가장 높은 플레이백 레이트를 선택함으로써 결정될 수 있다. 다운로드 레이트가 갑자기 하락하면, 수신기는 그 현재 리퀘스트를 취소하고 낮은 플레이백 레이트 미디어에 대한 새 리퀘스트를 할지 아니면 현재 리퀘스트가 플레이 아웃하게 할지를 결정한다. 높은 플레이백 레이트 리퀘스트를 취소하고 그것을 낮은 플레이백 레이트 리퀘스트로 대체하는 것은 프리젠테이션 버퍼의 콘텐츠가 오래 지속되게 할 수 있고, 그러나 도중에 리퀘스트를 취소하는 것은 그 리퀘스트에 대한 임의의 부분적으로 수신된 미디어의 손실을 초래할 수 있다.

[0355] [355] 그러한 한 실시예에서, 수신기는 수신기의 프리젠테이션 엘리먼트를 이용하여 플레이 아웃 할 미디어를 수신하고, 플레이 아웃은 미디어가 프리젠테이션 버퍼로부터 플레이백 레이트로 소비되는 것을 초래하고, 수신기는 복수의 플레이백 레이트들로부터 선택하도록 구성된다. 리퀘스트 행동을 결정하는 것은, 프리젠테이션 버퍼를 모니터링하고, 프리젠테이션 버퍼는 적어도 미디어 데이터가 수신되는 시간 및 미디어 데이터가 수신기에 연관된 프리젠테이션 엘리먼트에 의해 소비되는 시간 사이에 미디어 데이터를 저장하고, 버퍼 레벨의 표시를 저장하고, 버퍼 레벨은 얼마나 많은 프리젠테이션 버퍼가 수신되었지만 아직 프리젠테이션 엘리먼트에 의해 소비되지 않은 미디어 데이터에 의해 차지되었는지에 대응하고, 미디어 데이터의 선택된 제 1 청크를 다운로드하기 위한 이슈된 리퀘스트의 상태를 유지하고, 이슈된 리퀘스트가 아웃스탠딩이면, 네트워크 상태 및 이슈된 리퀘스트의 상태에 기초하여, 리퀘스트를 계속할지 리퀘스트를 취소할지를 결정하는 것을 포함한다.

[0356] [356] 리퀘스트를 계속할지 리퀘스트를 취소할지를 결정하는 것은, 제 1 미디어 데이터가 플레이 아웃되어야 하기 전에 리퀘스트에 대한 다운로드를 완료할 충분한 시간이 있는지 여부를 결정하고, 충분한 시간이 없다면, 리퀘스트를 취소하는 것을 포함할 수 있다. 리퀘스트를 계속할지 리퀘스트를 취소할지를 결정하는 것은, 다운로드 레이트들 및 미디어 소비 레이트들에 기초하여, 스톨이 발생할 것임을 검출하고, 프리젠테이션 엘리먼트가 소비되고 있는 미디어에 의해 지시된 레이트에서 미디어 데이터를 소비할 수 없을 때의 시간 및 프리젠테이션 엘리먼트가 소비되고 있는 미디어에 의해 지시된 레이트에서 미디어 데이터를 소비하는 것을 재개할 수 있는 시

간 사이의 스톨 주기를 추정하고, 계속 또는 취소가 스톨 주기에 미치는 영향을 결정하고, 리퀘스트를 취소하는 것이 스톨 주기를 줄이면, 리퀘스트를 취소하는 것을 더 포함할 수 있다.

[0357] [357] 다른 특징들은 미디어 데이터의 제 2 청크를 선택하고, 미디어 데이터의 제 2 청크는 시작 프리젠테이션 시간을 갖고 그 시작 프리젠테이션 시간은 미디어 데이터의 제 1 청크와 동일한 시작 프리젠테이션 시간이고, 미디어 데이터의 제 2 청크의 다운로드를 리퀘스트하는 것, 미디어 데이터의 제 2 청크를 선택하고, 미디어 데이터의 제 2 청크는 시작 프리젠테이션 시간을 갖고 그 시작 프리젠테이션 시간은 미디어 데이터의 제 1 청크보다 늦고, 미디어 데이터의 제 2 청크의 다운로드를 리퀘스트하는 것을 포함할 수 있다. 미디어 데이터의 제 2 청크는 리시버에 의해 그 시작 프리젠테이션 시간이 제 1 청크의 시작 프리젠테이션 시간의 그것에 비교하여 리시버에 이용 가능한 가장 낮은 차이이고, 및/또는 그 플레이백이 그 시작 프리젠테이션 시간과 미디어 데이터의 제 1 청크의 시작 프리젠테이션 시간 사이의 미리 결정된 최대 갭을 갖는 최대 플레이백 레이트 이도록 선택될 수 있다.

[0358] [358] 어떤 실시예들은 미디어 데이터의 제 1 청크의 잔여 부분의 다운로드가 플레이백을 위해 적시에 완료될 수 없는지를 결정하고, 미디어 데이터의 제 2 청크의 다운로드가 플레이백을 위해 적시에 완료될 수 있는지를 결정하고, 리퀘스트를 계속할지 아니면 미디어 데이터의 제 1 청크에 대한 리퀘스트를 취소하고 대신에 미디어 데이터의 제 2 청크를 리퀘스트할 지에 대한 결정을, 미디어 데이터의 제 1 청크의 잔여 부분의 다운로드가 플레이백을 위해 적시에 완료될 수 없는지 및 미디어 데이터의 제 2 청크의 다운로드가 플레이백을 위해 적시에 완료될 수 있는지에 기초를 두는 것을 포함할 수 있다. 데이터의 제 2 청크의 미디어 데이터의 플레이백 레이트는 수신기에서 지원되는 가장 높은 플레이백 레이트 이도록 선택될 수 있다. 수신기는 이미 프리젠테이션 버퍼에 있는 적어도 얼마간의 미디어 데이터의 프리젠테이션 시간을 커버링하는 미디어 데이터를 요청하고, 리퀘스트된 미디어 데이터를 다운로드하고, 리퀘스트된 미디어 데이터를 플레이 아웃하고, 이미 프리젠테이션 버퍼에 있는 대응하는 미디어 데이터의 적어도 얼마간을 폐기할 수 있다. 리퀘스트된 미디어 데이터의 플레이백 레이트는, 프리젠테이션 버퍼로부터 폐기된 대응 미디어 데이터의 최대 프리젠테이션 지속 시간에 대한 제약을 조건으로, 최대 플레이백 레이트일 수 있다. 리퀘스트된 미디어 데이터는 그 시작 프리젠테이션 시간이 수신기에 이용 가능한 가장 빠른 시작 프리젠테이션 시간 이도록 선택될 수 있다.

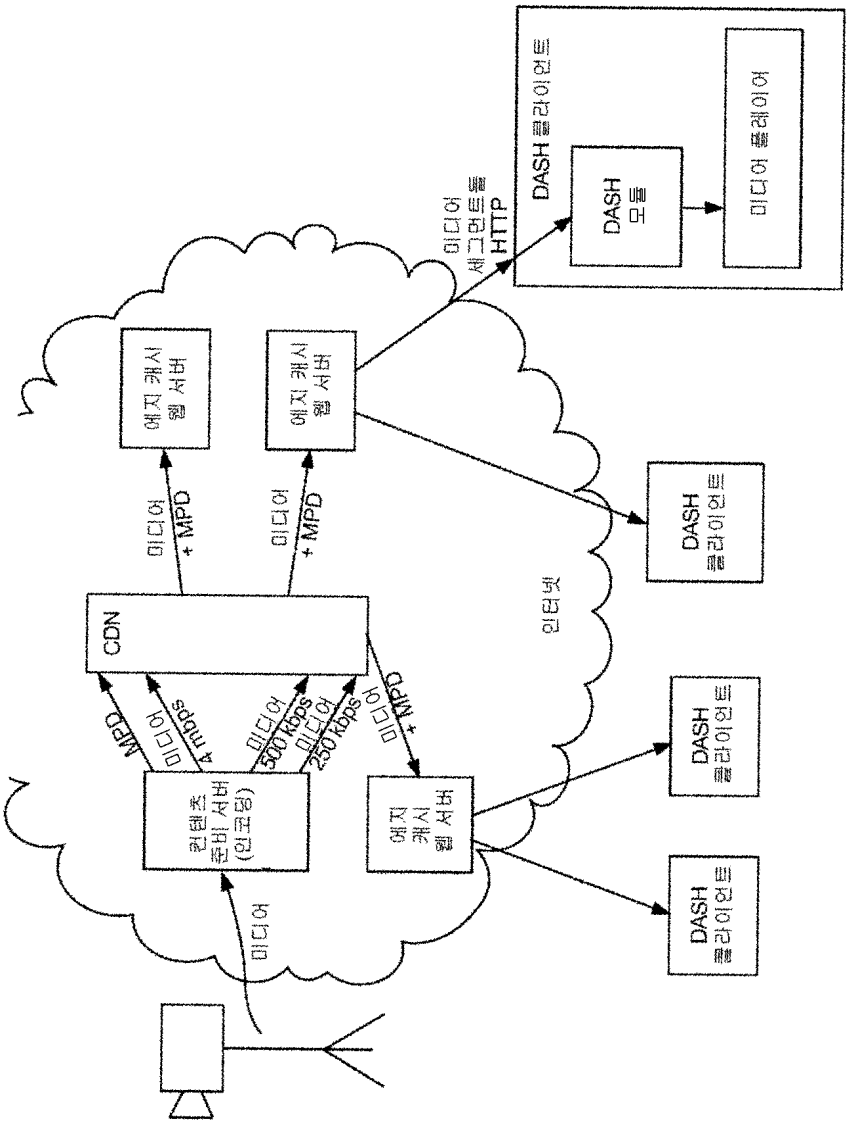
[0359] [359] 어떤 수신기들에서, 다운로드는 버퍼 레벨에 의존하고, 수신기들은 높은 워터마크 및 낮은 워터마크의 개념을 사용한다. 그러한 수신기에서, 미디어 데이터는 소스로부터 다운로드되고 수신기의 프리젠테이션 버퍼에 저장된다. 프리젠테이션 버퍼의 채워진 레벨(또는 단지 “레벨”)이 결정되고, 채워진 레벨은 프리젠테이션 버퍼의 어떤 부분이 프리젠테이션 엘리먼트에 의해 아직 소비되지 않은 미디어 데이터를 포함한다는 것을 나타낸다. 채워진 레벨이 높은 채움 임계치(“높은 워터마크”)를 넘으면, 다운로드는 중단되고, 채워진 레벨이 낮은 채움 임계치(“낮은 워터마크”) 아래이면, 다운로드는 다시 시작한다. 채워진 레벨은 미디어 데이터가 프리젠테이션 엘리먼트에 의해 소비됨에 따라 업데이트될 수 있다. 채워진 레벨은 메모리 저장 용량의 단위 및/또는 프리젠테이션 시간 단위로 측정될 수 있다. 다운로드는 추정된 라운드-트립 시간(“ERTT”)에 기초할 수 있고, ERTT는 미디어 데이터 다운로드가 재시작될 때 리셋된다. 다운로드가 복수의 TCP 접속들을 통해 발생하면, 사용된 복수의 TCP 접속들이 미디어 데이터 다운로드가 재시작될 때 리셋될 수 있다. 높은 채움 및 낮은 채움 임계치들은 시간에 따라 변할 수 있다.

