(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2016년02월18일
(11) 등록번호 10-1591238
(24) 등록일자 2016년01월28일

(51) 국제특허분류(Int. Cl.)
H04L 29/08 (2006.01) *H03M 13/37* (2006.01)
(21) 출원번호 10-2014-7014698
(22) 출원일자(국제) 2012년11월01일
심사청구일자 2014년05월30일
(85) 번역문제출일자 2014년05월30일
(65) 공개번호 10-2014-0089405
(43) 공개일자 2014년07월14일
(86) 국제출원번호 PCT/US2012/063115
(87) 국제공개번호 WO 2013/067219
국제공개일자 2013년05월10일
(30) 우선권주장
13/563,590 2012년07월31일 미국(US)
(뒷면에 계속)
(56) 선행기술조사문헌
US20050182842 A1
US20100217887 A1
David Gomez-Barquero et al. "Multicast Delivery of File Download Services in Evolved 3G Mobile Networks With HSDPA and MBMS", IEEE Transactions on Broadcasting, Vol.55, No.4, pp.742-751(2009.12.01.)

(73) 특허권자
퀄컴 인코포레이티드
미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
(72) 발명자
루비, 마이클 조지
미국 92121 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
레웅, 니콜라이, 콘라드
미국 92121 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
(뒷면에 계속)
(74) 대리인
특허법인 남앤드남

전체 청구항 수 : 총 44 항 심사관 : 강철수

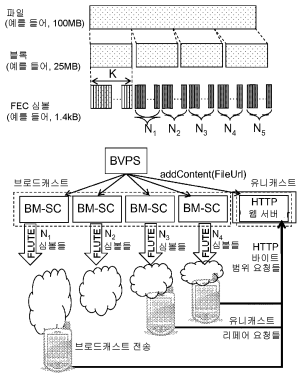
(54) 발명의 명칭 **ＨＴＴＰ 서버들 사이의 소스 데이터 및 리페어 데이터의 할당에 의한 컨텐츠 전달 시스템**

(57) 요 약

데이터 객체들은 패킷 교환 네트워크를 통해 전달되고, 수신기들은, 어떤 소스 심볼들 또는 서브-심볼들이 필요로 하거나 누락하고 있는 지에 기반으로 하여 필요에 따라 추가적인 심볼들에 대한 요청들을 형성하기 위한 충분한 정보를 갖는, 브로드캐스팅 또는 멀티캐스팅된 리페어 심볼들과 같은 인코딩된 심볼들을 수신한다. 요청들은

(뒷면에 계속)

대 표 도 - 도29



유니캐스트 또는 요청 방식으로 행해질 수 있다. 요청하는 것 및 브로드캐스팅하는 것은 상이한 엔티티들에 의해 행해질 수도 있다. 브로드캐스트 서버는 리페어 심볼들을 발생 및 저장할 수 있는 한편, 소스 서버는 컨텐츠를 소스 형태로 저장할 수 있다. 요청은 URL, 시작 위치 및 길이와 같은 유니캐스트 HTTP 바이트-범위 요청일 수 있다. 요청들은 파일들의 시작 위치들과 정렬될 수도 있다. 수신기는 파일에서 심볼들 또는 서브-심볼들의 시작 및 종료 바이트 위치들을 계산할 수 있고, 기존의 HTTP 서버들이 파일 리페어를 위하여 이용가능하다는 표시들을 얻을 수 있다. 다수의 수신기들로부터의 바이트-범위 요청들이 중첩할 때, 리페어 서버들은 리페어 데이터의 브로드캐스트를 요청할 수 있다.

(72) 발명자

골미에, 랄프, 아크람

미국 92121 캘리포니아주 샌 디에고 모어하우스 드라이브 5775

스톡해머, 토마스

미국 92121 캘리포니아주 샌 디에고 모어하우스 드라이브 5775

(30) 우선권주장

61/554,434 2011년11월01일 미국(US)
61/589,855 2012년01월23일 미국(US)
61/614,408 2012년03월22일 미국(US)
61/645,562 2012년05월10일 미국(US)
61/647,414 2012년05월15일 미국(US)

특허청구의 범위

청구항 1

전자 디바이스 또는 시스템을 포함하는 클라이언트(client) 디바이스에 의해패킷-교환 네트워크를 통해 수신되는 하나 또는 그보다 많은 멀티미디어(multimedia) 객체들을 플레이아웃(playout)하는 방법으로서,

상기 하나 또는 그보다 많은 멀티미디어 객체들의 소스 데이터는, 상기 소스 데이터가 인코딩된 심볼들로부터 복원가능하도록 패킷들 내의 인코딩된 심볼들에 의해 표현되고,

상기 방법은,

a) 브로드캐스트 채널을 통해 인코딩된 심볼들을 수신하는 단계 - 멀티미디어 객체에 대해 수신된 각각의 인코딩된 심볼의 값은, 상기 멀티미디어 객체에 대한 소스 심볼들의 값들로부터 유도된(derived) 리페어(repair) 심볼 또는 상기 멀티미디어 객체에 대한 소스 심볼 중 하나를 포함함 -;

b) 재생(play back)되지 않을 데이터가 요청되지 않기 위해, 플레이아웃될 데이터에 대해 상기 클라이언트 디바이스에 의해 요청되는 임의의 추가적인 데이터를 제한하기(narrow)위해 상기 멀티미디어 객체 중 어떤 부분들이 플레이아웃하기 위한 것인지에 대한 표시를 결정하는 단계 - 상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들은, 상기 표시된 부분들의 복원을 위해 요청하기 위한 상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터의 양을 결정하기 위해 사용됨 -;

c) 상기 표시를 이용하여, 상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들을 복원하기 위해 상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터의 양을 결정하는 단계;

d) 하나 또는 그보다 많은 파일들의 하나 또는 그보다 많은 바이트 범위들의 대응하는 세트를 결정하는 단계 - 상기 대응하는 세트는 상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들을 복원하기 위해 필요한 상기 하나 또는 그보다 많은 파일들 내의 추가적인 데이터에 대응함 -;

e) 서버로 보내진(directed) 하나 또는 그보다 많은 요청들을 이용하여, 상기 대응하는 세트 중 적어도 일 부분에 대한 하나 또는 그보다 많은 요청들을 생성하는 단계 - 각각의 요청은 하나 또는 그보다 많은 바이트 범위들을 특정함 -;

f) 상기 하나 또는 그보다 많은 요청들을 전송하는 단계;

g) 상기 전송된 하나 또는 그보다 많은 요청들에 응답하여, 상기 요청된 추가적인 데이터의 적어도 일부를 수신하는 단계; 및

h) 상기 클라이언트 디바이스를 이용하여 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들을 복원함에 있어서, 상기 브로드캐스트(broadcast) 채널을 통해 수신된 상기 인코딩된 심볼들과 조합하여 상기 수신된 추가적인 데이터를 이용하는 단계를 포함하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 2

삭제

청구항 3

삭제

청구항 4

삭제

청구항 5

삭제

청구항 6

삭제

청구항 7

제 1 항에 있어서,

상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터는,

상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들의 소스 심볼들을 포함하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 8

제 1 항에 있어서,

상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터는,

상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들의 리페어 심볼들의 적어도 일부를 포함하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 9

제 1 항에 있어서,

상기 브로드캐스트 채널을 통해 수신된 인코딩된 심볼들은 모두 리페어 심볼들인,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 10

삭제

청구항 11

삭제

청구항 12

삭제

청구항 13

삭제

청구항 14

제 1 항에 있어서,

상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터는,

상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들의 소스 서브-심볼들을 포함하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 15

삭제

청구항 16

패킷-교환 네트워크를 통해 수신되는 하나 또는 그보다 많은 멀티미디어(multimedia) 객체들을 플레이아웃할 수 있는 클라이언트 디바이스로서,

상기 하나 또는 그보다 많은 멀티미디어 객체들의 소스 데이터는, 상기 소스 데이터가 인코딩된 심볼들로부터 복원가능하도록 패킷들 내의 인코딩된 심볼들에 의해 표현되고,

상기 클라이언트 디바이스는,

a) 브로드캐스트 채널을 통해 인코딩된 심볼들을 수신하기 위한 브로드캐스트 수신기 입력 - 멀티미디어 객체에 대해 수신된 각각의 인코딩된 심볼의 값은, 상기 멀티미디어 객체에 대한 소스 심볼들의 값들로부터 유도된(derived) 리페어(repair) 심볼 또는 상기 멀티미디어 객체에 대한 소스 심볼 중 하나를 포함함 -;

b) 재생(play back)되지 않을 데이터가 요청되지 않기 위해, 플레이아웃될 데이터에 대해 상기 클라이언트 디바이스에 의해 요청되는 임의의 추가적인 데이터를 제한하기(narrow)위해 상기 멀티미디어 객체 중 어떤 부분들이 플레이아웃하기 위한 것인지에 대한 표시를 결정하기 위한 로직 - 상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들은, 상기 표시된 부분들의 복원을 위해 요청하기 위한 상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터의 양을 결정하기 위해 사용됨 -;

c) 상기 표시를 이용하여, 상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들을 복원하기 위해 상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터의 양을 결정하기 위한 로직;

d) 하나 또는 그보다 많은 파일들의 하나 또는 그보다 많은 바이트 범위들의 대응하는 세트를 결정하기 위한 로직 - 상기 대응하는 세트는 상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들을 복원하기 위해 필요한 상기 하나 또는 그보다 많은 파일들 내의 추가적인 데이터에 대응함 -;

e) 서버로 보내진(directed) 하나 또는 그보다 많은 요청들을 이용하여, 상기 대응하는 세트 중 적어도 일 부분에 대한 하나 또는 그보다 많은 요청들을 생성하기 위한 로직 - 각각의 요청은 하나 또는 그보다 많은 바이트 범위들을 특정함 -;

f) 패킷-교환 네트워크 상에서 상기 하나 또는 그보다 많은 요청들을 전송하기 위한 네트워크 인터페이스;

g) 상기 전송된 하나 또는 그보다 많은 요청들에 대응하여, 상기 요청된 추가적인 데이터의 적어도 일부를 수신하기 위한 로직; 및

h) 상기 클라이언트 디바이스를 이용하여 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들을 복원함에 있어서, 상기 브로드캐스트(broadcast) 채널을 통해 수신된 상기 인코딩된 심볼들과 조합하여 상기 수신된 추가적인 데이터를 처리하기 위한 프로세서를 포함하는,

클라이언트 디바이스.

청구항 17

삭제

청구항 18

삭제

청구항 19

삭제

청구항 20

삭제

청구항 21

제 16 항에 있어서,

상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터는,

상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들의 소스 심볼들을 포함하는,

클라이언트 디바이스.

청구항 22

제 16 항에 있어서,

상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터는,

상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들의 리페어 심볼들의 적어도 일부를 포함하는,

클라이언트 디바이스.

청구항 23

제 16 항에 있어서,

상기 브로드캐스트 채널을 통해 수신된 인코딩된 심볼들은 모두 리페어 심볼들인,

클라이언트 디바이스.

청구항 24

삭제

청구항 25

삭제

청구항 26

삭제

청구항 27

삭제

청구항 28

제 16 항에 있어서,

상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터는,

상기 플레이아웃하기 위한 상기 멀티미디어 객체 중 표시된 부분들의 소스 서브-심볼들을 포함하는,

클라이언트 디바이스.

청구항 29

삭제

청구항 30

삭제

청구항 31

삭제

청구항 32

삭제

청구항 33

삭제

청구항 34

삭제

청구항 35

삭제

청구항 36

삭제

청구항 37

삭제

청구항 38

삭제

청구항 39

삭제

청구항 40

삭제

청구항 41

삭제

청구항 42

삭제

청구항 43

삭제

청구항 44

삭제

청구항 45

삭제

청구항 46

삭제

청구항 47

삭제

청구항 48

삭제

청구항 49

삭제

청구항 50

삭제

청구항 51

삭제

청구항 52

삭제

청구항 53

삭제

청구항 54

제 1 항에 있어서,

상기 클라이언트 디바이스를 이용하여 플레이아웃 하는 것은,

유니캐스트(unicast) 리페어 서버로부터 수신된 데이터에 기초하여 플레이아웃 하는 것을 포함하고,

상기 멀티미디어 객체 중 일 부분을 복원하기 위해 유니캐스트 리페어 서버 및 브로드캐스트로부터 충분한 데이터가 이용가능하다고 상기 클라이언트 디바이스가 결정할 때, 상기 멀티미디어 객체 중 다른 부분들을 유니캐스트 리페어 서버로부터 동시에 수신하는 동안 상기 일 부분을 플레이아웃 하는 것을 포함하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 55

제 1 항에 있어서,

상기 멀티미디어 객체 중 어떤 부분들이 플레이아웃하기 위한 것인지에 대한 표시를 결정하는 단계는,

상기 클라이언트 디바이스 상에서 실행하는 애플리케이션(application)에 의해 일어나는 액션들 또는 상기 멀티미디어 객체 중 어떤 부분들이 플레이아웃 될지를 표시하기 위해 상기 클라이언트 디바이스의 사용자에 의해 일어나는 액션들에 기초하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 56

제 55 항에 있어서,

상기 애플리케이션 또는 사용자에 의해 일어나는 액션들은,

시간적으로 나누어져 있고, 서로 다른 시간들에서 플레이아웃 하기 위해 상기 멀티미디어 객체의 서로 다른 부분들을 참조하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 57

제 55 항에 있어서,

상기 하나 또는 그보다 많은 요청들을 생성하는 단계, 상기 하나 또는 그보다 많은 요청들을 전송하는 단계, 상기 요청된 추가적인 데이터의 적어도 일부를 수신하는 단계 및 상기 수신된 추가적인 데이터를 이용하는 단계는 상기 애플리케이션 또는 사용자에 의해 일어나는 액션들에 대응하여 발생하는(occur),

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 58

제 55 항에 있어서,

상기 애플리케이션은 전체의 멀티미디어 객체가 플레이아웃하기 위한 것이라고 결정하고, 플레이아웃 하는 것은 상기 클라이언트 디바이스 상의 저장공간에 상기 복원된 멀티미디어 객체를 기록하는(write)것인,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 59

제 55 항에 있어서,

상기 멀티미디어 객체는 오디오/비디오 객체이고,

플레이아웃 하는 것은 디스플레이 스크린에 디스플레이 하기 위해 상기 클라이언트 디바이스 상의 멀티미디어 플레이어에 상기 멀티미디어 객체의 부분들에의 접근을 제공하는 것을 포함하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 60

제 1 항에 있어서,

상기 하나 또는 그보다 많은 파일 중 적어도 하나는 확장된-원래-순서(extended-original-order) HTTP 파일 포맷인,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 61

제 60 항에 있어서,

파일이 확장된-원래-순서 HTTP 파일 포맷임을 표시하는 시그널링(signaling)을 수신하고 처리하는 단계를 더 포함하고,

상기 시그널링은 유니버설 객체 심볼 식별자(universal object symbol identifier) 범위의 표시를 포함하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 62

제 1 항에 있어서,

상기 하나 또는 그보다 많은 파일들 중 적어도 하나는 상기 멀티미디어 객체의 원래-순서 HTTP 파일이고, 상기 적어도 하나의 파일 내의 데이터는 네이티브(native) 포맷인,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 63

제 1 항에 있어서,

상기 하나 또는 그보다 많은 파일들 중 적어도 하나는 상기 멀티미디어 객체의 유니버설 파일 심볼 식별자-순서 HTTP 파일인,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 64

제 1 항에 있어서,

상기 서버는,

파일 요청들 및 바이트 범위 요청들에 응답하고, 파일들 및 파일들의 바이트 범위들을 제공(server)하는 하이퍼텍스트(HyperText) 전송 프로토콜 서버인,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 65

제 64 항에 있어서,

HTTP 요청들은 HTTP 1.1 포맷 요청들이고,

HTTP 서버는 상기 HTTP 1.1 프로토콜을 지원하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 66

제 1 항에 있어서,

상기 멀티미디어 객체는 소스 블록들로 분할(partitioned)되고,

상기 플레이아웃하기 위해 멀티미디어 객체들 중 표시된 부분들을 포함하는 소스 블록들의 최소의 세트는 복원되는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 67

제 66 항에 있어서,

상기 하나 또는 그보다 많은 파일들 중 적어도 하나는 상기 멀티미디어 객체의 원래-순서 HTTP 파일이고,

최대(at most) 1 바이트 범위의 요청이 생성되고, 상기 소스 블록들의 최소의 세트 내의 각각의 소스 블록에 전송되고,

상기 소스 블록에 대한 바이트 범위의 요청은 상기 소스 블록으로부터의 소스 심볼들의 연속 프리픽스(consecutive prefix)에 대응하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 68

제 66 항에 있어서,

상기 멀티미디어 객체의 소스 블록들은 소스 서브-블록들 내에서 추가로 분할되고,

상기 플레이아웃하기 위해 멀티미디어 객체들 중 표시된 부분들을 포함하는 소스 서브-블록들의 최소의 세트는 복원되는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 69

제 68 항에 있어서,

상기 하나 또는 그보다 많은 파일들 중 적어도 하나는 상기 멀티미디어 객체의 원래-순서 HTTP 파일이고,

최대 1 바이트 범위의 요청이 생성되고, 상기 소스 서브-블록들의 최소의 세트 내의 각각의 소스 서브-블록에 전송되고,

상기 소스 서브-블록에 대한 바이트 범위의 요청은 상기 소스 서브-블록으로부터의 소스 서브-심볼들의 연속 프리픽스(consecutive prefix)에 대응하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 70

제 1 항에 있어서,

상기 하나 또는 그보다 많은 요청들 중 적어도 하나는 하나 또는 그보다 많은 리페어 심볼들을 포함하는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 71

제 1 항에 있어서,

상기 표시된 부분들의 복원을 위해 요청하기 위한 상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터의 양은,

각각의 멀티미디어 객체에 대해 상기 클라이언트 디바이스에 제공된 FEC(forward-error correcting) 객체 전송 정보(FEC Object Transmission Information)에 기초하여 결정되는,

하나 또는 그보다 많은 멀티미디어 객체들을 플레이아웃 하는 방법.

청구항 72

제 16 항에 있어서,

상기 멀티미디어 객체 중 어떤 부분들이 플레이아웃하기 위한 것인지에 대하여 표시하기 위해, 상기 클라이언트 디바이스의 사용자에 의해 실행가능하거나 상기 클라이언트 디바이스에 의해 실행가능한 애플리케이션을 더 포함하는,

클라이언트 디바이스.

청구항 73

제 72 항에 있어서,

복원된 멀티미디어 객체를 저장하기 위한 저장공간(storage)를 더 포함하는,

클라이언트 디바이스.

청구항 74

제 16 항에 있어서,

상기 멀티미디어 객체는 오디오/비디오 객체이고,

상기 클라이언트 디바이스는 상기 멀티미디어 객체를 디스플레이 하기 위한 디스플레이 스크린을 더 포함하는,

클라이언트 디바이스.

청구항 75

제 16 항에 있어서,

상기 하나 또는 그보다 많은 파일 중 적어도 하나는 확장된-원래-순서(extended-original-order) HTTP 파일 포맷인,

클라이언트 디바이스.

청구항 76

제 75 항에 있어서,

수신된 신호들은 파일이 확장된-원래-순서 HTTP 파일 포맷임을 표시하고,

상기 수신된 신호들은 유니버설 객체 심볼 식별자(universal object symbol identifier) 범위의 표시를 포함하는,

클라이언트 디바이스.

청구항 77

제 16 항에 있어서,

상기 하나 또는 그보다 많은 파일들 중 적어도 하나는 상기 멀티미디어 객체의 원래-순서 HTTP 파일이고, 상기 적어도 하나의 파일 내의 데이터는 네이티브(native) 포맷인,

클라이언트 디바이스.

청구항 78

제 16 항에 있어서,

상기 하나 또는 그보다 많은 파일들 중 적어도 하나는 상기 멀티미디어 객체의 유니버설 파일 심볼 식별자-순서 HTTP 파일인,

클라이언트 디바이스.

청구항 79

제 16 항에 있어서,

상기 서버는,

파일 요청들 및 바이트 범위 요청들에 응답하고, 파일들 및 파일들의 바이트 범위들을 제공(server)하는 하이퍼텍스트(HyperText) 전송 프로토콜 서버인,

클라이언트 디바이스.

청구항 80

제 79 항에 있어서,

HTTP 요청들은 HTTP 1.1 포맷 요청들이고,

상기 HTTP 서버는 상기 HTTP 1.1 프로토콜을 지원하는,

클라이언트 디바이스.

청구항 81

제 16 항에 있어서,

상기 하나 또는 그보다 많은 요청들 중 적어도 하나는 하나 또는 그보다 많은 리페어 심볼들을 포함하는,

클라이언트 디바이스.

청구항 82

제 16 항에 있어서,

상기 표시된 부분들의 복원을 위해 요청하기 위한 상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터의 양은,

각각의 멀티미디어 객체에 대해 상기 클라이언트 디바이스에 제공된 FEC(forward-error correcting) 객체 전송 정보(FEC Object Transmission Information)에 기초하여 결정되는,

클라이언트 디바이스.

청구항 83

전자 디바이스 또는 시스템을 포함하는 클라이언트(client) 디바이스에 의해, 적어도 패킷-교환 네트워크를 통해 하나 또는 그보다 많은 멀티미디어 객체들을 수신하는 방법으로서,

상기 하나 또는 그보다 많은 멀티미디어 객체들의 소스 데이터는, 상기 소스 데이터가 인코딩된 심볼들로부터 복원가능하도록 패킷들 내의 인코딩된 심볼들에 의해 표현되고,

상기 방법은,

a) 심볼들의 전송 타입을 표시하는 신호를 수신하는 단계 - 적어도 하나의 전송 타입은 심볼들이 브로드캐스트

채널을 통해 수신될, 그리고 추가적인 데이터 가 유니캐스트 채널을 통해 수신될 수 있는 전송타입이고, 상기 브로드캐스트 채널을 통해 수신된 심볼들 각각은 리페어 심볼임 -;

b) 상기 브로드캐스트 채널을 통해 리페어 심볼들을 수신하는 단계 - 각각의 리페어 심볼의 값은 상기 멀티미디어 객체에 대한 소스 심볼들의 값들로부터 유도되는(derived) 값임 -;

c) 재생(play back)되지 않을 데이터가 요청되지 않기 위해, 플레이아웃 될 데이터에 대해 상기 클라이언트 디바이스에 의해 요청되는 임의의 추가적인 데이터를 제한하기(narrow)위해 상기 멀티미디어 객체 중 어떤 부분들이 플레이아웃 하기 위한 것인지에 대한 표시를 결정하는 단계 - 상기 플레이아웃 하기 위한 상기 멀티미디어 객체 중 표시된 부분들은, 상기 표시된 부분들의 복원을 위해 요청하기 위한 상기 수신된 인코딩된 심볼들을 초과하여 필요한 추가적인 데이터의 양을 결정하기 위해 사용됨 -;

d) 상기 표시를 이용하여, 상기 플레이아웃 하기 위한 상기 멀티미디어 객체 중 표시된 부분들을 복원하기 위해 상기 수신된 리페어 심볼들을 초과하는 추가적인 데이터의 양을 결정하는 단계;

e) 하나 또는 그보다 많은 파일들의 하나 또는 그보다 많은 바이트 범위들의 대응하는 세트를 결정하는 단계 - 상기 대응하는 세트는 상기 추가적인 데이터에 대응함 -;

f) 서버로 보내진(directed) 하나 또는 그보다 많은 요청들을 이용하여, 상기 대응하는 세트 중 적어도 일 부분에 대한 하나 또는 그보다 많은 요청들을 생성하는 단계 - 각각의 요청은 하나 또는 그보다 많은 바이트 범위들을 특정함 -;

g) 상기 하나 또는 그보다 많은 요청들을 전송하는 단계;

h) 상기 전송된 하나 또는 그보다 많은 요청들에 응답하여, 상기 요청된 추가적인 데이터의 적어도 일부를 수신하는 단계; 및

i) 상기 클라이언트 디바이스를 이용하여 플레이아웃 하기 위한 상기 멀티미디어 객체 중 표시된 부분들을 복원함에 있어서, 상기 브로드캐스트 채널을 통해 수신된 상기 리페어 심볼들과 조합하여 상기 수신된 추가적인 데이터를 이용하는 단계를 포함하는,

하나 또는 그보다 많은 멀티미디어 객체들을 수신하는 방법.

청구항 84

제 83 항에 있어서,

상기 리페어 심볼들을 포함하는 복수의 브로드캐스트 스트림들을 수신하는 단계를 더 포함하고,

상기 복수의 브로드캐스트 스트림들은 독립적인,

하나 또는 그보다 많은 멀티미디어 객체들을 수신하는 방법.

청구항 85

클라이언트 디바이스에 메타데이터(metadata)를 전송하는 방법으로서,

상기 메타데이터는 유니캐스트 인코딩된 심볼들과 결합된 브로드캐스트된 인코딩된 심볼들로서 상기 클라이언트 디바이스에 의해 수신될 멀티미디어 객체에 대한 정보를 제공하고,

상기 인코딩된 심볼들은 상기 멀티미디어 객체가 수신된 인코딩된 심볼들로부터 복원가능하도록 이루어지고,

상기 방법은,

a) 브로드캐스트 채널을 통한 전송에 대한 브로드캐스트 정보를 표시하는 제 1 신호를 전송하는 단계 - 상기 브로드캐스트 채널을 통해 전송된 각각의 심볼은 리페어 심볼이고, 각각의 리페어 심볼의 값은 상기 멀티미디어 객체에 대한 소스 심볼들의 값들로부터 유도된(derived) 값임 -; 및

b) 재생(play back)되지 않을 데이터가 요청되지 않기 위해, 플레이아웃 하기 위해 표시된 상기 멀티미디어 객체의 부분들을 복원하기 위해 수신된 리페어 심볼들을 초과하여 필요한 추가적인 심볼들에 대한 하나 또는 그보다 많은 요청들을 생성하기 위한 정보를 표시하는 제 2 신호를 전송하는 단계를 포함하고,

추가적인 심볼들에 대한 요청은 파일의 표시 및 바이트 범위 요청을 포함하고, 상기 파일은 적어도 부분적으로

상기 브로드캐스트 채널을 통해 수신되는 상기 멀티미디어 객체의 부분들에 대응하는,

클라이언트 디바이스에 메타데이터를 전송하는 방법.

청구항 86

제 85 항에 있어서,

상기 추가적인 심볼들에 대한 요청은 상기 파일에 대한 URL 및 상기 파일 내의 바이트 범위를 포함하는 HTTP 1.1 프로토콜 요청을 포함하는,

클라이언트 디바이스에 메타데이터를 전송하는 방법.

청구항 87

제 86 항에 있어서,

복수의 전송기(transmitter)들로부터 복수의 브로드캐스트 스트림들을 전송하는 단계를 더 포함하고,

상기 복수의 브로드캐스트 스트림들은 중첩(overlap)되지 않고, 리페어 심볼들을 포함하는,

클라이언트 디바이스에 메타데이터를 전송하는 방법.

명 세 서

기 술 분 야

[0001] 관련 출원들에 대한 상호-참조

[0002] 본 출원은 "Unicast Repair Service and Server Augmenting Broadcast File Delivery System"(안건 참조 번호 121519P1) 이라는 명칭으로 2011년 11월 1일자로 출원된 미국 특허 가출원 제61/554,434호, "Unicast Repair Service and Server Augmenting Broadcast File Delivery System"(안건 참조 번호 121519P2) 이라는 명칭으로 2012년 1월 23일자로 출원된 미국 특허 가출원 제61/589,855호, "Unicast Repair Service and Server Augmenting Broadcast File Delivery System"(안건 참조 번호 121519P3) 이라는 명칭으로 2012년 3월 22일자로 출원된 미국 특허 가출원 제61/614,408호, "Content Delivery System with Allocation of Source Data and Repair Data Among HTTP Servers"(안건 참조 번호 121519P4) 이라는 명칭으로 2012년 5월 10일자로 출원된 미국 특허 가출원 제61/645,562호, 및 "Content Delivery System with Allocation of Source Data and Repair Data Among HTTP Servers"(안건 참조 번호 121519P5) 이라는 명칭으로 2012년 5월 15일자로 출원된 미국 특허 가출원 제61/647,414호를 우선권 주장하는 정규 출원이며, 그 전체 개시 내용은 모든 목적들을 위하여 본원에 인용에 의해 포함된다.

[0003] 본 개시 내용은 다음의 공통으로 양도된 특허들 또는 출원들에 관한 것일 수도 있고, 그 각각은 모든 목적들을 위하여 그 전체가 본원에 인용에 의해 포함된다:

[0004] 1) "Information Additive Code Generator and Decoder for Communication Systems"라는 명칭으로 Michael G. Luby 에게 허여된 미국 특허 제6,307,487호(이하 "Luby I");

[0005] 2) "Information Additive Group Code Generator and Decoder for Communication Systems"라는 명칭으로 Michael G. Luby 에게 허여된 미국 특허 제 6,320,520호(이하 "Luby II");

[0006] 3) "Systems and Processes for Decoding a Chain Reaction Code Through Inactivation"이라는 명칭으로 M. Amin Shokrollahi 에게 허여된 미국 특허 제6,856,263호(이하 "Shokrollahi I");

[0007] 4) "Systematic Encoding and Decoding of Chain Reaction Codes"라는 명칭으로 M. Amin Shokrollahi 에게 허여된 미국 특허 제6,909,383호(이하 "Shokrollahi II");

[0008] 5) "Multi-Stage Code Generator and Decoder for Communication Systems"라는 명칭으로 M. Amin Shokrollahi 에게 허여된 미국 특허 제7,068,729호(이하 "Shokrollahi III");

[0009] 6) "File Download and Streaming System"이라는 명칭으로 Michael G. Luby, M. Amin Shokrollahi 및 Mark Watson에게 허여된 미국 특허 제7,418,651호(이하 "Luby III");

[0010] 7) "In-Place Transformations with Applications to Encoding and Decoding Various Classes of Codes"라는 명칭인, Michael G. Luby 및 M. Amin Shokrollahi 명의의 미국 특허 공개 제2006/0280254호(이하 "Luby IV");

[0011] 8) "Multiple-Field Based Code Generator and Decoder for Communications Systems"라는 명칭인, M. Amin Shokrollahi 명의의 미국 특허 공개 제2007/0195894호(이하 "Shokrollahi IV");

[0012] 9) "Method and Apparatus Employing FEC Codes with Permanent Inactivation of Symbols for Encoding and Decoding Processes"라는 명칭으로, M. Amin Shokrollahi 등의 명의로, 2009년 10월 23일자로 출원된 미국 특허 출원 제12/604,773호(이하 "Shokrollahi V");

[0013] 10) "Methods and Apparatus Employing FEC Codes with Permanent Inactivation of Symbols for Encoding and Decoding Processes"라는 명칭으로, Michael G. Luby 명의로, 2009년 8월 19일자로 출원된 미국 가출원 제61/235,285호(이하 "Luby V"); 및

[0014] 11) "Methods and Apparatus Employing FEC Codes with Permanent Inactivation of Symbols for Encoding and Decoding Processes"라는 명칭으로, Michael G. Luby, M. Amin Shokrollahi 명의로, 2010년 8월 18일자로 출원된 미국 특허 출원 제12/859,161호(이하 "Shokrollahi VI").

[0015] 12) "Broadcast Multimedia Storage and Access Using Page Maps when Asymmetric Memory is Used"라는 명칭으로, Michael G. Luby, Thadi M. Nagaraj 명의로, 2011년 8월 9일자로 출원된 미국 특허 출원 제13/206,418호(이하 "Luby VI").

[0016] 13) "Content Delivery System with Allocation of Source Data and Repair Data Among HTTP Servers"라는 명칭으로, Michael G. Luby, Nikolai Leung, Ralph Gholmieh, Thomas Stockbammer 명의로, 2012년 5월 10일자로 출원된 미국 특허 출원 제61/645,562호(이하 "Luby VII").

[0017] 참조 문헌들

[0018] 이하의 참조 문헌들 각각은 모든 목적들을 위하여 그 전체가 인용에 의해 본원에 포함된다:

[0019] [Albanese96], 또는 [PET] "Priority Encoding Transmission", Andres Albanese, Johannes Blomer, Jeff Edmonds, Michael Luby, 및 Madhu Sudan, IEEE Transactions on Information Theory, vol. 42, no. 6 (November 1996);

[0020] [Albanese97], 또는 [PET-Patent] "System for Packetizing Data Encoded Corresponding to Priority Levels Where Reconstructed Data Corresponds to Fractionalized Priority Level and Received Fractionalized Packets"라는 명칭의 A. Albanese, M. Luby, J. Blomer, J. Edmonds에 대한 미국 특허 제5,617,541호(1997년 4월 1일자로 허여됨);

[0021] [ALC] Luby, M., Watson, M., Vicisano, L., "Asynchronous Layered Coding (ALC) Protocol Instantiation", IETF RFC 5775 (2010년 4월);

[0022] [FEC BB] Watson, M., Luby, M., 및 L. Vicisano, "Forward Error Correction (FEC) Building Block", IETF RFC 5052 (2007년 8월);

[0023] [FLUTE] Paila, T., Luby, M., Lehtonen, R., Roca, V., Walsh, R., "FLUTE -File Delivery over Unidirectional Transport", IETF RFC 3926 (2004년 10월);

[0024] [LCT] Luby, M., Watson, M., Vicisano, L., "Layered Coding Transport (LCT) Building Block", IETF RFC 5651 (2009년 10월)

[0025] [Luby2007], 또는 [Raptor-RFC-5053] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, "Raptor Forward Error Correction Scheme for Object Delivery", IETF RFC 5053 (2007년 9월);

[0026] [Luby2002] Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley, M., 및 J. Crowcroft, "The Use of Forward Error Correction (FEC) in Reliable Multicast", IETF RFC 3453 (2002년 12월);

[0027] [Matsuoka], 또는 [LDPC-Extensions] "Low-Density Parity-Check Code Extensions Applied for Broadcast-Communication Integrated Content Delivery", Hosei Matsuoka, Akira Yamada, 및 Tomoyuki Ohya, Research Laboratories, NTT DOCOMO, Inc., 3-6, Hikari-No-Oka, Yokosuka, Kanagawa, 239-8536, Japan; 및

[0028] [RaptorQ-RFC-6330] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, L. Minder, "RaptorQ Forward Error Correction Scheme for Object Delivery", IETF RFC 6330, Reliable Multicast Transport (2011년 8월);

[0029] [Roca], 또는 [LDPC-RFC-5170] V. Roca, C. Neumann, D. Furodet, "Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes", IETF RFC 5170 (2008년 6월).

[0030] 본 발명은 통신 시스템들에서의 데이터의 인코딩 및 디코딩에 관한 것으로, 특히, 통신된 데이터 내의 에러들 및 갭(gap)들을 설명하기 위한 데이터를 효율적인 방식으로 인코딩 및 디코딩하며 상이한 파일 전달 방법들을 처리하는 통신 시스템들에 관한 것이다.

배 경 기 술

[0031] 통신 채널을 통한 전송자 및 수신자 사이의 파일들의 송신을 위한 기술들은 많은 문헌의 주제이다. 바람직하게는, 수신자는 전송자에 의해 채널을 통해 송신된 데이터의 정확한 복사본을 일부의 확실성 레벨로 수신할 것을 희망한다. 채널이 완벽한 충실도(모든 물리적으로 실현가능한 시스템들을 대부분 커버함)를 가지지 않는 경우, 하나의 관심은 송신 시에 손실 또는 왜곡된 데이터를 처리하는 방법이다. 손실된 데이터(소거들)는 오류를 일으킨 데이터(에러들)보다 처리하기가 종종 더 용이하며, 이것은 수신자가 오류를 일으킨 데이터가 언제 에러로 수신된 데이터인지를 항상 말할 수 없기 때문이다. 소거들 및/또는 에러들을 수정하기 위하여 많은 에러-수정 코드들이 개발되었다.

[0032] 전형적으로, 이용되는 특정한 코드는 채널을 통해 데이터가 송신되고 있는 그러한 채널의 불충실도(infidelity)들과, 송신되고 있는 데이터의 성질에 대한 일부 정보에 기반으로 하여 선택된다. 예를 들어, 채널이 불충실도의 긴 기간들을 갖는 것으로 알려진 경우, 버스트 에러 코드가 그 애플리케이션에 최적으로 적합할 수도 있다. 오직 짧은 빈번하지 않은 에러들이 예상될 경우, 간단한 패리티 코드(parity code)가 최적일 수도 있다.

[0033] 여기에서 이용되는 바와 같이, "소스 데이터(source data)"는, 하나 또는 그보다 많은 전송기들에서 이용가능하며, 에러들 및/또는 소거들 등을 갖거나 갖지 않는 송신된 시퀀스(sequence)로부터의 복원에 의해, 수신기가 얻기 위해 이용되는 데이터를 지칭한다. 여기에서 이용되는 바와 같이, "인코딩된 데이터"는, 전송되어 소스 데이터를 복원하거나 얻기 위하여 이용될 수 있는 데이터를 지칭한다. 간단한 경우, 인코딩된 데이터는 소스 데이터의 복사본이지만, 수신된 인코딩된 데이터가 (에러들 및/또는 소거들로 인해) 송신된 인코딩된 데이터와 상이할 경우에는, 이 간단한 경우에 있어서 소스 데이터는 소스 데이터에 대한 추가적인 데이터가 부재하여 완전히 복원가능하지 않을 수도 있다. 송신은 공간 또는 시간을 통한 것일 수도 있다. 더욱 복잡한 경우, 인코딩된 데이터는 변환 시에 소스 데이터에 기반으로 하여 발생되고, 하나 또는 그보다 많은 전송기들로부터 수신기들로 송신된다. 소스 데이터가 인코딩된 데이터의 일부인 것으로 판명될 경우에 인코딩은 "체계적(systematic)"인 것으로 말해진다. 체계적 인코딩의 간단한 예에서, 소스 데이터에 대한 중복 정보는 인코딩된 데이터를 형성하기 위하여 소스 데이터의 끝에 첨부된다.

[0034] 또한 여기에서 이용되는 바와 같이, "입력 데이터"는 순방향-에러 정정(forward-error correcting; FEC) 인코더 장치 또는 FEC 인코더 모듈, 컴포넌트, 단계, 등("FEC 인코더")의 입력에 존재하는 데이터를 지칭하고, "출력 데이터"는 FEC 인코더의 출력에 존재하는 데이터를 지칭한다. 이에 대응하여, 출력 데이터는 FEC 디코더의 입력에 존재할 것으로 예상될 것이고, FEC 디코더는 프로세싱한 출력 데이터에 기반으로 하여 입력 데이터, 또는 그 대응하는 것을 출력할 것으로 예상될 것이다. 일부의 경우들에 있어서, 입력 데이터는 소스 데이터이거나 이를 포함하고, 일부의 경우들에 있어서, 출력 데이터는 인코딩된 데이터이거나 이를 포함한다. 예를 들어, FEC 인코더의 입력 이전에 프로세싱이 없을 경우, 입력 데이터는 소스 데이터일 것이다. 그러나, 일부의 경우들에 있어서, 소스 데이터는 소스 데이터 대신에 FEC 인코더에 제공되는 중간 데이터를 발생하기 위하여 상이한 형태(예를 들어, 정적 인코더, 역 인코더 또는 또 다른 프로세스)로 프로세싱된다.

[0035] 일부의 경우들에 있어서, 전송기 디바이스 또는 전송기 프로그램 코드는 하나를 초과하는 FEC 인코더를 포함할 수도 있고, 즉, 소스 데이터는 일련의 복수의 FEC 인코더들에서 인코딩된 데이터로 변환된다. 유사하게 수신기에서는, 수신된 인코딩된 데이터로부터 소스 데이터를 발생하기 위해 적용된 하나를 초과하는 FEC 디코더가 있을 수도 있다.