[0360] [360] 추가의 실시예들이 이 개시물을 읽은 후에 당업자에게 상상될 수 있다. 다른 실시예들에서, 위에 개시된 발명의 조합 및 서브-조합들이 유리하게 만들어 질 수 있다. 실례의 목적으로 컴포넌트들의 예시적 배열들이 제시되고 조합들, 추가들, 재배열들 및 기타 등등은 본 발명의 대안적 실시예들에 의도된다. 따라서, 발명은 예시적인 실시예에 대하여 설명되었으며, 당업자는 많은 변형들이 가능함을 인식할 것이다.

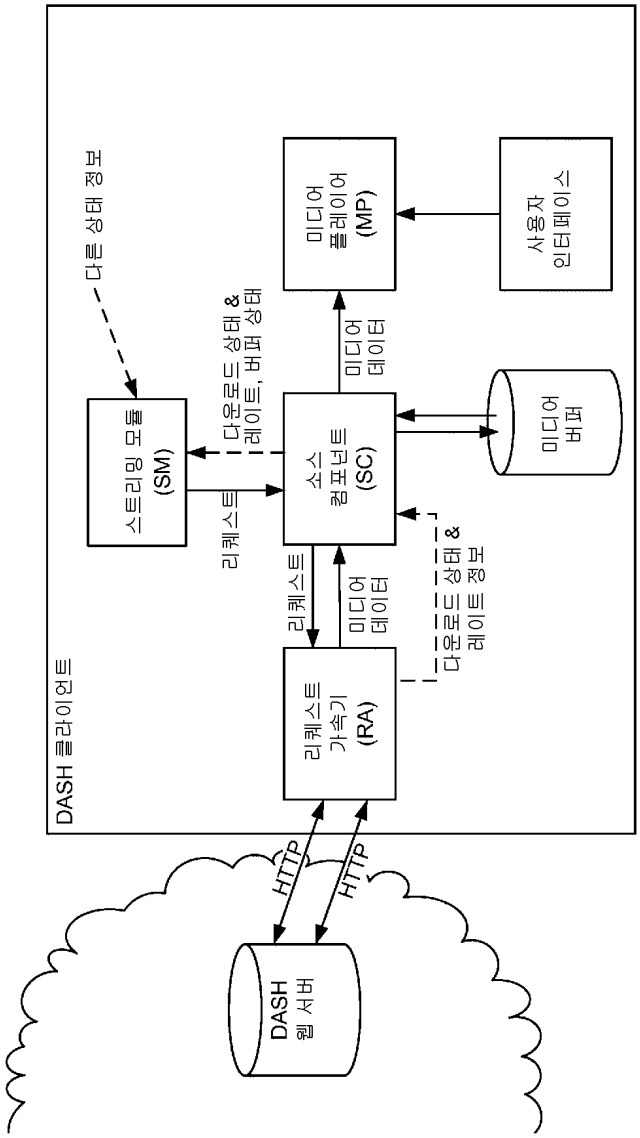
[0361] [361] 예를 들어, 여기에 설명된 프로세스들은 하드웨어 컴포넌트들, 소프트웨어 컴포넌트들, 및/또는 그 임의의 결합을 사용하여 구현될 수 있다. 설명 및 도면들은, 따라서, 한정적 의미가 아니라 예시적인 것으로 간주되어야 한다. 그러나 다양한 변경들 및 변형들이 청구항들에 제시된 바와 같은 본 발명의 더 넓은 사상 및 범위로부터 벗어남 없이 거기에 될 수 있고 본 발명은 다음 청구항들의 범위 내에 모든 변경들 및 균등물들을 포함하는 것으로 의도되었다는 것이 명백할 것이다.

도면

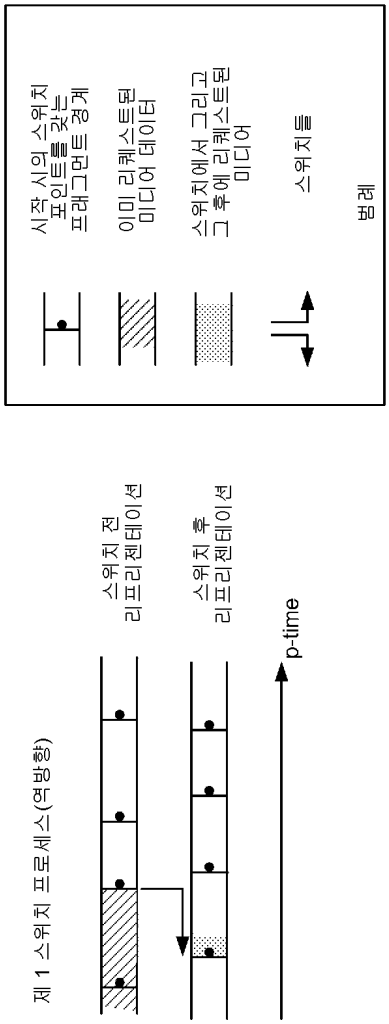
도면1


DASH 클라이언트
DASH 모듈
미디어 플레이어
미디어 세그먼트들 HTTP
에지 캐시 웹 서버
에지 캐시 웹 서버
미디어 + MPD
미디어 + MPD
CDN
인터넷
DASH 클라이언트
DASH 클라이언트
DASH 클라이언트
MPD
4 mbps 미디어
500 kbps 미디어
250 kbps 미디어
미디어 + MPD
컨텐츠 준비 서버 (인코딩)
에지 캐시 웹 서버
미디어

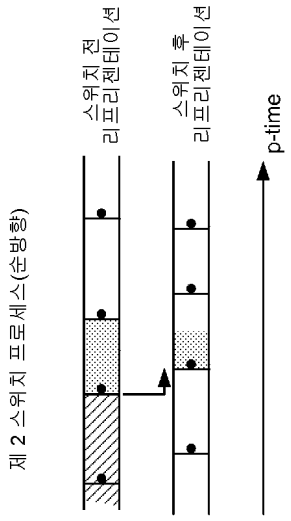
도면2


DASH 클라이언트
스트리밍 모듈 (SM)
다른 상태 정보
다운로드 상태 & 레이트, 버퍼 상태
리퀘스트
소스 컴포넌트 (SC)
미디어 데이터
미디어 플레이어 (MP)
사용자 인터페이스
미디어 버퍼
리퀘스트
미디어 데이터
다운로드 상태 & 레이트 정보
리퀘스트 가속기 (RA)
HTTP
HTTP
DASH 웹 서버

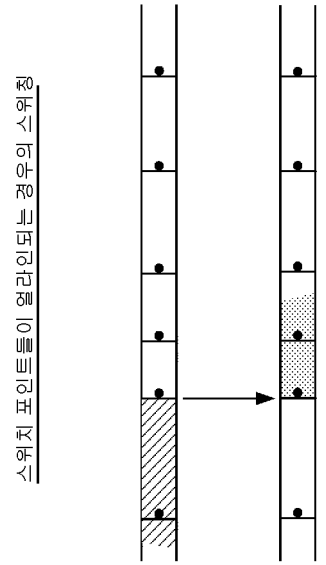
도면3a


범례
스위치들
스위치에서 그리고
그 후에 리퀘스트된
미디어
이미 리퀘스트된
미디어 데이터
시작 시의 스위치
포인트를 갖는
프래그먼트 경계
제 1 스위치 프로세스(역방향)
스위치 전
리프리젠테이션
스위치 후
리프리젠테이션
p-time

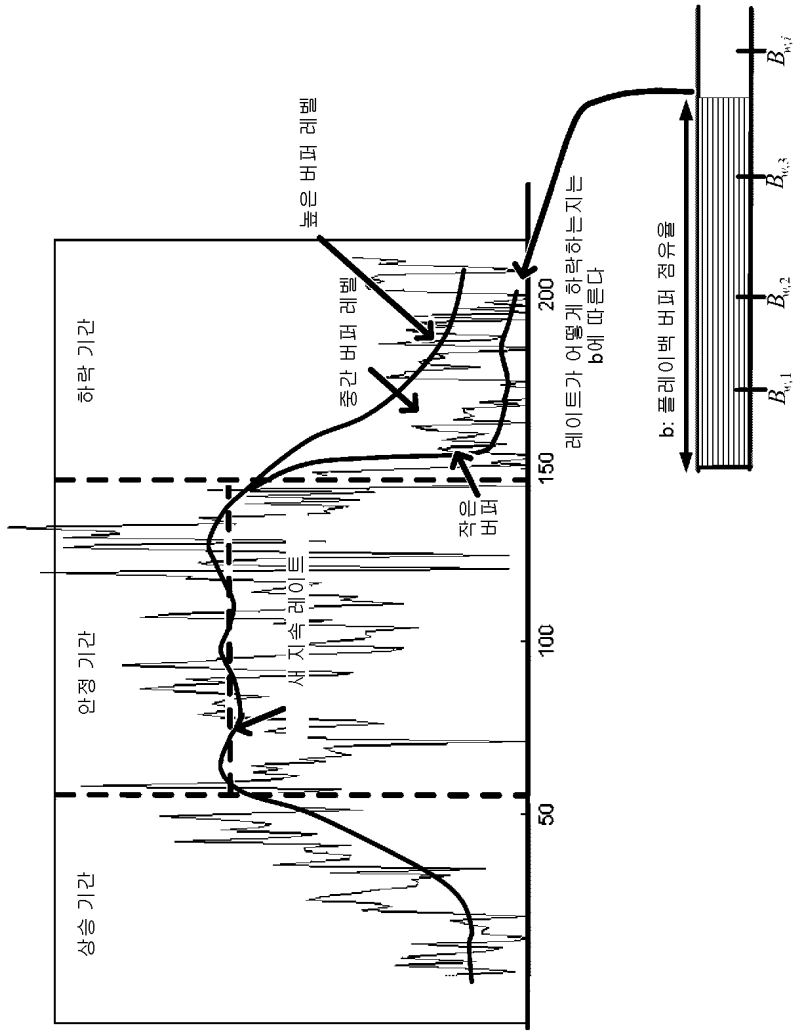
도면3b


제 2 스위치 프로세스(순방향)
스위치 전
리프리젠테이션
스위치 후
리프리젠테이션
p-time

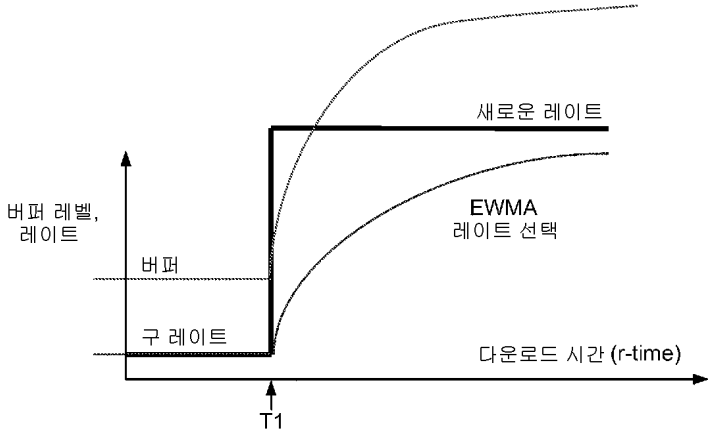
도면4


스위치 포인트들이 얼라인되는 경우의 스위칭

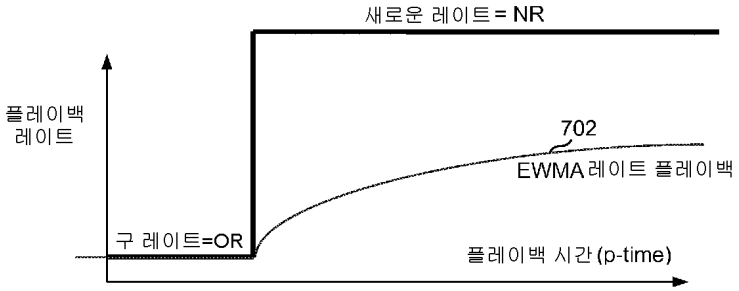
도면5


하락 기간
안정 기간
상승 기간
높은 버퍼 레벨
중간 버퍼 레벨
낮은 버퍼
새 지속레이트
레이트가 어떻게 하락하는지는
b에 따른다
b: 플레이백 버퍼 점유율
50
100
150
200

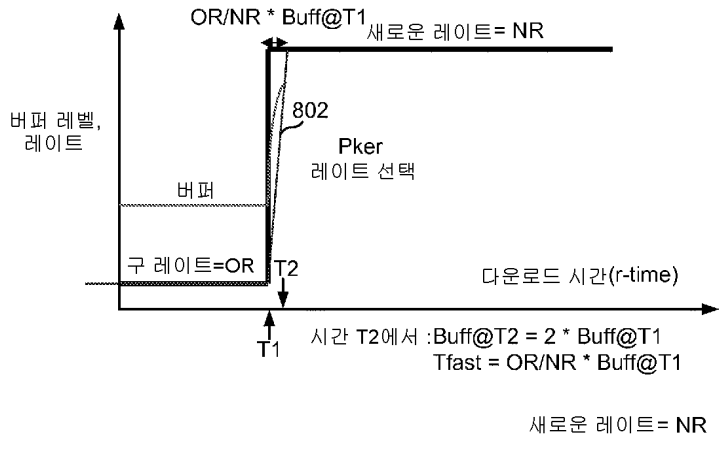
도면6


새로운 레이트
EWMA
레이트 선택
버퍼 레벨,
레이트
버퍼
구 레이트
다운로드 시간 (r-time)
T1

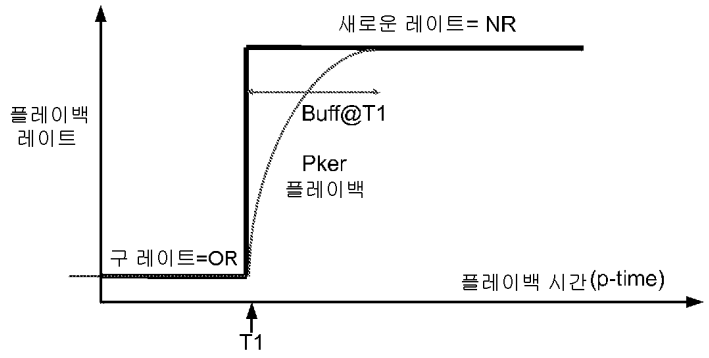
도면7


새로운 레이트 = NR
플레이백
레이트
702
EWMA 레이트 플레이백
구 레이트=OR
플레이백 시간 (p-time)

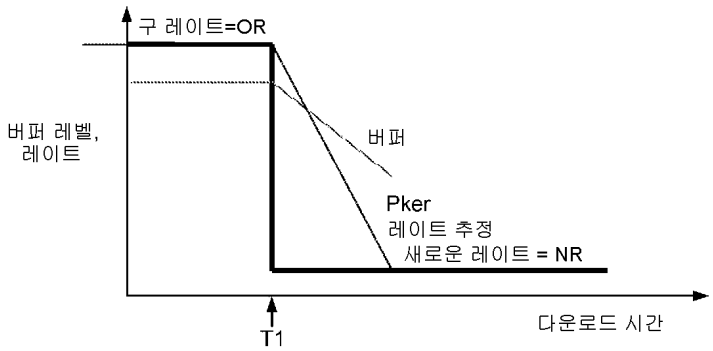
도면8


OR/NR * Buff@T1
새로운 레이트= NR
802
Pker
레이트 선택
버퍼 레벨,
레이트
버퍼
구 레이트=OR
T2
다운로드 시간(r-time)
T1
시간 T2에서 :Buff@T2 = 2 * Buff@T1
Tfast = OR/NR * Buff@T1
새로운 레이트= NR