[0036] 데이터는 심볼(symbol)들로 구획(partition)된 것으로 생각될 수 있다. 인코더는 소스 심볼들 또는 입력 심볼들의 시퀀스로부터 인코딩된 심볼들 또는 출력 심볼들을 발생하는 컴퓨터 시스템, 디바이스, 전자 회로, 등이고, 디코더는 수신된 또는 복원된 인코딩된 심볼들 또는 출력 심볼들로부터 소스 심볼들 또는 입력 심볼들

의 시퀀스를 복원하는 대응부분이다. 인코더 및 디코더는 채널에 의해 시간 및/또는 공간에 있어서 분리되어 있고, 임의의 수신된 인코딩된 심볼들은 대응하는 송신된 인코딩된 심볼들과 정확하게 동일하지는 않을 수도 있으며 이들은 이들이 송신되었던 것과 정확하게 동일한 시퀀스에서 수신되지 않을 수도 있다. 심볼이 실제로 비트 스트림으로 부서지든지 그렇지 않든지 간에, 심볼의 "크기"는 비트들로 측정될 수 있고, 여기서 심볼은 2^M 심볼들의 알파벳으로부터 선택될 때에 M 비트들의 크기를 가진다. 여기의 예들 중의 다수에서는, 심볼들이 옥테트(octet)들로 측정되고 코드들은 256 개의 가능성들의 필드 상에 있을 수도 있지만(각각의 옥테트 내에는 256 개의 가능한 8-비트 패턴들이 있음), 데이터 측정의 상이한 단위들이 이용될 수 있다는 것을 이해해야 하고, 데이터를 다양한 방식들로 측정하는 것은 잘 알려져 있다. 일반적인 문헌에서는, 8-비트 값을 표시하기 위하여 용어 "바이트"가 때때로 용어 "옥테트"와 상호 교환가능하게 이용되지만, 일부 문맥들에서는 "바이트"가 X-비트 값을 표시하고 여기서 X는 8과 동일하지 않고, 예를 들어, X = 7이다. 여기서, 용어 "옥테트" 및 "바이트"는 상호 교환가능하게 이용된다. 이와 달리 표시되지 않으면, 여기의 예들은 심볼당 특정한 정수 또는 비정수(noninteger) 수의 비트들로 제한되지 않는다.

[0037] Luby I 은 에러 정정을 연산-효율적, 메모리-효율적 및 대역폭-효율적인 방식으로 다루기 위하여 연쇄 반응(chain reaction) 코드들과 같은 코드들의 이용을 설명한다. 연쇄 반응 인코더에 의해 생성된 인코딩된 심볼들의 하나의 특성은, 충분한 인코딩된 심볼들이 수신되자마자 수신기가 원래의 파일을 복원할 수 있다는 것이다. 구체적으로, 원래의 K 소스 심볼들을 높은 확률로 복원하기 위하여, 수신기는 대략 K+A 인코딩된 심볼들을 필요로 한다.

[0038] 주어진 상황에 대한 "절대 수신 오버헤드"는 값 A에 의해 나타내어지는 반면, "상대 수신 오버헤드"는 비율 A/K로서 계산될 수 있다. 절대 수신 오버헤드는 정보 이론의 최소 데이터량 이상으로 얼마나 많은 여분의 데이터가 수신될 필요가 있는지에 대한 척도이고, 그것은 디코더의 신뢰성에 의존할 수도 있으며 소스 심볼들의 수 K의 함수로서 변동할 수도 있다. 유사하게, 상대 수신 오버헤드 A/K는 복원되고 있는 소스 데이터의 크기에 대한 정보 이론의 최소 데이터량 이상으로 얼마나 많은 여분의 데이터가 수신될 필요가 있는지에 대한 척도이고, 디코더의 신뢰성에 또한 의존할 수도 있으며 소스 심볼들의 수 K의 함수로서 변동할 수도 있다.

[0039] 연쇄 반응 코드들은 패킷 기반 네트워크를 통한 통신에 매우 유용하다. 그러나, 이들은 때때로 상당히 연산에 있어서 집약적일 수 있다. 소스 심볼들이 연쇄 반응 또는 또 다른 레이트리스 코드(rateless code)를 이용하여 인코딩하는 동적 인코더 이전에 정적 인코더를 이용하여 인코딩될 경우, 디코더는 더욱 자주 또는 더욱 용이하게 디코딩할 수 있을 수도 있다. 이러한 디코더들은 예를 들어, Shokrollahi I에 도시되어 있다. 거기에 도시된 예들에서, 소스 심볼들은 인코딩된 심볼들인 출력 심볼들을 생성하는 동적 인코더에 대한 입력 심볼들인 출력 심볼들을 생성하는 정적 인코더에 대한 입력 심볼들이고, 동적 인코더는 입력 심볼들의 수에 대한 고정된 레이트가 아닌 수량으로 출력 심볼들의 수를 발생할 수 있는 레이트리스 인코더이다. 정적 인코더는 하나를 초과하는 고정된 레이트 인코더를 포함할 수도 있다. 예를 들어, 정적 인코더는 해밍 인코더(Hamming encoder), 저밀도 패리티-검사("low-density parity-check; LDPC") 인코더, 고밀도 패리티-검사("high-density parity-check; HDPC") 인코더, 및/또는 유사한 것을 포함할 수도 있다.

[0040] 연쇄 반응 코드들은, 일부 심볼들이 수신된 심볼들로부터 디코더에서 복원되므로, 그러한 심볼들이 결국 더욱 더 많은 심볼들을 복원하기 위해 이용될 수도 있는 추가적인 심볼들을 복원하기 위해 이용될 수도 있다는 특성을 가진다. 바람직하게는, 희망하는 심볼들의 전부가 수신된 심볼들의 풀(pool)이 모두 이용되기 전에 복원되도록, 디코더에서 구하는 심볼의 연쇄 반응이 계속될 수 있다. 바람직하게는, 연쇄 반응 인코딩 및 디코딩 프로세스들을 수행하는 연산 복잡도가 낮다. 희망하는 심볼들은 원래의 소스 심볼들의 전부를 완전히 복원하기 위해 필요한 심볼들의 전부, 또는 원래의 소스 심볼들의 전부보다는 적은 일부의 희망하는 완성도의 레벨일 수도 있다.

[0041] 디코더에서의 복원 프로세스는 어느 심볼들이 수신되었는지를 결정하는 것, 원래의 입력 심볼들을 수신되었던 그러한 인코딩된 심볼들에 맵핑할 행렬을 생성하는 것, 그 다음으로 행렬을 반전하는 것, 그리고 반전된 행렬과 수신된 인코딩된 심볼들의 벡터와의 행렬 승산을 수행하는 것을 포함할 수도 있다. 전형적인 시스템에서는, 이것의 무차별 구현은 과도한 연산 노력 및 메모리 요건들을 소비할 수 있다. 물론, 수신된 인코딩된 심볼들의 특정한 세트에 대해서는, 원래의 입력 심볼들의 전부를 복원하는 것이 불가능할 수도 있지만, 가능한 경우에도, 그것은 결과를 연산하기에 매우 연산적으로 비용이 많이 들 수도 있다.

[0042] 순방향 에러 정정("FEC") 객체 송신 정보("Object Transmission Information; OTI"), 또는 "FEC OTI"

[0043] 수신기가 수신하는(또는 추론할 수 있는) FEC OTI에 기반으로 하여, 수신기는 파일 전송의 소스 블록 및 서브-

블록 구조를 결정할 수 있다. [Raptor-RFC-5053] 및 [RaptorQ-RFC-6330]에서는, FEC 페이로드 ID가 (SBN, ESI)이고, 여기의 도 1에 예시된 바와 같이, [Raptor-RFC-5053]에서는, 소스 블록 번호(SBN)가 16 비트들이고 인코딩 심볼 ID(ESI)가 16 비트들인 반면, [RaptorQ-RFC-6330]에서는, SBN이 8 비트들이고 ESI가 24 비트들이다. 이 FEC 페이로드 ID 포맷의 하나의 단점은, SBN 및 ESI에 할당하기 위한 FEC 페이로드 ID의 비트들의 수를 사전-결정해야 하고 모든 파일 전달 파라미터들에 대해 적합할 적당한 혼합(mix)을 결정하는 것이 때때로 어렵다는 점이다.

[0044] 예를 들어, [Raptor-RFC-5053]을 이용할 때, 일부의 경우들에 있어서, 8,192 소스 심볼들을 갖는 소스 블록이 있을 수도 있고 이에 따라 인코딩된 심볼들의 수는 불과 8 배 더 커서, 이것은 이용될 수 있는 가능한 코드 레이트를 제한하여 이 경우에 1/8 미만으로 내려가지 않도록 제한되므로, 이용가능한 2^{16} = 65,536 ESI들만을 갖는 것은 일부 상황들에서 제한적일 수도 있다. 이 예에서는, 이용가능한 2^{16} = 65,536 소스 블록들이 언제나 이용될 것보다 더 많을 수 있을 수도 있고, 예를 들어, 1,024 옥테트들 각각의 8,192 소스 심볼들에 있어서, 지원될 수 있는 파일의 크기는 524 GB이며, 이것은 많은 애플리케이션들에서 필요한 것보다 100배 더 크다.

[0045] 또 다른 예로서, [RaptorQ-RFC-6330]을 이용할 때에는, 4 GB 파일에 대하여, 각각의 소스 블록이 8 MB로 제한될 경우(최대 서브-블록 크기가 256 KB이고, 최소 서브-심볼 크기가 32 옥테트들이고, 심볼 크기가 1,024 옥테트들일 경우에 그러할 수도 있음), 소스 블록들의 수를 256으로 제한하는 것은 결국 파일 크기를 2 GB로 제한하므로, 이용가능한 2^{8} = 256 SBN들만을 갖는 것은 일부 상황들에서 제한적일 수도 있다. 이 예에서는, 이용가능한 2^{24} = 16,777,216 가능한 인코딩된 심볼들을 갖는 것이 언제나 이용될 것보다 더 많을 수 있을 수도 있고, 예를 들어, 8,192 소스 심볼들에 있어서, 가능한 인코딩된 심볼들의 수는 2,048배 더 크며, 이것은 일부 애플리케이션들에서 결코 필요하지 않을 수도 있다.

[0046] 또 다른 바람직한 특성은, 때때로 파일의 상이한 부분들 사이의 비균등 에러 보호("unequal error protection; UEP")라고 또한 칭해지는 우선순위화된 인코딩 송신을 위한 능력을 제공하는 것이다. 예를 들어, 파일의 최초 10 %를 나머지 90 %보다 패킷 손실에 대해 더욱 강하게 보호하는 것이 바람직할 수도 있다. 예를 들어, [LDPC-Extensions]는 [LDPC-RFC-5170]이 UEP에 대한 지원을 제공하기 위하여 어떻게 확장될 수 있는지를 설명한다. 이 경우, 실제적인 FEC 코드 자체는 파일의 상이한 부분들에 대한 패리티 보호의 상이한 레벨들을 제공하기 위하여 수정된다. 그러나, 이 접근법에 대한 단점들이 있다. 예를 들어, UEP를 제공하기 위하여 FEC 코드 자체를 수정해야 하는 것은 FEC 코드 자체를 구현 및 테스트하는 것을 복잡하게 하므로 바람직하지 않다. 또한, [LDPC-Extensions]의 도 6에 도시된 결과들과 같이, 파일의 상이한 부분들에 대해 제공되는 패킷 손실에 대한 복원력의 측면에서 이러한 접근법의 결과적인 성능은 결코 최적이 아니다.

[0047] [PET] 및 [PET-Patent]에 설명된 바와 같이, UEP 파일 전달 능력들을 제공하기 위한 하나의 방법은 그 우선순위 및 크기에 따라 파일의 상이한 부분들에 대하여 각각의 패킷의 상이한 부분들을 할당하는 것이다. 그러나, 예를 들어, 파일의 각각의 부분의 작은 메모리 디코딩을 지원하기 위하여, 그리고 또한 이와 동시에, 파일의 각각의 부분에 대한 어느 심볼이 각각의 패킷 내에 포함되는지를 수신기가 결정하도록 하는 FEC 페이로드 ID를 각각의 패킷 내에 제공하기 위하여, 파일의 각각의 상이한 부분이 파일의 다른 부분들에 관계없이 소스 블록들 및 서브-블록들로 구획될 수 있도록, 이러한 UEP 방법들을 어떻게 통합할 것인지가 관심사이다. 이것은, 파일의 각각의 부분에 대하여, 파일의 제 1 부분에 대한 패킷 내의 심볼에 대한 대응하는 SBN 및 ESI가 파일의 제 2 부분에 대한 패킷 내의 심볼에 대한 SBN 및 ESI와는 상이할 수도 있으므로, 포맷의 FEC 페이로드 ID(SBN, ESI)를 이용하는 것을 지원하기가 매우 어렵다.

[0048] 일부의 경우들에 있어서는, 특화된 서버들이 요구되고, 그것은 컨텐츠 전달을 지원하기 위하여 더욱 보편적인 기존의 하드웨어 시스템들을 이용하는 것보다 구현, 지원 및 유지하기가 더욱 비용이 들 수 있다. 그러므로, 구현하기가 덜 복잡한 컨텐츠 및 리페어 심볼들을 전달하기 위한 방법들을 가지는 것이 바람직하다.

발명의 내용

[0049] 파일 전달 방법 및 장치의 실시예들에서는, 컨텐츠를 나타내는 소스 블록들 및 심볼들이 유니캐스트 방식으로 이용가능하고 리페어 블록들 및 심볼들이 브로드캐스트 또는 멀티캐스트 방식으로 제공되도록, 컨텐츠가 파일 전달 시스템에 제공된다. 다른 통로들 및 중복성들이 제공될 수도 있다. 소스 블록들 및 심볼들의 유니캐스트 제공은 하나의 엔티티(entity)에 의해 행해질 수도 있고, 브로드캐스트 또는 멀티캐스트 부분은 제 1 엔티티에 의해 행해지는 특정한 처리에 의해 또는 이 특정한 처리 없이 또 다른 엔티티에 의해 제공될 수도 있다.

[0050] 특정한 실시예에서는, 컨텐츠(데이터, 이미지들, 오디오, 비디오, 등)의 세트가 컨텐츠를 그 사용자들에게 프리젠팅(presenting) 하기 위하여 많은 수의 사용자들에 의해 또는 이 사용자들을 위해 동작되는 많은 수의 최종-사용자 디바이스들("UE들")에 이용가능하게 되고, 전형적으로, 사용자가 컨텐츠에 대한 요청을 비동기식으로 행한 바로 직후에 주어진 사용자에 대한 프리젠테이션(presentation)을 시작하고, 바람직하게는 (사용자가 그것을 종결시키지 않으면) 그 종료시까지 프리젠테이션을 계속한다. 그 실시예에서는, 컨텐츠가 하나 또는 그보다 많은 유니캐스트 서버들에서 소스 형태로 저장되고, 컨텐츠에 대한 리페어 심볼들이 브로드캐스트 서버에서 발생 및 저장되고, 거기로부터 다수의 UE들로 브로드캐스팅 또는 멀티캐스팅된다. 대안적으로, 저장장치가 전혀 없고, 컨텐츠는 리페어 심볼들이 발생되자마자 브로드캐스팅된다. 그 다음으로, UE는 브로드캐스트 서버로부터 일부의 수의 리페어 심볼들을 수신하고, 리페어 심볼들이 프로세스에서 손실되었는지 여부를 결정하고, 추가적인 심볼들 및 수신된 리페어 심볼들로부터 컨텐츠를 완전히 복원하기 위해 필요한 추가적인 심볼들의 수를 (적어도 대략적으로) 결정하고, 그 다음으로, 유니캐스트 서버로부터 심볼들의 그 수를 요청할 것이다. 또 다른 대안으로서, 적어도 일부의 소스 심볼들은 브로드캐스트 서버로부터 브로드캐스팅 또는 멀티캐스팅되고, UE가 재생하고자 하는 컨텐츠의 일부분들을 완전히 복원하기 위해 필요한 추가적인 심볼들의 수를 (적어도 대략적으로) 결정하고, 그 다음으로, 리페어 서브-심볼들 또는 리페어 심볼들일 수도 있는 심볼들 또는 서브-심볼들의 그 수를 유니캐스트 서버로부터 요청한다.

[0051] 일부의 경우들에 있어서, 유니캐스트 서버로의 요청은 HTTP 바이트 범위 요청의 형태이다. HTTP 바이트-범위 요청이 파일의 URL, 파일에서의 시작 위치 및 (요청된 심볼들의 수와 같은) 요청의 길이, 또는 요청된 서브-심볼들의 수, 또는 서브-심볼들 또는 심볼들의 연속 세트에 대응하는 바이트 범위 요청을 특정하는 경우, 요청들은, 요청들의 전부 또는 다수가 시작 위치로서 파일의 초기 위치를 이용하도록 될 수도 있다. 이것은 다운스트림 캐쉬들이 요청들을 더욱 빈번하게 충족시키게 할 것인데, 이것은 제공할 필수 데이터 모두를 가질 것이므로, 이것이 주어진 파일에 대하여 최대 요청을 캐시하였으면, 작은 요청들 모두가 그 캐시된 바이트 범위의 서브세트일 것이기 때문이다.

[0052] 특정한 실시예들에서는, 유니캐스트 서버들이 간단하고, 기존의 HTTP 웹 서버들은 바이트-범위 요청들을 처리할 수 있다. 이와 같이, 그러한 유니캐스트 서버들은 임의의 특정한 브로드캐스팅이 수행되고 있는 것을 알지 못하도록 설계될 수 있다.

[0053] 첨부한 도면들과 함께, 다음의 상세한 설명은 본 발명의 성질 및 장점들의 더욱 양호한 이해를 제공할 것이다.

도면의 간단한 설명

[0054] 도 1은 기존의 FEC 페이로드 ID들을 예시하는 도면이고; 도 1a는 Raptor-RFC 5053에 대한 FEC 페이로드 ID를 예시하는 한편, 도 1b는 RaptorQ-RFC-6330에 대한 FEC 페이로드 ID를 예시한다.

도 2는 기본적인 유니버셜 파일 전달("universal file delivery; UFD") 방법에 대한 FEC 페이로드 ID를 예시하는 도면이다.

도 3은 전송기의 기본적인 UFD 방법을 예시하는 흐름도이다.

도 4는 수신기의 기본적인 UFD 방법을 예시하는 흐름도이다.

도 5a 내지 도 5b는 파일의 심볼들의 (SBN, ESI) 식별과, 파일의 심볼들의 대응하는 유니버셜 파일 심볼 식별자("universal file symbol identifier; UFSI")들로의 그리고 이들로부터의 맵핑을 예시하는 예들이다.

도 6은 전송기 유니버셜 파일 전달의 비균등 에러 보호("UFD-UEP") 방법을 예시하는 흐름도이다.

도 7은 수신기 UFD-UEP 방법을 예시하는 흐름도이다.

도 8a 내지 도 8b는 상이한 우선순위를 각각 갖는 2 개의 부분들을 포함하는 파일의 (SBN, ESI) 식별의 예를 예시한다.

도 9는 파일의 2 개의 부분들로부터의 인코딩된 심볼들의 (SBN, ESI) 식별자들과, 각각의 패킷 내에 포함된 UFSI와 함께 부분들에 대한 인코딩된 심볼들을 포함하는 패킷들과의 사이의 맵핑의, 도 8a 및 도 8b에 대응하는 예를 예시한다.

도 10은 [RaptorQ-RFC-6330]을 이용하는 간단한 UEP 파일 전달 방법의 성능을 예시한다.

도 11은 둘 모두가 [RaptorQ-RFC-6330]을 이용하는 간단한 UEP 파일 전달 방법과 UFD-UEP 파일 전달 방법과의

사이의 일 예의 성능 비교를 예시한다.

도 12는 모두가 [RaptorQ-RFC-6330]을 이용하는 하나의 파일의 파일 전달, 다수의 파일들의 파일 전달, 및 UFD-번들제공된(UFD-bundled) 파일 전달 방법 사이의 일 예의 성능 비교를 예시한다.

도 13은 파일 전달의 부분으로서 Raptor, RaptorQ 또는 다른 패킷들을 발생, 전송 및 수신하기 위해 이용될 수도 있는 통신 시스템의 블록도이다.

도 14는 파일 전달이 행해질 수도 있는 통신 시스템으로서, 하나의 수신기가 다수의 보통은 독립적인 전송기들로부터 출력 심볼들을 수신하는 통신 시스템의 예시도이다.

도 15는 파일 전달이 행해질 수도 있는 통신 시스템으로서, 다수의 아마도 독립적인 수신기들이 오직 하나의 수신기 및/또는 오직 하나의 전송기가 이용되는 경우보다 더 적은 시간 내에 입력 파일을 수신하기 위하여 다수의 보통은 독립적인 전송기들로부터 출력 심볼들을 수신하는 통신 시스템의 예시도이다.

도 16은 HTTP 스트리밍 서버들을 이용하여 파일 전달을 제공하기 위하여 이용될 수도 있는 블록-요청 스트리밍 시스템의 엘리먼트들을 도시한다.

도 17은 파일 전달을 위해 이용될 수도 있는 바와 같이, 컨텐츠 수집 시스템에 의해 프로세싱되는 데이터를 수신하기 위하여 블록 서빙 기반구조("block serving infrastructure; BSI")에 결합되는 클라이언트 시스템의 엘리먼트들에서 더욱 세부사항을 도시하는, 도 16의 블록-요청 스트리밍 시스템의 엘리먼트들을 예시한다.

도 18은 파일 전달을 위한 파일들을 준비하기 위하여 이용될 수도 있는 수집 시스템의 하드웨어/소프트웨어 구현예를 예시한다.

도 19는 클라이언트 시스템에 전달되는 파일들을 수신하기 위하여 이용될 수도 있는 클라이언트 시스템의 하드웨어/소프트웨어 구현예를 예시한다.

도 20은 공통 FEC OTI 엘리먼트 포맷의 일 예를 예시한다.

도 21은 방식-특정 FEC OTI 엘리먼트 포맷의 일 예를 예시한다.

도 22는 기본적인 FLUTE 파일 전달을 예시한다.

도 23은 기본적인 FLUTE 패킷 포맷을 예시한다.

도 24는 서브-블록들을 이용하는 전송기에서 발생하는 서브-블록 인코딩을 예시한다.

도 25는 서브-블록들을 이용하는 수신기에서 발생하는 서브-블록 디코딩을 예시한다.

도 26은 서브-블록킹(sub-blocking)을 이용하는 파일 전달을 예시한다.

도 27은 다수의 소스 블록들의 처리를 예시한다.

도 28은 FEC 및 FLUTE를 이용한 작업 흐름을 예시한다.

도 29는 브로드캐스트/리페어, 유니캐스트/소스 구성을 예시한다.

도 30은 시스템이 MBMS 베어러를 통해 어떻게 리페어 심볼들만을 브로드캐스팅할 것인지를 예시한다.

도 31은 HTTP 바이트 범위 요청들을 통해 유니캐스트 리페어를 이용하는 것을 예시한다.

도 32는 제 1 소스 블록에 대한 서브-블록 범위 요청 계산들의 일 예를 예시한다.

도 33은 제 2 소스 블록에 대한 서브-블록 범위 요청 계산들의 일 예를 예시한다.

도 34는 원래-순서 HTTP 파일 포맷 및 구획 구조의 일 예를 예시한다.

도 35는 UOSI 순서의 소스 심볼들의 일 예를 예시한다.

도 36은 UOSI 순서의 리페어 심볼들의 일 예를 예시한다.

도 37은 확장된-원래-순서 HTTP 파일 포맷의 일 예를 예시한다.

도 38은 확장된-원래-순서 HTTP 파일 포맷의 또 다른 예를 예시한다.

도 39는 서브-블록킹을 갖지 않는 원래-순서 HTTP 파일 포맷에 대한 바이트 범위 계산들의 일 예를 예시한다.

도 40은 서브-블록킹을 갖지 않는 확장된-원래-순서 HTTP 파일 포맷에 대한 바이트 범위 계산들의 일 예를 예시한다.

도 41은 서브-블록킹을 갖는 확장된-원래-순서 HTTP 파일 포맷에 대한 바이트 범위 계산들의 일 예를 예시한다.

발명을 실시하기 위한 구체적인 내용

[0055] 여기의 실시예들에서, 파일 전달은 파일을 전송하는 인코더/송신기 시스템과, 파일을 수신하는 수신기/디코더 시스템에 의해 수행된다. 송신들의 포맷은 인코더가 인코딩한 것을 디코더가 이해할 수 있도록 조정된다. 이하의 다양한 예들에서 도시된 바와 같이, 파일 전달은 일반적인 객체 전달의 예이고, 달리 표시되지 않으면, 이 예들로부터, 객체들이 파일들로서 취급될 수도 있고 아마도 그 반대도 성립할 수도 있다는 것은 분명해야 한다.

[0056] 패킷 전달 시스템에서는, 데이터가 패킷들로 조직화(organize)되고 패킷들로서 송신된다. 각각의 패킷은 패킷 내에 무엇이 있고 그것이 어떻게 배치되어 있는지를 수신기가 결정하도록 하는 엘리먼트들을 가진다. 여기에 설명된 기술들을 이용하면, 순방향 에러 정정 ("FEC")이 이용되는 패킷들을 송신하기 위한 유연성이 제공된다.

[0057] 이 기술들을 이용하면, 비균등 FEC 보호와, 번들제공된 파일들의 전달도 제공될 수 있다. 다수의 파일들이 별개의 파일들로서 전달될 때, 패킷 손실에 대한 전달들의 복원력은 모든 파일들이 더 큰 파일로 함께 연접(concatenate)되고 더 큰 파일은 전달에서 보호될 경우보다 훨씬 더 적을 수 있다는 것은 잘 알려져 있다. 그러나, 더 작은 파일들의 조합으로서 더 큰 파일의 구조의 시그널링이 요구되고, 수신기가 더 작은 파일들의 서브세트(subset)를 복원하는 것에만 관심이 있더라도, 수신기는 일반적으로 큰 파일 내의 더 작은 파일들 중의 임의의 것을 복원하기 위하여 전체의 큰 파일을 복원할 필요가 있다.

[0058] 따라서, 바람직한 파일 전달 시스템 또는 방법은 소스 블록들의 수와, 파일에 대한 파일 전달 구조로서 이용되는 소스 블록당 인코딩 심볼들의 수의 임의의 유연성 있는 조합을 허용해야 한다. 전형적인 구현예에서, 소스 블록은 리페어 심볼들을 발생하는 것과 같은, FEC 동작의 범위이다. 예를 들어, 리페어 심볼은 하나의 소스 블록으로부터 하나 또는 그보다 많은 소스 심볼들 모두의 함수로서 발생될 수도 있다. 이러한 경우들에 있어서, 각각의 소스 블록은 독립적으로 디코딩가능할 수도 있다. 이것은, 모든 데이터가 수신되기 전에 디코더가 전달되고 있는 일부 데이터를 디코딩 및 프로세싱할 필요가 있을 경우에 디코더에서 유용하다. 이 개시 내용으로부터 분명해야 하는 바와 같이, 소스 블록들이 매우 작은 경우에는, 수신된 리페어 심볼은 더 작은 수의 소스 심볼들의 복원을 위해 이용가능할 것이지만, 소스 블록들이 매우 클 경우에는, 더 큰 소스 블록을 디코딩하기 위해 더 오래 걸릴 수도 있으므로, 수신기가 소스 블록 내의 소스 심볼들의 임의의 것을 디코딩 및/또는 프로세싱 및/또는 이용하는 것에 더 많은 시간이 걸릴 수도 있다.

[0059] 파일 전달 방법들은 패킷 손실에 대하여 효율적인 보호를 제공해야 하고, 상이한 우선순위들로 보호된 파일의 상이한 부분들을 갖는 파일의 전달을 지원해야 하고, 파일의 각각의 부분은 파일 다른 부분들과는 상이한 소스 블록 구조 및 서브-블록 구조를 가질 수도 있다. 또한, 파일들은 일부의 경우들에 있어서 객체들의 특정한 예들로 간주되지만, 여기에서는, 파일들을 전송 및 처리를 설명하기 위해 여기에서 이용된 예들이 데이터베이스로부터의 데이터의 큰 청크(chunk)들, 비디오 시퀀스의 일부분, 등과 같이, 아마도 파일들로서 지칭되지 않은 데이터 객체들을 위해 또한 이용될 수 있다는 것을 이해해야 한다.

[0060] 파일/객체 전달 시스템 또는 방법은 큰 파일/객체의 보호 효율을 갖는 다수의 더 작은 파일들/객체들의 전달, 더 작은 파일/객체 구조들의 간단한 시그널링, 및 수신기가 더 작은 파일들/객체들의 전부를 복원하지 않으면서 더 작은 파일들/객체들의 서브세트만을 독립적으로 복원하기 위한 능력들을 제공해야 한다.

[0061] 파일 전달 시스템은 브로드캐스트부 및 유니캐스트부를 포함할 수도 있다. 판독가능성을 위하여, "브로드캐스트"는 "복수의 사용자들에게 공통의 데이터를 서빙하기 위한 브로드캐스트, 멀티캐스트, 및/또는 다른 메커니즘들"을 의미하는 것으로 판독될 수도 있다. "유니캐스트"는 하나의 소스로부터 하나의 목적지로의 데이터의 이동을 지칭하지만, 하나의 논리적 소스는 다수의 컴포넌트들을 포함할 수도 있고 하나의 논리적 목적지는 다수의 컴포넌트들을 포함할 수도 있다. 유니캐스트 구성에서는, 전형적으로 소스 서버 및 목적지 클라이언트가 있고, 소스 서버는 클라이언트들로부터의 요청들을 대기하고 (이와 달리 허용될 경우) 요청된 데이터를 요청 클라이언트에 분명히 전송함으로써 수신된 요청에 응답한다. 당해 기술에서 잘 알려진 바와 같이, 큰 수의 목적지들로의 유니캐스트 전달은 그러한 동일한 큰 수의 목적지들로의 브로드캐스트 전달보다 더 많은 확장성 과제들을 생성할 수 있다. 전형적으로, HTTP 전달을 위하여, 다수의 캐싱 서버(caching server)들이 서버 전달 확장성을 증가시키기 위해 네트워크 전반에 걸쳐 설치된다. 그러나, 이러한 접근법은 이동 디바이스들로의 컨텐츠의 전

달을 위한 공통적인 장애인 네트워크 용량을 반드시 증가시키지는 않는다.

[0062] 이 바람직한 품질들을 갖는 시스템들의 예들이 지금부터 설명될 것이다.

[0063] 기본적인 유니버셜 파일 전달("UFD") 방법 및 시스템

[0064] 기본적인 유니버셜 파일 전달("UFD") 방법 및 대응하는 시스템(들)이 지금부터 설명될 것이고, UFD는 현존하는 파일 전달 방법들에 비해 상당한 장점들을 포함한다. 기본적인 UFD 방법에 대한 순방향 에러 정정("FEC") 페이로드 ID는 예를 들어, 32-비트 필드일 수 있는 유니버셜 파일 심볼 식별자("UFSI") 필드를 포함한다. 기본적인 UFD 방법에 대한 전송기 및 수신기 방법들이 지금부터 차례로 설명될 것이다. 또한, 파일들이 객체들이라고 지칭되는 경우, "UFSI"는 그 대신에 "UOSI"(유니버셜 객체 심볼 식별자)라고 지칭될 수도 있다.

[0065] 도 1은 기존의 FEC 페이로드 ID들을 예시하는 도면이고; 도 1a는 Raptor-RFC 5053에 대한 FEC 페이로드 ID를 예시하는 한편, 도 1b는 RaptorQ-RFC-6330에 대한 FEC 페이로드 ID를 예시한다.

[0066] 도 2는 기본적인 유니버셜 파일 전달("UFD") 방법에 대한 FEC 페이로드 ID를 예시하는 도면이다. 후자의 접근법이 더욱 유연할 수 있다.

[0067] 도 3은 전송기의 기본적인 UFD 방법을 예시하는 흐름도이다. 전송기는 예를 들어, [Raptor-RFC-5053] 또는 [RaptorQ-RFC-6330]에 설명된 바와 같이, FEC 객체 송신 정보("OTI")를 발생하기 위하여 현존하는 방법들을 이용할 수 있고(예를 들어, [RaptorQ-RFC-6330]의 섹션 4.3 참조), 파일을 송신하기 위해 이용될 소스 블록 및 서브-블록 구조를 결정하기 위하여, 그리고 (SBN, ESI) 쌍들 및 파일의 인코딩된 심볼들 사이의 관계를 결정하기 위하여, FEC OTI를 이용할 수 있다(예를 들어, [RaptorQ-RFC-6330]의 섹션 4.4 참조).

[0068] 예를 들어, [RaptorQ-RFC-6330]에 설명된 바와 같이, 발생된 FEC OTI는 (F, Al, T, Z, N)일 수 있고, 여기서 F는 송신될 파일의 크기이고, Al은 서브-심볼들이 Al의 배수들인 메모리 경계들 상에 정렬되는 것을 보장하기 위해 이용되는 정렬 인자(alignment factor)이고, T는 송신 시에 발생 및 전송될 심볼들의 크기이고, Z는 송신을 위해 파일이 구획되어야 할 소스 블록들의 수이고, N은 송신을 위해 각각의 소스 블록이 구획되어야 할 서브-블록들의 수이다. 이것은 도 3의 단계(300)에서 도시된 바와 같다.

[0069] 전송기는 전송될 인코딩된 심볼들을 패킷들에서 형성할 수 있고 소스 블록에 기반한 현존하는 방법들을 이용하여 이 인코딩된 심볼들에 대한 SBN들 및 ESI들을 발생할 수 있고, 서브-블록킹(sub-blocking)이 이용될 경우에는, 예를 들어, [RaptorQ-RFC-6330]에 설명된 바와 같이, 서브-블록 구조를 또한 발생할 수 있다. 인코딩된 심볼이 전송되어야 할 때마다, 전송기는 도 3의 단계(310)에서 도시된 바와 같이, 발생될 인코딩된 심볼에 대한 SBN A 및 ESI B를 결정할 수 있고, 그 다음으로, 도 3의 단계(320)에서 도시된 바와 같이, 전송기는 예를 들어, [RaptorQ-RFC-6330]에 설명된 것들의 현존하는 기술들을 이용하여 (SBN, ESI) = (A, B)에 기반으로 하여 인코딩된 심볼의 값을 발생할 수 있다. 다음으로, 도 3의 단계(330)에서 도시된 바와 같이, 그 인코딩된 심볼에 대한 UFSI C는 C = B*Z+A로서 연산된다.

[0070] 도 3의 단계(340)에서 도시된 바와 같이, 전송기는 인코딩된 심볼의 UFSI C로 설정된 패킷의 FEC 페이로드 ID를 갖는 패킷으로 인코딩된 심볼을 전송할 수 있다. 다음으로, 도 3의 단계(350)에서 도시된 바와 같이, 전송기는 더 많은 인코딩된 심볼들이 전송되어야 하는지를 결정할 수 있고, 그러하다면, 도 3의 단계(310)로의 "예" 가지에 의해 도시된 바와 같이, 전송기는 전송하기 위한 추가적인 인코딩된 심볼들을 발생할 수 있고, 그렇지 않다면, 도 3의 단계(360)로의 "아니오" 가지에 의해 도시된 바와 같이, 전송기는 종료할 수 있다.

[0071] 전송기의 기본적인 UFD 방법의 다수의 변형들이 있다. 예를 들어, 전송기는 패킷들의 적어도 일부에서 하나를 초과하는 인코딩된 심볼들을 전송할 수 있고, 상기 경우에 있어서 FEC 페이로드 ID는 패킷 내에 포함된 제 1 인코딩된 심볼의 UFSI로 설정될 수 있고, 패킷 내에 포함된 추가적인 심볼들은 그 대응하는 UFSI 값들이 연속적이도록 선택될 수 있다. 예를 들어, 패킷에서 운반되는 3 개의 인코딩된 심볼들이 있고 이러한 제 1 심볼이 UFSI = 4,234를 가지는 경우, 다른 2 개의 인코딩된 심볼들은 UFSI들 4,235 및 4,236을 각각 갖는 것들일 수 있다. 다른 대안들의 예들로서, 전송기는 얼마나 많은 인코딩된 심볼들을 발생할 것인지를 미리 결정할 수도 있고, 인코딩된 심볼들 중의 임의의 것을 발생하기 전에 발생되어야 할 모든 인코딩된 심볼들에 대한 (SBN, ESI) 값들을 결정할 수도 있다. 또 다른 예로서, UFSI 값들은 (SBN, ESI) 값들을 발생하는 중간 단계 없이 바로 발생될 수도 있다.

[0072] 변형의 또 다른 예로서, FEC OTI 정보의 다른 형태들이 발생될 수도 있다. 예를 들어, 기본 UFSI BU는 다음과 같이 이용될 수 있는 파일에 대한 FEC OTI에 포함될 수 있고: 패킷 내에 포함된 인코딩된 심볼에 대한 FEC 전송

기 및 수신기에 의해 이용될 UFSI는 U+BU이고, U는 인코딩된 심볼을 운반하는 패킷에서 운반되는 UFSI이다. 따라서, 예를 들어, 패킷이 U = 1,045를 운반하고 FEC OTI 내의 기본 UFSI 가 BU = 2,000,000인 경우, 인코딩된 심볼 UFSI는 2,001,045이다. 기본 UFSI의 이용은 몇몇 장점들을 가진다. 하나에 대하여, [FLUTE], [ALC], [LCT], [FEC BB]에서 설명된 프로토콜 슈트(suite)는 TOI라고 또한 칭해지는 송신 객체 식별자(Transmission Object Identifier)를 전송될 파일 또는 객체의 FEC OTI와 연관시킨다. 동일한 파일에 대한 인코딩된 심볼들은 상이한 시간들 또는 상이한 세션들에서 전송될 수도 있고 상이한 TOI들과 연관될 수 있다는 것이 가능하다. 또한, 각각의 상이한 TOI와 연관된 패킷들에 대해 UFSI = 0으로 시작하는 인코딩된 패킷들을 전송할 수 있다는 것이 유리하다. 기본 UFSI를 FEC OTI의 부분으로서 특정하기 위한 능력을 가짐으로써, 상이한 기본 UFSI가 각각의 TOI와 연관될 수 있고, 이 TOI에 대한 인코딩된 심볼들은 상이한 TOI들에 대한 이중의 인코딩된 심볼들을 전송하지 않으면서 파일에 대해 전송되어야 한다. 예를 들어, 동일한 파일에 대한 인코딩된 심볼들은 TOI = 1과 연관되고 TOI = 2와 연관된 패킷들에서 전송될 수도 있고, TOI = 1과 연관된 기본 UFSI는 0으로 설정되고, TOI = 2와 연관된 기본 UFSI는 1,000,000으로 설정된다. 다음으로, TOI = 1 및 TOI = 2의 둘 모두에 대한 인코딩된 패킷들은 UFSI들 0, 1, 2, 등의 시퀀스를 포함할 수 있고, 1,000,000 보다 적은 인코딩된 심볼들이 TOI = 1인 파일에 대해 전송되는 한, 2 개의 TOI들과 연관되어 전송된 인코딩된 심볼들 사이에서 전송된 이중의 인코딩된 심볼들이 전혀 없을 것이다.