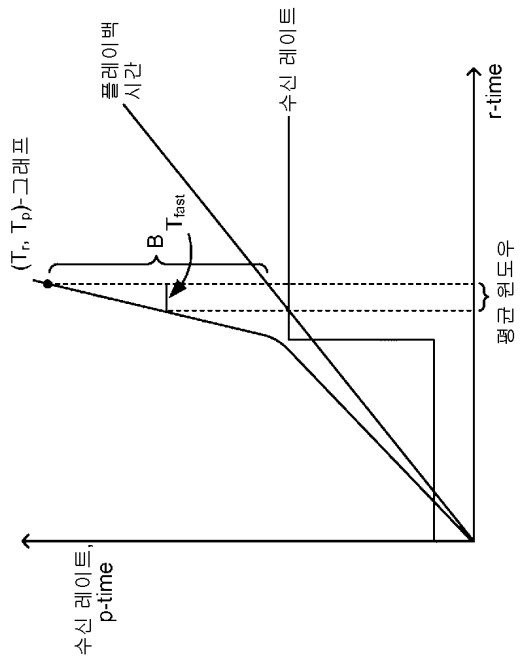
도면9



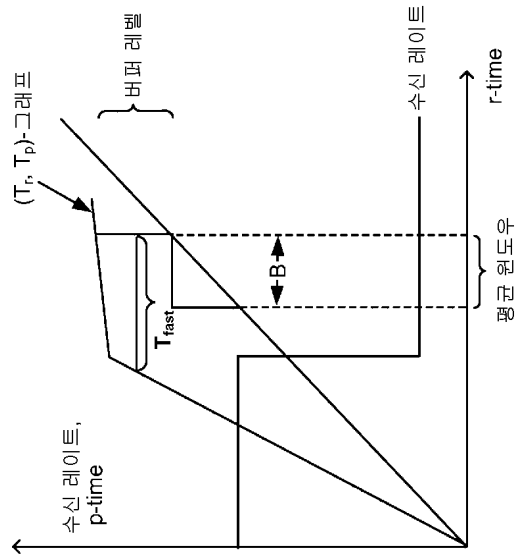
도면10



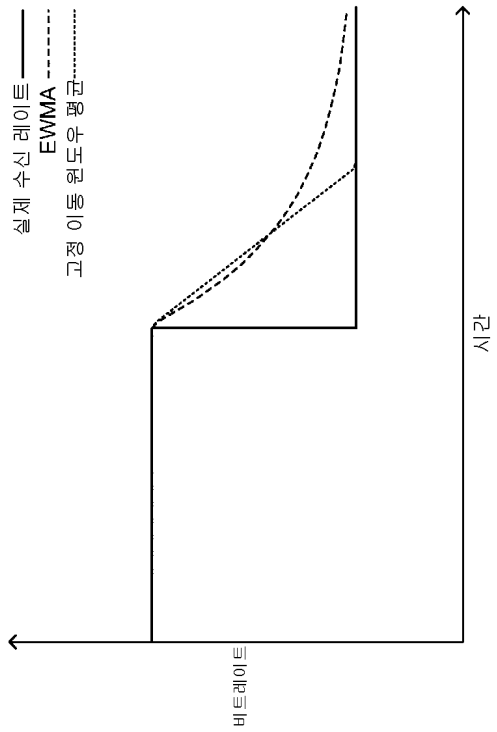
도면11



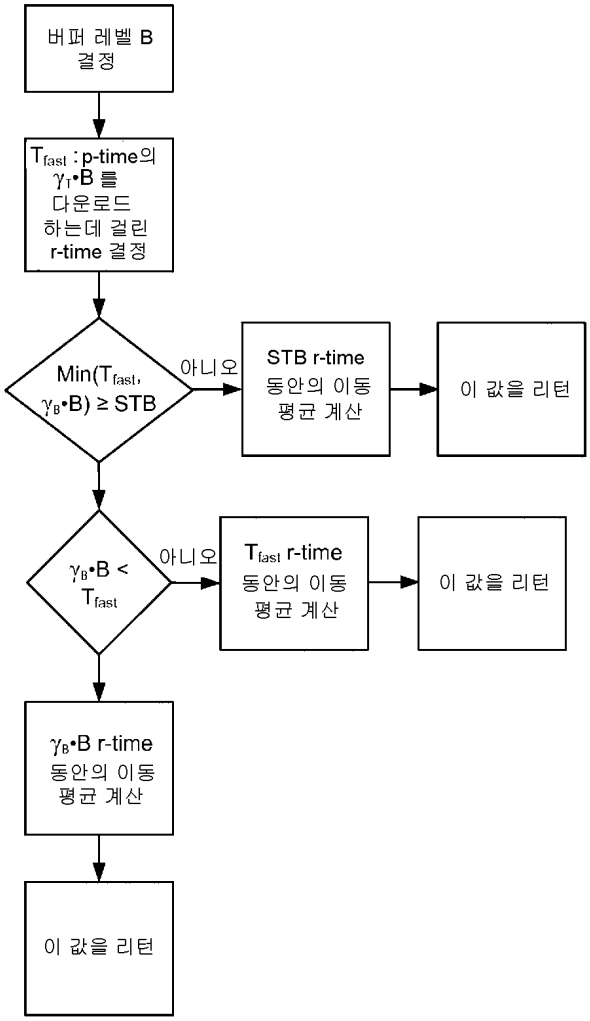
도면12


(Tr, Tp)-그래프
버퍼 레벨
수신 레이트
r-time
B
Tfast
평균 윈도우
수신 레이트,
p-time

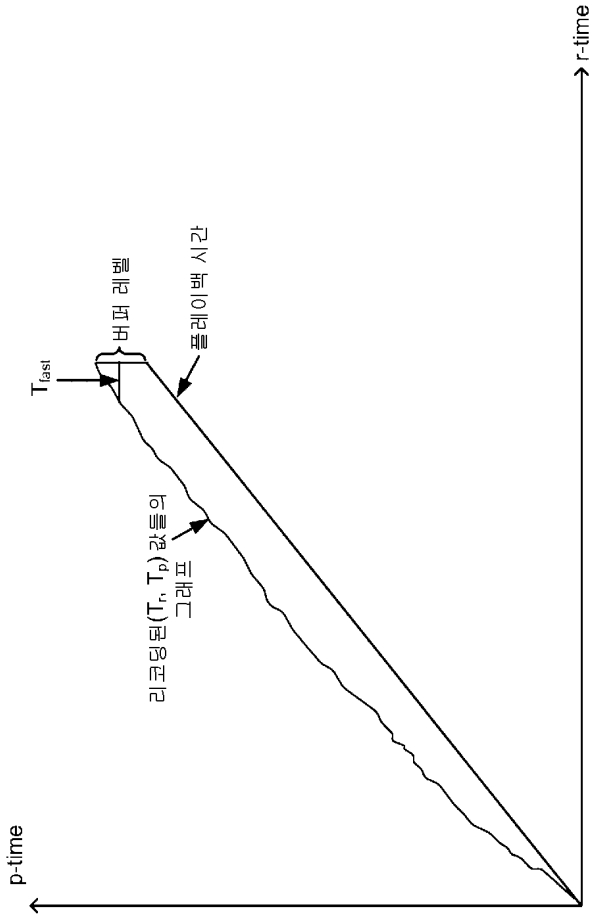
도면13


실제 수신 레이트
EWMA
고정 이동 윈도우 평균
시간
비트레이트

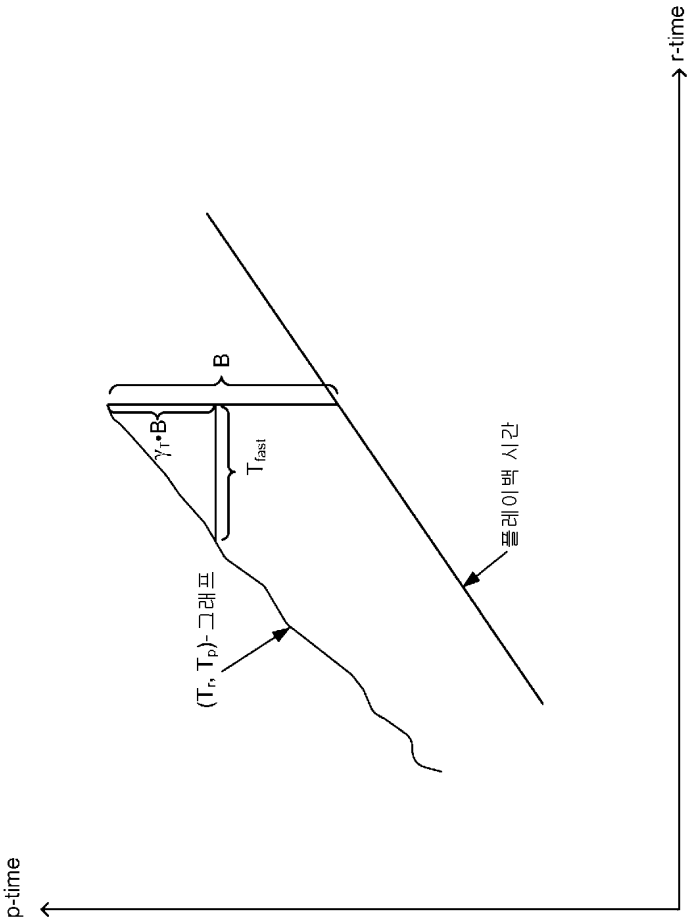
도면14


버퍼 레벨 B 결정
T_{fast} : p-time의 γ_T•B 를 다운로드 하는데 걸린 r-time 결정
Min(T_{fast}, γ_B•B) ≥ STB
아니오
STB r-time 동안의 이동 평균 계산
이 값을 리턴
γ_B•B < T_{fast}
아니오
T_{fast} r-time 동안의 이동 평균 계산
이 값을 리턴
γ_B•B r-time 동안의 이동 평균 계산
이 값을 리턴

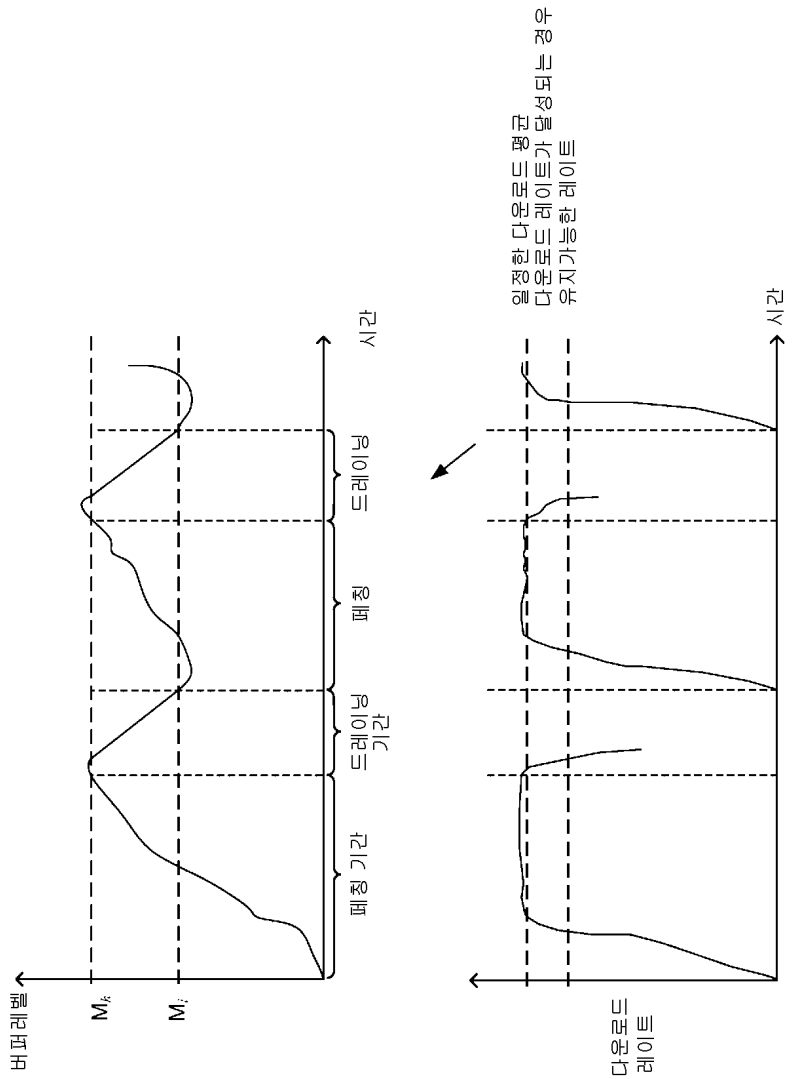
도면15



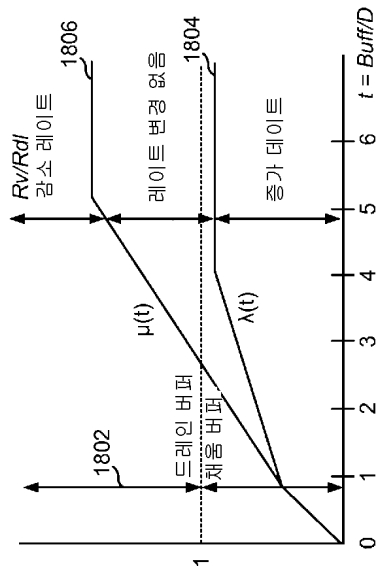
도면16



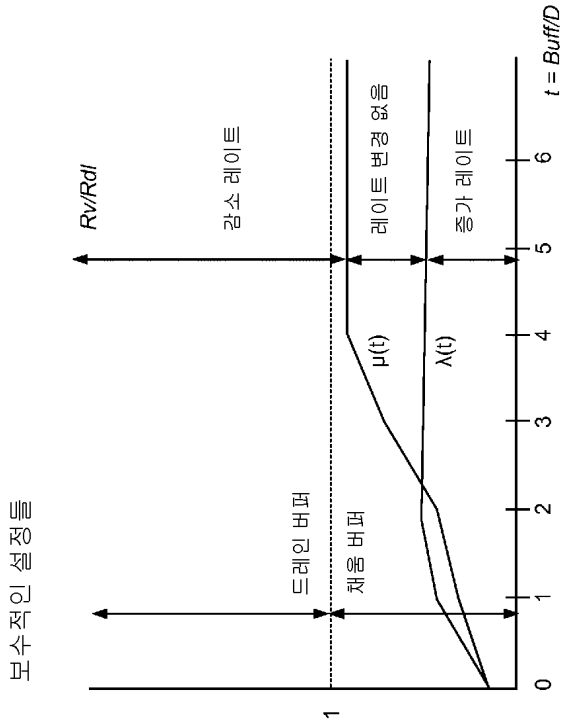
도면17


일정한 다운로드 평균
다운로드 레이트가 달성되는 경우
유지가능한 레이트
시간
다운로드
레이트
시간
버퍼레벨
M_k
M_i
폐칭 기간
드레이닝
기간
폐칭
드레이닝

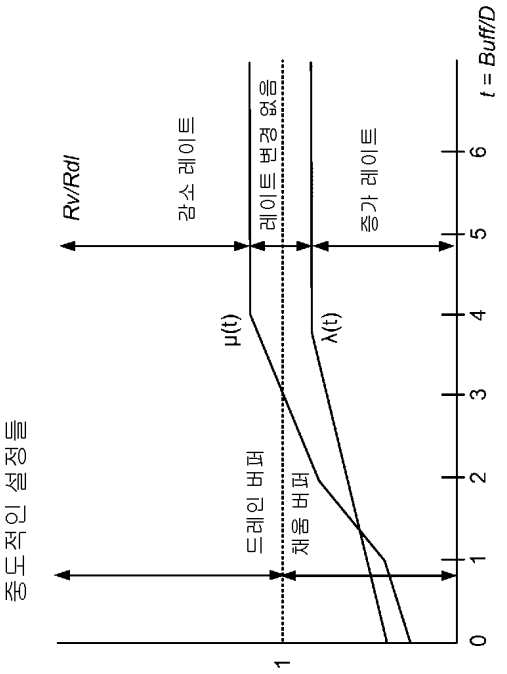
도면18


Rv/Rdl
감소 레이트
1806
레이트 변경 없음
1804
증가 데이트
t = Buff/D
μ(t)
λ(t)
드레인 버퍼
채움버퍼
1802
0
1
2
3
4
5
6
1

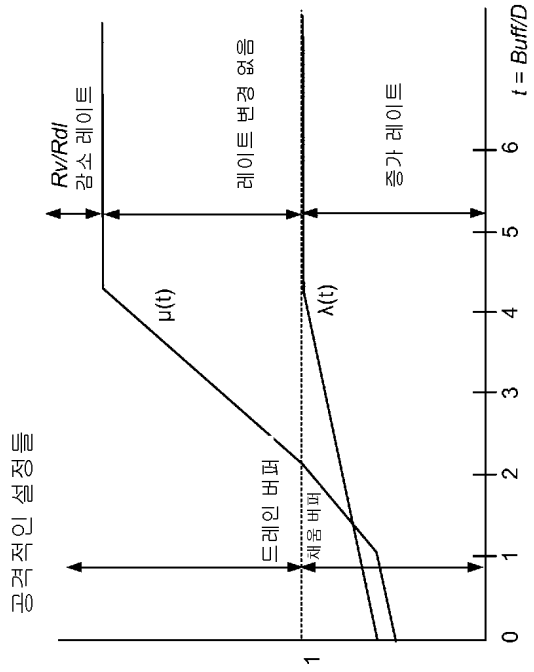
도면19


보수적인 설정들
Rv/Rdl
감소 레이트
레이트 변경 없음
증가 레이트
t = Buff/D
μ(t)
λ(t)
드레인 버퍼
채움 버퍼
0
1
2
3
4
5
6
1

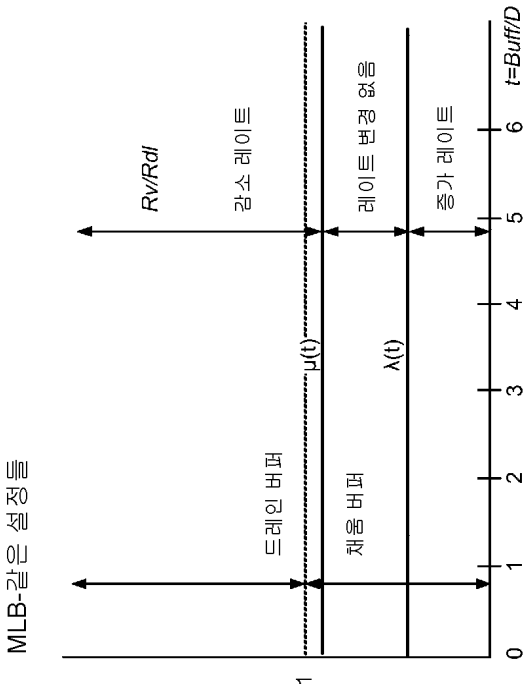
도면20


중도적인 설정들
Rv/Rdl
감소 레이트
레이트 변경 없음
증가 레이트
μ(t)
λ(t)
드레인 버퍼
채움 버퍼
1
0
1
2
3
4
5
6
t = Buff/D