[0073] 수신기의 기본적인 UFD 방법이 도 4를 참조하여 설명된다. 도 4의 단계(400)에서 도시된 바와 같이, 수신기는 전송기에 관하여 위에서 설명된 것과 동일한 포맷에서 FEC OTI(F, A1, T, Z, N)를 결정하기 위하여 현존하는 기술들을 이용할 수 있다. 예를 들어, FEC OTI가 FLUTE 세션 설명 내에 내장될 수도 있거나, 또는 FEC OTI가 URL로 인코딩될 수도 있거나, 또는 FEC OTI가 SDP 메시지를 통해 얻어질 수도 있다. 단계(410)에서는, 수신기가 더 많은 인코딩된 심볼들이 수신되는지를 판단하고, 수신기가 또 다른 인코딩된 심볼을 수신하는 경우에 수신기가 단계(430)로 진행하거나, 수신기가 더 많은 인코딩된 심볼들이 수신되지 않을 것이라고 결정하는 경우에 수신기가 단계(420)로 진행하며 다른 수단을 이용하여, 예를 들어, 파일 리페어 서버로의 HTTP 요청들을 이용하여 파일을 복원하는 것을 시도할 때까지, 수신기는 이 단계에 머무를 수도 있거나, 또는 수신기는 추후에 더 많은 인코딩된 심볼들을 수신하기 위한 또 다른 세션을 대기할 수도 있거나, 또는 수신기는 파일이 복원될 수 없다고 판정할 수도 있다.

[0074] 또 다른 인코딩된 심볼이 이용가능할 때, 단계(430)에서는, 수신기가 인코딩된 심볼의 UFSI C를 결정하고 인코딩된 심볼의 값을 수신한다. 단계(440)에서, 수신기는 소스 블록들의 수 Z 및 UFSI C에 기반으로 하여 A = C modulo Z, 및 B = floor(C/Z)을 계산하고, 단계(450)에서, 수신기는 인코딩된 심볼에 대한 (SBN, ESI)를 (A, B)로 설정하고, 단계(460)에서, 수신기는 파일 복원를 위해 이용될 인코딩된 심볼의 값 및 (A, B)를 저장한다. 단계(470)에서는, 수신기가 파일을 복원하기 위해 수신된 충분한 인코딩된 심볼들이 있는지를 결정하고, 그러하다면, 단계(480)에서 파일을 복원하도록 진행하고, 그렇지 않다면, 단계(410)에서 더 많은 인코딩된 심볼들을 수신하도록 진행한다.

[0075] 수신기의 기본적인 UFD 방법의 다수의 변형들이 있다. 예를 들어, 수신기는 패킷들의 적어도 일부에서 하나를 초과하는 인코딩된 심볼을 수신할 수 있고, 이 경우, FEC 페이로드 ID는 패킷 내에 포함된 제 1 인코딩된 심볼의 UFSI로 설정될 수도 있고, 패킷 내의 추가적인 심볼들은 연속적인 대응하는 UFSI 값들을 가질 수도 있다. 예를 들어, 패킷에서 운반되는 3 개의 인코딩된 심볼들이 있고 제 1 의 이러한 심볼이 UFSI = 4,234를 가질 경우, 다른 2 개의 인코딩된 심볼들은 각각 UFSI들 4,235 및 4,236인 것들일 수 있고, 패킷에서 운반되는 UFSI는 제 1 인코딩된 심볼의 UFSI = 4,234일 수도 있다. 다른 대안들의 예들로서, 수신기는 복원를 시도하기 전에 얼마나 많은 인코딩된 심볼들을 수신할 것인지를 미리 결정할 수도 있다. 또 다른 예로서, 수신기는 파일을 복원하기 위하여 충분한 인코딩된 심볼들이 수신되었는지를 결정하기 위해 이용된 FEC 코드에 특정한 일부의 프로세싱을 행할 수도 있다. 또 다른 예로서, UFSI 값들은 복원 프로세스에서 (SBN, ESI) 값들을 발생하는 중간 단계 없이 바로 이용될 수도 있다. 또 다른 예로서, 파일의 복원은 인코딩된 심볼들의 수신과 동시에 일어날 수도 있다. 또 다른 예로서, FEC OTI 정보의 다른 형태들이 이용될 수도 있다.

[0076] 기본적인 UFD 방법을, 소스 블록 및 서브-블록 구조를 결정하기 위하여 [RaptorQ-RFC-6330]에서 설명된 기술들과 조합하는 것은 다수의 장점들을 제공한다. 예를 들어, 이전의 방법들이 파일을 송신하는 목적들을 위하여 SBN 및 ESI의 조합에 의해 식별되었던 소스 블록들을 칭하였던 것은 기본적인 UFD 방법을 이용할 때에 UFSI에 의해 식별되는 파일 심볼들로서 간주될 수 있다. F를 송신될 옥테트들 내의 파일의 크기라고 하고, T를 파일을 송신할 때에 FEC 인코딩/디코딩 목적들을 위해 이용될 심볼 크기라고 하고, 이에 따라 KT = ceil(F/T)는 파일 내의 심볼들의 총 수이고, ceil(x)는 x 이상의 최소 정수이다.

[0077] 소스 블록 구조 및 서브-블록킹 구조가 예를 들어, [RaptorQ-RFC-6330]에서 설명된 바와 같이 결정되고, 위에서 설명된 기본적인 UFD 방법이 심볼의 식별을 (SBN, ESI) 포맷으로부터 UFSI 포맷으로 그리고 UFSI 포맷으로부터 변환하기 위해 이용될 때, 파일 심볼들에 대한 UFSI들의 범위는 0, 1, 2, ..., KT-1이고, 파일로부터 발생된 임의의 리페어 심볼들은 범위 KT, KT+1, KT+2, 등의 UFSI들을 가질 것이다. 이 특성은 그 UFSI를 KT의 값과 간단하게 비교함으로써 심볼이 원래의 파일의 부분인지 또는 파일로부터 발생된 리페어 심볼인지의 결정을 허용한다. 이것은 예를 들어, FEC 디코딩을 지원하지 않는 수신기들이 패킷들에서 운반된 UFSI 값과, 파일에 대한 KT의 값에 기반으로 하여, 어느 심볼들이 원래의 파일의 부분(및 파일 내의 그 위치)인지와, 어느 것이 리페어 심볼들로서 무시될 수 있는지를 결정하는 것을 허용하기에 유용할 수 있다.

[0078] 도 5a 및 도 5b는 일 예를 예시하고, 이 경우에, 파일 크기는 F = 28,669 옥테트들이고, 심볼 크기는 T = 1,024 옥테트들이고, 이에 따라 KT = ceil(F/T) = 28이다. 이 2 개의 도면들에서, 소스 블록들의 수는 Z = 5이다. 도 5a 및 도 5b에서, 파일의 심볼들의 (SBN, ESI) 라벨링은 상부 및/또는 측면에 도시되어 있고, 각각의 행(row)은 소스 블록에 대응하고 각각의 열(column)은 동일한 ESI 값을 갖는 심볼들에 대응한다. 심볼들의 대응하는 UFSI 라벨링은 하부에 도시되어 있다. 이 경우, UFSI = 27인 심볼, 즉, UFSI 라벨에 대한 28번째 심볼은 FEC 인코딩 및 디코딩의 목적들을 위하여 제로(zero)들로 패딩된(padded out) 그 최종 (KT*T)-F = 3 옥테트들을 가지지만, 이 심볼의 이 최종 3 개의 패딩된 옥테트들은 송신되지 않을 수도 있다. UFSI 28 이상인 임의의 인코딩된 심볼은 이 예에서 리페어 심볼이고, UFSI 28인 인코딩된 심볼은 SBN = 3인 소스 블록으로부터 발생되고, UFSI 29를 갖는 인코딩된 심볼은 SBN = 4인 소스 블록으로부터 발생되고, UFSI 30인 인코딩된 심볼은 SBN = 0인 소스 블록으로부터 발생되는 등등과 같다. 당업자가 인식하는 바와 같이, 이 속성의 다수의 다른 장점들이 있다.

[0079] 기본적인 UFD 방법의 또 다른 장점은, 인코딩된 심볼들이 그 UFSI에 의해 정의된 순서로, 즉, UFSI 순서 0, 1, 2, 3, 4, ...로 전송될 경우, Z 소스 블록들에 대한 인코딩된 심볼들이 인터리빙된 순서(interleaved order)로 전송되며, 즉, Z 소스 블록들의 각각으로부터의 ESI 0인 Z 인코딩된 심볼들이 먼저 전송되고, 그 다음으로, Z 소스 블록들의 각각으로부터의 ESI 1인 Z 인코딩된 심볼들이 전송되는 등등과 같다는 것이다. 대부분의 송신들에 대하여, 이 간단한 전송 순서는 충분하고 바람직하다. 그러나, 패킷 손실이 소스 블록들 Z 의 수와 동기화될 수도 있다는 일부의 주기성을 경험할 경우, 잠재적으로 더 양호한 전송 순서는 Z UFSI-연속 인코딩된 심볼들의 각각의 세트를 전송하기 전에 무작위로 치환하는 것이며, 즉, UFSI들 0, ..., Z-1인 제 1 Z 인코딩된 심볼들이 무작위 치환된 순서로 전송되고, 그 다음으로, UFSI들 Z, ..., 2*Z-1인 다음 Z 인코딩된 심볼들이 무작위 치환된 순서로 전송되는 등등과 같다. 이 설명을 통해, "무작위" 는 달리 표시되지 않으면 의사무작위(pseudorandom)를 포함할 수 있다는 것이 이해된다.

[0080] 기본적인 UFD 방법을 이용하는 것은 이전의 알려진 방법들에 비해 다수의 추가적인 장점들을 제공한다. 예를 들어, 미리 결정된 크기들의 SBN 및 ESI 필드들을 포함하는 FEC 페이로드 ID를 이용할 때, 가능한 소스 블록들의 별개의 미리 결정된 수 또는 소스 블록당 가능한 인코딩된 심볼들의 수가 있다. 예를 들어, 8-비트 SBN 및 24-비트 ESI가 32-비트 FEC 페이로드 ID로 귀착하는 것은 가능한 소스 블록들의 수를 256으로 제한하고 소스 블록당 가능한 인코딩된 심볼들의 수를 16,777,216으로 제한한다. 그 대신에, UFSI 필드를 포함하는 FEC 페이로드 ID는 파일의 소스 블록 구조에 관계없이, 파일에 대한 가능한 인코딩된 심볼들의 총 수만을 제한한다. 예를 들어, 32-비트 FEC 페이로드 ID로 귀착되는 32-비트 UFSI를 이용할 때, 파일이 구획되는 소스 블록들이 얼마나 많은 지에 관계없이, 그리고 서브-블록킹이 이용될 경우에 파일의 서브-블록 구조에 관계없이, 파일에 대해 발생될 수 있는 인코딩된 심볼들의 총 수는 4,294,967,296이다. 따라서, 심볼이 크기에 있어서 1,024 옥테트들일 경우, 이 예에서는, 파일이 하나의 소스 블록, 16,384 소스 블록들, 또는 4,194,304 소스 블록들로 구획되는지에 관계없이, 파일 크기가 4GB까지 이를 수 있고 인코딩된 심볼들의 수는 파일 크기의 1,024배일 수도 있다. 또 다른 예로서, 파일 크기는 2 TB일 수 있고 인코딩된 심볼들의 수는 파일 크기의 2 배일 수 있다. 모든 경우들에 있어서, 파일에 대해 발생될 수 있는 인코딩된 심볼들의 수는 서브-블록킹 구조가 이용될 경우, 파일의 서브-블록킹 구조에 관계없다.

[0081] 비균등 에러 보호 파일 전달 서비스들을 위한 유니버셜 파일 전달 방법

[0082] 대부분의 이전의 파일 전달 방법들은 비균등 에러 보호("UEP") 파일 전달을 지원하지 않는다. 일부 이전의 방법들, 예를 들어, 파일의 상이한 부분들을 인코딩하기 위해 이용된 실제적인 FEC 코드를 변화시킴으로써 UEP를 지원하는, [LDPC-Extensions]에서 설명된 방법들을 이용하는 ISDB-Tmm(Terrestrial mobile multi-media; 지상 이동 멀티-미디어) 표준에서 현재 특정된 것들이 있다. UEP를 이용할 때에 실제적인 FEC 코드를 변화시켜야 하

는 단점 외에, 추가의 단점은 이 방법들에 의해 제공되는 보호가 이상적인 것이 아니라는 것이다.

[0083] 일 예로서, 결과들이 [LDPC-Extensions]의 도 6에 도시된 예를 고려한다. 그 예에서는, 패킷들로 전송되어야 하는 크기 1,000 KB의 파일에 대한 2 개의 부분들이 있고, 각각의 패킷은 1KB 심볼을 포함하며: 파일의 제 1 부분은 크기 30 KB이고 100 패리티 또는 리페어 패킷들에 의해 보호되고, 파일의 제 2 부분은 970 KB이고, 전체적인 1,000 소스 패킷들 및 1,000 패리티 패킷들이 파일에 대해 전송된다. 제공되는 보호는 2 개의 쟁점들로 인해 이상적인 것이 아니다. 하나의 쟁점은, [LDPC-RFC-5170]에 기반으로 하여 이용되는 FEC 코드가 종종 소스 블록을 복원하기 위하여 상당한 반복 오버헤드를 요구하는 것이며, 즉, 파일에 소스 패킷들이 있는 것보다 파일을 복원하기 위하여 더 많은 패킷들이 수신될 것이 요구된다는 것이다. 두 번째 쟁점은, 방법이 파일의 제 2 부분을 전송 및 보호하기 위해 이용되기보다는, 패킷들의 별개의 세트를 이용하여 파일의 제 1 부분을 필수적으로 전송 및 보호한다는 것이다. 두 번째 쟁점에 대하여, 파일의 제 1 부분이 이러한 작은 수의 패킷들을 통해 전송되기 때문에, 파일의 제 1 부분에 대해 전송된 130개의 패킷들 중 30개의 패킷들을 수신할 때 분산(variance)이 클 수 있다.

[0084] 누군가는 여기에서 "간단한 UEP" 파일 전달 방법이라고 칭해지는 확장안에 의해 [LDPC-Extensions]에 설명된 UEP 파일 전달 방법을 개선시킬 수 있다. 간단한 UEP 파일 전달 방법은 파일 전달을 위한 현존하는 기술들을 이용하여 그리고 그 우선순위에 기반으로 파일의 각각의 부분에 대한 상이한 보호의 양들을 이용하여 파일의 부분들을 별개의 파일 전달들로서 전달하고, 그 다음으로, 파일의 부분들 사이의 논리적 접속은, 수신기가 전달된 파일들이 동일한 파일의 부분들인 것을 알도록 시그널링될 수 있다. 예를 들어, 간단한 UEP 파일 전달 방법은 제 1 부분으로부터 발생된 크기 1,024 옥테트들의 인코딩된 심볼을 각각 포함하는 총 130 패킷들을 전송함으로써 파일의 제 1 30 KB 부분을 전달하기 위하여 상기 예에서 [RaptorQ-RFC-6330]을 이용할 수 있고, 그 다음으로, 파일의 제 2 970KB 부분은 제 2 부분으로부터 발생된 크기 1,024 옥테트들의 인코딩된 심볼을 각각 포함하는 총 1870 패킷들을 전송함으로써 별개의 파일로서 전달될 수 있다. 따라서, 총 2,000 패킷들이 별개의 파일들로서 전송된 파일의 2 개의 부분들에 대해 전송된다. 간단한 UEP 파일 전달 방법은, FEC 코드 자체가 수정되지 않기 때문에, 그리고 여기에서 도 10에 도시된 바와 같이, 상이한 패킷 손실 조건들 하에서 파일의 2 개의 부분들의 전달의 성능이 [LDPC-Extensions]의 도 6에 도시된 것보다 우수하기 때문에, [LDPC-Extensions]에 설명된 방법에 비해 개선된 것이다.

[0085] 하나의 가능한 상이함의 원천은 [RaptorQ-RFC-6330]에 특정된 FEC 코드가 [LDPC-RFC-5170]에 특정된 FEC 코드보다 우수한 복원 특성들을 가지기 때문이다. 그러나, 간단한 UEP 파일 전달 방법은 여전히 위에서 설명된 두 번째 쟁점을 여전히 겪고 있다.

[0086] [PET] 및 [PET-Patent]는 UEP 파일 전달 서비스를 제공하기 위한 잠재적으로 개선된 방법들을 제공하고, 각각의 패킷은 그 우선순위에 기반으로 하여 파일의 각각의 부분으로부터의 특정된 양의 인코딩 데이터를 포함한다. [PET]의 간단한 통합은, 각각의 패킷에서 파일의 각각의 부분에 대한 적절한 크기의 인코딩된 심볼을 포함하고, 그 다음으로, 파일의 각각의 부분에 대한 (SBN, ESI) 쌍을 포함하는 별개의 FEC 페이로드 ID를 포함하는 것일 것이다. 그러나, 이 방법은 몇몇 이유들로 유리하지 않다.

[0087] 예를 들어, 각각의 부분에 대한 미리 결정된 크기들의 SBN 및 ESI 필드들을 포함하는 FEC 페이로드 ID를 이용할 때, 파일의 각각의 부분에 대하여 가능한 소스 블록들의 별개의 미리 결정된 수 또는 소스 블록당 가능한 인코딩된 심볼들의 수가 있다. 예를 들어, d 부분들의 각각에 대한 8-비트 SBN 및 24-비트 ESI는, 부분당 가능한 소스 블록들의 수를 256으로 그리고 소스 블록당 가능한 인코딩된 심볼들의 수를 16,777,216으로 제한하는 (32*d)-비트 FEC 페이로드 ID로 귀착된다. 또한, FEC 페이로드 ID 크기가 d 부분들의 각각에 대하여 32 비트들인 경우, 이것은 각각의 패킷 내의 모든 부분들에 대해 FEC 페이로드 ID들의 총 32*d 비트들을 의미할 것이고, 예를 들어, d = 10일 경우, 이것은 각각의 패킷 내의 FEC 페이로드 ID 헤더들에 대해서만 320비트들이고, 또는 등가적으로 40 옥테트들이다.

[0088] 파일 전달을 위한 기본적인 UFD 방법은 이전의 UEP 파일 전달 방법들에 대해 상당한 장점들을 제공하는, 아래에서 상세하게 설명된 바와 같이, 비균등 에러 보호(UEP) 파일 전달 서비스들을 제공하도록 확장될 수 있다. 여기에서, 이 확장된 방법들은 "UFD-UEP" 파일 전달 방법들이라고 지칭된다. 이 UFD-UEP 파일 전달 방법들은 [PET] 및 [PET-Patent]에 설명된 방법들의 일부를 이용할 수 있다.

[0089] 일 예의 UFD-UEP 파일 전달 방법은 지금부터 더욱 상세하게 설명될 것이다. 이러한 방법에서, 전송기는 크기 F의 파일을 크기들 F_0, F_1, ..., F_{d-1}의 d>1 부분들로 구획하고, 이에 따라 F 는 F_i의 i에 걸친 합과 동일하다.

전송기는 패킷 크기 T를 크기들 T_0, T_1, ..., T_{d-1}의 d 부분들로 구획하고, 이에 따라 T 는 T_i의 i에 관한 합과 동일하다. T의 이 구획은 F_0, F_1, ..., F_{d-1} 및 대응하는 파일 부분들의 우선순위들에 기반으로 한다. 비율 F_i/T_i는, 이상적인 FEC 코드가 하나의 소스 볼록으로서 파일의 부분 i를 보호하기 위해 이용되는 것으로 가정할 때, 파일의 부분 i를 복원하기 위하여 얼마나 많은 패킷들이 수신될 필요가 있는지를 결정하고, 이에 따라, 비율 F_i/T_i가 더 작을수록, 파일의 부분 i의 우선순위가 더 높다. 실제로는, 예를 들어, FEC 코드가 완전하지 않고 일부의 수신 오버헤드를 나타내므로, 또는 파일의 그 부분이 다수의 소스 블록들로 구획되며 일부 소스 블록들에 대한 인코딩된 심볼들이 다른 소스 블록들에 대한 것보다 더 높은 레이트에서 손실되므로, 또는 F_i/T_i가 정수가 아니므로, F_i/T_i보다 약간 더 많은 패킷들이 파일의 부분 i를 복원하기 위해 필요할 수도 있다. UEP의 예로서, F = 1 MB, T = 1,024 옥테트들, d = 2, F_0 = 32KB, F_1 = F - F_0 = 992KB, 및 T_0 = 64 옥테트들이라고 가정하면, T_1 = T - T_0 = 960 옥테트들이다. 이 예에서, F_0/T_0 = 512이고, 이에 따라 이상적으로는 512 패킷들의 수신이 파일의 부분 0의 복원을 허용하는 반면, F_1/T_1 = 1,058.13이고, 이에 따라 이상적으로는 1,059 패킷들의 수신이 파일의 부분 1의 복원을 허용한다. 따라서, 이 예에서, 파일의 부분 0은 파일의 부분 1을 복원하기 위해 필요한 많은 패킷들의 대략 절반으로부터 복원될 수 있다.

[0090] 이 예에서는, UEP가 이용되지 않는 경우, 즉, d = 1이고 이에 따라 F_0 = F = 1 MB, 및 T_0 = T = 1,024 옥테트인 경우, F_0/T_0 = 1,024 패킷들이 파일을 복원하기 위해 필요하다는 것에 주목해야 한다. 따라서, 이전의 단락에서 설명된 UEP 예에서, 파일의 부분 1의 복원은 파일의 부분 0의 더 높은 우선순위로 인해, UEP가 이용되지 않을 때보다 약간 더 많은 패킷들을 필요로 한다. 이 기본적인 절충의 분석적인 연구는 [PET]에서 발견될 수 있다.

[0091] F_0, F_1, ..., F_{d-1} 및 상이한 파일 부분들의 우선순위들에 기반으로 하여 T의 구획 T_0, T_1, ..., T_{d-1}을 발생하기 위한 전송기의 UFD-UEP 방법에 대한 다양한 방식들이 있다. T_i가 $F_i/T_i \approx F/T$가 되도록 선택될 경우, 파일의 부분 i는 평균 우선순위일 것으로 간주되고, 부분 i의 우선순위는 각각 $F_i/T_i < F/T$ 또는 $F_i/T_i > F/T$인지에 따라 상대적으로 더 높거나 더 낮을 것이라는 것에 주목해야 한다.

[0092] FEC OTI의 발생을 위한 전송기의 UFD-UEP 방법은 도 6을 참조하여 지금부터 설명된다. d, F_0, F_1, ..., F_{d-1}, T_0, T_1, ..., T_{d-1}, Al, WS의 값들이 주어지면, FEC OTI는 현존하는 방법들을 이용하여, 예를 들어, [RaptorQ-RFC-6330]의 섹션 4.3에 설명된 방법들을 이용하여 파일의 d 부분들의 각각에 통상적으로 적용되는 바와 같이 독립적으로 연산될 수 있으며, 즉, 각각의 i = 0, ..., d-1에 대하여, 전송기는 F_i를 파일 크기로서 그리고 T_i를 각각의 패킷에서 부분 i에 대한 정보를 운반하기 위해 이용될 심볼 크기로서 취급하면서, 예를 들어, [RaptorQ-RFC-6330]의 섹션 4.3에 설명된 방법들을 이용하여 그것이 어떻게 소스 블록들 및 서브-블록들로 구획되어야 하는지를 결정하는 파일 부분 i에 대한 FEC OTI를 발생할 수 있다. 따라서, 전송기는 파일의 다른 부분들에 관계없이 파일의 부분 i에 대한 FEC OTI를 발생한다. 이 프로세스는 여기에서 도 6 의 단계(600)에 도시되어 있다.

[0093] 전송기는 파일의 부분 i의 소스 블록들 및 서브-블록들로의 구획을, 그리고 파일의 부분 i에 대한 인코딩된 심볼의 (SBN, ESI)와, 현존하는 방법들 예를 들어, 여기에서 [RaptorQ-RFC-6330]의 섹션 4.4 및 섹션 5에 설명된 방법들을 이용하여 인코딩된 심볼이 파일의 부분 i로부터 발생되는 방법과의 사이의 맵핑을 또한 발생할 수도 있다. 이 UFD-UEP 방법들은 파일의 다른 부분들에 관계없이 파일의 부분 i에 적용되고, 이에 따라, 파일의 상이한 부분들은 상이한 소스 블록 및 서브-블록 구조들을 가질 수도 있고, 특히, 방법들이 파일의 각각의 부분에 관계없이 적용되므로, 소스 블록당 상이한 수들의 소스 심볼들과, 파일의 상이한 부분들 사이에 상이한 수들의 소스 블록들이 있을 수도 있다.

[0094] 정렬 인자 Al는 파일의 부분들의 전부에 대해 바람직하게는 동일하고, 특히, 그것은 각각의 부분 i에 대하여 T_i의 값이 Al의 배수인 경우에 바람직하다. 또한, 예를 들어, [RaptorQ-RFC-6330]의 섹션 4.5에 설명된 방법들이 FEC OTI를 유도하기 위해 이용되는 경우, 그것은 파일의 부분들의 각각에 대해 소스 블록 및 서브-블록 구조를 유도할 때에 Al 및 WS의 동일한 값이 이용될 경우에 바람직하다. Al에 대한 동일한 값의 이용은 수신기에서 Al 옥테트들의 배수들에 대해 정렬된 메모리 상에서 디코딩이 발생할 수 있다는 것을 보장하고, WS에 대한 동일한 값의 이용은 수신기의 랜덤 액세스 메모리(RAM)에서 디코딩될 필요가 있는 최대 블록 크기가 파일의 모든 부분들에 대해 동일하다는 것을 보장한다. 그러나, 이것은 여기에 설명된 방법들의 요건이 아니며, 즉, 방법들은 아래에서 더욱 설명되는 바와 같이, 상이한 Al 값들이 상이한 파일들에 대해 이용될 경우에 수정 없이

적용된다. 예를 들어, 파일의 더 높은 우선순위의 부분들을 복원하기 위하여 더 적은 메모리를 이용하는 것이 바람직할 경우, 파일의 상이한 부분들에 대한 WS의 상이한 값을 이용하는 것이 FEC OTI를 유도하기에 바람직한 일부의 애플리케이션들이 있다.

[0095] 상이한 부분들에 대한 상이한 정렬 인자들의 이용이 유리한 애플리케이션들이 있을 수도 있다. 예를 들어, 높은 우선순위의 부분들은 4-옥테트 정렬된 메모리를 가지는 로우-엔드(low-end) 수신기들과, 8-옥테트 정렬된 메모리를 가지는 하이-엔드 수신기들에 의해 디코딩될 수도 있는 반면, 낮은 우선순위의 부분들은 오직 하이-엔드 수신기들 둘 다에 의해 디코딩될 수도 있다. 이 예에서는, 로우-엔드 수신기들이 이 부분들을 효율적으로 디코딩할 수 있도록, 높은 우선순위의 부분들에 대해 Al = 4를 이용하는 것이 유리할 수도 있는 반면, 하이-엔드 수신기들은 Al = 4 보다는 Al = 8로 이 부분들을 더 효율적으로 디코딩할 수 있으므로, 낮은 우선순위의 부분들에 대해 Al = 8을 이용하는 것이 유리할 수도 있다.

[0096] 파일 부분 i에 특정한 전송기의 UFD-UEP 방법에 의해 발생된 대응하는 FEC OTI는 F_i, T_i, Z_i, N_i를 포함하고, 여기서 Z_i는 파일의 부분 i가 구획되는 소스 블록들의 수이고, N_i는 파일의 부분 i의 각각의 소스 블록이 구획되는 서브-블록들의 수이다. 따라서, 전송기의 UFD-UEP 방법이 파일에 대해 발생하는 전체적인 FEC OTI는 (d, Al, F_0, T_0, Z_0, N_0, F_1, T_1, Z_1, N_1, ..., F_{d-1}, T_{d-1}, Z_{d-1}, N_{d-1})을 포함할 수 있다. 예를 들어, d가 고정되어 있고 이에 따라, FEC OTI에서 명시적으로 열거될 필요가 없을 때, 또는 소스 블록 구조와, 이용된다면, 서브-블록 구조를 표시하기 위하여 다른 방법들이 이용될 때에는, FEC OTI의 다른 버전(version)들이 또한 이용가능하다.

[0097] UFD-UEP 방법을 이용하여, 전송기는 패킷에서 전송될 파일의 각각의 부분에 대해 하나의 인코딩된 심볼을 어셈블링(assembling) 하고, 패킷에 대한 FEC 페이로드 ID는 UFSI 값 C를 포함한다. 패킷이 전송되어야 할 때, 전송기는 도 6의 단계(610)에서 도시된 바와 같이, 패킷에 대한 FEC 페이로드 ID로서 이용될 UFSI 값 C를 결정한다. 예를 들어, 이용될 UFSI 값들은 연속적일 수 있고, 예를 들어, UFSI 값들 0, 1, 2, 3, ..., 등이다. 도 6의 단계(620)에서 도시된 바와 같이, 주어진 UFSI 값 C에 대하여, 파일의 부분 i에 대한 패킷에서 패킷의 i번째 부분에 배치될 크기 T_i의 인코딩된 심볼은 A_i = C modulo Z_i, 및 B_i = floor(C/Z_i)로서 연산된 SBN A_i 및 ESI B_i를 가진다. 도 6의 단계들(630, 640 및 650)에서 도시된 바와 같이, i = 0, ..., d-1에 대하여, 이 d 인코딩된 심볼들은 패킷의 d 부분들의 각각에 대해 발생되고, 그 다음으로, 총 크기 T의 이 d 인코딩된 심볼들과 함께 UFSI C는 패킷에서 함께 전송된다. 전송기의 UFD-UEP 방법은 도 6의 단계(660)에서 행해진 판정에서 도시된 바와 같이, 인코딩된 패킷들을 발생 및 전송하는 것을 계속한다.

[0098] 수신기의 UFD-UEP 방법은 도 7을 참조하여 설명된다. 도 7의 단계(700)에서 도시된 바와 같이, 수신기는 전송기에 대해 위에서 설명된 것과 동일한 포맷으로 FEC OTI(d, Al, F_0, T_0, Z_0, N_0, F_1, T_1, Z_1, N_1, ..., F_{d-1}, T_{d-1}, Z_{d-1}, N_{d-1})를 결정하기 위하여 현존하는 기술들을 이용할 수 있다. 예를 들어, FEC OTI가 FLUTE 세션 설명 내에 내장될 수도 있거나, 또는 FEC OTI가 URL로 인코딩될 수도 있거나, 또는 FEC OTI가 SDP 메시지를 통해 얻어질 수도 있다. 단계(710)에서는, 수신기가 더 많은 패킷들이 수신되는지를 판단하고, 수신기가 또 다른 패킷을 수신하는 경우에 수신기가 단계(730)로 진행하거나, 수신기가 더 많은 패킷들이 수신되지 않을 것이라고 결정하는 경우에 수신기가 단계(720)로 진행하여, 파일의 충분한 부분들이 복원된 것으로 결정하고, 다른 수단을 이용하여, 예를 들어, 파일 리페어 서버로의 HTTP 요청들을 이용하여 파일의 추가적인 부분들을 복원하는 것을 정지하거나 또는 시도할 때까지, 수신기는 이 단계에 머무를 수도 있거나, 또는 수신기는 추후에 더 많은 패킷들을 수신하기 위한 또 다른 세션을 대기할 수도 있다.

[0099] 또 다른 패킷이 이용가능할 때, 단계(730)에서는, 수신기가 수신된 패킷의 UFSI C를 결정하고, 각각의 i = 0, ..., d-1에 대하여, 각각의 i = 0, ..., d-1에 대한 패킷으로부터 크기 T_i의 인코딩된 심볼을 추출한다. 단계(740)에서, 각각의 i = 0, ..., d-1에 대하여, 수신기는 소스 블록들의 수 Z_i 및 UFSI C에 기반으로 하여 A_i = C modulo Z_i, 및 B_i = floor(C/Z_i)을 계산하고, 단계(750)에서, 수신기는 부분 i에 대한 인코딩된 심볼에 대한 (SBN, ESI)를 (A_i, B_i)로 설정하고, 단계(760)에서, 수신기는 파일의 부분 i를 복원하기 위해 이용될 부분 i에 대한 인코딩된 심볼의 값 및 (A_i, B_i)를 저장한다. 단계(770)에서는, 수신기가 각각의 i = 0, ..., d-1에 대하여, 파일의 부분 i를 복원하기 위해 수신된 충분한 인코딩된 심볼들이 있는지를 결정하고, 그러하다면, 단계(780)에서 파일의 부분 i를 복원하도록 진행하고, 그렇지 않다면, 단계(710)에서 더 많은 패킷들을 수신하도록 진행한다.

[0100] 수신기의 UFD-UEP 방법의 다수의 변형들이 있다. 예를 들어, 패킷들의 적어도 일부에서 파일의 각각의 부분에 대한 하나를 초과하는 인코딩된 심볼을, 전송기는 전송할 수 있고 이에 따라 수신기는 수신할 수 있으며, 이 경우, FEC 페이로드 ID는 패킷 내에 포함된 각각의 부분에 대한 제 1 인코딩된 심볼에 대응하는 UFSI로 설정될 수도 있고, 패킷 내의 각각의 부분에 대한 추가적인 심볼들은 연속적인 대응하는 UFSI 값들을 가질 수도 있다. 예를 들어, 패킷에서 운반되는 파일의 각각의 부분에 대한 3 개의 인코딩된 심볼들이 있고 각각의 부분에 대한 제 1 심볼이 UFSI = 4,234에 대응하는 경우, 각각의 부분에 대한 다른 2 개의 인코딩된 심볼들은 각각 UFSI들 4,235 및 4,236에 대응할 수 있고, 패킷에서 운반되는 UFSI는 UFSI = 4,234일 수도 있다.

[0101] 다른 대안들의 예들로서, 수신기는 복원을 시도하기 전에 얼마나 많은 인코딩된 심볼들을 수신할 것인지를 미리 결정할 수도 있거나, 세션 동안에 패킷 손실 통계를 계산하고 이것에 기반으로 하여 파일의 어느 부분들을 복원하도록 시도할 것인지를 판정할 수도 있다. 또 다른 예로서, 수신기는 파일의 각각의 부분을 복원하기 위하여 충분한 인코딩된 심볼들이 수신되었는지를 결정하기 위해 이용된 FEC 코드에 특정한 일부의 프로세싱을 행할 수도 있다. 또 다른 예로서, UFSI 값들은 각각의 파일 부분에 대한 복원 프로세스에서 (SBN, ESI) 값들을 발생하는 중간 단계 없이 바로 이용될 수도 있다. 또 다른 예로서, 파일의 부분들의 복원은 인코딩된 심볼들의 수신과 동시에 일어날 수도 있다.

[0102] 또 다른 예로서, FEC OTI 정보의 다른 형태들이 이용될 수도 있다. 예를 들어, 다음과 같이 이용될 수 있는 기본 UFSI BU_i는 다른 부분들에 관계없이 부분 i에 대한 FEC OTI에서 특정될 수도 있으며: 패킷 내에 포함된 부분 i에 대한 인코딩된 심볼에 대한 FEC 전송기 및 수신기에 의해 이용될 UFSI는 $U+BU_i$이고, U는 인코딩된 심볼을 운반하는 패킷에서 운반되는 UFSI이다. 따라서, 예를 들어, 패킷이 U = 1,045를 운반하고, 부분 i에 대한 FEC OTI에서의 기본 UFSI가 BU_i = 2,000,000인 경우, 인코딩된 심볼 UFSI는 2,001,045이다.

[0103] 기본 UFSI의 이용은 몇몇 장점들을 가진다. 예를 들어, (나중에 설명되는 이유들로) 리페어 심볼들만 상이한 부분들에 대해 송신되어야 할 경우, $BU_i = KT_i$를 설정하는 것이 유리할 수 있고, KT_i는 부분 i 내의 파일 심볼들의 수이다. 이 경우, 전송된 패킷 시퀀스에서의 UFSI들의 시퀀스는 0, 1, 2, 3, 등일 수 있고, 그럼에도 불구하고 각각의 부분에 대하여, 그 부분으로부터 발생된 리페어 심볼들만이 패킷들로 전송될 것이다.

[0104] 이 단락은 기본 UFSI의 이용의 또 다른 일 예의 장점을 설명한다. [FLUTE], [ALC], [LCT], [FEC BB]에서 설명된 프로토콜 슈트는 TOI로 또한 칭해지는 송신 객체 식별자를 전송될 파일 또는 객체의 FEC OTI와 연관시킨다. 동일한 부분들에 대한 인코딩된 심볼들은 상이한 시간들 또는 상이한 세션들에서 전송될 수도 있고 상이한 TOI들과 연관될 수도 있다는 것이 가능하다. 또한, 각각의 상이한 TOI와 연관된 패킷들에 대해 UFSI = 0으로 시작하는 인코딩된 패킷들을 전송할 수 있다는 것이 유리하다. 기본 UFSI를 각각의 부분에 관계없이 FEC OTI의 부분으로서 특정하기 위한 능력을 가짐으로써, 각각의 부분에 대한 상이한 기본 UFSI가 각각의 TOI와 연관될 수 있고, 이 TOI에 대한 인코딩된 심볼들은 상이한 TOI들에 대한 이중의 인코딩된 심볼들을 전송하지 않으면서 파일에 대해 전송되어야 한다. 예를 들어, 동일한 부분에 대한 인코딩된 심볼들은 TOI = 1과 연관되며 TOI = 2와 연관된 패킷들에서 전송될 수도 있고, TOI = 1과 연관된 부분에 대한 기본 UFSI는 0으로 설정되고, TOI = 2와 연관된 동일한 부분에 대한 기본 UFSI는 1,000,000으로 설정된다. 다음으로, TOI = 1 및 TOI = 2의 둘 모두에 대한 인코딩된 패킷들은 UFSI들 0, 1, 2, 등의 시퀀스를 포함할 수 있고, 1,000,000 보다 적은 인코딩된 심볼들이 TOI = 1인 부분에 대해 전송되는 한, 2 개의 TOI들과 연관되어 전송된 인코딩된 심볼들 사이에서 전송된 부분에 대한 이중의 인코딩된 심볼들이 전혀 없을 것이다. 각각의 부분에 대한 상이한 기본 UFSI를 특정하는 대신에, FEC OTI에서 부분들의 전부에 의해 이용될 기본 UFSI를 가지는 것이 또한 유리할 수도 있는데, 이것은 특히, 이 방법과 함께 이용된 FEC 코드가 예를 들어, Luby I에서 설명된 것과 같은 정보 부가 FEC 코드일 때, 이 경우에는 모든 부분들에 대한 UFSI들의 유효한 범위가 매우 클 수 있으므로, FEC OTI에서 각각의 부분에 대한 별개의 기본 UFSI를 특정하는 장점들의 다수를 동시에 공유하면서, FEC OTI를 전달하기 위해 필요한 옥테트들의 수를 감소시킬 수 있기 때문이다.

[0105] UFD-UEP 방법을, 소스 블록 및 서브-블록 구조를 결정하기 위하여 [RaptorQ-RFC-6330]에서 설명된 기술들과 조합하는 것은 다수의 장점들을 제공한다. 특히, 기본적인 UFD 방법의 장점들의 전부는 UFD-UEP 방법에 대해 또한 유효하다. 예를 들어, 이전의 방법들이 파일의 UEP 부분들 중의 하나를 송신하기 위하여 SBN 및 ESI의 조합에 의해 식별되었던 소스 블록들을 칭하였던 것은 UFD-UEP 방법을 이용할 때에 UFSI에 의해 식별되는 그 부분에 대한 파일 심볼들로서 간주될 수 있다. $KT_i = ceil(F_i/T_i)$는 파일의 부분 i에서의 파일 심볼들의 총 수인 것에 주목해야 한다. 소스 블록 구조 및 서브-블록킹 구조가 예를 들어, [RaptorQ-RFC-6330]에서 설명된 바와 같이

파일의 각각의 부분에 대해 결정되고, 위에서 설명된 UFD-UEP 방법이 파일의 부분에 대한 심볼의 식별을 (SBN, ESI) 포맷으로부터 UFSI 포맷으로 그리고 UFSI 포맷으로부터 변환하기 위해 이용될 때, 파일 심볼들에 대한 UFSI들의 범위는 0, 1, 2, ..., KT_i-1이고, 파일로부터 발생된 임의의 리페어 심볼들은 KT_i, KT_i+1, KT_i+2, 등의 범위의 UFSI들을 가질 것이다. 이 특성은, 그 UFSI를 KT_i의 값과 간단하게 비교함으로써 심볼이 파일의 원래의 부분 i의 부분인지 또는 파일의 부분 i로부터 발생된 리페어 심볼인지의 결정을 허용한다. 이것은 예를 들어, FEC 디코딩을 지원하지 않는 수신기들이 패킷들에서 운반된 UFSI 값과, 파일의 각각의 부분 i에 대한 값들 T_i 및 KT_i에 기반으로 하여, 패킷의 어느 부분들이 원래의 파일의 부분들(및 파일 내의 그 위치)을 포함하는지와, 패킷의 어느 부분들이 무시될 수 있는 리페어 심볼들을 포함하는지를 결정하는 것을 허용하기에 유용할 수 있다.

[0106] 도 8a 및 도 8b는 이 경우에 파일이 2 개의 부분들을 포함하는 예를 예시한다. 제 1 부분은 5 개의 소스 블록들로 구획되고, 이 소스 블록들의 최초 3 개는 각각 6 개의 소스 심볼들을 가지고, 나머지 2 개의 소스 블록들은 각각 5 개의 소스 블록들을 가지고, 이 심볼들의 각각은 크기가 예를 들어, 48 옥테트들이고, 이에 따라, 제 1 부분은 28*48 = 1,344 옥테트들의 크기이다. 제 2 부분은 4 개의 소스 블록들로 구획되고, 이 소스 블록들의 최초 3 개는 각각 4 개의 소스 심볼들을 가지고, 나머지 1 개의 소스 블록은 각각 3 개의 소스 심볼들을 가지고, 이 심볼들의 각각은 크기가 예를 들어, 256 옥테트들이고, 이에 따라, 제 2 부분은 15*256 = 3,840 옥테트들의 크기이다.

[0107] 도 9는 도 8a 및 도 8b에 예시된 파일 구조에 대한 가능한 패킷화를 도시한다. 이 예에서, 각각의 패킷은 UFSI C, 도 6을 참조하여 이전에 설명된 바와 같이 C에 기반으로 하여 도 8a 및 도 8b에 도시된 파일 구조의 제 1 부분으로부터 발생된 크기 T_1 = 48 옥테트들인 제 1 인코딩된 심볼, 및 도 6을 참조하여 이전에 설명된 바와 같이 C에 기반으로 하여 도 8a 및 도 8b에 도시된 파일 구조의 제 2 부분으로부터 발생된 T_2 = 256 옥테트들인 제 2 인코딩된 심볼을 포함한다. 패킷들의 음영처리되지 않은 부분들은 파일의 대응하는 부분의 소스 심볼들을 운반하는 반면, 음영처리된 부분들은 파일의 대응하는 부분으로부터 발생된 리페어 심볼들을 운반한다. 이 예에서, 파일의 제 1 부분을 복원하기 위해 필요한 패킷들의 최소 수는 28인 반면, 파일의 제 2 부분을 복원하기 위해 필요한 패킷들의 최소 수는 15이다.

[0108] 도 9는 UFSI들 0, ..., 27인 28 개의 패킷들을 도시하고, 이에 따라, 이 패킷들에서 운반되는 파일의 제 1 부분에 대한 인코딩된 심볼들의 전부는 소스 심볼들이다. UFSI 값들 적어도 28로 발생된 임의의 추가적인 패킷들은 파일의 제 1 부분에 대한 리페어 심볼들을 운반할 것이다.

[0109] UFD-UEP 방법의 또 다른 장점은, 인코딩된 심볼들이 그 UFSI에 의해 정의된 순서로, 즉, UFSI 순서 0, 1, 2, 3, 4, ...로 전송될 경우, 파일의 부분 i의 Z_i 소스 블록들에 대한 인코딩된 심볼들이 인터리빙된 순서로 전송되며, 즉, Z_i 소스 블록들의 각각으로부터 ESI 0인 Z_i 인코딩된 심볼들이 먼저 전송되고, 그 다음으로, Z_i 소스 블록들의 각각으로부터 ESI 1인 Z_i 인코딩된 심볼들, 등이 전송된다는 것이다. 이 특성은 각각의 부분이 독립적인 소스 블록 구조를 가지더라도, 파일의 모든 부분들에 대해 유효하다. 대부분의 송신들에 대하여, 이 간단한 전송 순서는 충분하고 바람직하다. 그러나, 패킷 손실이 소스 블록들 Z_i 의 수와 동기화될 수도 있다는 일부의 주기성을 경험할 경우, 잠재적으로 더 양호한 전송 순서는 Z UFSI-연속 인코딩된 심볼들의 각각의 세트를 전송하기 전에 무작위로 치환하는 것이며, 여기서 Z는 모든 i = 0, ..., d-1에 대한 Z_i의 최대 값이고, 즉, UFSI들 0, ..., Z-1인 제 1 Z 인코딩된 심볼들이 무작위 치환된 순서로 전송되고, 그 다음으로, UFSI들 Z, ..., 2*Z-1인 다음 Z 인코딩된 심볼들이 무작위 치환된 순서로 전송되는 등등과 같다. 또 다른 잠재적인 전송 순서는 전송될 인코딩된 심볼들의 전부를 전송하기 전에 무작위로 치환하는 것이다.

[0110] UFD-UEP 방법을 이용하는 것은 이전의 방법들 또는 이전의 방법들의 간단한 확장안들에 비해 다수의 추가적인 장점들을 제공한다. 예를 들어, UFD-UEP 방법이 UFSI 필드를 포함하는 FEC 페이로드 ID 필드를 이용하면, 파일의 각각의 부분에 대한 가능한 인코딩된 심볼들의 총 수는 UFSI 필드의 크기에 의해서만 제한되고, 파일의 각각의 부분의 소스 블록 구조에 독립적이다. 또한, UFSI 필드의 이용은 파일의 각각의 부분에 대한 완전히 상이한 소스 블록 구조들로부터 발생된 심볼들의 동시 식별을 허용하는 유니버셜하고 간결한 FEC 페이로드 ID를 제공한다. 예를 들어, 32-비트 FEC 페이로드 ID로 귀착되는 32-비트 UFSI를 이용할 때, 파일이 얼마나 많은 소스 블록들로 구획되는지에 관계없이, 그리고 파일의 각각의 부분에 대하여 서브-블록킹이 이용될 경우의 파일의 서브-블록 구조에 관계없이, 파일에 대해 발생될 수 있는 인코딩된 심볼들의 총 수는 4,294,967,296이다. 따라서,

파일의 제 1 부분에 대한 심볼이 크기에 있어서 256 옥테트들이고, 파일의 제 2 부분에 대한 심볼이 크기에 있어서 1,024 옥테트들일 경우, 이 예에서는, 파일의 제 1 부분이 1 GB 크기이며 파일의 제 2 부분이 4 GB 크기이고, 각각의 파일 부분이 하나의 소스 블록, 16,384 소스 블록들, 또는 4,194,304 소스 블록들로 구획되는지에 관계없이, 인코딩된 심볼들의 수가 각각의 파일 부분에 대한 소스 심볼들의 수의 1,024배일 수 있다.

[0111] 도 10은 [RaptorQ-RFC-6330]을 이용하는 간단한 UEP 파일 전달 방법의 성능을 예시한다. 거기의 예에서, UEP 파일 전달은 100 리페어 패킷들로 보호된 30 소스 패킷들과, 900 리페어 패킷들로 보호된 970 소스 패킷들을 위한 것이다.

[0112] 도 11은 UFD-UEP 파일 전달 방법이 도 10의 간단한 UEP 파일 전달 방법에 비해 제공하는 개선들의 예를 예시한다. 이 예에서, 1 MB의 파일은 32 KB의 제 1 부분 및 992 KB의 제 2 부분으로 구획된다. 두 방법들에 대하여, [RaptorQ-RFC-6330]에서 특정된 FEC 코드들이 이용되고, 인코딩된 심볼들을 운반하기 위한 각각의 패킷 내의 크기는 1,024 옥테트들이고, 총 2,048 패킷들이 송신된다.

[0113] 도 11에 도시된 간단한 UEP 파일 전달 방법의 예에 대하여, 파일의 제 1 부분 및 파일의 제 2 부분은 독립적으로 프로세싱 및 전달되고, 여기서 두 경우들에 있어서, 크기 1,024 옥테트들의 정확하게 하나의 인코딩된 심볼이 각각의 패킷에서 운반된다. 파일의 제 1 부분에 대한 소스는 32 패킷들로 운반되고, 총 128 패킷들이 인코딩된 심볼들을 포함하여 전송된다. 파일의 제 2 부분에 대한 소스는 992 패킷들로 운반되고, 총 1,920 패킷들이 인코딩된 심볼들을 포함하여 전송된다.

[0114] 도 11에 도시된 UFD-UEP 파일 전달 방법의 예에 대하여, 파일의 제 1 부분 및 파일의 제 2 부분은 조합된 방식으로 프로세싱 및 전달되고, 즉, 전송되는 각각의 패킷은 2 개의 부분들의 각각에 대한 인코딩된 심볼을 포함하고, 여기서 제 1 부분에 대하여, 인코딩된 심볼의 크기는 64 옥테트들이고, 제 2 부분에 대하여, 인코딩된 심볼의 크기는 960 옥테트들이다. 파일의 제 1 부분에 대한 소스는 512 패킷들로 운반되고, 모두 2,048 패킷들이 제 1 부분에 대한 인코딩된 심볼을 포함한다. 파일의 제 2 부분에 대한 소스는 1059 패킷들로 운반되고(소스의 최종 패킷에서의 인코딩된 심볼, 992 옥테트들의 전체 심볼 크기에 제로들로 패딩된 제 2 부분에 대한 인코딩된 심볼), 모두 2,048 패킷들이 제 2 부분에 대한 인코딩된 심볼을 포함한다.

[0115] 도 11에서 알 수 있는 바와 같이, 간단한 UEP 파일 전달 방법 및 UFD-UEP 파일 전달 방법의 복원 성능은 패킷 손실 레이트의 함수로서 파일의 제 2 부분에 대해 실제적으로 동일하고, 즉, 두 경우들에 있어서, 파일의 제 2 부분은 48 %에 접근하는 패킷 손실 레이트까지 상당히 지속적으로 복원된다. 다른 한편으로, UFD-UEP 파일 전달 방법의 복원 성능은 파일의 제 1 부분에 대하여 간단한 UEP 파일 전달 방법의 복원 성능보다 상당히 더 양호하며: 간단한 UEP 파일 전달 방법은 65 %보다 더 적은 패킷 손실 레이트들에 대하여 파일의 제 1 부분을 지속적으로 복원할 수 있는 반면, UFD-UEP 파일 전달 방법은 75 %에 접근하는 패킷 손실 레이트들에 대하여 파일의 제 1 부분을 지속적으로 복원할 수 있다.

[0116] 유니버셜 파일 전달 방법 및 번들제공된 파일 전달 서비스들을 위한 시스템

[0117] 가장 이전의 파일 전달 방법들은 번들제공된 파일 전달, 즉, 단일의 번들제공된 파일로서의 다수의 파일들의 전달을 지원하지 않는다. 몇 개의 파일들을 전달하는 간단한 방법은 각각의 파일을 독립적으로 전달하는 것이다. 그러나, 이 간단한 방법은 일부의 단점들을 가진다. 예를 들어, 각각의 파일에 대한 인코딩된 심볼들을 포함하는 패킷들의 수가 작을 경우에는 패킷 손실 통계에 있어서 큰 분산이 있을 수 있으므로, 제공되는 보호는 파일들이 작을 경우에 이상적인 것이 아니다.

[0118] 도 12는 이 쟁점을 예시한다. 도 12에는, 32 KB 파일의 파일 전달의 신뢰성이 전달 동안의 네트워크에서의 패킷 손실의 백분율의 함수로서 도시되어 있다. 이 파일 전달 예에서는, 심볼들이 크기 1,024이고, 파일의 32 소스 심볼들이 [RaptorQ-RFC-6330]에서 특정된 FEC 코드들을 이용하여 64 개의 인코딩된 심볼들로 인코딩되고, 각각의 인코딩된 심볼은 별개의 패킷에서 전송된다. 도 12에서 알 수 있는 바와 같이, 파일의 신뢰성 있는 전달이 달성될 수 있는 손실의 백분율은 이런 분산으로 인해 50 %보다 훨씬 적다.

[0119] 또한, 다수의 작은 파일들이 독립적으로 인코딩 및 송신될 경우, 모든 파일들이 신뢰성 있게 수신되는 패킷 손실의 백분율은 훨씬 적다. 도 12는 크기가 각각 32 KB인 32 개의 파일들의 전달을 위한 이러한 거동을 도시하고, 여기서 각각의 파일의 인코딩 및 송신은 이전의 단락에서 설명된 것과 동일한 파라미터들을 이용하여 다른 파일들에 관계없이 수행된다. 알 수 있는 바와 같이, 모두 32 개의 파일들의 전달은 패킷 손실이 50 %보다 훨씬 적은 약 25 % 미만일 때에만 신뢰성 있게 달성될 수 있다.

[0120] UFD-UEP 파일 전달 방법은 UFD-번들제공된 파일 전달 방법을 제공하기 위하여 아래와 같이 확장될 수 있다.

UFD-번들제공된 파일 전달 방법은, 동일한 파일의 d 부분들의 전달을 시그널링하는 대신에, 각각의 부분이 별개의 파일이고 d 파일들은 번들로서 제공되고 있다는 것을 시그널링하는 대신에, UFD-UEP 파일 전달 방법과 동일한 방법들을 이용할 수 있다. 전송기가 각각 크기들 F_0, F_1, ..., F_{d-1}의 d 파일들의 번들제공된 전달을 제공하는 것을 원한다고 가정한다. 전송기의 UFD-번들제공된 방법은 패킷 크기 T를 크기들 T_0, T_1, ..., T_{d-1}의 d 부분들로 구획하고, 이에 따라 T는 T_i의 i에 걸친 합과 동일하다. T의 이 구획은 F_0, F_1, ..., F_{d-1}과 대응하는 파일들의 우선순위들에 기반으로 한다.

[0121] 비율 F_i/T_i는, 이상적인 FEC 코드가 하나의 소스 블록으로서 파일의 부분 i를 보호하기 위하여 이용되는 것으로 추정할 때, 파일 i를 복원하기 위하여 얼마나 많은 패킷들이 수신될 필요가 있는지를 결정하고, 이에 따라, 비율 F_i/T_i가 더 작을수록, 파일의 부분 i의 우선순위가 더 높다. 실제로는, 예를 들어, FEC 코드가 완전하지 않고 일부의 수신 오버헤드를 나타내기 때문에, 또는 파일의 그 부분이 다수의 소스 블록들로 구획되고 일부의 소스 블록들에 대한 인코딩된 심볼들이 다른 소스 블록들에 대한 것보다 더 높은 레이트에서 손실되기 때문에, 또는 F_i/T_i가 정수가 아니기 때문에, F_i/T_i보다 약간 더 많은 패킷들이 파일의 부분 i를 복원하기 위하여 필요할 수도 있다. 전달될 모든 파일들의 우선순위가 동일하기를 희망하는 경우, T_i는 $F_i/T_i \approx F/T$가 되도록 설정된다. 전송기 및 수신기 둘 모두에 대한, UFD-번들제공된 파일 전달 방법의 다수의 세부사항들은 UFD-UEP 파일 전달 방법과 거의 동일하고, 이에 따라 생략된다.

[0122] UFD-번들제공된 파일 전달 방법은 번들제공된 파일 전달 및 UEP 파일 전달 둘 모두를 동시에 제공하도록 확장될 수 있고, 즉, 번들에서의 각각의 파일의 우선순위는 상이하게 설정될 수도 있다. 또한, UFD-번들제공된 파일 전달 방법은 적당한 시그널링으로, 다수의 파일들의 우선순위화된 전달 및 파일의 부분들의 우선순위화된 전달의 둘 모두의 전달을 지원할 수 있다. 예를 들어, 3 개의 객체들이 UFD-번들제공된 파일 전달 방법들을 이용하여 인코딩 및 전송되어야 할 경우, 최초 2 개의 객체들은 상이한 우선순위들을 갖는 동일한 파일의 상이한 부분들일 수도 있고, 제 3 객체는 또한 상이한 우선순위를 갖는 상이한 파일일 수도 있다. 수신기는 다수의 인자들에 기반으로 하여, 번들제공된 파일들 및/또는 파일 부분들의 어느 것이 복원하는 것에 관심이 있는지를 판정할 수 있고, 다른 파일들 또는 파일들의 부분들에 관계없이 그러한 파일들 또는 파일들의 부분들만을 복원할 수 있다. 당업자가 인식하는 바와 같이, 이 개시내용을 판독 시에, UFD-번들제공된 파일 전달 방법들의 다수의 가능한 대안적인 버전들이 있다.

[0123] UFD-번들제공된 파일 전달의 간단한 예로서, 각각 크기 32 KB인 32 개의 파일들의 번들이 전달되어야 하고, 여기서 각각의 파일은 동일한 우선순위를 가진다고 가정한다. T = 1,024 옥테트들이라고 가정한다. 이 경우, 각각의 i = 0, ..., 31에 대하여, T_i = 32 옥테트들의 값이다. 각각의 패킷은 32 개의 파일들의 각각에 대하여 32-옥테트 인코딩된 심볼을 포함할 것이고, UFSI는 패킷에서 32 개의 인코딩된 심볼들을 식별한다. 이 예에서는, 32 개의 동일-크기 파일들의 각각으로부터의 소스 심볼들을 포함하는 1,024 패킷들이 있을 것이고, 이들은 범위 0 내지 1,023의 UFSI들을 갖는 패킷들이다. 이 예에서, 1,024 개의 추가적인 리페어 심볼들이 각각의 파일에 대해 발생되고 1,024 내지 2,047 범위의 UFSI 들을 갖는 추가적인 1,024 패킷들에서 전송된다고 가정한다.

[0124] 도 12는 패킷 손실의 함수로서 이 UFD-번들제공된 파일 전달의 예의 복원 특성들을 예시한다. 이 예에서는, 모든 32 개의 파일들이 50 %에 접근하는 패킷 손실 레이트들에 대하여 UFD-번들제공된 파일 전달 방법을 이용하여 신뢰성 있게 복원될 수 있고, 이 50 %는 각각의 파일의 별개의 인코딩 및 전송을 이용하여 모든 32 개의 파일들의 전달을 신뢰성 있게 허용하는 대략 25 % 패킷 손실 레이트에 비해 실질적인 개선이다.

[0125] UFD 및 UEP 파일 전달 방법들은 다수의 다른 애플리케이션들을 가질 수도 있다. 예를 들어, DASH 포맷팅된 컨텐츠의 세그먼트들은 3GPP eMBMS 표준에 따른 네트워크들과 같은 멀티캐스트 또는 브로드캐스트 네트워크를 통해 전달될 수도 있다. 그 경우, DASH 포맷팅된 컨텐츠는 나중에 재생될 최종 사용자 이동 디바이스들로 전달될 수도 있고, DASH 포맷팅된 컨텐츠의 각각의 세그먼트는 1 Mbps에서 인코딩된 비디오 컨텐츠의 8 초를 포함할 수도 있고, 즉, 각각의 세그먼트는 1 MB 크기이다.

[0126] DASH 포맷팅된 컨텐츠가 궁극적으로 이동 디바이스 상에서 최종 사용자에 의해 재생될 때, 재생의 높은 충실도 즉, 비디오 아티팩트들 또는 스킵들 또는 버퍼링이 없는 것이 바람직하고, 최종 사용자는 전달 컨텐츠의 일부분들을 재생하는 것을 희망하기만 할 수도 있다. 예를 들어, 컨텐츠를 일단 DASH 포맷으로 포맷팅하고 재포맷팅 없이 다양한 전송들을 통해 그것을 전달할 수 있기 위한, 전송 및 재생에서의 효율 및 유연성을 위해서는, 각각의 세그먼트가 별개의 파일로서 전달되는 것이 바람직하다.

[0127] 네트워크 효율을 위해서는, 각각의 DASH 세그먼트가 FEC 보호되고 다른 세그먼트들과 완전히-인터리빙되어 전달되는 것이 바람직하고, 바람직하게는, 패킷들 내에서 UFD 및 UEP 전달 방법들에 의해 제공되는 인터리빙이 사용될 수 있다. 예를 들어, 전달될 150 개의 세그먼트들이 있고, 이에 따라, 전달될 DASH 컨텐츠의 전체 크기는 1200 초 또는 동등하게는 20분의 재생 프리젠테이션 시간을 포함하는 150 MB 라는 것을 가정한다. 컨텐츠가 1200 바이트들의 페이로드들을 각각 갖는 패킷들로 전달되어야 한다고 가정한다. 다음으로, 여기에서 설명된 UFD 및 UEP 전달 방법을 이용하면, 각각의 패킷의 8 바이트들이 150 개의 세그먼트들의 각각에 할당될 수도 있고, 즉, 각각의 패킷은 150 개의 DASH 세그먼트들의 각각에 대하여 8-바이트 심볼을 운반한다. 다음으로, 패킷 손실 패턴들에 관계없이, 각각의 DASH 세그먼트는 대략 131,072 패킷들의 이동 수신 디바이스로의 도달 시에, 즉, 150 개의 세그먼트들의 각각에 대한 1 MB 데이터의 도달 시에 복원될 수 있다. 이 예에서, 150 개의 DASH 세그먼트들의 각각은 다른 DASH 세그먼트들을 디코딩해야 할 필요없이 수신 이동 디바이스에서 디코딩되고 선택적으로 재생될 수 있다.

[0128] 여기에서 설명된 UFD 및 UEP 파일 전달 방법들의 또 다른 예로서, 3GPP eMBMS 표준에 따른 네트워크와 같은 멀티캐스트 또는 브로드캐스트 네트워크를 통해 전달되는 DASH 포맷팅된 컨텐츠를 전달하는 것을 고려하고, DASH 컨텐츠는 비디오 세그먼트들 및 오디오 세그먼트들의 시퀀스를 포함하고, 각각의 비디오 세그먼트는 1 Mbps에서 인코딩된 비디오 컨텐츠의 8 초를 포함하고, 즉, 각각의 비디오 세그먼트는 1 MB 크기이고, 각각의 오디오 세그먼트는 32 Kbps에서 인코딩된 대응하는 오디오 컨텐츠의 8 초를 포함하고, 즉, 각각의 오디오 세그먼트는 32 KB 크기이고, 비디오 세그먼트 및 대응하는 오디오 세그먼트는 8 초의 프리젠테이션 시간의 주로 동일한 간격에 대한 컨텐츠를 포함한다.

[0129] 이 예에서, DASH 컨텐츠는 스트리밍되어야 하고, 즉, 각각의 비디오 세그먼트 및 대응하는 오디오 세그먼트는 그 프리젠테이션 시간들의 순서로 시퀀스로 전달되어야 한다. 컨텐츠는 1200 바이트들의 페이로드들을 각각 갖는 패킷들로 전달되어야 한다. 다음으로, 여기에서 설명된 UFD 및 UEP 전달 방법들을 이용하면, 각각의 패킷의 1160 바이트들은 비디오 세그먼트에 할당될 수도 있고, 각각의 패킷의 나머지 40 바이트들은 대응하는 오디오 세그먼트에 할당될 수도 있고, 즉, 각각의 패킷은 비디오 세그먼트에 대한 1160-바이트 심볼 및 대응하는 오디오 세그먼트에 대한 40-바이트 심볼을 운반한다. 다음으로, 패킷 손실 패턴들에 관계없이, 비디오 세그먼트는 그 세그먼트에 대한 심볼들을 포함하는 대략 905 패킷들의 수신으로부터 복원될 수 있고, 대응하는 오디오 세그먼트는 그 세그먼트에 대한 심볼들을 포함하는 대략 820 패킷들로부터 복원될 수 있다.

[0130] 따라서, 대응하는 비디오 및 오디오 세그먼트가 있는 프리젠테이션 시간의 8 초 간격에 대응하는 임의의 820 패킷들의 전체적인 수신은 이동 수신 디바이스가 오디오 세그먼트를 복원하도록 하고, 추가적인 85 패킷들, 또는 총 905 패킷들의 수신은 대응하는 비디오 세그먼트의 복원을 또한 허용한다. 이 예에서, 오디오는 비디오보다 더욱 보호되고, 이것은 다수의 경우들에 있어서 바람직한데, 예를 들어, 비디오 재생의 타이밍이 오디오 재생의 타이밍에 의해 기술되고, 다수의 경우들에 있어서, 오디오 및 비디오 둘 모두를 재생하기 위하여 충분한 데이터가 수신되지 않는 경우에 오디오를 적어도 재생하는 것이 바람직하기 때문이다.

[0131] 유니캐스트 리페어 요청들의 네이티브 HTTP 지원을 위한 방법들

[0132] 파일 또는 파일 부분은 수신기들에 의한 파일 또는 파일 부분들에 대한 액세스를 저장 및 제공하기 위하여 표준 HTTP 웹 캐쉬 서버들에 의해 이용될 수 있는 연관된 URL을 갖는 "HTTP 파일"로서 조직화되고 이용가능하게 될 수도 있다. HTTP 파일로서 이용가능하게 된 그 원래의 순서에서의 파일 또는 파일 부분은 여기에서 "원래-순서 HTTP 파일"이라고 칭해진다. 일반적으로 유니캐스트 리페어 요청들의 네이티브 HTTP 지원을 위한 방법은 SBN들 및 ESI들에 기반으로 하여, HTTP 파일의 소스 블록들의 심볼들 및 서브-심볼들에 대한 리페어 요청들을 HTTP 파일 내의 표준 HTTP(예를 들어, 네이티브 HTTP 1.1) 옥테트 범위 요청들(종종 특정 바이트 범위를 갖는 HTTP 부분 GET 요청들이라 칭함)로 전환할 수 있다. 이것은 표준 HTTP 웹 캐쉬 서버들이 이 리페어 요청들을 서비스할 수 있도록 하고, 이것은 HTTP 파일이 소스 블록들 및 소스 블록 내의 소스 심볼들로 어떻게 구획되는지를 이해하는 특화된 HTTP 웹 캐쉬 서버들을 대량으로 설치할 필요성을 회피한다.

[0133] 이러한 시나리오들에서, 체계적인 FEC 코드가 이용될 때, 세션들, 예를 들어, 브로드캐스트/멀티캐스트 세션들 중의 일부에서 리페어 심볼들을 전송하기만 하는 것이 종종 유리한데, 이것은 수신기들이 유니캐스트 리페어 요청들에서 소스 심볼들만을 요청하도록 하기 때문이다. 이것은 HTTP 파일, 또는 표준 HTTP 웹 캐쉬 서버들일 수 있는 유니캐스트 리페어 서버들에 제공될, 아래에서 더욱 상세하게 설명되는 바와 같은 HTTP 파일의 간단한 재순서화 변환(reordering transformation)을 요구하기만 하는 장점을 가지며, 이것은 HTTP 파일을 정의하고 HTTP 파일을 분배하는 실행계획들을 더 간단하게 하는데, 이 실행계획들이 FEC 인코딩에 독립적이기 때문이다.

또 다른 장점은, 소스 심볼들이 세션들에서 전송되지 않는 경우, 소스 심볼들의 전부는 "누락" 되어 있으므로, 수신기는 유니캐스트 리페어 요청들에서 소스 심볼들의 임의의 시퀀스를 요청할 수 있는데, 이것은 소스 심볼들의 연속적인 시퀀스에 대한 리페어 요청들을 허용하므로 바람직할 수 있다는 것이다. 예를 들어, 완전한 복원 특성들을 갖는 FEC 코드가 이용되고 부분의 소스 블록이 1,000 개의 소스 심볼들을 포함하고, (리페어 심볼들의 일부가 송신 시에 손실될 수도 있고, 이에 따라 수신된 리페어 심볼들의 ESI들의 패턴이 연속적이지 않을 수도 있더라도) 700 개의 리페어 심볼들이 소스 블록에 대해 수신된다고 가정한다. 다음으로, 수신기는 유니캐스트 리페어 요청들에서 소스 블록에 대한 최초 300 개의 연속적인 소스 심볼들을 요청할 수 있고, 소스 블록을 복원하기 위하여 이 소스 심볼들을 700 개의 리페어 심볼들과 조합할 수 있다. 소스 심볼들이 HTTP 파일에서 연속적일 경우, 단일의 HTTP 옥테트 범위 요청이 모든 300 개의 연속적인 소스 심볼들을 요청 및 수신하기 위하여 이용될 수 있다.

[0134] 수신기는 프리픽스 요청(즉, 최초 바이트로 시작하는 파일의 설정된 바이트들의 수에 대한 요청)을 반드시 행할 필요가 없지만, 다수의 수신 디바이스들이 있고 브로드캐스트 또는 멀티캐스트 세션으로부터 패킷들을 수신할 때에 상이한 수신 디바이스들에 의해 경험되는 임의적으로 상이한 손실 패턴들이 있더라도, 모든 UE들이 프리픽스 요청들을 행할 것을 요구하는 것은, UE들로부터의 결과적인 요청들이 매우 중첩될 것이므로 HTTP 서버들에서 캐싱 효율을 상당히 개선시킨다.

[0135] 설명된 그러한 이점들 중의 일부에 기반으로 하여, 위에서 설명된 기술들은 리페어 심볼들만이 브로드캐스팅되고 있음을 수신기들에게 시그널링하기 위한 네트워크 또는 네트워크 디바이스에 유리할 수 있다. 이 표시를 시그널링받을 때, 수신기들은 자신들이 어떤 소스 심볼들을 수신하는지를 계속 추적해야 할 필요가 없고, 자신들이 수신하는 심볼들의 수를 추적하기만 하면 될 수도 있다. 예를 들어, 수신된 총 심볼들에 기반으로 하여, 수신기들은 리페어 서버로부터 얼마나 더 많은 소스 심볼들을 요청할 것인지를 결정할 수 있다. 캐싱 효율을 개선시키기 위하여, 수신기들은 리페어 심볼들만이 브로드캐스팅되고 있다는 표시의 정보에 기반으로 하여 누락된 소스 심볼들의 수에 대한 프리픽스 요청들을 행하도록 구성될 수 있다.

[0136] 대안적으로, 프리픽스 요청들을 행하기 위하여 별개의 시그널링 표시가 수신기들에 제공될 수 있다. 또 다른 실시예에서, 수신기들은, 수신기들이 누락된 소스 블록들의 상이한 서브세트들 사이에서 선택하기 위한 유연성을 가질 때마다 최저 심볼 인덱스 값들(즉, 소스 블록 또는 서브-블록의 시작에 가장 근접한 것들)을 갖는 누락된 소스 심볼들을 항상 선택하도록 구성될 수 있다. 이것은, 소스 및 리페어 심볼들의 둘 모두가 브로드캐스트 채널을 통해 전송될 경우에도 그러할 수 있다.

[0137] 또 다른 실시예에서는, 수신 디바이스들이 수신된 심볼 식별자들, ESI들을 계속 추적하고, 예를 들어, FEC OTI에 기반으로 하여, 또는 수신 패킷들에서 운반된 FEC 페이로드 ID에 기반으로 하여, 자신들이 리페어 심볼들에만 대응하는 ESI들을 수신하는지를 결정한다. 그러한 경우, 이러한 수신기들은 프리픽스 요청들을 행할 수 있다. 이 실시예에서는, 명시적인 시그널링을 이용하지 않으면서, 수신기는 리페어 심볼들을 수신하기만 할 경우에 리페어 요청들에서 소스 블록들 및 서브-블록들의 프리픽스들에 대한 요청들을 전송할 것이다. 수신 디바이스들이 브로드캐스트 채널로부터 리페어 심볼들을 수신하는 경우에도, 수신된 심볼들의 패턴과, 간단한 요청들을 행하는 것과 브로드캐스트 채널을 통해 이미 수신된 중복 데이터를 잠재적으로 수신하는 것과의 사이의 절충들에 따라서는, 리페어 요청들이 여전히 대체로 또는 완전히 프리픽스 요청들일 수 있다.

[0138] 리페어 심볼들만이 브로드캐스팅되고 있음을 시그널링하고, 수신기들이 프리픽스 요청들을 행하고 있음을 시그널링할 때에는, HTTP 바이트 범위 파일 리페어 요청들 또는 심볼-기반 파일 리페어 요청들이 이용될 때에 일부의 장점들이 발생한다. 그러므로, 이 방법들은 어느 하나 또는 둘 모두의 요청 포맷들을 지원하는 수신기들 및 네트워크들에 적용될 수 있다.

[0139] 리페어 심볼들만이 브로드캐스팅되고 있음을 표시하기 위한 시그널링은 (유니캐스트를 통해) 대역외(out-of-band)로 또는 (브로드캐스트를 통해) 대역내(in-band)로 전송될 수 있다. 표시는 희망하는 입도(granularity)의 레벨에 따라 서비스 레벨, 세션 레벨, 미디어 레벨, 또는 심지어 파일 레벨에 적용가능할 수 있다. 예를 들어, 시그널링은 사용자 서비스 설명("USD : User Service Description")에서 추가될 수 있다. USD는 식별된 서비스들 전부의 전달을 위하여 리페어 심볼들만이 브로드캐스팅 된다는 것을 표시할 수도 있다. 시그널링은 미디어 프리젠테이션 설명("MPD : Media Presentation Description")에서 또한 추가될 수 있고, 여기서 MPD는 리페어 심볼들만을 이용하여 어느 미디어가 브로드캐스팅 되는지를 표시한다. 시그널링은 전체 세션 또는 세션에서의 미디어 컴포넌트들의 전달에 적용되는지를 표시하기 위하여 세션 설명 프로토콜("SDP : Session Description Protocol")에서 추가될 수 있다. 시그널링은 리페어 심볼들의 브로드캐스트가 파일 레벨 상에서

적용되기만 하는 것을 표시하기 위하여 파일 설명 테이블("FDT : File Description Table")에서 또한 추가될 수 있다.

[0140] 수신기들이 리페어 데이터에 대한 프리픽스 요청들을 행하도록 요구하기 위한 시그널링은 대역내 및 대역외 전송들을 통해 또한 전달될 수 있다. 표시는 희망하는 입도의 레벨에 따라 서비스 레벨, 세션 레벨, 미디어 레벨, 또는 심지어 파일 레벨에 적용하도록 또한 행해질 수도 있다.

[0141] 따라서, 이러한 표시를 제공하기 위한 다양한 방법들 및 장치가 있고, 수신기들은, 이러한 표시가 수신되는 경우와 이러한 표시가 수신되지 않는 경우에 상이하게 응답하도록 프로그래밍 및/또는 구성될 수 있다.

[0142] 요청은 바이트들 또는 심볼들의 특정한 수에 대한 것일 수 있다. 일부의 경우들에 있어서, 수신기는 수신된 리페어 심볼들의 수와, 수신될 파일들 또는 객체들에 포함될 것으로 예상되는 소스 심볼들의 수에 기반으로 하여 요청하기 위한 소스 심볼들의 수를 결정할 것이다. 다른 경우들에 있어서, 수신기는 스케줄링 동작을 수행할 수도 있고, 수신기는 이미 수신한 리페어 심볼들만을 이용하여 어떻게 디코딩할 것인지를 결정하고, 이에 따라, 필요한 소스 심볼들의 더욱 특정한 수를 언급할 수 있다. 예를 들어, 리페어 심볼들의 일부는 다른 리페어 심볼들의 중복인 것일 수도 있고, 이 경우에 더 많은 소스 심볼들이 요구될 수도 있다. 다른 사례들에서는, 더 적은 소스 심볼들이 필요한 것으로 판명될 수도 있다.

[0143] 수신기는 FEC OTI에 기반으로 하여, 특정한 (SBN, ESI) 쌍들에 대응하는 원래-순서 HTTP 파일의 소스 심볼들에 대한 요청들을 HTTP 옥테트 범위 요청들로 전환할 수 있다. 예를 들어, 파일에 대한 FEC OTI가 (F, Al, T, Z, N)이고, 요청될 소스 심볼에 대한 SBN이 A 이며 소스 심볼에 대한 ESI가 B라고 가정하고, 간단함을 위하여, N = 1, 즉, 소스 블록들이 파일 구조의 이 예에서 서브-블록들로 더 구획되지 않는다고 가정한다. 그 다음으로, 수신기는 (KL, KS, ZL, ZS)를 결정하기 위하여 예를 들어, [RaptorQ-RFC-6330]의 섹션 4.4에서 설명된 방법들을 적용할 수 있고, 여기서 최초 ZL 소스 블록들은 KL 소스 심볼들을 가지고 나머지 ZS 소스 블록들은 KS 소스 심볼들을 가진다. 다음으로, A, B, 및 심볼 크기 T에 기반으로 하여, 수신기는 파일 내의 소스 심볼의 시작 옥테트 인덱스 SI가 수학식 1에 도시된 바와 같다고 결정할 수 있고, 여기서 파일 내의 소스 심볼의 종료 옥테트 인덱스는 EI = SI + T-1이다. 다음으로, 수신기는 옥테트들 SI 내지 EI를 요청하는, 소스 심볼에 대한 표준 HTTP 옥테트 범위 요청을 이용할 수 있다.

수학식 1

[0144] $$SI = T*(\mathrm{KL}*\min\{A, \mathrm{ZL}\} + \mathrm{KS}*\max\{A-\mathrm{ZL},0\} + B)$$

[0145] 당업자가 인식하는 바와 같이, 이 방법들에 대한 다수의 확장들 및 개선들이 있다. 예를 들어, 단일의 HTTP 옥테트 범위 요청은, 서브-블록킹이 이용되지 않을 경우, 다수의 연속적인 소스 심볼들이 원래-순서 HTTP 파일로부터 요청될 경우에 행해질 수도 있다. 또 다른 예로서, HTTP 웹 캐쉬 서버들은 원래-순서 HTTP 파일에 추가적으로 또는 그 대신에, 리페어 심볼들을 포함하는 파일들을 가질 수도 있거나, 원래-순서 HTTP 파일이 리페어 심볼들을 포함하도록 이미 확장되었을 수도 있고, 수신기는 리페어 심볼들에 대한 HTTP 옥테트 범위 요청들을 행하기 위하여 여기에서 설명된 것들과 유사한 방법들을 이용할 수 있다. 개선의 또 다른 예로서, 이 방법들은 당업자들에 의해 인식되는 것과 유사한 방법들을 이용하여, 서브-블록킹이 이용될 경우를 처리하도록 확장될 수 있다. 예를 들어, 수신기는 (TL, TS, NL, NS)를 결정하기 위하여 [RaptorQ-RFC-6330]의 섹션 4.4에서 설명된 방법들을 이용할 수 있고, 여기서 소스 블록의 최초 NL 서브-블록들은 크기가 TL*Al이고, 소스 블록의 나머지 NS 서브-블록들은 크기가 TS*Al이다. 다음으로, A, B, 및 심볼 크기 T에 기반으로 하여, 수신기는 소스 심볼에 대응하는 파일 내의 N 서브-심볼들의 시작 및 종료 옥테트 인덱스를 결정할 수 있고, 표준 HTTP 옥테트 범위 요청들을 이용하여 이 옥테트들에 대한 요청들을 행할 수 있다.

[0146] 또 다른 예로서, 수신기는 FEC OTI에 기반으로 하여 특정한 UFSI들에 대응하는 파일 또는 파일 부분의 소스 심볼들에 대한 요청들을 HTTP 옥테트 범위 요청들로 전환할 수 있다.

[0147] 다음의 방법들은 HTTP 파일의 심볼들을 복원하기 위하여 수신기에 의한 표준 HTTP 바이트 범위 요청들을 이용하는 한편, 이와 동시에, 요청된 옥테트 범위 요청들을 수신기들에 전달하기 위하여 표준 HTTP 웹 캐쉬 서버들을 이용하는 효율을 대폭 개량시킬 수 있다.