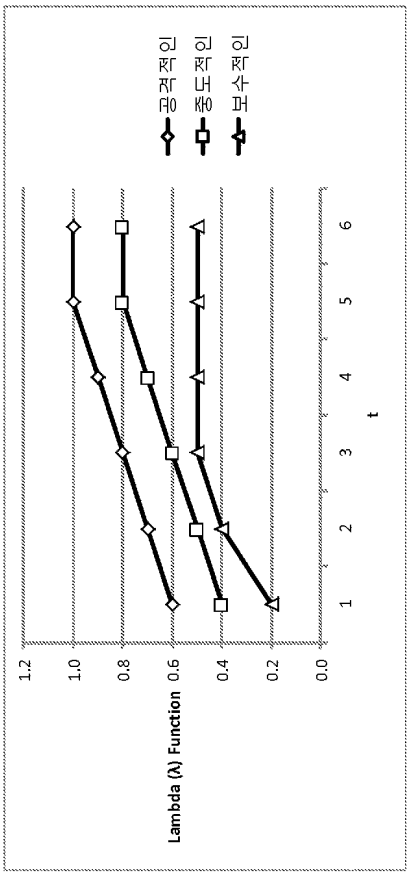
도면21


공격적인 설정들
Rv/Rdl
감소 레이트
레이트 변경 없음
증가 레이트
μ(t)
λ(t)
드레인 버퍼
채움 버퍼
1
0
1
2
3
4
5
6
t = Buff/D

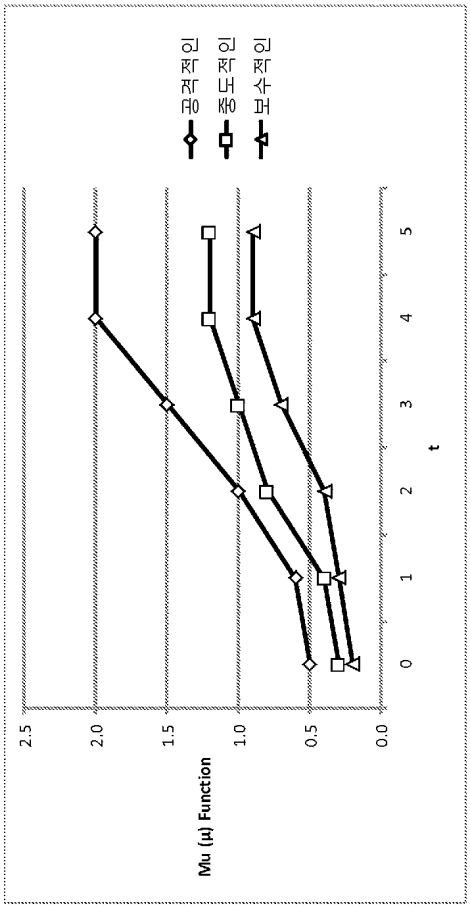
도면22


MLB-같은 설정들
드레인 버퍼
채움 버퍼
μ(t)
λ(t)
Rv/Rdl
감소 레이트
레이트 변경 없음
증가 레이트
t=Buff/D
0
1
2
3
4
5
6
1

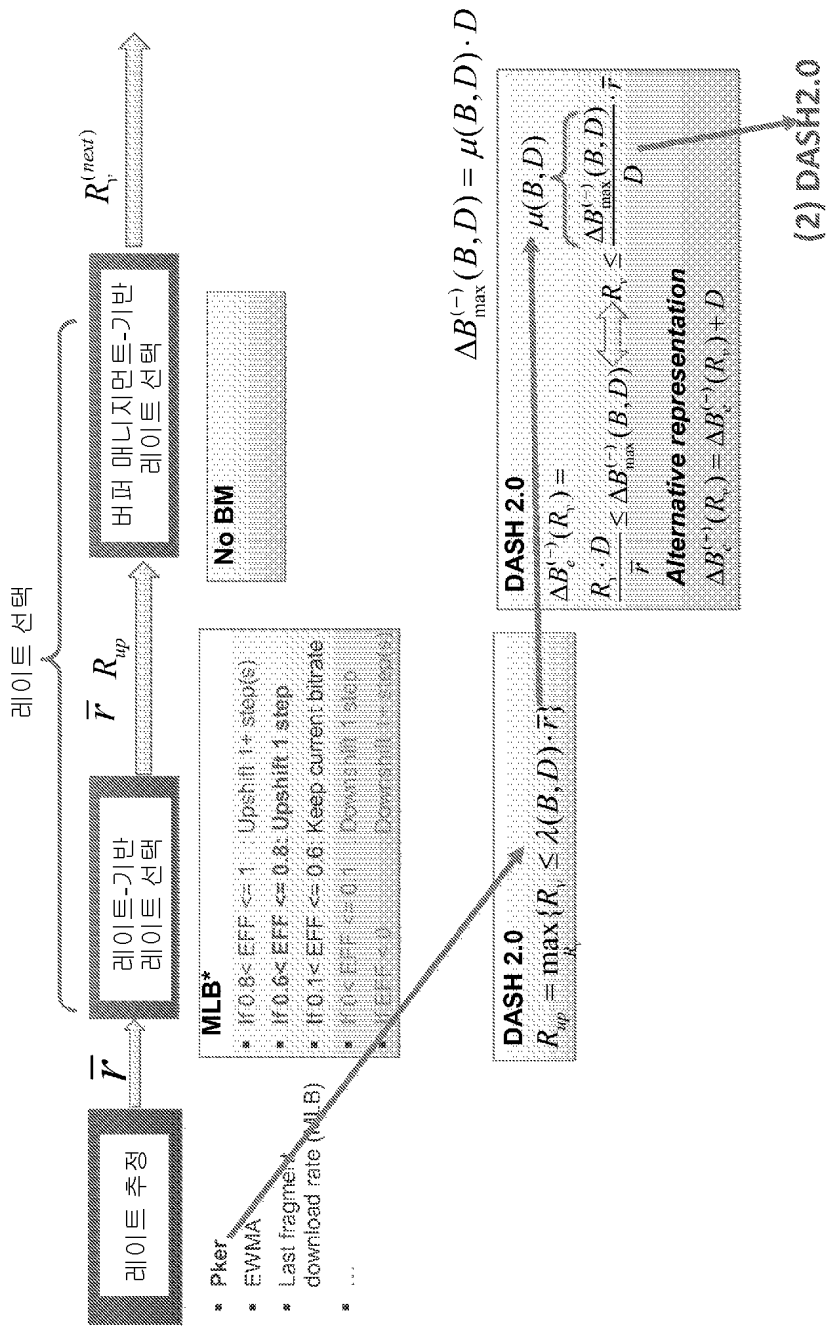
도면23


공격적인
중도적인
보수적인
Lambda (λ) Function
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
6
t

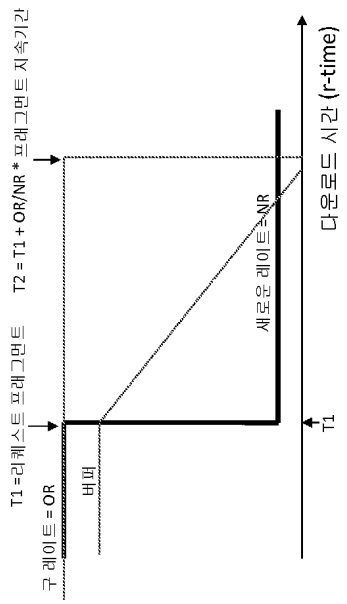
도면24



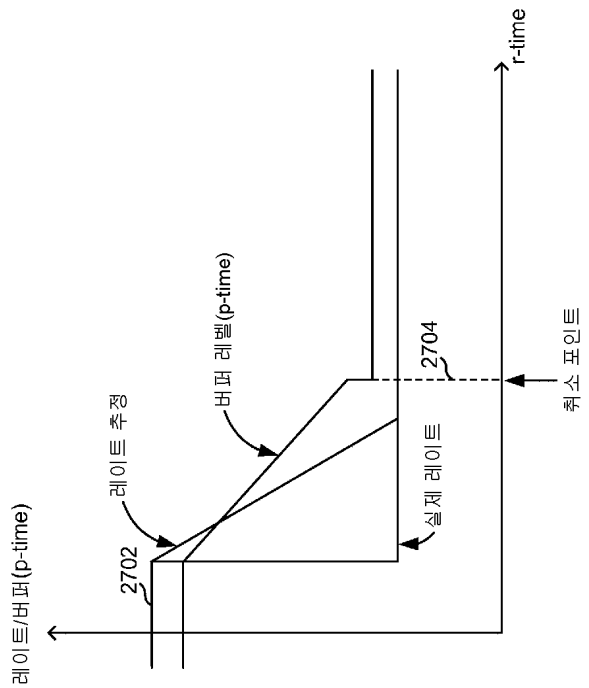
도면25


레이트 선택
레이트 추정
레이트-기반 레이트 선택
버퍼 매니지먼트-기반 레이트 선택
$\bar{r}$
R_{up}
$R_i^{(next)}$
No BM
MLB*
If 0.8< EFF <= 1 : Upshift 1+ step(s)
If 0.6< EFF <= 0.8: Upshift 1 step
If 0.1< EFF <= 0.6: Keep current bitrate
If 0 < EFF <= 0.1 : Downshift 1 step
If EFF = 0 : Downshift to lowest
Pker
EWMA
Last fragment download rate (MLB)
...
DASH 2.0
$R_{up} = \max_{\lambda} \{R_\lambda \le \lambda(B,D) \cdot \bar{r}\}$
$\Delta B_{\max}^{(-)}(B,D) = \mu(B,D) \cdot D$
DASH 2.0
$\Delta B_e^{(-)}(R_i) =$
$\frac{R_i \cdot D}{\bar{r}} \le \Delta B_{\max}^{(-)}(B,D) \Leftrightarrow R_i \le \frac{\Delta B_{\max}^{(-)}(B,D)}{D} \cdot \bar{r}$
$\mu(B,D)$
Alternative representation
$\Delta B_e^{(-)}(R_i) = \Delta B_e^{(-)}(R_i) + D$
(2) DASH2.0