[0148] 설명된 바와 같이 HTTP 옥테트 범위 요청들을 지원하기 위한 간단한 방법은 원래-순서 HTTP 파일을 이용하는 것

이다. 이 방법은, HTTP 웹 캐쉬 서버들에 대한 원래-순서 HTTP 파일을 발생하기 위하여 원래의 파일 또는 파일 부분의 변환이 필요하지 않다는 장점을 가지지만, 각각의 소스 블록에 대해 연속적인 소스 심볼들만이 요청되더라도, 소스 심볼들을 요청하기 위하여 다수의 상이한 HTTP 옥테트 범위 요청들이 필요할 수도 있다. 이것에 대한 적어도 2 개의 이유들이 있으며: (1) 다수의 소스 블록들과, 각각의 소스 블록으로부터의 누락된 소스 심볼들이 있을 수도 있고, 이 경우에 별개의 옥테트 범위 요청들이 각각의 소스 블록에 대해 요구될 수도 있고; (2) 소스 블록당 다수의 서브-블록들이 있을 수도 있고, 이에 따라, 소스 심볼에 대한 서브-심볼들이 원래-순서 HTTP 파일에서 연속적이지 않을 수도 있으므로, 하나의 소스 블록으로부터 단일의 소스 심볼을 요청하는 것조차 소스 심볼을 포함하는 다수의 서브-심볼들에 대한 다수의 HTTP 옥테트 범위 요청들을 요구할 수 있다.

[0149] 위에서 설명된 HTTP 옥테트 범위 요청들을 지원하기 위한 유리한 방법은 우선, 원래의 파일 또는 파일 부분에 대한 FEC OTI에 기반으로 하여, 파일 또는 파일 부분을 여기에서 "UFSI-순서 HTTP 파일"이라고 칭하는 새로운 포맷으로 재조직화(reorganize) 한다. 이 방법은, 다수의 소스 블록들 및/또는 소스 블록당 다수의 서브-블록들이 있을 때에 유용하다. UFSI-순서 HTTP 파일에서의 데이터의 순서는 UFSI 0을 갖는 파일 심볼, UFSI 1을 갖는 파일 심볼, UFSI 2를 갖는 파일 심볼, 등의 순서이고, 즉, UFSI-순서 HTTP 파일에서의 데이터의 순서는 FEC OTI에 의해 결정되는 바와 같이, 증가하는 연속적인 UFSI들에 따라 순서화된 파일 심볼들로 조직화된다. URL은 UFSI-순서 HTTP 파일과 연관될 수 있고, URL은 HTTP 웹 캐쉬 시스템에 제공될 수 있다. 그 다음으로, 수신기는 필요에 따라 UFSI-순서 HTTP 파일의 일부분들을 요청하기 위하여 HTTP 옥테트 범위 요청들을 이용할 수 있다. UFSI-순서 HTTP 파일의 하나의 장점은, 수신기가 소스 블록들의 각각으로부터 대략 동일한 수의 리페어 심볼들을 수신하는 경우, 원래의 파일 또는 파일 부분을 복원하기에 충분한 소스 심볼들을 얻기 위해 필요한 HTTP 옥테트 범위 요청들의 수는 최소일 수 있고, 예를 들어, 정확히 동일한 수의 리페어 심볼들이 각각의 소스 블록에 대해 수신될 경우에는, 하나의 HTTP 옥테트 범위 요청이 충분할 수도 있다는 것이다. 예를 들어, UFSI-순서 HTTP 파일의 최초 L*T*Z 연속적인 옥테트들에 대한 요청은 원래의 파일 또는 파일 부분의 Z 소스 블록들의 각각으로부터 크기 T의 최초 L 소스 심볼들을 수신하기에 충분하다. K-L 리페어 심볼들이 Z 소스 심볼들의 각각에 대한 하나 또는 그보다 많은 세션들에서 수신될 경우와, FEC 코드가 이상적인 복원 특성들을 가지는 경우에는, HTTP 옥테트 범위 요청으로부터 수신된 L*T*Z 옥테트들이 HTTP 파일을 FEC 디코딩하기에 충분하고, 여기서 이 예에서는, K가 Z 소스 블록들의 각각에서의 소스 심볼들의 수인 것으로 간주된다.

[0150] 위에서 설명된 HTTP 옥테트 범위 요청들을 지원하기 위한 또 다른 유리한 방법은 우선, 원래의 파일 또는 파일 부분에 대한 FEC OTI에 기반으로 하여, 파일 또는 파일 부분을 여기에서 "SS-순서 HTTP 파일"이라고 칭해지는 상이한 새로운 포맷으로 재조직화한다. 이 방법은 소스 블록당 다수의 서브-블록들이 있을 때에 유용한데, 이것은, 이 경우에 각각의 소스 심볼이 원래의 파일 또는 파일 부분의 연속적인 일부분이 아닐 수도 있고, 즉 소스 심볼의 서브-심볼들이 원래의 파일 순서화에서 소스 블록의 전반에 걸쳐 산재될 수 있기 때문이다. SS-순서 HTTP 파일에서의 데이터의 순서는 제 1 소스 블록의 모든 소스 심볼들, 그 다음으로 제 2 소스 블록의 모든 소스 심볼들, 그 다음으로 제 3 소스 블록의 모든 소스 심볼들, 등의 순서이고, 즉, SS-순서 HTTP 파일에서의 데이터의 순서는 소스 블록 내의 증가하는 연속적인 ESI 순서에 따라 순서화된 소스 심볼들로 조직화되고, FEC OTI에 의해 결정되는 바와 같이, 증가하는 연속적인 SBN 순서에서의 소스 블록들의 연접이다. URL은 SS-순서 HTTP 파일과 연관될 수 있고, URL은 HTTP 웹 캐쉬 시스템에 제공될 수 있다. 다음으로, 수신기는 필요에 따라 SS-순서 HTTP 파일의 일부분들을 요청하기 위하여 HTTP 옥테트 범위 요청들을 이용할 수 있다. SS-순서 HTTP 파일은, 수신기가 소스 블록들의 각각으로부터 상이한 수들의 리페어 심볼들을 수신할 경우에 특히 유리하다. 예를 들어, 각각 1,000 개의 소스 심볼들의 2 개의 소스 블록들이 있는 경우와, 수신기가 제 1 소스 블록에 대한 900 개의 리페어 심볼들과, 제 2 소스 블록에 대한 100 개의 리페어 심볼들을 하나 또는 그보다 많은 세션들로부터 수신할 경우, 수신기는 SS-순서 HTTP 파일의 최초 T*100 옥테트들에 대한 하나의 요청을 행할 수 있고 제 1 소스 블록에 대한 100 개의 소스 심볼들을 수신할 수 있고, 그리고 수신기는 SS-순서 HTTP 파일의 옥테트들 T*1,000 내지 T*1,900-1에 대한 또 다른 요청을 행할 수 있고 제 2 소스 블록에 대한 900 개의 소스 심볼들을 수신할 수 있고, 이 예에서 T는 두 소스 블록들에 대한 옥테트들로 된 심볼 크기이다.

[0151] 방금 설명된 방법들의 둘 모두의 조합이 또한 이용될 수도 있고, 즉, UFSI-순서 HTTP 파일 및 SS-순서 HTTP 파일의 둘 모두가 수신기들로의 상이한 URL들을 통해 이용가능하게 될 수도 있고, 그 다음으로, 수신기는 2 개의 상이하게 포맷팅된 HTTP 파일들에 대한 HTTP 옥테트 요청들의 조합을 이용할 수도 있다. 예를 들어, 각각 1,000 개의 소스 심볼들을 갖는 10 개의 소스 블록들이 있을 경우, 그리고, 하나 또는 그보다 많은 세션들에서, 800 개의 리페어 심볼들이 최초 8 개의 소스 블록들의 각각에 대해 수신되고, 820 개의 리페어 심볼들이 제 9 소스 블록에 대해 수신되고, 500 개의 리페어 심볼들이 제 10 소스 블록에 대해 수신될 경우에는, 수신기는 10 개의 소스 블록들의 각각에 대한 200 개의 소스 심볼들을 수신하기 위한 UFSI-순서 HTTP 파일에 대한 HTTP 옥테

트 범위 요청들과, 제 10 소스 블록에 대한 추가적인 300 개의 소스 심볼들을 수신하기 위한 SS-순서 HTTP 파일에 대한 추가적인 HTTP 옥테트 범위 요청을 행할 수도 있다. 이 예에서는, 수신기가 (이상적인 복원 특성들을 갖는 FEC 코드를 가정하여) 제 9 소스 블록에 대한 필요하지 않은 20 개의 소스 심볼들을 수신할 수도 있지만, 일부의 경우들에 있어서, 더 작은 수의 HTTP 옥테트 범위 요청들을 이용하여 일부 소스 블록들에 대한 최소의 필요한 옥테트들보다 더 많이 요청하는 것은, 훨씬 더 큰 수의 상이한 HTTP 옥테트 범위 요청들을 요구할 수도 있는, 각각의 소스 블록에 대한 정확하게 필요한 수의 소스 심볼들을 요청하는 것보다 더욱 효율적일 수도 있다.

[0152] 파일들을 복원하기 위하여 이 HTTP 바이트-범위 요청이 수신기들에 의해 언제 이용될 수 있는지를 시그널링하는 것이 유리할 수 있다. 이것은 파일 리페어를 위하여 다른 요청 메시지들을 현재 이용하고 있는 설치된 네트워크들로의 이 특징의 점진적인 도입을 가능하게 한다. 그 네트워크 동작들을 기존의 HTTP-기반 파일 리페어 서버들로 전이하는 것을 계획하고 있는 시스템 운영자는, 특징이 자신의 전체 네트워크에 걸쳐 또는 로밍 상대방 네트워크들에서 완전히 지원되기 전에 HTTP 바이트-범위 요청들을 행할 수 있는 수신기들을 설치하는 것을 시작할 수 있다. 그 다음으로, 설치된 HTTP-가능 수신기들의 수가 증가함에 따라, 운영자는 이 단말들로부터의 추정된 요청 부하를 정합하기 위하여 HTTP 서버들을 설치하기 시작할 수 있다. 시그널링은 수신기들이 그 네트워크에서 HTTP 바이트-범위 요청들을 이용할 수 있다는 것을 수신기들에 표시하기 위하여 특정한 네트워크에서 이용될 수 있다.

[0153] HTTP 바이트-범위 요청들의 네트워크 지원을 시그널링하기 위한 또 다른 접근법은 서버가 HTTP 바이트-범위 요청들만을 지원한다고 표시하는 리페어 서버 URI와 함께(예를 들어, 서버 URI를 "byteRangeServiceURI" 유형 객체로서 태그함) 디스크립터(descriptor)를 제공하는 것이다. 이 방법은 "serviceURI" 디스크립터에 의해 전형적으로 식별되는 심볼-기반 요청들을 받아들이기만 하는 리페어 서버들을 이미 설치한 네트워크들에서 유용하다. 새로운 "byteRangeServiceURI"의 정의는, 이 디스크립터를 인식하지 않는 레거시(legacy) 수신기들이 이 HTTP 서버들로부터의 심볼-기반 요청들을 실수로 행하는 것을 방지하면서, 새로운 수신기들이 이 서버들과의 바이트 범위 요청들을 이용하도록 한다.

[0154] HTTP 바이트-범위 요청의 네트워크 지원을 시그널링하기 위한 또 다른 실시예는, 서버가 HTTP 바이트-범위 요청들만을 지원하거나, 심볼-기반 요청들과 같이, HTTP-바이트 범위 요청들 및 다른 요청 포맷들을 지원하는지 여부를 표시하는 리페어 서버 URI와 함께 디스크립터를 제공하는 것이다. 또 다른 실시예에서, 디스크립터는 3 개의 가능성들 중의 하나를 표시하며: (1) 리페어 서버가 HTTP 바이트-범위 요청들만을 지원하는지 여부, (2) 리페어 서버가 또 다른 유형의 요청 포맷만을 (예를 들어, 심볼-기반) 지원하는지 여부, 또는 (3) 리페어 서버가 HTTP-바이트 범위 요청들 및 다른 요청 포맷들을 지원하는지 여부. 디스크립터는 요청 포맷들 중의 어느 것이 바람직한지 또는 수신기에 의해 지원될 경우에 이용되도록 요구되는지를 또한 표시할 수 있다. 서버들에 의해 지원되는 요청 포맷들의 이 디스크립터는 리페어 서버 URI들의 리스트와 함께 전송될 수 있다.

[0155] 또 다른 디스크립터(예를 들어, "useFileURI" 엘리먼트)는 수신기들에 또한 전송될 수 있고, 수신기들이 파일이 저장되어 있는 컨텐츠 서버, 예를 들어, 인터넷 상의 위치로부터 직접 파일의 일부분들 또는 파일의 전부를 검색하기 위하여 HTTP 바이트-범위 요청들을 이용할 수 있다는 것을 표시할 수 있다. 이것은, (이동 네트워크 운영자와 같은) 네트워크 운영자가 전용 파일 리페어 서버들을 설치해야 할 필요가 없고 그 대신에, 리페어 절차들에 대한 소스 데이터를 제공하기 위하여 파일들의 컨텐츠 서버들에 의존할 수 있는 아키텍처를 가능하게 한다. 파일들이 브로드캐스트 채널을 통해 다운로딩될 때, 파일들은 파일의 URI, 예를 들어, www.<newsprovidersite>.com/latest_news.3gp를 포함하는 파일 디스크립터 테이블과 함께 전송될 수도 있다. "useFileURI" 엘리먼트의 존재는, 단말들이 www.<newsprovidersite>.com/latest_news.3gp로부터 소스 데이터를 검색하기 위하여 HTTP 바이트-범위 요청들을 이용할 수 있다는 것을 단말들에 표시할 것이다. 따라서, 수신기를 www.<mobile-oper-cached-file-server-of-news>.com/newsfile_003.3gp로 지시하는 대신에, 수신기는 전형적으로, 기존의 HTTP 프로토콜들을 이용하여 컨텐츠를 액세스하는 브라우저들을 위하여 컨텐츠가 이미 이용가능한 사이트로 바로 갈 것이다. 파일 포맷, 예를 들어, 원래-순서 또는 UFSI-순서 또는 SS-순서 또는 확장된-원래-순서 HTTP 파일 포맷을 시그널링하는 또 다른 엘리먼트가 있을 수도 있다.

[0156] 전용 리페어 절차들에 관계없이, 컨텐츠 전달 시스템은 브로드캐스팅된 컨텐츠에 추가하여, 파일의 컨텐츠의 일부 또는 전부가 유니캐스트에 의해 이용가능하다는 것을 수신기에 시그널링하기 위한 메커니즘들을 포함할 수도 있다. 이 시그널링은 브로드캐스트의 일부로서 제공되는 데이터일 수도 있고, 데이터는 유니캐스트 위치를 가리킨다. 예를 들어, 브로드캐스트 데이터는 파일 이용가능성의 표시와 파일에 대한 URL을 포함할 수도 있다. 상이한 실시예들은 이 이용가능한 파일들을 상이하게 이용할 수도 있다. 예를 들어, 브로드캐스팅되는 URL에

의해 표시된 파일은 리페어 프로세싱을 위해 이용되는, 다운로딩의 속도를 높이기 위해 이용되는, 및/또는 더 신속한 초기 플레이아웃(playout)을 지원하기 위한 파일일 수 있다.

[0157] 일부 실시예들에서, 파일은 하나를 초과하는 네트워크 또는 물리적 위치에 존재한다. 그것을 위하여, 브로드캐스트 채널 상에서의 시그널링은 파일이 액세스가능한 복수의 위치들을 표시하는 URL들의 리스트를 포함할 수도 있다. 일부 실시예들에서는, 동일한 물리적 위치를 실제로 가리키고, 이에 따라 동일한 파일 또는 객체를 가리키는 다수의 URL들이 있다.

[0158] URL(들)에 의해 표시된 자원(들)은 예를 들어, 제한된 시간량 동안에만 이용가능하게 됨으로써 한정될 수 있다. (브로드캐스트 분배에서 식별자를 통해 특정된) 객체의 아마도 다수의 URL들로의 맵핑을 가능하게 하는 것이 유익할 수 있다. URL들의 각각에 대하여, 파일이 HTTP 프로토콜을 이용하여 하나의 위치에서 물리적으로 액세스가능하고 다른 프로토콜들을 이용하여 다른 위치들 또는 동일한 위치에서 이용가능하다는 표시와 같은 추가적인 정보가 제공될 수도 있다. 이러한 추가적인 정보의 다른 예들은 자원이 특정한 URL에서 액세스가능한 시간, 및/또는, 자원을 액세스하기 위하여 복수의 URL들 사이에서의 URL의 선택의 우선순위에 관한 규칙들, 예를 들어, 우선순위 리스트들, 또는 이용가능한 액세스 네트워크에 따른 우선순위 리스트들, 등을 포함한다.

[0159] 또 다른 디스크립터는, 리페어 데이터를 요청할 때에 수신기가 어느 서버들 또는 도메인들을 먼저 시도해야 하는지에 대한 우선순위화(prioritization)를 제공하기 위하여 서버 URI 리스트와 함께 또한 전송될 수 있다. 바람직한 실시예에서는, 서버를 선택할 시에 URI들이 수신기에 의해 우선순위화되어야 함을 표시하기 위하여, "priorityURI" 엘리먼트가 어떤 URI들과 연관될 수 있다. 이것은 우선순위화의 2 개의 레벨들을 가능하게 하고, 이것에 의해, 수신기는 "priorityURI"를 갖는 서버들을 먼저 시도하고, 우선순위화된 서버들이 반응이 없을 경우에 다른 서버들로 요청들을 전송하기만 한다. 또 다른 실시예에서는, 서버 URI들의 그룹들이 감소하는 우선순위의 순서로 열거된 그룹들을 갖는 "priorityGroups"로 구조화될 수 있다. 이것은 최고 우선순위 그룹부터 먼저 선택해야 한다는 것을 수신기에 표시한다. 이 서버들의 전부에 대한 요청들이 실패할 경우, 수신기는 서버들의 다음 그룹들, 등으로부터 서버를 선택하도록 진행하고, 이것에 의해, 우선순위화의 다수의 레벨들을 가능하게 한다.

[0160] 2 개의 일반적인 방법들이 지금부터 설명되고, 이 방법들을 통해, 디스크립터들 및 서버 URI들의 리스트가 수신기에 시그널링될 수 있다. 하나에서는, 시그널링이 모든 수신기들에 대해 브로드캐스트 채널을 통한 것이고(대역내), 다른 하나에서는, 시그널링이 각각의 수신기로의 유니캐스트 채널을 통한 것이다(대역외). 정보는 연관된 절차 설명(구성 파일), 사용자 서비스 설명("USD"), 미디어 프리젠테이션 설명("MPD"), 세션 설명 프로토콜("SDP"), 또는 파일 설명 테이블("FDT")과 같은 파일-리페어 특징에서 이용되는 다양한 현존하는 메시지 유형들에 추가될 수 있다.

[0161] 서버가 이러한 정보를 분배하기 위하여 FDT들 및 USD들을 시그널링하도록 구성되고, 수신기들이 그러한 FDT들 및 USD들을 판독 및 이해하도록 구성될 때, 이것은 다양한 유용한 정보를 전송하기 위한 설비를 제공한다.

[0162] 예를 들어, 정보는, 동일한 파일에 대한 다수의 대안적인 URL들을 포함할 수 있는, FLUTE의 파일 엘리먼트에서 "Alternative-Content-Location" 엘리먼트를 포함할 수도 있다. 이 엘리먼트는 (자원 이용가능성, 이용가능한 포맷들, 이용가능한 시간, 우선순위, 등의 표시자들과 같은) 다른 파라미터들을 또한 추가하도록 확장될 수도 있다.

[0163] 또 다른 예로서, "BaseURL" 엘리먼트는, FDT 내의 파일들의 각각이 BaseURL 엘리먼트들 중의 임의의 것에 대해 분해될 수 있음을 수신기에 표시하기 위하여 FDT 엘리먼트 내에 포함될 수 있다(그리고 이에 따라 파일 엘리먼트에서 필요하지 않다). 위와 같은 플래그들 및 타이밍이 마찬가지로 존재할 수도 있다. 이것은 다수의 설치 옵션들을 포괄할 수 있다. 변형예에서, BaseURL 엘리먼트는 FLUTE 대신에 USD 레벨 상에서 프리젠팅된다. 따라서, USD는 FLUTE와 호환성이 있는 동안에, MBMS에 대한 BaseURL 분해를 행하는 방법으로서 제공될 수 있다.

[0164] 또 다른 실시예에서는, "Alternate-Content-Location-1" 엘리먼트 및 "Alternate-Content-Location-2" 엘리먼트와 같은, 일부의 선택적인 엘리먼트들이 파일 전달 테이블(FDT)의 파일 엘리먼트로 도입될 수 있다. 이 엘리먼트들의 각각은 컨텐츠 서버 상에서 또는 공통의 전용 서버 상에서 파일의 URL을 제공하기 위해 이용될 수 있다. 하나의 실시예에서, 이 URL들은 이 엘리먼트들 중의 어느 하나가 존재할 때에 심볼-기반 파일 리페어 서버들로부터의 요청을 행하기 전에 단말에 의해 우선순위화된다. 2 개의 엘리먼트들이 존재할 때에는, "Alternate-Content-Location-1" 엘리먼트가 "Alternate-Content-Location-2" 엘리먼트 전에 선택되는 것을 요구하는 것이 또한 가능하다. URL들은 RFC 3986에서 설명된 바와 같이 절대적인 URL들 또는 상대적인 참조들일

수 있다.

[0165] 또 다른 실시예에서는, 알려져 있다면, 이용가능성 시간을 표시하기 위하여, 선택적인 "Availability-Time" 엘리먼트가 상기 "Alternate-Content-Location-x" 속성들의 각각과 연관된다.

[0166] 또 다른 실시예에서는, 선택적인 엘리먼트들 "Base-URL-1" 및 "Base-URL-2"가 파일 전달 테이블에서 이용될 수도 있다. 존재할 때, "Base-URL-1"은, FDT 파일 엘리먼트들의 임의의 "Alternate-Content-Location-1" 엘리먼트에 포함된 상대적인 참조를 그것에 대해 분해하기 위한 기본 URL을 제공한다. 존재할 때, "Base-URL-2"는, FDT 파일 엘리먼트들의 임의의 "Alternate-Content-Location-2" 엘리먼트에 포함된 상대적인 참조를 그것에 대해 분해하기 위한 기본 URL을 제공한다.

[0167] 다른 옵션들이 고려될 수도 있고, 상기 옵션들의 조합들이 마찬가지로 고려될 수도 있다. 예를 들어, 파일 포맷 유형을 이용가능한 파일 포맷에 따라 그룹화된 다른 URL들 내로 내장한다.

[0168] 운영자의 네트워크 내의 단말들이 처닝(churning)하고 더 많은 수신기들이 HTTP 바이트-범위 요청들을 할 수 있음에 따라, 운영자는 파일 리페어 용량을 더 많이 기존의 HTTP 서버들로 쉬프트시킬 수 있고 그 능력들을 이 시그널링 방법을 통해 표시할 수 있다.

[0169] 어떤 리페어 서버들이 HTTP 바이트-범위 요청들을 지원한다는 표시의 수신 시에, HTTP 바이트-범위 요청들을 지원하는 단말은 지원하는 서버들 중의 하나로부터 데이터를 요청하기 위하여 바이트-범위 메시지를 이용할 수도 있다.

[0170] 디스크립터가 이용하기 위한 요청 포맷에 대한 선호도 또는 요건을 표시하는 또 다른 실시예에서는, 수신기가 표시된 선호도들 또는 요건들을 따른다.

[0171] 또 다른 접근법은 어느 특정한 서버들이 이 능력을 가지는지를 식별하지 않으면서 HTTP-바이트 범위 요청들이 리페어 서버들에 의해 지원된다는 것을 수신기들에 시그널링하는 것이다. 시그널링은, 모든 리페어 서버들이 HTTP 바이트-범위 요청들만을 할 수 있거나, 모두가 HTTP 바이트-범위 요청들 및 다른 요청 메시지 포맷들, 예를 들어, 심볼-기반 요청들을 할 수 있다는 것을 표시할 것이다.

[0172] 또 다른 실시예에서, 신호는 위에서 열거된 것과 같은 3 개의 가능성들 중의 하나를 표시하며, 즉: (1) 모든 리페어 서버들이 HTTP 바이트-범위 요청들만을 지원하는지 여부, (2) 리페어 서버들이 또 다른 유형의 요청 포맷만을 (예를 들어, 심볼-기반) 지원하는지 여부, 또는 (3) 리페어 서버들이 HTTP-바이트 범위 요청들 및 다른 요청 포맷들을 지원하는지 여부. 이 간략화된 시그널링이 이용될 때, 운영자는 이것을 수신기들에 시그널링하기 전에 표시된 요청 포맷(들)을 지원하기 위하여 파일 리페어 서버들의 모두를 업그레이드할 수도 있다. 파일 포맷 유형은 예를 들어, 원래-순서, UFSI-순서, SS-순서, 확장된-원래-순서 또는 다른 유형들의 HTTP 파일 포맷들로 또한 시그널링될 수도 있다. 파일 포맷 유형이 명시적으로 시그널링되지 않을 경우에는, 디폴트(default) 파일 포맷 유형이 있을 수도 있고, 예를 들어, 파일 포맷 유형이 시그널링되지 않을 경우에는 원래-순서 HTTP 파일 포맷이 디폴트일 수도 있다.

[0173] 디스크립터 및 서버 URI 정보와 유사하게, 이 시그널링은 2 개의 일반적인 방법들을 이용하여 수신기에 또한 시그널링될 수 있다. 하나에서는, 시그널링이 모든 수신기들에 대해 브로드캐스트 채널을 통한 것이고(대역내), 다른 하나에서는, 시그널링이 각각의 수신기로의 유니캐스트 채널을 통한 것이다(대역외). 정보는 연관된 절차 설명(구성 파일), 사용자 서비스 설명("USD"), 미디어 프리젠테이션 설명("MPD"), 세션 설명 프로토콜("SDP"), 또는 파일 설명 테이블("FDT")과 같은 파일-리페어 특징에서 이용되는 다양한 현존하는 메시지 유형들에 추가될 수 있다.

[0174] 2 개의 일반적인 방법들이 지금부터 설명되고, 이 방법들을 통해, 디스크립터들 및 서버 URI들의 리스트가 수신기에 시그널링될 수 있다.

[0175] 당업자가 인식하는 바와 같이, 이 방법들에 대한 다수의 변형들이 있다. 예를 들어, 원래-순서 HTTP 파일, UFSI-순서 HTTP 파일 및 SS-순서 HTTP 파일이 모두 요청들에 대해 이용가능하게 될 수 있다. 또 다른 예로서, SS-순서 HTTP 파일은 각각의 소스 블록에 대한 하나의 이러한 HTTP 파일인 다수의 HTTP 파일들로서 분할되고 이용가능하게 될 수도 있다. 변형들의 다른 예로서, RTP/UDP에 기반한 방법들과 같이, HTTP에 기반한 것들 이외의 방법들 및 프로토콜들이 이용될 수도 있거나, UDP의 상부에 구축된 독점적인 프로토콜들이 이용될 수도 있다.

[0176] 하드웨어 시스템들 및 예들

[0177] 위에서 설명된 방법들 및 시스템들은 하드웨어, 소프트웨어, 및/또는 프로그램 코드 또는 다른 명령들을 포함하는 적절하게 조직된 컴퓨터-판독가능 매체들로 구현될 수 있다. 프로그램 코드는 적당한 하드웨어 플랫폼 상에서 실행되도록 하기 위하여, 비-일시적인 매체들 상에서 제공될 수도 있거나 다운로드 등과 같이 전송될 수도 있다. 상기 교시 내용들을 이용할 수도 있는 시스템들의 일부의 예들이 여기에서 제공된다.

[0178] 도 13은 여기에서 설명된 바와 같은 파일 전달 시스템의 일부로서 패킷들을 전송하기 위하여 이용될 수도 있는 바와 같은, 멀티-스테이지(multi-stage) 코딩을 이용하는 통신 시스템(1300)의 블록도이다. 물론, 다른 코드들 및/또는 하드웨어가 그 대신에 이용될 수도 있다.

[0179] 통신 시스템(1300)에서는, 입력 파일(1301) 또는 입력 스트림(1305)이 입력 심볼 발생기(1310)에 제공된다. 입력 심볼 발생기(1310)는 입력 파일 또는 스트림으로부터 하나 또는 그보다 많은 입력 심볼들(IS(0), IS(1), IS(2), ...)의 시퀀스를 발생하고, 각각의 입력 심볼은 값 및 위치(괄호로 묶인 정수로서 도 13에 나타냄)를 가진다. 위에서 설명된 바와 같이, 입력 심볼들에 대한 가능한 값들, 즉, 그 알파벳은 전형적으로 입력 파일의 M 비트들에 대한 각각의 입력 심볼 코드들이 되도록, 2^M 심볼들의 알파벳이다. M의 값은 통신 시스템(1300)의 이용에 의해 일반적으로 결정되지만, 범용 시스템은 M이 이용시마다 변동될 수 있도록 한 입력 심볼 발생기(1310)에 대한 심볼 크기 입력을 포함할 수도 있다. 입력 심볼 발생기(1310)의 출력은 인코더(1315)에 제공된다.

[0180] 정적 키이 발생기(1330)는 정적 키이들(S_0, S_1, ...)의 스트림을 생성한다. 발생된 정적 키이들의 수는 일반적으로 제한되고 인코더(1315)의 특정한 실시예에 의존한다. 정적 키이들의 발생은 추후에 더욱 상세하게 설명될 것이다. 동적 키이 발생기(1320)는 인코더(1315)에 의해 발생될 각각의 출력 심볼에 대한 동적 키이를 발생한다. 각각의 동적 키이는, 동일한 입력 파일에 대한 동적 키이들의 대부분이 고유하도록 발생된다. 난수 발생기(1335)는 발생기(1320) 및/또는 발생기(1330)에 입력을 제공할 수도 있다. 예를 들어, Luby I은 이용될 수 있는 키이 발생기들의 실시예들을 설명한다. 동적 키이 발생기(1320) 및 정적 키이 발생기(1330)의 출력은 인코더(1315)에 제공된다.

[0181] 동적 키이 발생기(1320)에 의해 제공된 각각의 키이 I로부터, 인코더(1315)는 입력 심볼 발생기에 의해 제공된 입력 심볼들로부터, 값 B(I)을 갖는 출력 심볼을 발생한다. 인코더(1315)의 동작은 아래에서 더욱 상세하게 설명될 것이다. 각각의 출력 심볼의 값은 그 키이, 입력 심볼들 중의 하나 또는 그보다 많은 일부의 함수, 및 아마도 입력 심볼들로부터 연산되었던 하나 또는 그보다 많은 중복 심볼들에 기반으로 하여 발생된다. 입력 심볼들과, 특정한 출력 심볼을 야기시키는 중복 심볼들의 집합은 출력 심볼의 "연관된 심볼들" 또는 단지 그 "연관들"이라고 여기에서 지칭된다. 함수("값 함수") 및 연관들의 선택은 아래에서 더욱 상세하게 설명되는 프로세스에 따라 행해진다. 전형적이지만, 항상 그렇지 않게도, M은 입력 심볼들 및 출력 심볼들에 대해 동일하고, 즉, 이들은 모두 동일한 비트들의 수에 대한 코드이다.

[0182] 일부 실시예들에서는, 연관들을 선택하기 위하여 입력 심볼들의 수 K가 인코더(1315)에 의해 이용된다. 입력이 스트리밍 파일인 경우와 같이, K가 미리 알려져 있지 않은 경우, K는 단지 추정치일 수 있다. 입력 심볼들과, 인코더(1315)에 의해 발생된 임의의 중간 심볼들에 대한 저장을 할당하기 위하여, 값 K는 인코더(1315)에 의해 또한 이용될 수도 있다.

[0183] 인코더(1315)는 출력 심볼들을 송신 모듈(1340)에 제공한다. 송신 모듈(1340)에는, 동적 키이 발생기(1320)로부터의 각각의 이러한 출력 심볼의 키이가 또한 제공된다. 송신 모듈(1340)은 출력 심볼들을 송신하고, 이용되는 키잉(keying) 방법에 따라, 송신 모듈(1340)은 송신된 출력 심볼들의 키이들에 대한 일부의 데이터를 채널(1345)을 통해 수신 모듈(1350)로 또한 송신할 수도 있다. 채널(1345)은 소거 채널인 것으로 가정되지만, 그것은 통신 시스템(1300)의 적당한 동작을 위한 요건이 아니다.

[0184] 송신 모듈(1340)이 출력 심볼들 및 그 키이들에 대한 임의의 필요한 데이터를 채널(1345)로 송신하도록 적응되고 수신 모듈(1350)이 채널(1345)로부터 심볼들 및 잠재적으로 그 키이들에 대한 일부의 데이터를 수신하도록 적응되어 있는 한, 모듈들(1340, 1345 및 1350)은 임의의 적당한 하드웨어 컴포넌트들, 소프트웨어 컴포넌트들, 물리적 매체들, 또는 그 임의의 조합일 수 있다. 연관들을 결정하기 위하여 이용될 경우, K의 값은 채널(1345)을 통해 전송될 수 있거나, 인코더(1315) 및 디코더(1355)의 일치에 의해 미리 설정될 수도 있다. 다른 엘리먼트들이 도 13에 도시되어 있다(그리고 여기의 다른 곳에서는, 모듈, 단계, 프로세스, 등으로 설명되어 있든지 간에, 하드웨어, 소프트웨어, 및/또는 전자적으로-판독가능한 매체들 상에 저장된 프로그램 코드를 이용하여 또

한 구현될 수 있음).

[0185] 위에서 설명된 바와 같이, 채널(1345)은 인터넷을 통한 경로, 또는 텔레비전 송신기로부터 텔레비전 수신자로의 브로드캐스트 링크 또는 하나의 지점으로부터 또 다른 지점으로의 전화 접속과 같은 실시간 채널일 수 있거나, 채널(1345)은 CD-ROM, 디스크 드라이브, 웹 사이트, 등과 같은 저장 채널일 수 있다. 채널(1345)은 심지어, 한 사람이 전화선을 통해 퍼스널 컴퓨터로부터 인터넷 서비스 제공자(ISP)로 입력 파일을 송신하고, 입력 파일이 웹 서버 상에 저장되고, 다음으로 인터넷을 통해 수신자에게 송신될 때에 형성되는 채널과 같이, 실시간 채널 및 저장 채널의 조합일 수도 있다.

[0186] 채널(1345)이 소거 채널인 것으로 가정되므로, 통신 시스템(1300)은 수신 모듈(1350)에서 나오는 출력 심볼들과, 송신 모듈(1340)로 들어가는 출력 심볼들과의 사이의 일대일 대응관계를 가정하지 않는다. 실제로, 채널(1345)이 패킷 네트워크를 포함할 경우, 통신 시스템(1300)은 심지어, 임의의 2 개 이상의 패킷들의 상대적인 순서가 채널(1345)을 통한 송신 시에 보존된다고 가정할 수 없을 수도 있다. 그러므로, 출력 심볼들의 키이는 위에서 설명된 키이 방식들 중의 하나 또는 그보다 많은 것을 이용하여 결정되고, 출력 심볼들이 수신 모듈(1350)에서 나오는 순서에 의해 반드시 결정되는 것은 아니다.

[0187] 수신 모듈(1350)은 출력 심볼들을 디코더(1355)에 제공하고, 수신 모듈(1350)이 이 출력 심볼들의 키이들에 대해 수신하는 임의의 데이터는 동적 키이 재발생기(1360)에 제공된다. 동적 키이 재발생기(1360)는 수신된 출력 심볼들에 대한 동적 키이들을 재발생하고 이 동적 키이들을 디코더(1355)에 제공한다. 정적 키이 발생기(1363)는 정적 키이들(S_0, S_1, ...)을 재발생하고 이들을 디코더(1355)에 제공한다. 정적 키이 발생기는 인코딩 및 디코딩 프로세스 둘 모두의 동안에 사용되는 난수 발생기(1335)에 대해 액세스한다. 이것은, 난수들이 이러한 디바이스 상에서 발생될 경우에 동일한 물리적 디바이스에 대한 액세스의 형태, 또는 동일한 거동을 달성하기 위하여 난수들의 발생을 위한 동일한 알고리즘에 대한 액세스의 형태일 수 있다. 디코더(1355)는 입력 심볼들(다시 IS(0), IS(1), IS(2), ...)을 복원하기 위하여, 대응하는 출력 심볼들과 함께, 동적 키이 재발생기(1360) 및 정적 키이 발생기(1363)에 의해 제공된 키이들을 이용한다. 디코더(1355)는 복원된 입력 심볼들을, 입력 파일(1301) 또는 입력 스트림(1305)의 복사본(1370)을 발생하는 입력 파일 리어셈블러(reassembler; 1365)에 제공한다.

[0188] 파일 전달은 다수의 수신기들 및/또는 다수의 전송기들로 행해질 수 있다. 이 개념들은 도 14 내지 도 15에 예시되어 있다.

[0189] 도 14는 하나의 수신기(2402)가 3 개의 채널들(2406)을 통해 3 개의 송신기들(2404; 개별적으로 "A", "B" 및 "C"로 나타냄)로부터 데이터를 수신하는 배치를 예시한다. 이 배치는 수신기에 의해 이용가능한 대역폭을 3 배로 하거나, 임의의 하나의 송신기로부터 전체 파일을 얻을 정도로 충분히 오래 이용가능하지 않은 송신기들을 처리하기 위해 이용될 수 있다. 표시된 바와 같이, 각각의 송신기(2404)는 값들의 스트림, S()를 전송한다. 각각의 S() 값은 출력 심볼 B(I) 및 키이 I를 나타내고, 그 이용은 위에서 설명되어 있다. 예를 들어, 값 S(A, n_A)는 송신기(2402(A))에서 발생된 출력 심볼들의 시퀀스에서 "n_A"번째 출력 심볼 및 "n_A"번째 키이이다. 하나의 송신기로부터의 키이들의 시퀀스는 다른 송신기들로부터의 키이들의 시퀀스와는 바람직하게는 별개이므로, 송신기들은 노력들을 중복하지 않는다. 이것은 시퀀스 S()가 송신기의 함수라는 사실에 의해 도 14에 예시되어 있다.

[0190] 송신기들(2402)은 노력들을 중복하지 않기 위하여 동기화되거나 조정될 필요가 없음을 주의하라. 실제로는, 조정 없이, 각각의 송신기가 그 시퀀스에서 상이한 위치에 있을 가능성이 있다(즉, $n_A \neq n_B \neq n_C$).

[0191] 도 15에는, 하나의 입력 파일(2502)의 복사본들이 복수의 송신기들(2504)(이들 중의 2 개가 도면에 도시되어 있음)에 제공된다. 송신기들(2504)은 입력 파일(2502)의 컨텐츠들로부터 발생된 출력 심볼들을 채널들(2506)을 통해 수신기들(2508)로 독립적으로 송신한다. 도시된 2 개 중의 각각의 송신기는, 수신기의 디코더가 전체 입력 파일을 복원할 수 있기 전에 (K + A)/2 출력 심볼들을 송신하기만 할 필요가 있을 수도 있다.

[0192] 2 개의 수신기들 및 2 개의 송신기들을 이용하면, 수신기 유닛(2510)에 의해 수신된 전체 정보량은 하나의 채널(2506)을 통해 이용가능한 정보의 4 배만큼 많을 수 있다. 정보량은, 예를 들어, 송신기들이 동일한 데이터를 두 수신기들에 브로드캐스팅할 경우에는 단일의 채널 정보의 4 배보다 적을 수도 있다. 그 경우, 수신기 유닛(2510)에서의 정보량은 적어도 2 배이고, 데이터가 임의의 채널에서 손실될 경우에는 종종 더 많다. 송신기들이 하나의 신호만을 브로드캐스팅 하지만, 수신기들이 상이한 시간들에서 계획하더라도, 각각의 송신기에 리스닝(listening)하는 하나를 초과하는 수신기를 가진다는 장점이 있다는 것에 주목해야 한다. 도 15에서는, 수신

기 유닛(2510)이 도 13에 도시된 수신 모듈(1350), 디코더(1355), 동적 키이 재발생기(1360) 및 입력 파일 리어셈블러(1365)의 기능들과 유사한 기능들을 수행한다.

[0193] 일부 실시예들에서, 입력 파일(2502)은 2 개의 인코더들을 갖는 하나의 연산 디바이스에서 인코딩되므로, 연산 디바이스는 하나의 송신기에 대한 하나의 출력과, 다른 송신기에 대한 또 다른 출력을 제공할 수 있다. 이 예들의 다른 변형들은 이 개시내용의 검토 시에 분명해야 한다.

[0194] 여기에서 설명된 코딩 장치 및 방법들은 다른 통신 상황들에서 또한 이용될 수도 있고, 인터넷과 같은 통신 네트워크들로 제한되지 않는다는 것을 이해해야 한다. 예를 들어, 컴팩트 디스크 기술은 스크래칭된(scratched) 디스크들의 문제를 처리하기 위하여 소거 및 에러-정정 코드들을 또한 이용하고, 정보를 거기에 저장함에 있어서 연쇄 반응 코드들의 이용으로부터 이익을 얻을 것이다. 또 다른 예로서, 위성 시스템들은 전력을 감소시킴으로써 더 많은 에러들을 목적의식적으로 허용하는, 송신을 위한 전력 요건들을 절충하기 위하여, 소거 코드들을 이용할 수도 있고, 연쇄 반응 코딩은 그 애플리케이션에서 유용할 것이다. 또한, 소거 코드들은 정보 저장의 신뢰성을 위하여 RAID(redundant arrays of independent disks; 독립 디스크들의 중복 어레이들) 시스템들을 개발하기 위해 이용되었다. 그러므로, 현재의 발명은 상기 예들과 같은 다른 애플리케이션들에서 유용한 것으로 판명될 수도 있고, 여기서 코드들은 잠재적으로는 손실 또는 오류 있는 데이터의 문제들을 처리하기 위하여 이용된다.

[0195] 일부 바람직한 실시예들에서, 위에서 설명된 통신 프로세스들을 수행하기 위한 명령들의 세트(또는 소프트웨어)가 아마도 손실 있는 통신 매체를 통해 통신하는 2 개 또는 그보다 많은 다목적 연산 머쉰(machine)들에 제공된다. 머쉰들의 수는 하나의 전송기 및 하나의 수신자로부터 전송 및/또는 수신하는 임의의 수의 머쉰들까지의 범위일 수도 있다. 머쉰들을 접속하는 통신 매체는 유선, 광, 무선, 등일 수도 있다. 위에서 설명된 통신 시스템들은 이 설명으로부터 분명해야 할 다수의 이용들을 가진다.

[0196] HTTP 스트리밍 서버를 이용하는 블록-요청 스트리밍 시스템은 위에서 설명된 바와 같이 파일들을 전달할 수도 있다. 아래에서는, 이러한 시스템의 일 예의 구현예가 설명된다. HTTP 스트리밍에서는, 소스들이 표준 웹 서버들 및 컨텐츠 전달 네트워크(CDN)들일 수도 있고, 표준 HTTP를 이용할 수도 있다.

[0197] 클라이언트 측 상에서는, HTTP를 이용하여, 클라이언트에 의해 심리스(seamless) 방식으로 함께 스플라이싱(splicing) 되어 있는 개별적인 세그먼트들에 대하여 요청들이 행해질 수도 있다. HTTP 스트리밍의 장점들은 표준화된 그리고 현존하는 기반구조의 이용을 포함한다.

[0198] 도 16은 파일들을 전달할 수도 있는 블록-요청 스트리밍 시스템의 간략화된 도면을 도시한다. 도 16에는, 블록 서빙 기반구조("BSI; block serving infrastructure")(1601)를 포함하는 블록-스트리밍 시스템(1600)이 예시되어 있고, 블록 서빙 기반구조(1601)는 궁극적으로, 컨텐츠(1602)를 수집하고, 그 컨텐츠를 준비하고, 그것을 수집 시스템(1603) 및 HTTP 스트리밍 서버(1604)의 둘 모두에 액세스가능한 컨텐츠 저장소(1610)에 저장함으로써 HTTP 스트리밍 서버(1604)에 의한 서비스를 위하여 그것을 패키징하기 위한 수집 시스템(1603)을 포함한다. 도시된 바와 같이, 시스템(1600)은 HTTP 캐쉬(1606)를 또한 포함할 수도 있다. 동작 시에, HTTP 스트리밍 클라이언트와 같은 클라이언트(1608)는 요청들(1612)을 HTTP 스트리밍 서버(1604)로 전송하고, HTTP 스트리밍 서버(1604) 또는 HTTP 캐쉬(1606)로부터 응답들(1614)을 수신한다. 각각의 경우에는, 도 16에 도시된 엘리먼트들이 적어도 부분적으로, 프로세서 상에서 또는 다른 전자기기들 상에서 실행되는 프로그램 코드를 그 내부에 포함하는 소프트웨어로 구현될 수도 있다.

[0199] 도 17에 예시된 바와 같이, 미디어 블록들은, 예를 들어, HTTP 서버, 컨텐츠 전달 네트워크 디바이스, HTTP 프록시, FTP 프록시 또는 서버, 또는 일부의 다른 미디어 서버 또는 시스템일 수 있는 블록 서빙 기반구조(1601(1)) 내에 저장될 수도 있다. 블록 서빙 기반구조(1601(1))는, 예를 들어, 인터넷과 같은 인터넷 프로토콜("IP") 네트워크일 수 있는 네트워크(1722)에 접속된다. 블록-요청 스트리밍 시스템 클라이언트는 6 개의 기능적인 컴포넌트들, 즉, 위에서 설명된 메타데이터가 제공되며 메타데이터에 의해 표시된 복수의 이용가능한 블록들 중으로부터 요청될 블록들 또는 부분적인 블록들을 선택하는 기능을 수행하는 블록 선택기(1723), 블록 선택기(1723)로부터 요청 명령들을 수신하며, 특정된 블록, 블록의 일부분들, 또는 다수의 블록들에 대한 요청을 네트워크(1722)를 통해 블록 서빙 기반구조(1601(1))로 전송하기 위해 그리고 궁극적으로 블록을 포함하는 데이터를 수신하기 위해 필요한 동작들을 수행하는 블록 요청기(1724) 뿐만 아니라, 블록 버퍼(1725), 버퍼 감시기(1726), 미디어 디코더(1727) 및 미디어 소비를 용이하게 하는 하나 또는 그보다 많은 미디어 트랜스듀서들(1728)을 갖는 것으로 도시되어 있다.

[0200] 블록 요청기(1724)에 의해 수신된 블록 데이터는 미디어 데이터를 저장하는 블록 버퍼(1725)에 일시적인 저장을 위해 전달된다. 대안적으로, 수신된 블록 데이터는 블록 버퍼(1725)에 바로 저장될 수 있다. 미디어 디코더(1727)에는 블록 버퍼(1725)에 의해 미디어 데이터가 제공되고, 미디어 디코더(1727)는 미디어를 사용자 또는 다른 소비를 위한 적당한 형태로 만드는 미디어 트랜스듀서들(1728)에 적당한 입력을 제공하기 위해 필요한 바와 같이 이 데이터에 대한 이러한 변환들을 수행한다. 미디어 트랜스듀서들의 예들은 이동 전화들, 컴퓨터 시스템들 또는 텔레비전들에서 발견되는 것들과 같은 시각적 디스플레이 디바이스들을 포함하고, 스피커들 또는 헤드폰들과 같은 오디오 렌더링(audio rendering) 디바이스들을 또한 포함할 수도 있다.

[0201] 버퍼 감시기(1726)는 블록 버퍼(1725)의 컨텐츠들에 관한 정보를 수신하고, 이 정보 및 아마도 다른 정보에 기반으로 하여, 여기에서 설명되는 바와 같이 요청에 대한 블록들의 선택을 결정하기 위해 이용되는 블록 선택기(1723)에 입력을 제공한다.

[0202] 블록-요청 스트리밍 시스템(BRSS; block-request streaming system)은 하나 또는 그보다 많은 컨텐츠 서버들(예를 들어, HTTP 서버들, FTP 서버들, 등)에 대한 요청들을 행하는 하나 또는 그보다 많은 클라이언트들을 포함한다. 수집 시스템은 하나 또는 그보다 많은 수집 프로세서들을 포함하고, 수집 프로세서는 컨텐츠를 (실시간이든 또는 아니든) 수신하고, BRSS에 의한 이용을 위해 컨텐츠를 프로세싱하고, 아마도 또한 수집 프로세서에 의해 발생된 메타데이터와 함께, 컨텐츠 서버들에 의해 액세스가능한 저장장치에 그것을 저장한다.

[0203] BRSS는 컨텐츠 서버들과 조정하는 컨텐츠 캐쉬들을 또한 포함할 수도 있다. 컨텐츠 서버들 및 컨텐츠 캐쉬들은 URL을 포함하는 HTTP 요청들의 형태로 파일들 또는 세그먼트들에 대한 요청들을 수신하는 기존의 HTTP 서버들 및 HTTP 캐쉬들일 수도 있고, URL에 의해 표시된 파일 또는 세그먼트의 전부보다 적게 요청하기 위하여 옥테트 범위를 또한 포함할 수도 있다. 클라이언트들은, HTTP 서버들의 요청들을 행하고 그러한 요청들에 대한 응답들을 처리하는 기존의 HTTP 클라이언트를 포함할 수도 있고, 여기서 HTTP 클라이언트는, 요청들을 형식화(formulate)하고, 이들을 HTTP 클라이언트에 전달하고, HTTP 클라이언트로부터 응답들을 얻고, 클라이언트 디바이스에 의한 플레이아웃을 위하여 응답들을 프리젠테이션 플레이어(presentation player)에 제공하기 위하여 응답들을 프로세싱(또는 저장, 변환, 등)하는 새로운 클라이언트 시스템에 의해 구동된다. 전형적으로, (필요성들이 사용자 입력, 사용자 입력에서의 변경들, 등에 의존할 수도 있으므로) 클라이언트 시스템은 어떤 미디어가 필요하게 될 것인지를 미리 알지 못하므로, 미디어가 수신되자마자, 또는 그 바로 후에 "소비"된다는 점에서 "스트리밍" 시스템이라고 말해진다. 그 결과, 응답 지연들 및 대역폭 제약들은, 스트림이 사용자가 프리젠테이션을 소비하고 있는 곳까지 따라잡을 때에 프리젠테이션에서 일시정지(pause)를 야기시키는 것과 같이, 프리젠테이션에서 지연들을 야기시킬 수 있다.

[0204] 양호한 품질인 것으로 인지되는 프리젠테이션을 제공하기 위하여, 다수의 세부사항들이 클라이언트 엔드, 또는 수집 엔드의 어느 하나에서, 또는 둘 모두에서, BRSS에서 구현될 수 있다. 일부의 경우들에 있어서, 구현되는 세부사항들은 네트워크에서 클라이언트-서버 인터페이스를 고려하여 그리고 이를 처리하기 위하여 행해진다. 일부 실시예들에서는, 클라이언트 시스템 및 수집 시스템의 둘 모두가 개량을 알고 있는 반면, 다른 실시예들에서는, 한 측만 개량을 알고 있다. 이러한 경우들에 있어서, 한 측이 그것을 알지 못하더라도 전체 시스템이 개량으로부터 이익을 얻는 한편, 다른 것들에서는, 양 측들이 그것을 알 경우에 이익이 누적되기만 하지만, 한 측이 알지 못할 때에는, 그것은 실패 없이 여전히 동작한다.

[0205] 도 18에 예시된 바와 같이, 수집 시스템은 다양한 실시예들에 따라 하드웨어 및 소프트웨어 컴포넌트들의 조합으로서 구현될 수도 있다. 수집 시스템은 시스템으로 하여금 여기에서 논의된 방법론들 중의 임의의 하나 또는 그보다 많은 것을 수행하도록 실행될 수 있는 명령들의 세트를 포함할 수도 있다. 시스템은 특정한 머쉰으로서 컴퓨터의 형태로 실현될 수도 있다. 시스템은 서버 컴퓨터, 퍼스널 컴퓨터(PC), 또는 그 시스템에 의해 취해질 작업들을 특정하는 명령들의 세트(순차적이거나 또는 이와 다름)를 실행할 수 있는 임의의 시스템일 수도 있다. 또한, 단일의 시스템만이 예시되어 있지만, 용어 "시스템"은 여기에서 논의된 방법론들 중의 임의의 하나 또는 그보다 많은 것을 수행하기 위한 명령들의 세트(또는 다수의 세트들)를 개별적으로 또는 공동으로 실행하는 시스템들의 임의의 집합을 포함하는 것으로 또한 받아들여져야 할 것이다.

[0206] 수집 시스템은 수집 프로세서(1802)(예를 들어, 중앙 프로세싱 유닛(CPU)), 실행 동안에 프로그램 코드를 저장할 수도 있는 메모리(1804), 및 디스크 저장장치(1806)를 포함할 수도 있고, 이들 모두는 버스(1800)를 통해 서로 통신한다. 시스템은 비디오 디스플레이 유닛(1808)(예를 들어, 액정 디스플레이(LCD) 또는 음극선관(CRT))을 더 포함할 수도 있다. 시스템은 영숫자 입력 디바이스(1810)(예를 들어, 키보드)와, 컨텐츠 소스(content source)를 수신하고 컨텐츠 저장소(content store)를 전달하기 위한 네트워크 인터페이스 디바이스(1812)를 또

한 포함할 수도 있다.

[0207] 디스크 저장 유닛(1806)은 여기에서 설명된 방법론들 또는 기능들 중의 임의의 하나 또는 그보다 많은 것을 구체화하는 명령들의 하나 또는 그보다 많은 세트들(예를 들어, 소프트웨어)이 그 상에 저장될 수도 있는 머쉰-판독가능 매체를 포함할 수도 있다. 명령들은 시스템에 의한 그 실행 동안에 메모리(1804) 내에 및/또는 수집 프로세서(1802) 내에 완전히 또는 적어도 부분적으로 또한 상주할 수도 있고, 메모리(1804) 및 수집 프로세서(1802)는 머쉰-판독가능 매체들을 또한 구성한다.

[0208] 도 19에 예시된 바와 같이, 클라이언트 시스템은 다양한 실시예들에 따라 하드웨어 및 소프트웨어 컴포넌트들의 조합으로서 구현될 수도 있다. 클라이언트 시스템은 시스템으로 하여금 여기에서 논의된 방법론들 중의 임의의 하나 또는 그보다 많은 것을 수행하게 하도록 실행될 수 있는 명령들의 세트를 포함할 수도 있다. 시스템은 특정한 머쉰으로서 컴퓨터의 형태로 실현될 수도 있다. 시스템은 서버 컴퓨터, 퍼스널 컴퓨터(PC), 또는 그 시스템에 의해 취해질 작업들을 특정하는 명령들의 세트(순차적이거나 또는 이와 다름)를 실행할 수 있는 임의의 시스템일 수도 있다. 또한, 단지 단일의 시스템이 예시되어 있지만, 용어 "시스템"은 여기에서 논의된 방법론들 중의 임의의 하나 또는 그보다 많은 것을 수행하기 위한 명령들의 세트(또는 다수의 세트들)를 개별적으로 또는 공동으로 실행하는 시스템들의 임의의 집합을 포함하는 것으로 또한 받아들여져야 할 것이다.

[0209] 클라이언트 시스템은 클라이언트 프로세서(1902)(예를 들어, 중앙 프로세싱 유닛(CPU)), 실행 동안에 프로그램 코드를 저장할 수도 있는 메모리(1904), 및 디스크 저장장치(1906)를 포함할 수도 있고, 이들 모두는 버스(1900)를 통해 서로 통신한다. 시스템은 비디오 디스플레이 유닛(1908)(예를 들어, 액정 디스플레이(LCD) 또는 음극선관(CRT))을 더 포함할 수도 있다. 시스템은 영숫자 입력 디바이스(1910)(예를 들어, 키보드)와, 요청들을 전송하고 응답들을 수신하기 위한 네트워크 인터페이스 디바이스(1912)를 또한 포함할 수도 있다.

[0210] 디스크 저장 유닛(1906)은 여기에서 설명된 방법론들 또는 기능들 중의 임의의 하나 또는 그보다 많은 것을 구체화하는 명령들의 하나 또는 그보다 많은 세트들(예를 들어, 소프트웨어)이 그 상에 저장될 수도 있는 머쉰-판독가능 매체를 포함할 수도 있다. 명령들은 시스템에 의한 그 실행 동안에 메모리(1904) 내에 및/또는 클라이언트 프로세서(1902) 내에 완전히 또는 적어도 부분적으로 또한 상주할 수도 있고, 메모리(1904) 및 클라이언트 프로세서(1902)는 머쉰-판독가능 매체들을 또한 구성한다.

[0211] 특정한 실시예의 예

[0212] [RaptorQ-RFC-6330]에 특정된 RaptorQ FEC 방식을 이용한 유니버셜 객체 전달을 위한 완전히-특정된 FEC 방식의 특정 실시예가 이 섹션에서 설명된다. 간략화된 그리고 개량된 객체 전달 능력들을 제공하기 위하여, UOSI FEC 페이로드 ID가 [RaptorQ-RFC-6330]에서의 RaptorQ FEC 방식(여기에서는 "UOD-RaptorQ FEC 방식"이라고 지칭됨)과 함께 이용될 수 있다. 특히, 기본 객체 전달에 대한 더욱 유연하고 더 간단한 지원이 제공되고, 비균등 에러 보호(UEP) 객체 전달에 대하여, 번들제공된 객체 전달에 대하여, 그리고 UEP 및 번들제공된 객체 전달의 조합들에 대하여 지원이 또한 제공된다. 통신 디바이스들 사이에서 "UOD-RaptorQ FEC 방식"을 구현하기 위하여 적당한 하드웨어 및/또는 소프트웨어가 이용될 수 있다는 것을 이해해야 한다.

[0213] FEC 페이로드 ID 포맷은 도 2에 예시된 바와 같이 4-옥테트 필드이고, 여기서 이 특정한 구현예에서는, 유니버셜 파일 심볼 식별자(UFSI)의 32-비트 무부호 정수가 유니버셜 객체 심볼 식별자(UOSI)의 또한 32-비트인 무부호 정수에 의해 일반화된다. UOSI는 FEC OTI와 함께, 그 페이로드 ID를 운반하는 패킷 내에 포함된 인코딩 심볼들을 식별하기 위해 이용되는 비-네거티브(non-nagative) 정수이다.

[0214] 단일의 객체, 또는 다수의 객체들, 또는 상이한 우선순위들을 갖는 부분들로 구획된 단일의 객체, 또는 이들의 임의의 조합의 전달을 위하여, FEC OTI는 아래에서 설명되는 바와 같다. 전달되는 각각의 객체에 대하여, FEC OTI는 RaptorQ FEC 방식에 의해 특정된 것과 동일할 수 있다는 것에 주목해야 한다. 전달은 일부의 포지티브 정수 d에 대하여 d 객체들을 포함할 수도 있다. 각각의 객체는 동일한 파일, 또는 상이한 파일들, 또는 그 조합들의 상이한 부분들을 포함할 수도 있다. 객체 i의 크기 F_i와, 객체 i에 대해 이용될 인코딩 심볼의 크기 T_i와의 사이의 관계는 송신에서 객체 i의 우선순위를 결정하기 위하여 이용될 수 있다.

[0215] 도 20은 공통 FEC OTI 필드의 예를 예시한다. 거기에서 이용되는 바와 같이, 전달되는 객체들의 수 d에 대하여 8-비트 무부호 정수가 있고, 도시된 예는 d = 2에 대한 것이다. 디폴트 값은 d = 1일 수도 있다. 포함하는 다른 서브필드들은 d 객체들의 각각에 대하여(즉, i = 1, ..., d에 대하여), 심볼 크기 T_i에 대한 16-비트 무부호 정수, 옥테트들의 단위들로 객체 i에 대한 심볼의 크기를 표시하는 2^{16}보다 작은 포지티브 정수, 최대 2^{40}인 비-

네거티브 정수이며 옥테트들의 단위들로 객체 i의 전송 길이를 표시하는 객체 i의 전송 길이 F_i에 대한 40-비트 무부호 정수를 포함하는 객체 i에 특정한 공통 FEC OTI 엘리먼트들이다. 적당한 패딩이 제공된다.

[0216] 공통 FEC OTI 필드와 대조적으로, 도 21에 도시된 바와 같이, 방식-특정 FEC OTI 엘리먼트가 있을 수도 있다. 그 예에서 도시된 바와 같이, 방식-특정 FEC OTI 엘리먼트는 심볼 정렬 파라미터(Al)(8 비트들, 무부호 정수), 객체 i에 대한 소스 블록들의 수 Z_i를 포함하는 각각의 객체에 대한 방식-특정 FEC OTI 엘리먼트들(12 비트들, 무부호 정수), 및 객체 i에 대한 서브-블록들의 수 N_i을 포함한다. 인코딩된 방식-특정 객체 송신 정보는 (1+3*d)-옥테트 필드이다. 도 21의 예는 d = 2에 대한 것이다. 다음으로, 인코딩된 FEC OTI는 인코딩된 공통 FEC OTI 및 인코딩된 방식-특정 FEC OTI 엘리먼트들의 연접을 포함하는 (2+10*d)-옥테트 필드일 수 있다.

[0217] 인코딩된 FEC OTI를 이용한 컨텐츠 전달은, UOD-RaptorQ FEC 방식을 이용하는 시스템들의 디바이스들, 컴퓨터들, 프로그램들과, 객체 전달을 위하여 UOD-RaptorQ FEC 방식을 이용하는 컨텐츠 전달 프로토콜("CDP")과의 사이의 정보 교환을 포함할 수 있다. CDP는 d, Al, 그리고 각각의 객체에 대하여, F_i, T_i(Al의 배수), Z_i 및 N_i를 인코더 디바이스 및 디코더 디바이스에 제공해야 한다. 인코더는 i에 의해 인덱스된 각각의 객체가 제공된다. UOD-RaptorQ 인코더 방식을 이용하는 인코더 디바이스는 전송될 각각의 객체에 대하여 CDP를, d 객체(들)에 대하여 패킷의 UOSI 및 인코딩 심볼(들)에 제공할 것이다.

[0218] 파라미터 선택의 예들

[0219] 하나의 예에서는, [RaptorQ-RFC-6330]의 섹션 4.3에서 설명된 일 예의 파라미터 유도가 이용될 수도 있고, d 객체들의 각각에 독립적으로 적용될 수도 있다. 일 예에서, Al = 4 옥테트들, SS = 8(각각의 서브-심볼이 적어도 SS*Al = 32 옥테트들일 것임을 암시함), 및 객체 i에 대한 T_i는 바람직하게는 적어도 SS*Al 옥테트들이고, T_i는 Al의 배수인 한편, 각각의 인코딩 패킷의 페이로드 크기는 바람직하게는 크기가 적어도 T이며, 여기서 T는 T_i의 i = 1, ..., d에 걸친 합이다.

[0220] 소스 블록 구성을 위하여, [RaptorQ-RFC-6330]의 섹션 4.4.1에서의 절차들이 d 객체들의 각각에 독립적으로 적용될 수도 있다.

[0221] 인코딩 패킷 구성을 위하여, 각각의 인코딩 패킷은 d 객체(들)에 대한 UOSI 및 인코딩 심볼(들)을 포함할 것이다. [RaptorQ-RFC-6330]의 RaptorQ FEC 방식에 의해 이용되는 FEC 페이로드 ID의 (SBN, ESI) 포맷과, UOD-RaptorQ FEC 방식에 의해 이용되는 UOSI 포맷과의 사이의 호환성을 위하여, 특정 포맷이 있을 수도 있다. 예를 들어, 각각의 객체에 대하여, UOSI 값 C로부터 객체 i에 대한 대응하는 (SBN, ESI) 값들(A, B)로의 맵핑은 $B = \text{floor}(C/Z_i)$ 및 $A = C - B*Z_i$인 경우에 있을 수도 있다. 유사하게, 각각의 객체에 대하여, 객체 i에 대한 (SBN, ESI) 값들(A, B)로부터 대응하는 UOSI 값 C로의 맵핑은 $C = A + B*Z_i$일 것이다.

[0222] 각각의 객체 i = 1, ..., d에 대하여, 0으로부터 KT_i-1로의 UOSI 값들은 소스 블록 인터리빙된 순서로 객체 i의 소스 심볼들을 식별하고, $KT_i = \text{ceil}(F_i/T_i)$이다. KT_i로부터 계속 진행하는 UOSI 값들은 RaptorQ 인코더를 이용하여 객체 i의 소스 심볼들로부터 발생된 리페어 심볼들을 식별한다.

[0223] 인코딩 패킷들은 소스 심볼들, 리페어 심볼들, 또는 소스 및 리페어 심볼들의 조합들을 포함할 수도 있다. 패킷은 객체 i의 동일한 소스 블록으로부터의 임의의 수의 심볼들을 포함할 수도 있다. 객체 i에 대한 소스 패킷에서의 최종 소스 심볼이 FEC 인코딩의 목적들을 위하여 추가된 패딩 옥테트들을 포함하는 경우, 이 옥테트들은 패킷에 포함되어야 하므로, 전체 심볼들만이 각각의 패킷에 포함된다.

[0224] 각각의 인코딩 패킷에서 운반되는 유니버셜 객체 심볼 식별자 C는 그 패킷에서 운반되는 각각의 객체에 대한 최초 인코딩 심볼의 UOSI이다. 각각의 객체에 대한 패킷에서의 추후의 인코딩 심볼들은 순차적인 순서로 C+1로부터 C+G-1까지 번호 부여된 UOSI들을 가지며, 여기서 G는 패킷에서의 각각의 객체에 대한 인코딩 심볼들의 수이다. 다른 실시예들에서는, 각각의 객체 i에 대한 FEC OTI 정보의 일부로서 특정된 기본 UOSI U_i가 있을 수도 있고, 이 경우에 객체 i에 대한 패킷에서의 최초 심볼에 대한 UOSI는 $C+U_i$이다.

[0225] 바람직한 구현예에서는, 각각의 인코딩 패킷에서의 d 객체들의 각각에 대하여 하나의 인코딩된 심볼이 있다. 바람직한 구현예에서는, 인코딩 패킷들이 다음의 절차에 따라 발생 및 전송된다. 각각의 UOSI 값 C = 0, 1, 2, 3, ...에 대하여, 인코더는 다음과 같이 인코딩 패킷을 발생 및 전송하고, 인코딩 패킷의 FEC 페이로드 ID의 값

은 UOSI 값 C로 설정되고, i = 1, ..., d에서의 각각의 객체 i에 대하여, 인코더는 UOSI 값 C에 대응하는 (SBN, ESI) 값들(A_i, B_i)을 결정하고, RaptorQ FEC 방식 [RaptorQ-RFC-6330]의 절차들에 따라 객체 i로부터의 (SBN, ESI) 값들(A_i, B_i)에 대응하는 크기 T_i의 인코딩 심볼 E_i를 발생하고, 인코딩 심볼 E_i를 인코딩 패킷의 페이로드에 추가하고, 인코딩 패킷을 전송한다. 수신기가 인코딩 패킷들의 총 수를 아는 것이 필요하지 않다는 것에 주목해야 한다.

[0226] 유니캐스트 소스 서버들 및 브로드캐스트 리페어 서버들

[0227] 이 섹션에서 설명되는 실시예들의 대부분에서는, 리페어 심볼들이 브로드캐스트 방식으로 전송되고 소스 심볼들은 소스 심볼들의 일부 세트들에 대한 UE 요청들에 응답하여 유니캐스트 방식으로 전송된다. 적어도 일부 소스 심볼들이 브로드캐스트 방식으로 전송되는, 일부가 이 섹션에서 설명되고 일부가 여기의 다른 곳에서 설명된 다른 실시예들이 있고, 적어도 일부의 리페어 심볼들이 UE 요청들에 응답하여 유니캐스트 방식으로 전송되는 실시예들이 또한 있다.

[0228] 일 예의 브로드캐스트 파일 전달 시스템은 eMBMS 파일 전달 시스템이다. 일 예의 유니캐스트 파일 전달 시스템은 HTTP 바이트-범위 요청 가능 시스템이다. 그러나, 전체 파일들만이 요청 및 제공될 수 있는 유니캐스트 시스템은 아마도 이용될 수 있지만, 많은 장점들이 손실될 것이다. 여기의 더 이전 섹션들은 일부의 예시적인 시스템들을 도시한다.

[0229] 이 섹션의 예들에서는, 리페어 데이터가 브로드캐스트 세션에서 전송되고, 소스 데이터만이 유니캐스트 리페어 세션에서 전송된다. 유니캐스트 리페어 요청들은 원래의 파일의 부분들에 대한 표준 HTTP 1.1 바이트 범위 요청들이다. 브로드캐스트 세션만이 리페어 심볼들을 가졌으므로, 2 개의 세션들 사이에는 중첩 데이터가 없을 것이다. 일부 구현예들에서는, 더 이전에 설명된 시스템 및 이 섹션의 시스템의 둘 모두가 공존 및/또는 조합될 수 있다는 것에 주목해야 한다.

[0230] 위에서 설명된 바와 같이, UE들(예를 들어, 이동 사용자 디바이스와 같은 최종-사용자 디바이스들)은 UE가 어떤 소스 심볼들을 누락하는지를 결정하고, 누락된 소스 심볼들의 소스 블록 번호, 인코딩 심볼 id로부터 표준 HTTP 1.1 바이트 범위 요청들로 변환하도록 구성될 수 있고, 그러한 요청들을 행할 수 있다.

[0231] 브로드캐스트 세션에서 리페어 심볼들을 전송함으로써, 유니캐스트 리페어 서버는 파일의 원래의 복사본만을 필요로 하고, 요청들은 개별적인 심볼들에 대한 요청들 또는 바이트들의 다수의 작은 범위들에 대한 요청들보다는, 소스 심볼들의 연속적인 범위에 대한 또는 연속적인 바이트들의 연속적인 큰 범위에 대한 간단한 요청들일 것이다. UE는 얼마나 많이 수신되는지와 얼마나 많이 필요한지를 간단히 카운트하는 것에 기반으로 하여, 또는 예를 들어, 소스 블록의 디코딩을 허용할 소스 심볼들의 프리픽스 번호, 또는 소스 서브-블록의 디코딩을 허용할 소스 서브-심볼들의 프리픽스 번호, 또는 서브-블록 또는 소스 블록의 디코딩을 허용할 프리픽스 바이트 범위 크기를 결정하기 위한 디코딩 스케줄을 실행하는 것에 기반으로 하여, 얼마나 많은 심볼들을 요청할 것인지를 결정할 수도 있다. 이 상황 및 여기의 다른 상황들에서는, "프리픽스"는 "서브-블록에 대응하는 파일의 일부분의 프리픽스" 또는 "소스 블록에 대응하는 파일의 일부분의 프리픽스"를 나타내도록 의도된 것이고, 즉, "프리픽스"는 전체 파일의 프리픽스가 아니라, 그 대신에 파일의 관련된 일부분의 프리픽스를 지칭할 수도 있고, 여기서 관련된 부분은 "프리픽스"가 이용되는 상황으로부터 추론될 수 있다.

[0232] 서브-블록킹이 이용되지 않을 때, 소스 블록의 연속적인 소스 심볼들에 대한 요청은 바이트들의 연속적인 시퀀스에 대한 요청으로 전환될 수 있다. 서브-블록킹이 이용될 때, 서브-블록의 소스 서브-심볼들의 연속적인 번호에 대한 요청이 파일의 바이트들의 연속적인 시퀀스에 대한 요청으로 전환될 수 있다.

[0233] 여기에서 설명된 바와 같이, UE는 서브-블록의 소스 서브-심볼들의 연속적인 번호에 대한 요청으로부터 파일에서의 바이트 범위 요청을 유도할 수 있다.

[0234] 일부 구현예들에서는, 상이한 브로드캐스트 서버들이 아마도 상이한 지형들에 기반으로 하여 리페어 심볼들의 상이한 세트들을 가진다. 이것은 동일한 디바이스에 대한 중복 수신이 없다는 것을 보장하기 위하여, 브로드캐스트 송신 동안에 하나의 지형으로부터 또 다른 지형으로 이동하는 UE들에 유익하다.

[0235] UE들이 브로드캐스트 세션에서 어느 리페어 서브-심볼들(서브-블록들이 이용되지 않을 경우에는 심볼들; 이 삽입어구는 아래에서 마찬가지로 적용됨)을 수신하는지와, 그리고 UE들이 브로드캐스트 세션에서 얼마나 많은 리페어 심볼들을 수신하였는지에 크게 관계없이, UE들은 HTTP 요청들에서 서브-블록(또는 서브-블록들이 이용되지 않을 경우에는 블록)의 바이트 범위 프리픽스를 요청한다. IETF RFC 5510에서 특정된 리드-솔로몬(Reed-

Solomon) 코드들과 같은 일부 코드들에 대하여, 요청된 바이트 범위 프리픽스는 브로캐스트 세션에서 어느 리페어 심볼들이 수신되는지에 완전히 관계없을 수 있다. IETF RFC 6330에서 특정된 RaptorQ 코드들과 같은 다른 코드들에 대해서는, 바이트 범위 프리픽스의 크기는 브로드캐스트 세션에서 어느 리페어 심볼들이 수신되는지에 약하게 의존할 수도 있다. 예를 들어, 소스 서브-블록에서 K 소스 서브-심볼들이 있고, 그 서브-블록에 대해 K-L 리페어 서브-심볼들이 수신되는 경우, 최초 L 소스 서브-심볼들에 대응하는 바이트 범위 프리픽스를 요청하는 것은 적어도 99% 확률로 디코딩을 허용할 것이고, L+1 서브-심볼들을 요청하는 것은 적어도 99.99% 확률로 디코딩을 허용할 것이고, L+2 서브-심볼들을 요청하는 것은 적어도 99.9999% 확률로 디코딩을 허용할 것이다. 디코딩이 가능한지의 여부는 디코딩을 위해 이용되는 서브-심볼들의 패턴에 의존하고, 수신 디바이스는 요청을 행하기 전에, 수신된 서브-심볼들의 특정한 시퀀스가 디코딩을 허용할 것인지 여부를 결정하고, 이에 따라, 디코딩을 보장하기 위하여 어느 요청들을 행할 것인지를 결정한다. 따라서, 리페어를 위해 요청된 프리픽스의 크기는 브로드캐스트 세션에서 수신된 서브-심볼들의 패턴에 약하게 의존할 수 있다.

[0236] 그러나, 간단함을 위하여, 수신 디바이스는 L+2 서브-심볼들에 대한 요청을 간단히 행할 수도 있고, 이 경우에 요청 크기는, 잠재적으로 2 개의 중복 서브-심볼들, 즉, 브로드캐스트 세션으로부터의 K-L 서브-심볼들과, K 소스 서브-심볼들을 복원하기 위해 수신된 총 K+2 서브-심볼들에 대한 리페어 요청으로부터의 L+2를 수신하는 오버헤드를 희생하면서, 그리고 디코딩에 실패할 매우 작은 확률을 희생하면서, 어느 서브-심볼들이 브로드캐스트 세션에서 수신되는지에 관계없다. 어느 데이터가 수신되는지에 관계없이 그리고 얼마나 많은 데이터가 브로드캐스트 세션에서 수신되는지에 따라서만 요청의 크기 및 패턴을 행하기 위한 일부의 이유들은 다음을 포함한다:

[0237] (a) UE 요청들은 간단할 수 있고, 소스 서브-블록(블록)의 시작으로부터 최대로 소스 서브-블록(블록)당 하나의 바이트 범위 요청을 항상 요청할 수 있다(일부의 경우들에 있어서, 충분한 리페어 서브-심볼들(심볼들)이 브로드캐스트 세션에서 수신되는 경우, 추가적인 HTTP 요청들이 필요하지 않음).

[0238] (b) UE는 리페어 서버로부터 HTTP를 통해 다운로딩하고 있는 동안에 서브-블록(블록)의 컨텐츠의 플레이아웃을 시작할 수 있고, 그 다음으로 일단 충분히 다운로딩 하였으면, 서브-블록의 나머지 일부분을 복원하기 위하여 브로드캐스트를 통해 이미 수신된 리페어 서브-심볼들을 이용할 수 있다(나머지 서브-블록(블록)의 크기는 본질적으로 브로드캐스트 세션에서 수신된 서브-블록(블록)에 대한 리페어 서브-심볼들(심볼들)의 크기임).

[0239] 수신기가 브로드캐스트를 통해 리페어 데이터를 동시에 수신하는 한편, 컨텐츠를 재생하도록 시도함과 동시에, 브로드캐스트를 통해 수신된 것과 조합될 때까지 각각의 서브-블록에 대해 HTTP를 통해 충분히 다운로딩하는 것은 서브-블록의 나머지 일부분을 디코딩 및 복원하기에 충분한 경우가 심지어 있을 수 있다. 이 경우, 브로드캐스트가 계속될 때, 각각의 서브-블록(블록)에 대해 필요한 다운로딩되는 HTTP의 양은, UE가 브로드캐스트를 통해 모든 서브-블록들(블록들)에 대해 점점 더 많은 리페어 서브-심볼들(심볼들)을 수신함에 따라 점점 더 적어진다.

[0240] (c) UE들은 소스 서브-블록들의 프리픽스들을 요청하고 있으므로, 파일의 일부분들에 대한 UE 요청들의 중첩은 가능한 한 높을 것이고, 예를 들어, 각각 수신하는 리페어 심볼들의 패턴이 독립적이고 완전히 상이할 수 있더라도, 소스 서브-블록(블록)에 대한 리페어 서브-심볼들(심볼들)의 동일한 양을 수신하는 2 개의 UE들은 그 소스 서브-블록(소스 블록)에 대해 동일한 바이트 범위 요청을 정확하게 행할 것이다.

[0241] 상이한 UE들이 서브-블록에 대한 브로드캐스트 세션으로부터 상이한 수의 리페어 서브-심볼들을 수신할 때, 리페어 서브-심볼들의 최소의 양을 수신하는 UE들은 일반적으로 소스 서브-블록에 대한 최대 프리픽스 바이트 범위 요청을 행할 것이고, 다른 UE들로부터의 모든 요청들은 이 요청의 서브세트일 것이다. 따라서, HTTP 서버가 서브-블록(블록)에 대해 하나의 UE에 대한 바이트 범위 요청을 이미 불러와서 서빙한 경우, HTTP 서버는 이 응답을 캐쉬할 수 있고, 동일한 서브-블록(블록)에 대하여 리페어 요청을 행하는 다음 UE에 대해 그것을 이용가능하게 한다. HTTP 캐쉬 서버는 UE에 의해 요청되는 것보다 더 큰 바이트 범위를 업스트림 서버로부터 사전에 요청할 수도 있고, 이에 따라, 현재의 요청을 넘어서서 데이터의 일부분들에 대하여 동일한 또는 다른 UE들로부터 추가의 요청들이 있는 경우에 요청의 크기 이상의 데이터가 이용가능할 때까지의 시간을 감소시킬 수 있다. 따라서, 하나의 UE가 파일의 관련된 일부분으로부터의 데이터의 프리픽스에 대한 요청을 행할 경우에는, 후속 UE가 파일의 동일한 관련된 일부분으로부터의 데이터의 더 큰 프리픽스에 대한 요청을 행할 때, HTTP 캐쉬 서버가 이미 캐쉬하였을 수도 있고, 데이터의 대응하는 프리픽스로 즉시 요청에 응답할 수 있을 수도 있다.