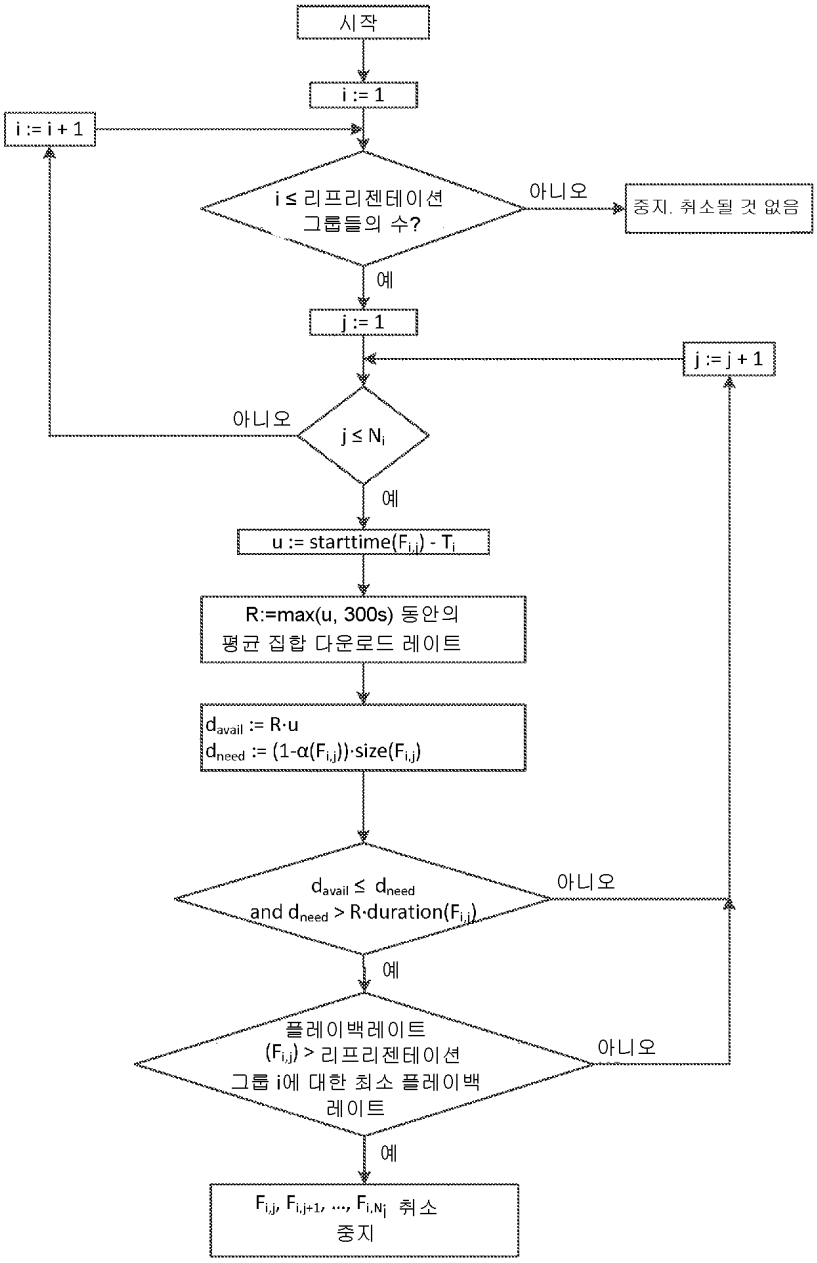
도면26


T1 =리퀘스트 프래그먼트
T2 = T1 + OR/NR * 프래그먼트 지속기간
구 레이트 = OR
버퍼
새로운 레이트 = NR
다운로드 시간 (r-time)
T1

도면27


레이트/버퍼(p-time)
r-time
2702
레이트 추정
버퍼 레벨 (p-time)
2704
실제 레이트
취소 포인트

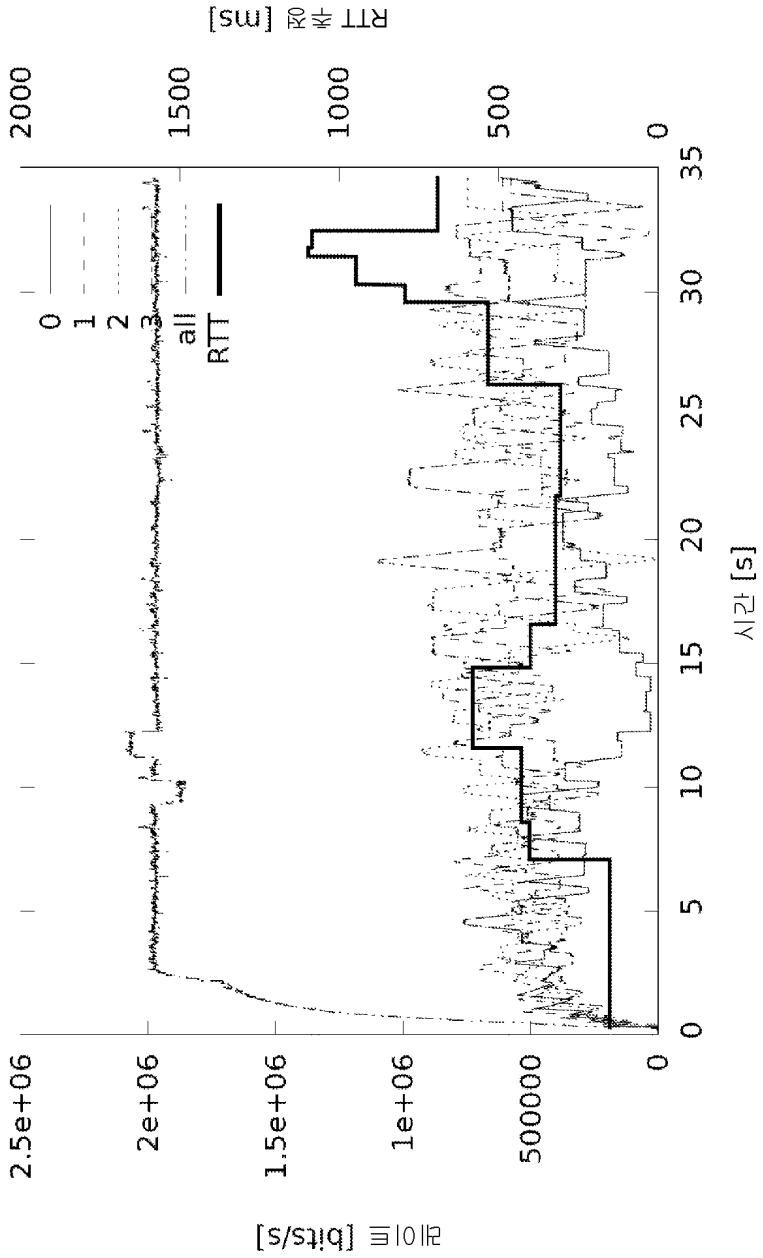
도면28


시작
i := 1
i := i + 1
i ≤ 리프리젠테이션 그룹들의 수?
아니오
중지. 취소될 것 없음
예
j := 1
j := j + 1
아니오
j ≤ Ni
예
u := starttime(Fi,j) - Ti
R:=max(u, 300s) 동안의 평균 집합 다운로드 레이트
davail := R·u
dneed := (1-α(Fi,j))·size(Fi,j)
davail ≤ dneed and dneed > R·duration(Fi,j)
아니오
예
플레이백레이트 (Fi,j) > 리프리젠테이션 그룹 i에 대한 최소 플레이백 레이트
아니오
예
Fi,j, Fi,j+1, ..., Fi,Ni 취소 중지