[0242] HTTP 리페어 서버가 중첩하는 바이트-범위들에 대하여, 예를 들어, 일부 임계치 이상의 다수의 요청들을 수신하는 경우, 서버는 유니캐스트 채널들을 이 수신기들에 중복적으로 로딩하는 것을 회피하기 위하여 데이터의 이 바이트 범위의 브로드캐스트를 요청하는 BMSC로 요청을 포워딩할 것을 판정하도록 또한 구성될 수 있다. 리페

어 서버는 BMSC에 의해 브로드캐스팅되지 않을 것이었던 소스 데이터를 각각의 수신기에 유니캐스팅 하기만 할 것이다. BMSC는 바이트-범위 요청을 요구되는 필요한 소스 심볼들로 변환할 수 있고, 그 다음으로, 이 소스 심볼들을 브로드캐스팅할 수 있거나, 또는 더 양호하게, 이 다수의 새로운 리페어 심볼들을 브로드캐스팅할 수 있다. 이것은 단말들이 유니캐스트 채널들로의 더 적은 로딩으로 그 파일들을 리페어하기 위하여 충분한 비-중복 심볼들을 수신하도록 한다.

[0243] 일부의 경우들에 있어서, UE가 그 소스 블록(서브-블록)과 연관된 컨텐츠를 재생하도록, 즉, 요청들을 행하기 위하여 "스트리밍 모드"를 이용하도록 준비되어 있을 때, UE가 소스 블록(서브-블록)에 대하여 소스 심볼들(서브-심볼들)에 대한 유니캐스트 요청들을 행하기만 하는 것은 장점이다. 하나의 이유는, UE가 실제로 재생할 컨텐츠에 대한 리페어 데이터를 요청하기만 하는 것이며, 일부의 이유로 UE가 컨텐츠의 일부를 전혀 재생하지 않더라도, UE가 그 블록(서브-블록)에 대한 데이터를 요청하는 유니캐스트 네트워크 자원들을 결코 낭비하는 것은 아니다.

[0244] 예를 들어, UE는 컨텐츠(파일)를 중간으로부터 재생하기 시작할 수도 있고, 컨텐츠의 1/4을 재생하기만 할 수도 있다. 이 예에서, 재생이 임박할 때에 요청들을 행하기만 하는 것은, 재생 전에 전체 컨텐츠 파일을 복원하기 위하여 유니캐스트 리페어 요청들이 행해질 경우에 필요하게 될 것의 약 1/4로 유니캐스트 네트워크 자원들을 감소시킨다. 물론, 일단 소스 블록(서브-블록)에 대한 충분한 소스 심볼들(서브-심볼들)이 수신되었으면, UE가 소스 블록(서브-블록)에 대하여 유니캐스트 리페어 서버로부터 추가적인 데이터를 요청하기 위한 임의의 필요성이 전혀 없다.

[0245] 시스템 운영자는 이 접근법들로부터 이익을 얻을 수 있다. 예를 들어, 유니캐스트 리페어 서버들의 설치가 간략화될 수 있고, 훨씬 덜 고가일 수 있다. 그것은 또한, HTTP 웹 서버들을 설치 및 관리하기 위해 훨씬 더 저렴하고 용이한 것에 의존하는 대신에, 특화된 리페어 서버들을 구입, 설치 및 운영하기 위한 필요성을 제거한다. 또 다른 예로서, 운영자는 인터넷 컨텐츠 파일들은 투명하게 캡처되고, 파일들에 대한 FEC 리페어 데이터는 UE들로 브로드캐스팅 되도록 하는 서비스를 설치할 수도 있고, 그 다음으로, UE들은, 자신들에게 브로드캐스팅된 수신된 FEC 리페어 데이터를 이용하고, 브로드캐스트를 통해 수신되는 FEC 리페어 데이터와 조합되는 것이 UE들이 재생할려고 하는 파일의 일부분들을 복원하도록 하기 위한 HTTP 1.1 바이트 범위 요청들을 결정하고, 현존하는 CDN들에서 이미 캐쉬될 수도 있는 인터넷-이용가능 컨텐츠 파일의 일부분들(반드시 운영자에 의해 운영되는 것이 아니고, 반드시 운영자에 의해 소싱되는 컨텐츠도 아님)에 대하여 인터넷을 통해 적절한 HTTP 1.1 바이트 범위 요청들을 행하기 위한 방법들을 포함한다.

[0246] 그렇게 함으로써, 운영자는 (파일들이 훨씬 더 신뢰성 있고 훨씬 더 신속하게 이용가능하므로) 더 양호한 사용자 경험을 제공하는 가치 있는 서비스를 제공할 수도 있는 한편, 동시에 (브로드캐스트 데이터가 모든 UE들에 유용하고, 각각의 개별적인 UE로의 다수의 개별적인 유니캐스트 접속들을 통해 데이터를 전송하는 것보다 전송하기 위해 더 적은 네트워크 자원을 취하므로) 파일들을 전달하기 위해 필요한 그 네트워크 상에서 전체 네트워크 트래픽의 양을 감소시키고, 이와 동시에, 컨텐츠는 다른 것들에 의해 제공되는 표준 HTTP 웹 서버들로부터 이미 이용가능하고 이것은 유니캐스트 리페어를 지원하기 위해 이용되는 것이므로, 운영자는 그 서비스를 지원하기 위하여 임의의 유니캐스트 리페어 서버들을 설치하는 비용을 회피할 수 있다.

[0247] 시스템 운영자는 현존하는 HTTP 웹 서버들을 목적에 맞게 또한 고칠 수 있고, 다른 컨텐츠 제공자들로부터 HTTP 웹 서버들을 통해 UE들로 전달되는 파일들에 대한 리페어 심볼들을 가로채고 브로드캐스팅하는 것과 같은 새로운 서비스 기회들을 만들 수 있다.

[0248] 추가적으로, 이 접근법은 운영자의 네트워크 대역폭 용량을 절약할 수 있다. 비용은 네트워크를 통해 전송된 브로드캐스트 데이터의 양인 반면, 이익은 그 UE에 의해 수신되는 브로드캐스트 데이터의 양의 컨텐츠를 재생하는 모든 UE들에 걸친 합이다. 이것은 운영자 네트워크 상에서의 더욱 양호한 사용자 경험뿐만 아니라, 컨텐츠의 더욱 신속하고 더욱 신뢰성 있는 전달을 제공할 수 있다.

[0249] 세션 설명 및 MBMS 다운로드 전달 프로토콜, FLUTE는 클라이언트(즉, UE)에 각각의 파일의 소스 블록 및 인코딩 심볼 구조를 결정하기 위한 충분한 정보를 제공한다. 이것으로부터, 클라이언트는 어느 소스 심볼들이 파일을 복원하기 위하여 요청 및 이용될 수 있는지를 결정할 수 있다. 따라서, MBMS 클라이언트는 (파일의) 소스 블록의 재구성을 완료할 요구된 소스 심볼들의 수를 식별할 수 있다.

[0250] 3GPP TS 26.346에서 특정된 바와 같이 MBMS FEC 방식이 이용될 때, MBMS 클라이언트는 요구되는 추가의 소스 심볼들의 결정을 행할 때에 이미 수신된 리페어 심볼들을 고려할 수 있다. 이 경우, 클라이언트는 (파일의) 각각

의 소스 블록에 대해 수 r을 식별해야 하고, MBMS FEC 디코더가 파일을 복원하도록 할 소스블록의 최초 r 소스 심볼들에 대한 리페어 요청을 행하여야 한다.

[0251] 일단 누락된 파일 데이터가 식별되면, MBMS 클라이언트는 하나 또는 그보다 많은 메시지들을, 누락된 파일 데이터의 복원을 허용하는 데이터의 송신을 요청하는 파일 리페어 서버로 전송한다. 특정한 MBMS 송신에 대한 모든 파일 리페어 요청들 및 리페어 응답들은 HTTP 프로토콜을 이용하는 단일의 TCP 세션에서 발생할 수 있다. 대안적으로, 바이트 범위 요청은 상이한 잠재적으로 중첩하는 HTTP/TCP 접속들, 또는 FTP 접속들, 또는 RTP 접속들 등을 통해 다수의 요청들로 구획될 수도 있다. 리페어 요청들에 대한 응답들은 네트워크 이용가능성과, 수신 디바이스가 얼마나 비지(busy) 한지와, 그 능력들에 따라, 매우 낮은 또는 높은 비트-레이트에서 전달될 수도 있다. 리페어 요청은 선택된 "serviceURI"로부터 분해된 파일 리페어 서버 IP 어드레스로 라우팅될 수 있다. 특정한 MBMS 클라이언트의 서버로의 TCP 접속의 개방 타이밍과, 최초 리페어 요청은 시간 윈도우 상에서 무작위화될 수 있으므로, 모든 UE들이 동시에 그 유니캐스트 요청들을 행하지는 않는다.

[0252] MBMS UE가 리페어 요청들에서 요청될 심볼들을 식별할 때, 그 답신은 그렇게 식별된 대응하는 심볼들일 것이고, 요청에서 식별된 관련된 소스 블록으로부터의 모든 심볼들을 포함해야 한다. 그 대신에, MBMS UE는 서브-블록킹이 이용될 때에 소스 서브-블록들의 서브-심볼들에 대한 요청들을 행할 수도 있다. 대안적으로, 여기에서 설명된 바와 같이, 바이트 범위 요청들은 심볼들 또는 서브-심볼들을 요청하기 위하여 행해질 수도 있다.

[0253] MBMS UE는 이것을 다른 방법들과 조합할 수도 있다. 예를 들어, Luby VI에서 설명된 바와 같이, MBMS UE는 브로드캐스트 세션에서 데이터를 수신할 수도 있고, 원래의 소스 파일을 디-인터리빙 또는 디코딩하지 않으면서 장기간 저장장치에 그것을 저장할 수도 있다. MBMS UE는 예를 들어, 소스 파일이 비디오 파일일 때, 최종 사용자가 소스 파일의 그 일부분들을 재생하기 위한 요청들을 행하는 것에 응답하여 원래의 소스 파일의 일부분들의 디-인터리빙 및 디코딩을 트리거(trigger)할 수도 있다. MBMS UE가 재생을 위해 요청되고 있는 원래의 소스 파일의 일부분들을 복원하기 위하여 브로드캐스트 세션으로부터 충분한 데이터를 수신하지 않는 경우, MBMS UE는 최종 사용자가 재생할 것을 요청하고 있는 소스 파일의 관련된 일부분들에 대하여 여기에서 설명된 바와 같은 리페어 요청들을 행할 수도 있다. 따라서, MBMS UE는 그들을 복원하기 위하여 충분한 데이터가 브로드캐스트 세션에서 수신되지 않을 경우에만, 어떤 서브-블록들 또는 소스 블록들에 대한 추가적인 데이터를 수신하기 위한 요청들을 행하기만 할 수도 있고, 요청들은 최종 사용자가 그 서브-블록들 또는 소스 블록들과 교차하는 소스 파일의 일부분들을 재생하기를 희망하는 시간에 또는 그 시간 근처에서 행해지기만 한다. 최종 사용자가 그 부분들의 재생을 요청할 때에 소스 파일의 일부분들에 대한 리페어 데이터가 요청되기만 하므로, 이 실시예는 네트워크 자원들을 최소화하고, 이에 따라, 파일의 일부분들이 결코 재생되지 않을 경우에는, 그 일부분들에 대한 리페어 요청들은 결코 행해지지 않는다. 또한, 이 동작들은 최종 사용자가 소스 파일의 이러한 일부분들의 재생을 요청할 경우에 수행되기만 하므로, 이 실시예는 장기간 저장장치로부터 판독하고, 서브-블록들 또는 소스 블록들을 복원하기 하여 데이터를 디-인터리빙 및 디코딩하기 위해 요구되는 디바이스 자원들을 최소화한다.

[0254] 대안적으로, MBMS UE가 이러한 리페어 요청들을 행하기 위하여 적절한 네트워크 접속 및 아이들 용량(idle capacity)을 가질 때, MBMS UE는 브로드캐스트 세션으로부터 복원하기 위하여 충분한 데이터가 수신되지 않은 서브-블록들 또는 소스 블록들에 대한 리페어 요청들을 행할 수도 있고, 그 다음으로, 이 서브-블록들 또는 소스 블록들에 대하여 브로드캐스트 세션으로부터 수신된 데이터에 추가로, 서브-블록들 또는 소스 블록들의 추후의 복원을 위하여 응답 데이터를 장기간 저장장치에 저장할 수도 있다. 사용자가 원래의 파일의 일부분들의 재생을 요청할 때, MBMS UE는 브로드캐스트 세션으로부터 그리고 파일의 사용자 요청된 일부분들과 교차하는 서브-블록들 또는 소스 블록들에 대한 리페어 요청들로부터 수신된 인터리빙된 그리고 인코딩된 데이터의 조합을 장기간 저장장치로부터 다시 판독할 수 있고, 그 서브-블록들 또는 소스 블록들을 복원할 수 있고, 이들을 재생을 위하여 바로 비디오 플레이어에 제공할 수 있다. 이 대안은, MBMS UE가 리페어 요청들을 행하고 응답들을 수신하기 위하여 적절한 네트워크에 접속되어 있을 때, 리페어 데이터가 요청될 수 있다는 장점을 가지는 반면, 사용자가 재생을 요청할 때, MBMS UE는 적절한 네트워크에 접속될 수도 없다. 또한, 재생을 위해 필요한 모든 데이터가 네트워크를 통해 전달을 요구하는 대신에, 디바이스 상에서의 사용자 요청 시에 국부적으로 저장되어 있으므로, 재생을 위한 사용자 요청들에 대한 재생 응답 시간은 더 작을 수도 있다.

[0255] 또 다른 대안으로서, MBMS UE는 복원하기 위해 추가적인 데이터가 요구되는 서브-블록들 또는 소스 블록들의 전부에 대해 리페어 데이터를 요청할 수 있고, 파일 전부를 복원할 수 있고, 추후의 재생 또는 이용을 위하여 장기간 저장장치에 그것을 저장할 수 있다. MBMS UE는 임의의 순서로 시간에 있어서의 임의의 시점에서 이 동작들을 수행할 수도 있고, 즉, 예를 들어, 파일의 서브-블록들 또는 소스 블록들을 완전히 복원하기 위하여 충분한 데이터가 브로드캐스트 세션을 통해 전달되지 않을 것이라는 점이 알려져 있는 경우, 또는 예를 들어, 최종

사용자가 브로드캐스트 세션이 종료되기 전에(그리고 잠재적으로 그것이 시작되기 전에) 컨텐츠의 재생을 시작하기를 희망하는 경우에는, 리페어 데이터가 일부의 경우에 있어서 브로드캐스트 세션으로부터 데이터를 수신하기 전에 요청될 수도 있다.

[0256] 일부 실시예들에서는, FEC를 수행하지 않는 UE들이 있다(그리고 이에 따라, 소스 심볼들이 모두 있고 리페어 심볼들이 전혀 없는 것을 선호할 것임). 이러한 경우들에 있어서, UE는 "No-code FEC"를 이용할 것이고 브로드캐스트 세션 및 임의의 리페어 요청 응답들에서 소스 심볼들을 수신할 것이지만, 이것은 다수의 상이한 UE들이 상이한 손실 패턴들을 경험하여 상이한 소스 심볼 리페어 요청들을 행하는 것만큼 확장가능하지 않을 수도 있다.

[0257] 일부 실시예들에서는, 리페어 브로드캐스트 세션들이 UE에서 Luby VI에서 예시된 바와 같이 브로드캐스트 데이터의 부분적인 재순서화를 이용한다. 거기서, 위에서 설명된 바와 같이 HTTP를 이용하여 유니캐스트 리페어 서비스의 조합은 Luby VI에서 설명된 방법들과 조합될 수 있다. 브로드캐스트 송신에서 수신된 심볼들/서브-심볼들은 Luby VI에서 설명된 바와 같이 플래쉬 또는 다른 메모리로 저장될 수 있다. 일부의 경우들에 있어서는, 브로드캐스트 세션에서 리페어 심볼들을 전송하기만 하는 것이 바람직하지만, 다른 경우들에 있어서는, 브로드캐스트 세션에서 소스 심볼들의 전부가 아니라면, 적어도 일부를 전송하는 것이 바람직하다. 심볼들 또는 서브-심볼들이 바이트 범위 요청들로 또는 다른 MBMS 방법들로 HTTP를 통해 리페어 서비스에서 요청될 때, 2 개의 상이한 가능성들이 야기된다(그리고 둘 모두 동시에 이용될 수도 있음):

[0258] (1) Luby VI에서 설명된 방법들은 브로드캐스트 세션으로부터 수신된 심볼들 또는 서브-심볼들을 플래쉬 메모리로 기록하기 위해 이용될 수 있고, 소스 블록 또는 서브-블록이 복원되어야 할 때, Luby VI에서 설명된 방법들은 브로드캐스트 세션으로부터 플래쉬 메모리에 저장된 소스 블록 또는 서브-블록에 대한 심볼들 또는 서브-심볼들을 RAM으로 판독하기 위해 이용될 수 있다. 다음으로, 이것들은 HTTP를 통해 수신되고 RAM으로 바로 저장된 심볼들 또는 서브-심볼들과 조합될 수 있고, 소스 블록 또는 서브-블록은 이 심볼들 또는 서브-심볼들의 조합을 이용하여 RAM에서 디코딩되고, 그 다음으로 복원된 소스 블록 또는 서브-블록은 재생을 위해 바로 제공될 수 있다.

[0259] (2) Luby VI에서 설명된 일부의 방법들은 HTTP를 통해 수신된 심볼들 또는 서브-심볼들을 플래쉬 메모리로 기록하기 위해 이용될 수 있고, 브로드캐스트 세션을 통해 수신된 심볼들 또는 서브-심볼들에 대해서도 동일할 수 있고, 그 다음으로, 시간에 있어서의 임의의 시점에, 소스 블록 또는 서브-블록은, HTTP 요청에 기반으로 하여 플래쉬 메모리에 저장된 소스 블록 또는 서브-블록에 대한 심볼들 또는 서브-심볼들과, 브로드캐스트 세션으로부터 플래쉬 메모리에 저장된 소스 블록 또는 서브-블록에 대한 심볼들 또는 서브-심볼들의 조합을 플래쉬 메모리로부터 RAM으로 판독함으로써 즉시 복원될 수 있다.

[0260] 상기 실시예들에서, 바람직하게는 브로드캐스트 세션을 통해 수신된 심볼들 또는 서브-심볼들과, 리페어 세션에서 수신된 심볼 또는 서브-심볼들은 심볼들 또는 서브-심볼들의 크게 상이한 세트들이다.

[0261] 도 22는 기본적인 FLUTE 파일 전달을 예시한다. FEC 페이로드 ID는 그 대신에 여기에서 설명된 UOSI 또는 UOFI를 포함할 수 있다는 것에 주목해야 한다.

[0262] 도 23은 기본적인 FLUTE 패킷 포맷의 일부의 부분들을 예시한다.

[0263] 도 24는 서브-블록들을 이용하는 전송기에서 발생하는 서브-블록 인코딩을 예시한다. 거기서 예시된 바와 같이, 파일은 하나 또는 그보다 많은 소스 블록(이 예에서는 하나)으로서 조직화될 수 있고, 서브-심볼들로부터 심볼들을 형성하도록 배치되는 서브-블록들로 분할될 수 있다.

[0264] 도 25는 서브-블록들을 이용하는 수신기에서 발생하는 서브-블록 디코딩을 예시한다. 일부의 경우들에 있어서, 수신기에서 저장하기 전에 부분적인 재순서화가 있다. 이것은 예를 들어, 판독과 기록 사이의 저장 엘리먼트에서 비대칭성이 있을 경우에 유용할 수도 있다.

[0265] 도 26은 서브-블록킹을 이용한 파일 전달을 예시한다. 전형적인 경우에 있어서, 수신기는 각각의 서브-블록 내에서 동일한 손실 패턴을 확인한다. 전송의 관점으로부터, 파일은 하나의 소스 블록일 수도 있고 각각의 패킷은 하나의 심볼을 포함할 수도 있다. 디코딩의 관점으로부터, 디코딩 동작들의 스케줄은 모든 서브-블록들에 대해 동일할 수 있고, 각각의 서브-블록에 대한 서브-심볼들의 손실 또는 수신 패턴이 동일할 수 있고, 수신기는 모든 서브-블록들에 대해 스케줄을 한번 연산한다. 다음으로, 디코딩 동작들의 스케줄은 각각의 서브-블록에 개별적으로 적용되어, 서브-심볼들에 대해 동작하고 동일한 스케줄을 모든 서브-블록들에 적용한다. 디코더의 CPU는 심지어 큰 소스 블록들에 대해 동작하는 것, 스크래치 RAM 메모리의 작은 양만을 이용하는 것, 이들이

재생될 때에 매체들의 서브-블록들의 액세스 및 디코딩을 허용하는 것을 양호하게 수행할 수 있다.

[0266] 도 27은 다수의 소스 블록들의 처리의 예를 예시한다. 일부의 경우들에 있어서, 양호한 네트워크 효율을 위하여, 심볼들은 라운드-로빈(round-robin) 방식으로 전송되고, 각각의 소스 블록에 대한 하나의 심볼이 각각의 라운드에서 전송된다. 다른 경우들에 있어서는, 더욱 용이한 클라이언트 복원 또는 엔드-투-엔드(end-to-end) 지연시간의 이유들로 인해, 심볼들은 순차적인 방식으로 전송되고, 즉, 각각의 소스 블록에 대하여, 모든 심볼들은 다른 소스 블록들로부터의 심볼들을 배치하지 않으면서 연속적인 시퀀스로 전송된다. 서브-블록킹은 모든 경우들에 이용될 수 있다.

[0267] 도 28은 FEC 및 FLUTE를 이용한 작업 흐름을 예시한다. 단계 1에서는, 파일이 FLUTE 세션에서 운반되도록 정의된다. 단계 2에서는, 파일이 블록들로 세그먼트화될 수도 있다. 세그먼트화(segmentation)를 위한 필요성은 FEC 방식이 처리할 수 있는 최대 크기에 의해 추진된다.

[0268] 단계 3에서는, FEC가 블록에 적용되고, 각각의 블록은 K 동일 크기의 체계적인 심볼들로 분할되고, R_1+R_2 리페어 FEC 심볼들이 발생된다. 단계 4에서는, K+R_1 심볼들이 브로드캐스트 송신을 통한 FLUTE를 통해, 또는 대안적으로 UDP를 이용하는 유니캐스트를 통해, 또는 HTTP를 이용하는 유니캐스트를 통해 전송된다. 각각의 IP/UDP/LCT 패킷은 심볼 크기에 따라 하나 또는 그보다 많은 심볼들을 운반한다. 단계 5에서, 수신기는 K+δ 심볼들이 FEC 디코딩을 위해 이용가능하도록 충분한 패킷들을 수집한다. RaptorQ를 이용하는 일부 실시예들에서는, δ = 2이다. 그러나, 다른 값들이 가능하고, 예를 들어, δ = 0 또는 δ = 0.01*K이다. 단계 3에서 발생된 추가적인 R_2 심볼들은 단계 4에서 송신으로부터 K+δ 심볼들을 수신하지 않는 수신기들에 대해 HTTP-기반 리페어 서비스를 제공하기 위하여 HTTP 서버들에 제공될 수도 있다.

[0269] 도 29는 위에서 설명된 브로드캐스트/리페어, 유니캐스트/소스 구성을 예시한다. 많은 수의 리페어 심볼들이 발생될 수 있다. UE는 성공적인 FEC 디코딩을 위하여 그 심볼들의 임의의 K+δ를 필요로 한다. 도시된 실시예에서, 각각의 브로드캐스트 BM-SC는 파일의 N 개의 상이한 Raptor 리페어 심볼들을 전송하고 각각의 BM-SC는 리페어 심볼들만, 즉, {K, K+1, ...} 범위의 ESI를 갖는 것들만을 전송한다.

[0270] 상기 실시예들의 하나의 장점은, UE가 상이한 BM-SC들의 커버리지 영역(coverage area)들 사이에서 로밍하고 있을 때, UE가 상이한 BM-SC들로부터의 심볼들을 이용할 수 있고, 심볼들의 중복 수신에 대해 걱정하지 않을 수 있어서, 더욱 효율적인 FEC 디코딩을 허용할 수 있다는 것이다. (비록 이것이 특화된 리페어 요청 서버들을 요구할 수도 있더라도) UE는 소스 심볼들에 대한 간단한 리페어 요청들을 행할 수 있지만, 여기의 다른 곳에서 더욱 상세하게 설명되는 바와 같이 원래의 소스 파일만을 갖는 기존의 HTTP 웹 서버들을 이용하여 HTTP 바이트 범위 요청들을 또한 행할 수 있다. 이것은 더욱 비용 효율적이고 융합된 인터넷/MBMS 해결책들을 허용한다.

[0271] 도 30은 시스템이 MBMS 베어러를 통해 어떻게 리페어 심볼들만을 브로드캐스팅할 것인지와, UE가 소스 심볼들의 인접 세트만을 어떻게 요청할 것인지를 예시한다. UE는 전형적으로 소스 블록을 성공적으로 디코딩하기 위하여 K+δ 심볼들을 필요로 하지만, MBMS 베어러를 통해 수신된 K+δ-R 리페어 심볼들만을 가진다. 소스 블록을 복원하기 위하여, UE는 소스 블록의 최초 R(또는 R>K일 때에는 K) 소스 심볼들을 요청한다. 그것은 일부의 다른 R 소스 심볼들을 요청할 수 있지만, 모든 UE들이 소스 심볼들의 중첩하는 세트들을 요청할 경우, 다운스트림 캐쉬들은 그 국부적인 캐쉬로부터 대부분의 수신기들로 요청된 소스 심볼들을 전달할 수 있을 가능성이 더 많다.

[0272] 이것은 리페어 서버에서 간단하고 효율적인 캐싱을 제공한다. 리페어 심볼들만이 브로드캐스트 채널을 통해 전송되므로, 소스 블록의 소스 심볼들은 UE에서 이미 수신된 심볼들과는 모두 구별된다. 그러므로, 리페어 서버는 소스 블록의 소스 심볼들을 저장하기만 하면 되고, 새로운 리페어 심볼들을 저장하거나 발생할 필요가 없다. 이것은 FEC 방식을 구현하거나 새로운 리페어 심볼들을 저장해야 할 필요가 없는 더 간단한 리페어 서버들을 허용한다.

[0273] 수신된 심볼들의 간단한 UE 프로세싱이 가능하다. UE는 수신한 리페어 심볼들의 수를 추적해야만 하고, 그 다음으로, 필요로 하는 소스 심볼들의 수(R이 K이하일 때에는 R, 또는 R이 K보다 클 때에는 K)를 계산해야만 한다. UE는 특정한 소스 심볼들이 누락되어 있는 것을 추적할 필요가 없고, 그 다음으로, 누락된 소스 심볼들만에 대한 요청을 발생할 필요가 없다.

[0274] 추가적으로, UE는 초기 ESI + (R의 값)을 이용하여 리페어 서버로부터 소스 심볼들의 인접 세트를 요청할 수 있다. 이것은 요청들이 매우 간단하고 짧게 유지한다. 이것은 또한 응답의 서버 구성을 간략하게 할 수 있는데, 그것은 서버가 소스 심볼들의 분리된 세트들을 검색 및 부가할 필요가 없기 때문이다. 이것은 "no-code" FEC

방식들을 이용하지 않는 설치들에서 특히 유리할 수 있는데, 그것은 서버가 단지 인접 소스 심볼 요청들을 처리해야만 하기 때문이다.

[0275] 도 31은 서브-블록킹이 이용될 때에 HTTP 바이트 범위 요청들을 통해 유니캐스트 리페어를 이용하는 것을 예시한다.

[0276] HTTP 바이트 범위 리페어 요청으로, UE는 동일한 소스 블록에 대하여 각각의 서브-블록에 대한 동일한 수의 바이트들을 요청할 수 있다. 각각의 서브-블록에 대한 바이트-범위의 하나의 요청은 브로드캐스트 세션에서 소스 심볼들이 전혀 전송되지 않는 대부분의 상황들에서 이용될 수 있다.

[0277] HTTP 요청들은 "필요에 따라"에 기반으로 하여 행해질 수 있다. UE는 그 서브-블록 내의 컨텐츠를 재생할 시간일 때에 각각의 서브-블록에 대한 요청을 행할 수 있다. UE가 서브-블록을 디코딩 및 복원하기 위한 브로드캐스트 세션과 같은 다른 전송들을 통해 서브-심볼 수신기와 조합하여 충분한 서브-심볼들을 가질 때까지, UE는 HTTP를 통해 서브-블록에 대한 컨텐츠를 요청 및 재생할 수 있다. 일단 충분한 서브-심볼들이 수신 및 디코딩되면, UE는 국부적으로 복원된 서브-블록들로부터의 재생을 심리스 방식으로 계속할 수 있다. UE는 결코 재생되지 않는 서브-블록들에 대한 HTTP 요청들을 행할 필요가 없다.

[0278] 바이트-범위 계산들의 예

[0279] 수신기는 어느 바이트 범위들을 요청할 것인지를 결정하기 위하여 이 섹션에서 설명되는 계산들을 수행할 수도 있다. 계산들은 FEC 구조에 의존할 수도 있다. 다음으로, 수신기는 바이트 범위 요청들에 기반으로 하여 수신기가 필요로 하는 심볼들을 얻기 위하여 요청을 서버로 전송할 수 있다.

[0280] 바람직한 실시예에서는, 수신기가 어느 심볼들을 수신하지 않았는지를 인식하기 위하여 FEC 객체 송신 정보("FEC OTI")를 이용한다. FEC OTI는 파일의 소스 블록, 서브-블록, 심볼, 및 서브-심볼 구조를 결정하고, 객체 전송 크기 F, 정렬 인자 Al, 심볼 크기 T, 소스 블록들의 수 Z, 및 소스 블록당 서브-블록들의 수 N을 포함한다. 수신기는 SBN, ESI 및 SuBN(각각 소스 블록 번호, 인코딩 심볼 식별자, 서브-블록 번호)에 의해 식별되는 각각의 심볼 또는 서브-심볼로, 얼마나 많은 심볼들 및 어느 심볼들을 필요로 하는지를 정확하게 결정하기 위하여 FEC 코드 자체의 특성들과 함께 이 정보의 전부를 이용한다. 이 실시예에서는, 소스 심볼들 또는 소스 서브-심볼들이 요청된다. 수신기는 다음에서 설명되는 바와 같이 이 심볼들 또는 서브-심볼들에 대응하는 파일에서 바이트-범위들을 계산하기 위하여 FEC OTI 정보를 이용한다.

[0281] 주어진 SBN 및 ESI에 대하여 서브-블록킹이 이용되지 않을 때(즉, N = 1일 때), 수신기는 (SBN, ESI) 쌍에 의해 식별되는 대응하는 소스 심볼의 파일에서 시작 바이트 번호 및 종료 바이트 번호를 연산할 수 있다.

[0282] 계산 프로세스는 지금부터 설명될 것이다. 이것은 수신기에서 또는 수신기의 로직에서 프로그램에 의해 행해질 수 있다. KT = ceil(F/T)는 파일에서 소스 심볼들의 수라고 한다(최근접 정수로 올림(round up)되고, 여기서 최종 소스 심볼은 전체 소스 심볼까지 영들로 패딩되는 것으로 생각될 수 있음).

[0283] KL = ceil(KT/Z) 및 KS = floor(KT/Z)이라고 하면, ZL = KT - KS*Z 및 ZS = Z - ZL이 되고, 여기서 KL은 최초 ZL 소스 블록들에서 소스 심볼들의 수이고, KS는 나머지 ZS 소스 블록들에서 소스 심볼들의 수이다.

[0284] 각각의 i = 0, ..., Z-1에 대하여, K_i이 소스 블록 i에서 소스 심볼들의 수라고 하고, 즉, i = 0, ..., ZL-1에 대하여 K_i = KL이고, i = ZL, ..., Z-1에 대하여, K_i = KS이다.

[0285] 다음으로, SBN 및 ESI에 의해 식별되는 심볼과 연관된 시작 바이트는 수학식 2에서와 같이 계산될 수 있다. 이 서브-심볼의 종료 바이트 위치는 T-1을 시작 바이트 위치에 추가함으로써 계산될 수 있다.

수학식 2

[0286]
$$\sum_{i<\mathrm{SBN}} K_i \cdot T + \mathrm{ESI} \cdot T$$

[0287] 주어진 SBN, SuBN 및 ESI에 대하여 서브-블록킹이 이용될 때(즉, N>1일 때), 수신기는 이 (SBN, SuBN, ESI) 삼중항에 의해 식별되는 대응하는 소스 서브-심볼의 파일에서 시작 바이트 번호 및 종료 바이트 번호를 연산할 수

있다. i = 0, ..., Z-1에 대한 KT, KL, KS, ZL, ZS, 및 K_i에 대한 계산들은 서브-블록킹이 이용되지 않는 경우(N = 1)에 대한 계산들과 동일할 수 있다.

[0288] 서브-블록 바이트 위치들을 식별하기 위한 추가적인 계산들은 아래에서 뒤따르는 바와 같이 수행될 수 있다.

[0289] TL = ceil(T/(N*Al)) 및 TS = floor(T/(N*Al))이라고 하면, NL = T/Al - TS*N 및 NS = N - NL이고, 여기서 TL*Al은 최초 NL 서브-블록들에서 서브-심볼들의 크기이고, TS*Al은 나머지 NS 서브-블록들에서 서브-블록들의 크기이다.

[0290] 각각의 j = 0, ..., N-1에 대하여, T_j는 서브-블록 j의 서브-심볼들의 크기라고 하고, 즉, j = 0, ..., NL-1에 대하여 T_j = TL*Al이고, j = NL, ..., N-1에 대하여 T_j = TS*Al이다.

[0291] 다음으로, SBN, ESI, 및 SuBN에 의해 식별되는 심볼과 연관된 시작 바이트는 수학식 3에서와 같이 계산될 수 있다. 이 서브-심볼의 종료 바이트 위치는 시작 바이트 위치에 T_{SuBN}-1을 추가함으로써 계산될 수 있다.

수학식 3

[0292]
$$\sum_{i<\mathrm{SBN}} K_i \cdot T + \sum_{j<\mathrm{SuBN}} T_j \cdot K_{SBN} + \mathrm{ESI} \cdot T_{SuBN}$$

[0293] 요청된 심볼들 또는 서브-심볼들의 시작 및 종료 바이트 위치들에 대한 이 계산들에 기반으로 하여, 수신기는 서버로 전송할 필요가 있는 바이트-범위 요청들을 결정할 수 있다. 인접하는 누락된 심볼들 또는 서브-심볼들이 있을 경우들에는, 수신기가 최초 누락된 심볼 또는 서브-심볼의 시작 바이트 위치로부터 바이트 범위 요청을 시작할 수 있고, 최종 누락된 심볼 또는 서브-심볼의 종료 바이트 위치에서 범위를 종료할 수 있다.

[0294] 서브-블록킹이 이용될 때에 상기 계산들이 어떻게 행해지는지를 예시하기 위하여, 수신기는 브로드캐스트 다운로드 세션을 가지고, 객체 크기 F = 20,000,000 바이트들, 정렬 인자 Al = 4 바이트들, 심볼 크기 T = 1320 바이트들, 소스 블록들의 수 Z = 2, 및 N = 12의 소스 블록당 서브-블록들의 수의 FEC OTI로 전달될, www.<mobile-operator>.com/news/weather.3gp의 URI를 갖는 3gp 파일을 얻기 위해 시도하는 것으로 가정한다. 이 예에서, 파일에서의 소스 심볼들의 총 수는 KT = 15,152이고, SBN = 0인 제 1 소스 블록 및 SBN = 1인 제 2 소스 블록은 각각 7,576 소스 심볼들을 가진다. 이 경우, 각각의 소스 블록은 7,576 소스 서브-심볼들을 각각 갖는 12 서브-블록들로 각각 더 구획되고, 최초 6 서브-블록들은 크기 112 바이트들의 서브-심볼들을 가지고 나머지 6 서브-블록들은 크기 108 바이트들의 서브-심볼들을 가진다. 브로드캐스트 다운로드 세션 후에, 수신기가 SBN = 0인 제 1 소스 블록으로부터 ESI들 12-29를 갖는 18 연속 소스 심볼들과, SBN = 1인 제 2 소스 블록으로부터 ESI들 12-29를 갖는 18 연속 소스 심볼들을 수신하지 않았고, 이 소스 블록들을 디코딩하기 위하여 이 소스 블록들의 각각으로부터 적어도 이러한 다수의 더 많은 심볼들을 필요로 한다는 것을 수신기가 인식한 것으로 가정한다.

[0295] SBN = 0인 제 1 소스 블록에 대하여, ESI = 12인 소스 심볼을 구성하는 12 서브-심볼들의 시작 바이트 위치들(18 손실된 소스 심볼들의 최초)은 도 32에 도시된 바와 같이 계산될 수 있다. 도 32는 ESI = 29인 소스 심볼을 구성하는 12 서브-심볼들의 종료 바이트 위치들(18 손실된 소스 심볼들의 최종)이 어떻게 계산될 수 있는지를 또한 도시한다.

[0296] 유사하게, 도 33은 ESI = 12인 심볼들을 구성하는 12 서브-심볼들의 시작 바이트 위치들이 SBN = 1인 제 2 소스 블록에 대해 어떻게 계산될 수 있는지를 도시한다. 도 33은 ESI = 29인 소스 심볼들을 구성하는 12 서브-심볼들에 대한 종료 바이트 위치들을 어떻게 계산할 것인지를 또한 도시한다.

[0297] 선택된 리페어 서버 URI가 http://mbmsrepair2.<mobile-operator>.com인 경우, HTTP GET 요청은 다음과 같을 수도 있다:

```
GET www.<mobile-operator>.com/news/weather.3gp
HTTP/1.1

Range: bytes = 1344-3359,849856-851871,1698368-
1700383,2546880-2548895,3395392-3397407,4243904-
4245919,5092368-5094311,5910576-5912519,6728784-
6730727,7546992-7548935,8365200-8367143,9183408-
9185351,10001664-10003679,10850176-10852191,11698688-
11700703,12547200-12549215,13395712-13397727,14244224-
14246239,15092688-15094631,15910896-15912839,16729104-
16731047,17547312-17549255,18365520-18367463,19183728-
19185671

Host: mbmsrepair2.<mobile-operator>.com
```

[0298]

[0299] 헤더: 통계 모음

[0300] 일부 실시예들에서는, 컨텐츠에 대해 행해지는 다양한 요청들과 관련될 때에 통계들이 모인다. 이 통계들은 예를 들어, 시스템 운영자가 컨텐츠의 브로드캐스트들로부터 얼마나 많은 파일들이 완전히 소싱되지 않는지를 말할 수 있기 위하여 유용할 수도 있다. 이것은 유니캐스트 방식으로 리페어 데이터를 제공하는 리페어 서버들에 대한 HTTP 로그들을 단지 봄으로써 행해질 수 있다. 그러나, 리페어 서버가 (브로드캐스트 심볼들의 수신을 시도하지 않거나 컨텐츠에 대한 1차 HTTP 소스로서 리페어 서버를 간단히 고려하는 디바이스로부터의 요청들로부터와 같이) 컨텐츠에 대한 직접적인 소스로서 또한 서빙하고 있는 경우, 통계들은 요청들을 과다 카운트할 것이다. 시스템 운영자는 디바이스들 자체가 브로드캐스트/유니캐스트 통계들을 보고하도록(그리고 유니캐스트-단독 요청을 보고하지 않도록) 할 수 있지만, 이것은 디바이스들에 의존하고, 그것은 번거로울 수 있다. 하나의 해결책에서는, HTTP 리페어 서버들이 통계들을 수집하고, 브로드캐스트 손실들로 인한 리페어 데이터에 대한 것인 요청들과, 컨텐츠에 대한 직접 유니캐스트 요청들인 요청들과의 사이를 구별할 수 있다.