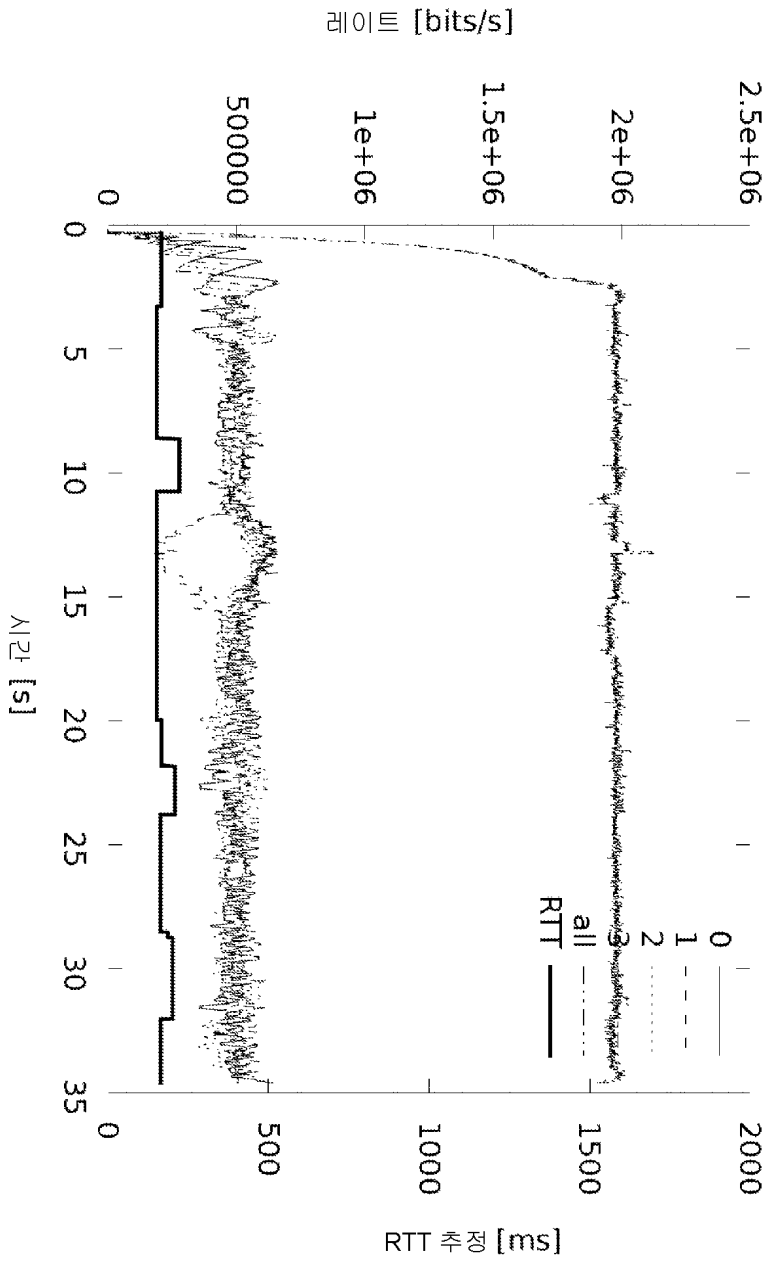
도면29

```
for every representation group i do
    m ← smallest available rate  for this group
    for j = 1, ... , N_i do
        if representation-rate(F_ij) = m then
            continue
        end if
        u ← starttime(F_ij) – T
        R ← aggregate rate over max(u, 300ms)
        if R cannot be computed  then
            continue
        end if
        d_avail ← R • u
        d_need ← (1 – α (F_ij)) •·size(F_ij)
        if d_avail < d_need and d_need > R •·duration (F_ij) then
            return (i,j)
        end if
    end for
end for
return nil
```

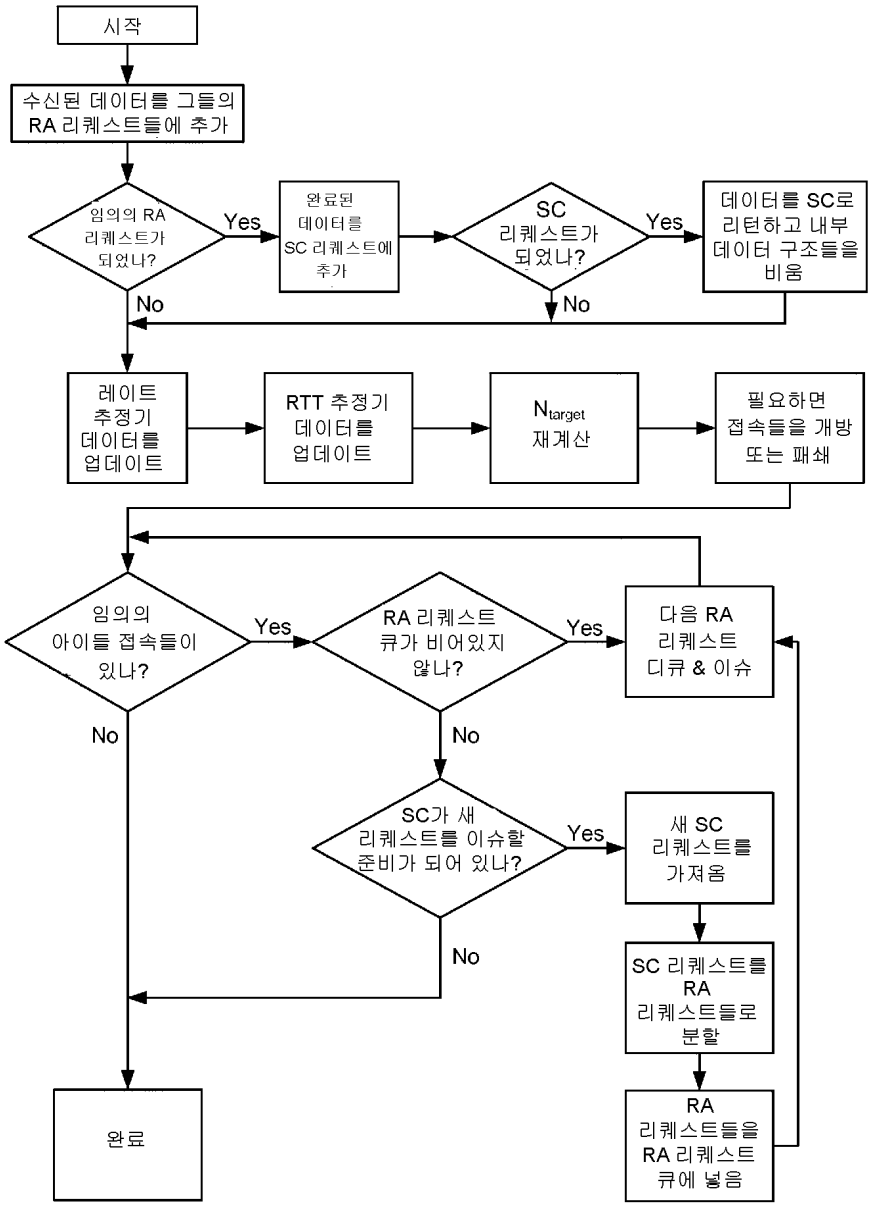
도면30


RTT 샘플 [ms]
2000
1500
1000
500
0
0
1
2
3
all
RTT
2.5e+06
2e+06
1.5e+06
1e+06
500000
0
0
5
10
15
20
25
30
35
시간 [s]
레이트 [bits/s]

도면31


레이트 [bits/s]
0
500000
1e+06
1.5e+06
2e+06
2.5e+06
시간 [s]
0
5
10
15
20
25
30
35
RTT 추정 [ms]
0
500
1000
1500
2000
RTT
all
0
1
2
3

도면32



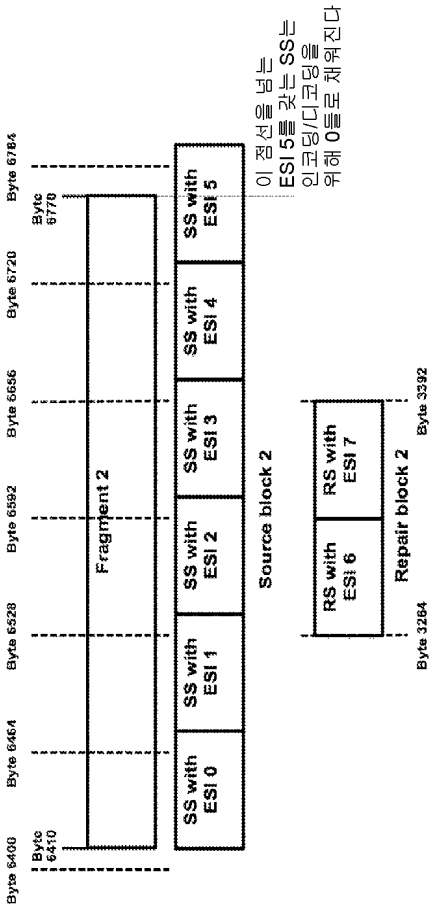
도면33

$S \leftarrow$ the size of the original request
$N_{target} \leftarrow$ the target number of connections
$R \leftarrow$ rate estimate over the last T_{wn}
$RTT \leftarrow$ RTT estimate
if no valid RTT estimate or no valid rate estimate is known **then**
    $n \leftarrow N_{target}$
    return n
end if
$D \leftarrow \max(D_{min}, c_s \cdot RTT)$
if $S < R \cdot D$ **then**
    $n \leftarrow \left\lceil \frac{N_{target} \cdot S}{R \cdot D} \right\rceil$
else
    $n \leftarrow N_{target}$
end if
return n

도면34

```
a := 1/(n + 1)
Let v[1], ..., v[n - 1] be iid random variables chosen uniformly
in the range [0,1]
Sort the v[] in ascending order
for i = 1, ..., n - 1:
      w[i] := (1 - a)*i/n + a*v[i]
w[n] := 1
prev := 0
for i = 1, ..., n
      x := floor(w[i] * S)
      s[i] := x - prev
      prev := x
Sort the vector s[] in decreasing size order
return s[1], ..., s[n]
```

도면35



도면36

MLB 파라미터 세트:

T	lambda	mu
0	0.4*R	0.9

보수적인 파라미터 세트:

t	lambda	mu
0	0.2	0.2
1	0.4	0.3
2	0.5	0.4
3	0.5	0.7
4	0.5	0.9

중도적인 파라미터 세트:

t	lambda	mu
0	0.4	0.3
1	0.5	0.4
2	0.6	0.8
3	0.7	1.0
4	0.8	1.2

공격적인 파라미터 세트:

t	lambda	mu
0	0.6	0.5
1	0.7	0.6
2	0.8	1.0
3	0.9	1.5
4	1.0	2.0