[0301] 하나의 특정한 접근법에서는, 리페어 서버가 요청의 유형(리페어 요청 대 직접 요청)과, 어느 단말이 요청을 행하고 있는지를 결정한다. 제 1 부분은 브로드캐스트 채널을 통해 수신된 파일을 리페어링하기 위한 것이며 수신하고 있는 HTTP 요청들과, 유니캐스트를 통해 수신한 파일을 단지 요청 또는 리페어링하고 있는 단말들로부터의 요청들과의 사이를 구별한다. 단말 식별은 서버가 동일한 단말로부터의 요청들을 맵핑하며 각각의 단말에 대한 상세한 수신 통계들을 또한 제공하도록 한다. 이것은 이중-카운팅을 감소시키며 또한 리페어 패턴들을 양호하게 맵핑하기 위하여 유용할 수도 있다.

[0302] 유형 및 단말을 결정하기 위한 서버의 이 능력을 구현하기 위하여, 이를 행하는 몇몇 방식들이 있다.

[0303] 서버 URL 또는 파일의 URL은 직접적인 유니캐스트-유형 요청들로부터 구별하기 위한 리페어-유형 요청들에 특정할 수도 있다. 예를 들어, 이 URL들은 브로드캐스트 서비스의 연관된 전달 절차들 또는 브로드캐스팅되고 있는 파일의 파일 전달 테이블에서 제공되기만 할 수도 있다. 이 URL들은 브로드캐스팅되지 않은 파일을 직접적으로 다운로딩하기 위한 단말들에 대해서는 제공되지 않을 것이다. 서버는 특정한 URL들로의 요청들이 브로드캐스팅된 파일들을 리페어링하기 위한 것으로 결정할 수 있고, 이 요청들에 기반으로 하여 리페어 통계들을 수집할 수 있다. 동일한 단말로부터의 요청들을 식별하기 위하여, 서버는 HTTP GET 요청의 소스 IP 어드레스를 이용할 수 있다.

[0304] 서버는 어느 단말이 요청을 행하고 있는지를 식별하기 위하여 단말의 IP 어드레스를 이용할 수 있다. 서버는, HTTP 서버가 어느 IP 어드레스들이 브로드캐스트 단말들에 속하는지를 아는 경우, 요청이 브로드캐스트 단말로부터 나오는지 여부를 결정하기 위하여 IP 어드레스를 또한 이용할 수 있다. 이 접근법은, 단말이 브로드캐스트를 통해 수신하지 않은 파일들에 대한 데이터를 요청하도록 허용될 경우에 브로드캐스트 통계들을 수집할 정도로 신뢰성이 있지 않음에 주목해야 한다. 따라서, 이 서버들로부터 파일을 직접 요청하는 브로드캐스트 단말들에 대해 약간의 한정들이 두어질 수 있다.

[0305] 또 다른 접근법에서는, HTTP 서버로부터 데이터를 요청하기 위한 단말에 의해 이용되는 HTTP GET 요청에 확장이 추가될 수 있다. 확장은 요청이 브로드캐스팅된 파일을 리페어링하기 위한 것임을 식별할 것이다. 확장은 이동 디렉토리 번호(Mobile Directory Number) 또는 국제 이동 가입자 식별번호(International Mobile

Subscriber Indentity; IMSI)와 같은 단말의 고유 ID를 또한 식별할 수 있다. 확장 헤더의 또 다른 예는 3GPP TS 24.109에서 정의된 바와 같이 "X-3GPP-Intended-Identity extension-header"를 이용하는 것이다. 운영자가 프라이버시(privacy) 또는 보안 목적들을 위하여 ID들을 명백하게 전송하는 것을 원하지 않을 경우, 해쉬(hash) 또는 암호화가 이 ID들에 적용될 수 있다.

[0306] 상기 접근법들의 전부는 리페어 서버들의 커스터마이제이션(customization)을 요구할 수 있고, 이에 따라 표준 HTTP 서버들을 이용하는 것을 어렵게 한다. 상기 절차들의 서버 지원은 선택적인 특징일 수도 있다. 그러한 방식으로, UE(즉, 단말들)로부터의 수신 보고에 의존하고자 하는 시스템 운영자들은 표준 HTTP 서버들을 여전히 이용할 수 있다.

[0307] 소스 심볼들 및 리페어 심볼들의 일반화된 할당

[0308] 위에서 설명된 예들에서, 컨텐츠 전달 시스템은 UE들과 같은 클라이언트들과, MBMS 브로드캐스트 서버 및/또는 HTTP 서버들과 같은 서버들, 바람직하게는 URL 파일 요청들에 추가하여 바이트-범위 요청들을 지원하는 것들을 포함할 수도 있다. 그 접근법에서는, 리페어 심볼들만이 브로드캐스팅되고 소스 심볼들만이 HTTP 서버로의 요청들을 통해 이용가능한 것일 수도 있다. 그러나, 더욱 일반화된 경우에는, 리페어 및/또는 소스 심볼들이 브로드캐스팅될 수도 있고 리페어 및/또는 소스 심볼들이 HTTP 서버들로부터 이용가능할 수도 있다.

[0309] 하나의 예에서, HTTP 서버로부터 이용가능한 것의 성질은 UOSI 값들(대안적으로, UOFI 값들이라고 칭함)의 범위로서 표현 및/또는 시그널링된다. UOSI 값들의 범위가 원래의 파일에 대응하는 UOSI 값들, 즉, UOSI 값들 0 내지 ceil(F/T)-1과 동일하고, 여기서 F는 옥테트들로 된 파일 크기이고 T는 옥테트들로 된 심볼 크기인 경우, 그것은 HTTP 리페어 서버로부터 이용가능한 원래의 포맷, 즉, 원래-순서 HTTP 파일 포맷인 원래의 파일이다. UOSI 값들의 범위가 리페어 심볼들 또는 리페어 및 소스 심볼들의 UOSI들에 걸친 범위일 경우, 그러한 심볼들은, 아래에서 더욱 설명되고 이후에는 "확장된-원래-순서 HTTP 파일 포맷"이라고 지칭되는 원래-순서 HTTP 파일 포맷의 자연적인 확장인 포맷으로 마찬가지로 이용가능하다. 이것은 심볼들의 다양한 유형들을 처리하고 그것을 심리스 방식으로 행하기 위한 요청의 하나의 유형을 허용한다. 예를 들어, HTTP 서버는 단지 UOSI 범위를 시그널링하는 것 또는 일부 다른 시그널링에 의해, 리페어 심볼들만을 갖는 리페어 파일들 또는 리페어 심볼들 및 소스 심볼들을 갖는 리페어 파일들(혼합된 포맷)과 동일하게 소스 심볼들만을 갖는 리페어 파일들을 처리할 수 있다. 이 방식은 서브-블록킹이 이용될 때, 그리고 블록 크기가 파일의 상이한 소스 블록들 사이에서 가변적일 때에도 작동할 것이라는 점에 주목해야 한다.

[0310] 심볼들의 UOSI-순서화에서는, 소스 심볼들만을 포함하는 리페어 파일이 0 내지 KT-1의 UOSI 범위를 가질 것이고, 여기서 KT는 원래의 파일에서의 소스 심볼들의 총 수이다. 이것은 시그널링될 수 있는 범위이다. 예를 들어, 파일이 20,000,000 바이트들이고 심볼 크기가 1,000 바이트들일 경우, KT = 20,000이고, 이에 따라, [0, 19,999]의 UOSI 범위를 시그널링하는 것은 리페어 파일이 (원래 파일에서의 소스 블록들의 수에 관계없는) 원래의 소스 파일이고, 즉, 이 경우에 확장된-원래-순서 HTTP 파일이 원래-순서 HTTP 파일과 동일하다는 것을 표시한다.

[0311] 그 대신에, 컨텐츠 전달 시스템은 리페어 파일이 리페어 심볼들만을 운반하는 것을 시그널링하기를 원하였다고 가정한다. 따라서, 그것은 [20,000, X]의 UOSI 범위를 이용할 수 있었고, 여기서 x는 20,000 이상이다. 또한, 이 소스 파일은 FEC 객체 송신 정보("OTI")에서 시그널링되는 바와 같이, Z = 19 소스 블록들로 구획되고, 이에 따라 최초 12 소스 블록들은 1053 소스 심볼들을 가지며 나머지 7 소스 블록들은 1052 소스 심볼들을 가진다고 가정한다. 따라서, 리페어 파일이 리페어 심볼들만을 운반하고 각각의 소스 블록에 대한 리페어 심볼들의 수는 그 소스 블록에서의 소스 심볼들의 수보다 적어도 3 개를 초과한다는 것을 시그널링하기 위하여, UOSI 범위는 [20,000, 40,063]으로서 시그널링될 수 있었고, 즉, 19 소스 블록들의 각각에 대한 리페어 파일에서 1056 리페어 심볼들이 있다는 것을 시그널링한다.

[0312] 또한, 서브-블록킹이 이용되는 경우, 각각의 서브-블록의 서브-심볼들은 각각의 소스 블록 내에서 순서대로 있고, 즉, 소스 블록에 대한 리페어 데이터 내에, 제 1 서브-블록에 대한 1056 서브-심볼들이 있을 것이고, 다음으로 제 2 서브-블록에 대한 1056 서브-심볼들이 있는 등등과 같을 것이고, 즉, 리페어 심볼들이 소스 블록에 대한 리페어 데이터 내에서 연속적이고 이에 따라, 원래의 소스 파일에서 소스 데이터의 것과 동일한 조직화를 가질 수 있는 경우일 필요가 없으며, 즉, 확장된-원래-순서 HTTP 파일 포맷은 원래-순서 HTTP 파일 포맷의 자연적인 확장이다. 예를 들어, 19 소스 블록들의 각각이 N = 3 서브-블록들로 구획되고, 서브-블록들에 대한 서브-심볼 크기들이 각각 336, 332, 및 332 옥테트들이라고 가정한다. (J, L)은 파일 내의 서브-블록에 대한 식별자라고 하고, 여기서 J는 소스 블록 번호이고 L은 그 소스 블록 내의 서브-블록 번호이다. 다음으로, 파일은

서브-블록 (0, 0)에 대하여 크기가 각각 336 바이트들인 1056 서브-심볼들, 다음으로, 서브-블록 (0, 1)에 대하여 크기가 각각 332 바이트들인 1056 서브-심볼들, 다음으로, 서브-블록 (0, 2)에 대하여 크기가 332 바이트들인 1056 서브-심볼들, 다음으로, 서브-블록 (1, 0)에 대하여 크기가 각각 336 바이트들인 1056 서브-심볼들, 다음으로, 서브-블록 (1, 1)에 대하여 크기가 각각 332 바이트들인 1056 서브-심볼들, 다음으로, 서브-블록 (1, 2)에 대하여 크기가 각각 332 바이트들인 1056 서브-심볼들, 등을 포함할 것이다. 이 예에서, 소스 블록들 J = 0, ..., 11 내의 서브-블록들은 ESI 1053에서 시작하고 ESI 2108에서 종료되고, 소스 블록 J = 12, ..., 18 내의 서브 블록들은 ESI 1052에서 시작하고 ESI 2107에서 종료된다.

[0313] 또 다른 예로서, 수신기들이 파일을 복원하기 위해 요구된 데이터의 최대 20%를 누락하고 있을 경우, 서비스는 HTTP 리페어 서버로부터 리페어 데이터를 요구하기 위해 복귀하는 수신기들을 지원하는 것만을 의도한 것이라고 가정한다. 다음으로, 리페어 데이터를 포함하는 리페어 파일은 훨씬 더 작을 수 있고, 예를 들어, [20,000, 24,027]의 UOSI 범위에 대응하고, 즉, 리페어 파일에서 19 소스 블록들의 각각에 대해 212 리페어 심볼들이 이용가능하다. 이 예에서, 최초 11 소스 블록들 내의 서브-블록들에 대한 서브-심볼들의 ESI-범위는 1053 내지 1264이고, 나머지 8 소스 블록들 내의 서브-블록들에 대한 서브-심볼들의 ESI-범위는 1052 내지 1263이다.

[0314] 전형적으로, 원래의 소스 파일 포맷은 순차적이고, 예를 들어, 데이터는 파일의 바이트들이 시작부터 종료까지 컨텐츠의 재생을 위해 소비될 순서이고, 즉, 그것은 원래-순서 HTTP 파일 포맷이다. 이것은, HTTP 웹 서버들 상에서의 호스팅, 전달, 파일 시스템들에서의 저장, 등을 위한 자연적인 포맷을 포함하는, 많은 이유들로 편리한 포맷이다. 이것은 eMBMS에 의해 또는 유니캐스트에 의해 전달되든지 간에, 소스 파일들에 대한 포맷일 수도 있다. 리페어 심볼들만이 eMBMS 채널을 통해 전송되는 경우, 순차적인 포맷의 소스 파일의 일부분이 전혀 전송되지 않고, 그러므로 HTTP 웹 서버들로부터의 요청들은 중복적이지 않을 것이고, 소스 블록들(또는 서브-블록들)의 연속 초기 일부분들에 대한 요청들은 유용한 방식으로 파일을 복원하기 위하여 eMBMS를 통해 수신되는 것과 조합하기 위해 행해질 수 있다.

[0315] 수신기들로의 eMBMS 채널을 통해 소스 심볼들 및 리페어 심볼들의 둘 모두를 전송하는 것이 바람직한 경우들에 있어서, 소스 파일의 일부의 부분들이 eMBMS를 통해 이미 수신되었을 수도 있으므로, 원래의 소스 파일만을 HTTP 웹 서버들 상에서 이용가능하게 하는 것은 마찬가지로 작동하지 않을 수도 있고, 이에 따라, 소스 파일의 일부분들을 요청하기 위한 HTTP 웹 서버들로의 요청 패턴은 소스 파일의 다수의 작은 일부분들을 요청하는 것을 종료할 수도 있고(eMBMS를 통해 수신되었던 일부분들을 스킵함), 이에 따라, HTTP 바이트 범위 요청들의 용량의 측면에서 상당한 비효율들을 잠재적으로 야기시킬 수 있고, HTTP 웹 서버들에서 비인접 데이터 부분들을 액세스할 수 있고, eMBMS 채널로부터의 상이한 패킷 손실 패턴들을 경험하는 상이한 수신기들로부터 상이한 데이터를 요청하는 것으로 인해 캐싱 효율을 감소시킬 수 있다.

[0316] 여기에서 설명된 바와 같이, 이것은 리페어 심볼들을 포함하기 위하여 또는 일부의 경우들에 있어서 전적으로 리페어 심볼들로 이루어지도록 하기 위하여 자연적인 방식으로 원래의 소스 파일 포맷을 확장하는 확장된-원래-순서 HTTP 파일 포맷을 이용함으로써 해결될 수 있다. 이 새로운 파일 포맷을 이용하면, 파일은 기존의 HTTP 웹 서버들 상에서 이용가능하게 될 수도 있고 융합된 서비스를 제공하기 위해 이용될 수도 있고, 리페어 요청들은 소스 및 리페어 심볼들의 둘 모두를 eMBMS를 통해 전달하는 것과 조합된 바이트 범위 요청들을 이용하여 기존의 HTTP 웹 서버들에 대해 행해질 수도 있어서, 캐싱 효율이 높고, HTTP 바이트 범위 요청들은 최소화되고, eMBMS 세션에서 상이한 수신기들에 의해 경험되는 손실 패턴들에 관계없이 파일에서 동일한 지점에서 시작하는 데이터의 인접 위치들을 요청한다.

[0317] 기술적으로, 파일 X는 연관된 FEC OTI 파라미터들(F, Al, T, Z, N)을 가질 수도 있다. 확장된-원래-순서 HTTP 파일 포맷에서는, 파일에서의 데이터의 범위가 대안적으로 UOFI라고 칭해지는 유니버셜 객체 심볼 식별자("UOSI")의 측면에서 표현될 수도 있다. 심볼의 UOSI는 심볼의 소스 블록 번호("SBN") 및 그 인코딩 심볼 식별자("ESI")에 대응하고, 즉, A의 SBN 및 B의 ESI를 갖는 심볼에 대하여, UOSI는 C = A + B*Z일 것이다. 역방향 맵핑은, C의 UOSI 값을 갖는 심볼에 대하여, ESI 값은 B = floor(C/Z)이고 SBN은 A = C - B*Z이다. UOSI 범위를 시그널링함에 있어서, 범위는 [SU, EU] 형태로 표현될 수도 있고, 여기서 SU는 파일에 대한 시작 UOSI 값이고 EU는 파일에 대한 종료 UOSI 값이다. SU는 EU 이하라는 것에 주목해야 한다. 다음으로, 파일 내에 포함되는 심볼들의 범위는 SU로부터 EU까지의 범위인 UOSI들을 갖는 그러한 심볼들이고, 이에 따라, 파일에서의 심볼들의 총 수는 EU - SU + 1이다. 이 새로운 파일 포맷의 특수한 경우는, SU = 0이고 EU = ceil(F/T) -1인 경우이고, 이 경우에 파일은 원래의 파일과 정확하게 동일하고, 즉, 확장된-원래-순서 HTTP 파일 포맷은 이 경우에 원래-순서 HTTP 파일 포맷과 일치한다(파일 크기 F가 심볼 크기 T의 배수가 아닐 경우에, 모듈로(modulo)는 아

마도 제로 패딩을 갖는 최종 심볼을 패딩하는 것임).

[0318] 이러한 파일은 FEC OTI 파라미터들을 포함하는 URL에 의해, 그리고 파일에서의 심볼들의 시작 및 종료 UOSI들에 의해 식별될 수도 있다. 새로운 확장된-원래-순서 HTTP 파일 포맷에서, 파일에서의 데이터는 SBN = 0인 소스 블록에 대한 ESI들과 연관된 모든 심볼들, 다음으로, SBN = 1인 소스 블록에 대한 ESI들과 연관된 모든 심볼들, 등으로부터 SBN = Z-1인 소스 블록에 대한 ESI들과 연관된 모든 심볼들까지일 것이다. 소스 블록들 내에서, 조직화는 소스-블록들의 순서에 있고, 즉, 소스 블록의 제 1 서브-블록과 연관된 모든 서브-심볼들은 증가하는 ESI의 순서이고, 다음으로, 소스 블록의 제 2 서브-블록과 연관된 모든 서브-심볼들, 등을 거쳐 소스 블록의 최종 (N-1) 서브 블록과 연관된 모든 서브-심볼들이 뒤따른다.

[0319] UOSI 범위 SU 내지 EU에 있는 SBN = J인 소스 블록에서의 소스 심볼들의 ESI들은 다음과 같이 계산될 수 있다. SESI(J)가 UOSI 범위 (SU, ..., EU)에서의 심볼들 중에서 소스 블록 J에 대한 시작 ESI라고 하고, EESI(J)가 UOSI 범위 (SU, ..., EU)에서의 심볼들 중에서 소스 블록 J에 대한 종료 ESI라고 한다. 다음으로, SESI(J)는 다음과 같이 계산될 수 있다:

[0320] SESI(J) = floor(SU/Z)

[0321] If ((SU-Z*floor(SU/Z)) > J) then SESI(J) = SESI(J) + 1.

[0322] 유사하게, EESI(J)는 다음과 같이 계산될 수 있다:

[0323] EESI(J) = floor(EU/Z)

[0324] If ((EU-Z*floor(EU/Z)) < J) then EESI(J) = EESI(J) - 1.

[0325] 다음으로, UOSI 범위 (SU, ..., EU)에서의 소스 블록 J로부터의 심볼들의 총 수는 NK(J) = EESI(J) - SESI(J) + 1이다. 파일의 상이한 소스 블록들에 대한 심볼들의 수는 값들 (SU, EU)의 범위에 관계없이 서로 중의 하나 내에 모두 있다는 것에 주목해야 한다.

[0326] 서브-블록 L에서의 서브-심볼들의 크기 TS(L)은 여기에서 도시된 바와 같이 계산될 수 있다. 아래의 공식에서, TS(L')은 서브-블록 L'에서의 서브-심볼들의 크기이다.

[0327] SESI(J) < = I < = EESI(J)인, 파일 내에서의 (SBN, SuBN, ESI) - 삼중항 (J, L, I)에 의해 인덱싱되는 서브-심볼의 시작 바이트 위치는 다음과 같이 계산될 수 있다:

[0328] T*(Sum{J' = 0, ..., J-1} NK(J')) + NK(J)*(Sum{L' = 0, ..., L-1} TS(L')) + TS(L)*(I-SESI(J)).

[0329] 파일 내에서 (J, L, I)에 의해 인덱싱되는 서브-심볼의 종료 바이트 위치는 시작 바이트 위치 플러스(plus) TS(L) - 1이다.

[0330] 일 예로서, 파일은 SU = ceil(F/T) 및 EU = SU + Z*M -1로 설정함으로써 형성되고, M은 임의의 수신 클라이언트에 의해 예상되는 소스 블록당 심볼들의 최대 손실이고, 이에 따라, 이 경우에 Z 소스 블록들의 각각에 대하여 파일에서 정확하게 M 심볼들이 있고, 파일은 전적으로 리페어 심볼들로 구성되고, 파일에서 최소 ESI 값들을 갖는 리페어 심볼들은 리페어 심볼들 중에서 최소의 가능한 ESI 값들을 가진다고 가정한다. 원래의 eMBMS 브로드캐스트에서, 소스 심볼들만이 전송된다고 가정할 경우, 전적으로 리페어 심볼들로 구성되는 이 파일은 HTTP 웹 서버들 상에서 업로딩될 수 있고, eMBMS에 대해 리페어 서비스를 제공하기 위하여 바이트 범위 요청들을 발행하는 클라이언트들에 의해 이용될 수 있다. 일부의 더욱 일반론에서는, eMBMS 세션에서 전송된 최대 UOSI가 X인 경우, 리페어를 위해 이용된 파일은 UOSI 범위(X', Y')에 의해 특정될 수 있고, 여기서 X'>X 및 (Y' - X')/Z은 임의의 클라이언트가 소스 블록 또는 소스 서브-블록을 복원하기 위하여 요청할 필요가 있을 심볼들 또는 서브-심볼들의 수에 대한 상한(upper bound)이다.

[0331] 도 34는 그 원래의 순서에서 파일을 예시하고, 여기서, 이 예에서는 파일이 Z = 3 소스 블록들로 구획되고, 각각의 소스 블록은 N = 2 서브-블록들로 더욱 구획되고, 최초 2 개의 소스 블록들의 각각에는 4 소스 심볼들이 있고, 제 3 소스 블록에는 3 소스 심볼들이 있다. 도 34에서, 각각의 소스 서브-심볼은 삼중항 (J, L, I)로 라벨이 붙여져 있고, 여기서 J는 SBN이고, L은 SuBN이고, I는 서브-심볼의 ESI이다.

[0332] 도 35는 도 34에 대응하는 파일에 대한 소스 심볼들을 도시하고, 여기서 소스 심볼들은 UOSI 순서로 열거된다. 따라서, UOSI = 0인 소스 심볼은 (0, 0, 0) 및 (0, 1, 0)으로 라벨이 붙여진 2 개의 서브-심볼들을 포함하고, UOSI = 5인 소스 심볼은 (2, 0, 1) 및 (2, 1, 1)로 라벨이 붙여진 2 개의 서브-심볼들을 포함한다.

[0333] 도 36은 도 34에 도시된 파일에 대한 11 내지 19 범위인 UOSI들을 갖는 리페어 심볼들을 도시한다. 이 리페어 심볼들은 원래의 파일로부터, 즉, 도 35에 도시된 소스 심볼들로부터 발생될 수 있다. 각각의 리페어 심볼은 동일한 ESI를 이용하여 주어진 소스 블록의 2 개의 서브-블록들의 각각으로부터 발생된 2 개의 서브-심볼들을 포함하고, 예를 들어, UOSI = 16인 리페어 심볼은 (1, 0, 5) 및 (1, 1, 5)로 라벨이 붙여진 2 개의 서브-심볼들을 포함한다.

[0334] 도 37은 도 34에 도시된 파일에 대한 UOSI 범위 11 내지 19에 대응하는 확장된-원래-순서 HTTP 파일 포맷을 도시한다. 이 순서화에서는, 특정된 UOSI 범위의 심볼들을 포함하는 서브-심볼들이 SuBN에 의한 것 내에서, 그리고 ESI에 의한 것 내에서, SBN에 의해 먼저 순서화된다. 따라서, 주어진 서브-블록에 대한 모든 서브-심볼들은 확장된-원래-순서 HTTP 파일 포맷 내에서 연속한다. 이 예에서는, ESI 범위가 SBN = 0 및 SBN = 1인 소스 블록들을 포함하는 서브-블록들의 서브-심볼들에 대해 4 내지 6인 반면, ESI 범위가 SBN = 2인 소스 블록을 포함하는 서브-블록들의 서브-심볼들에 대해서는 3 내지 5인 것에 주목해야 한다. 확장된-원래-순서 HTTP 파일 포맷은 단일의 서브-블록으로부터의 임의의 수의 서브-심볼들에서 그 서브-블록에 대한 파일에서 서브-심볼들의 수까지를 요청하기 위하여 단일의 바이트 범위를 이용하는 것을 지원한다는 것에 주목해야 한다.

[0335] 확장된-원래-순서 HTTP 파일 포맷은 소스 및 리페어 심볼들의 혼합을 포함하는 파일을 발생하기 위하여 또한 이용될 수 있다. 예를 들어, 도 38은 도 34에 도시된 파일에 대한 UOSI 범위 8 내지 12에 대한 확장된-원래-순서 HTTP 파일 포맷을 도시한다. 이 예에서는, UOSI들 8, 9 및 10이 소스 심볼들에 대응하고 UOSI들 11 및 12가 리페어 심볼들에 대응하는 것에 주목해야 한다. 또한, 이 예에서는, 소스 블록들의 일부에 대해 상이한 수들의 심볼들이 있고, 예를 들어, SBN = 0 및 SBN = 2인 소스 블록들에 대해 2 심볼들이 있고, SBN = 1인 소스 블록들에 대해 1 심볼이 있다는 것에 주목해야 한다.

[0336] 일부의 경우들에 있어서, 주어진 소스 파일에 대하여 하나를 초과하는 확장된-원래-순서 HTTP 파일을 제공하는 것이 바람직할 수도 있다. 예를 들어, 상이한 파일들은 상이한 CDN들에서 또는 상이한 영역들에서 이용가능할 수도 있다. 예를 들어, 원래의 소스 파일에 대응하는 UOSI 범위 [0, 19,999]를 갖는 하나의 파일이 있을 수도 있고, UOSI 범위들 [20,000, 22,000], [22,001, 26,354] 및 [45,651, 64,356]을 각각 갖는 3 개의 다른 파일들이 있을 수도 있다. 대안적으로, 원래의 소스 파일에 대응하는 다수의 파일들, 예를 들어, 20,000 소스 심볼들을 갖는 소스 파일에 대해 UOSI 범위들 [0, 7,000], [7,001, 14,000], [14,000, 19,999]을 각각 갖는 3 개의 파일들이 있을 수도 있다. 또 다른 예로서, 이용가능한 중첩하는 UOSI 범위들, 예를 들어, UOSI 범위들 [0, 15,000] 및 [10,000, 30,000]을 각각 갖는 2 개의 파일들이 있을 수도 있다. 일부의 경우들에 있어서, 확장된-원래-순서 HTTP 파일에서의 심볼들은 소스 및 리페어 심볼들의 혼합일 수도 있고, 예를 들어, UOSI 범위는 [10,000, 30,000]인 반면, 소스 파일에서의 소스 심볼들의 수는 20,000이다.

[0337] 원래-순서 HTTP 파일들에 대한 HTTP 바이트 범위 요청들의 예들

[0338] MBMS UE는 참조된 자원의 소스 심볼들 또는 서브-심볼들의 전부 또는 서브세트를 각각 요청하기 위하여 RFC 2616에서 정의된 바와 같이 기존의 HTTP 1.1 부분적인 GET 요청들을 이용할 수도 있다. 이 메시지 포맷들은, MBMS UE가 바이트 범위 요청들을 지원하는 파일 리페어 서버로부터 심볼들을 요청하고 있는 경우, 즉, 그 URI가 "byteRangeServiceURI" 또는 "priorityByteRangeServiceURI"로서 열거되어 있을 경우, 또는 FLUTE FDT 사례에서 "Content-Location" 속성의 서버 URI 부분으로부터 바로 소스 데이터를 요청할 때에 이용된다.

[0339] 원래-순서 HTTP 파일에 대하여, MBMS UE는 송신될 자원의 모든 소스 심볼들을 요청할 때에 HTTP GET 요청을 이용한다. MBMS UE가 소스 심볼들 또는 서브-심볼들의 서브세트의 송신을 요청하기만 할 경우에는, UE가 RFC 2626의 14.35.2에서 정의된 바와 같이 범위 요청 헤더를 갖는 HTTP 부분적인 GET 요청을 이용할 수도 있다. MBMS UE는 RFC 2616의 14.35.1에서 정의된 바와 같이 byte-range-spec으로서 특정 소스 심볼들 또는 서브-심볼들을 표시한다.

[0340] 메시징 효율성을 위하여, HTTP GET 방법은 UE가 단일의 부분적인 GET 요청 내에 다수의 바이트 범위 요청들을 포함하도록 한다. UE가 단일의 요청 내에 다수의 바이트 범위들을 포함하는 경우, HTTP GET 요청은 HTTP 서버에 의한 절단(truncation)을 회피하기 위하여 길이가 2048 바이트들을 초과하지 않아야 한다.

[0341] MBMS UE가 소스 심볼들 또는 서브-심볼들의 다수의 서브세트들 중에서 선택할 수 있는 것으로 결정하는 경우, MBMS UE는 이용가능한 최저 ESI 값들로 서브세트를 요청해야 하고, 즉, 소스 블록 또는 소스 서브-블록의 시작부터 누락된 소스 심볼들 또는 서브-심볼들을 각각 선택해야 한다. 이것은 HTTP 파일 리페어 서버들의 캐싱 효율을 개선시킨다. 다른 전략들이 또한 가능하고, 예를 들어, MBMS UE는 이용가능한 최고 ESI 값들로 서브세트

들을 요청한다.

[0342] 하나를 초과하는 파일이 특정한 MBMS 다운로드 세션에서 다운로딩되었을 경우, 그리고 MBMMS 클라이언트가 그 세션에서 수신된 하나를 초과하는 파일에 대한 리페어 데이터를 필요로 하는 경우, MBMS 클라이언트는 각각의 파일에 대한 별개의 HTTP GET 요청들을 전송할 수 있다.

[0343] 예를 들어, MBMS 다운로드 세션에서, www.example.com/news/sports.3gp로 설정된 FLUTE FDT 사례에서 "Content-Location" 속성을 갖는 3gp 파일이 객체 크기 F = 20,000,000 바이트들, Al = 4 바이트들의 정렬 인자, T = 1320 바이트들의 심볼 크기, Z = 20인 소스 블록들의 수, 및 N = 1의 소스 블록당 서브-블록들의 수의 FEC OTI로 전달되어야 한다고 가정한다. 그 예에서는, 파일에서의 소스 심볼들의 총 수가 KT = 15,152이고, SBN들 0-11을 갖는 최초 12 소스 블록들이 각각 758 소스 심볼들을 가지고, SBN들 12-19를 갖는 나머지 8 소스 블록들이 각각 757 소스 심볼들을 가진다.

[0344] MBMS 다운로드 세션 후에는, MBMS 클라이언트가 SBN = 5인 소스 블록으로부터 ESI들 12-29를 갖는 18 연속 소스 심볼들과, SBN = 19인 최종 소스 블록으로부터 ESI들 27-36을 갖는 10 연속 소스 심볼들을 수신하지 않았고, 이 소스 블록들을 디코딩하기 위하여 이 소스 블록들의 각각으로부터 적어도 이 다수의 더 많은 심볼들을 필요로 한다는 것을 MBMS 클라이언트가 인식하였다고 가정한다. SBN = 5인 소스 블록에 대하여, ESI = 12인 최초 누락된 소스 심볼의 시작 바이트 위치는 도 39에 도시된 바와 같이 계산될 수 있다. ESI = 29인 최종 누락된 심볼의 종료 바이트 위치에 대한 계산들은 도 39에 또한 도시되어 있다. SBN = 19인 소스 블록에 대한 누락된 심볼들에 대한 시작 및 종료 바이트 위치들에 대한 계산들은 도 39의 표의 최종 행(row)에 또한 도시되어 있다.

[0345] 선택된 리페어 서버가 바이트 범위 요청들을 받아들이고(예를 들어, 서버 URI는 연관된 전달 절차 메타 데이터 프래그먼트에서 "byteRangeServiceURI" 엘리먼트로 식별됨) 그 서버 URI는 http://httprepair1.example.com인 경우, 리페어 서버로의 HTTP GET 요청은 다음과 같다:

```
GET /www.example.com/news/sports.3gp HTTP/1.1
Range: bytes = 5018640-5042399,19037040-19050239
Host: httprepair1.example.com
```

[0346]

[0347] 확장된-원래-순서 HTTP 파일 포맷에서의 파일의 UOSI 범위는 다수의 상이한 접근법들 또는 실시예들을 통해 UE로 시그널링될 수 있다. 하나의 실시예에서, 이것은 예를 들어, 새로운 속성들 "Start-UOSI" 및 "End-UOSI"를 추가함으로써 파일과 연관된 FLUTE 사례에서 파일 설명 테이블의 일부로서 전달될 수 있다. 또 다른 실시예에서는, 예를 들어, "start_uosi = xxx" 및 "end_uosi = yyy" 속성들을 URL로 삽입/부가함으로써, UOSI 범위는 파일의 URL에서 직접 인코딩될 수 있다. 대안적으로, 예를 들어, "Start-UOSI" 및 "End-UOSI"에 대한 새로운 속성들을 아래 설명들 중의 하나에 추가함으로써, 파일의 UOSI 범위는 파일에 대한 전달 세션과 연관된 세션 설명 프로토콜(SDP), 미디어 프리젠테이션 설명(MPD), 또는 사용자 서비스 설명(USD)을 통해 시그널링될 수 있다.

[0348] 파일 포맷을 적절하게 디코딩하기 위하여 UE에 대해 필요하게 되는 보조 파일 포맷 파라미터들은 UOSI 범위에 추가하여 시그널링될 수 있다. 이러한 추가적인 파라미터들의 예들은 일부의 FEC 객체 송신 정보를 포함한다. 이 보조 파일 포맷 파라미터들은 이 파라미터들에 대한 속성들을 정의함으로써 FLUTE 사례의 파일 설명 테이블(FDT)에서 또한 전송될 수 있다. 또 다른 실시예에서, 보조 파일 포맷 파라미터들은 파일의 URL에서 직접 인코딩될 수 있다. 대안적으로, 보조 파일 포맷 파라미터들은 예를 들어, 파라미터들에 대한 새로운 속성들을 아래 설명들 중의 하나에 추가함으로써, 파일에 대한 전달 세션과 연관된 세션 설명 프로토콜(SDP), 미디어 프리젠테이션 설명(MPD), 또는 사용자 서비스 설명(USD)을 통해 시그널링될 수 있다.

[0349] 하나의 실시예에서는, FEC OTI 정보, UOSI 범위, 또는 임의의 일반적인 보조 파일 포맷 정보가 MIME 유형으로서 시그널링될 수 있다. 예를 들어, FEC OTI 파라미터들은, FLUTE 심볼들이 특정된 FEC OTI 값들을 갖는 MP4 파일 내로 포함되었음을 표시하기 위하여, RFC 6381에 따라 스트링 "application/mp4 profiles='flut' fec-oti = 'XYZ'"을 이용하여 시그널링될 수 있다. 또 다른 실시예에서는, 새로운 파일 포맷이 정의되고 파라미터를 "fec-oti"로서 MIMO 유형 등록에 추가한다. 이 시그널링의 예는 스트링 "application/flut fec-oti = 'XYZ'"이다.

[0350] 일부 실시예들에서는, 요구되지는 않지만, HTTP 서버 상에 저장된 파일들에 대한 보조 파일 포맷 파라미터들이 브로드캐스트를 통해 전송되는 파일에 대한 보조 파일 포맷 파라미터들과 동일하다. 대안적인 실시예에서는, 보조 파일 포맷 파라미터들의 일부가 브로드캐스팅되었던 파일과, 리페어를 가능하게 하기 위한 HTTP 서버 상에

저장되어 있는 파일에 대해 상이할 수 있다.

[0351] 예를 들어, HTTP 서버 상에 저장된 파일 포맷에 대한 FEC OTI는 브로드캐스트를 포맷하기 위하여 이용되었던 FEC OTI와는 상이할 수 있다. 실제적인 예는 브로드캐스트 데이터가 상이한 영역들에서 상이한 FEC OTI 파라미터들을 이용하여 인코딩될 때이고; 상이한 커버리지 영역들 사이를 이동하는 디바이스는 디바이스의 현재의 위치에 따라 상이하게 인코딩된 파일들을 서빙하는 상이한 서버들로 라우팅된 그 유니캐스트 리페어 요청들을 가질 수도 있다. 파라미터 값들의 이 상이한 세트들을 구별하기 위하여, 파라미터 값들은 위에서 설명된 바와 같이 리페어 서버 상에 저장된 파일명 또는 URI로 직접 인코딩될 수 있다. 대안적으로, 그 속성들은 FDT, SDP, MPD, 또는 USD의 일부로서 단말에 시그널링될 때, 상이한 명칭들, 예를 들어, "FEC_OTI_bcast" 및 "FEC_OTI_file"로 정의될 수 있다. 다음으로, 단말(예를 들어, UE 단말)은 브로드캐스트 또는 HTTP 서버로부터 검색된 파일을 디코딩하기 위한 적절한 절차들을 결정할 수 있다. 하나의 실시예에서, 단말은 상이한 파일 포맷 구성들을 이용하여 다수의 파일들이 프리젠팅될 때에 HTTP 서버들 상의 어느 파일들로부터 데이터를 검색할 것인지를 판정하기 위하여 보조 파일 파라미터들을 이용할 수도 있다. 또 다른 실시예들에서, 파일에 대한 HTTP 요청은 요구되는 OTI 파라미터들을 표시하는 프라그마 디렉티브(pragma directive)들을 포함할 수도 있고; 다음으로, HTTP 서버는 정확하게 인코딩된 파일로부터의 요청을 서빙하거나, 이러한 인코딩된 파일이 이용가능하지 않을 경우에 요청을 거절하기 위하여 내장된 디렉티브들을 이용할 것이다.

[0352] 또 다른 실시예에서는, 단말들이 보조 파일 포맷 파라미터들을 시그널링하기 위하여 변경되거나 포맷팅될 수 없는 URL들로부터 리페어 데이터를 요청하는 것이 필요할 수도 있고, 예를 들어, 소스 파일들이 인터넷 URL들을 갖는 서버들 상에 위치된다. 또 다른 접근법은 URL에서 선택적인 파라미터들을 시그널링하기 위한 능력을 가지는 것이고, 이들이 URL에서 명시적으로 시그널링되지 않는 경우에는 디폴트 파라미터 값들을 가정한다. 예를 들어, 시그널링될 잠재적인 파라미터들이 포맷 유형(예를 들어, SS-순서 HTTP 파일 포맷, 또는 확장된-원래-순서 HTTP 파일 포맷), FEC OTI 파라미터들, 및 UOSI 범위이었던 것으로 가정한다.

[0353] URL www.example.com을 갖는 유니캐스트 파일이 있다고 가정하고, 즉, 이 파라미터들 중의 어느 것도 URL에서 시그널링되지 않는다. 그 사례에서, 클라이언트에 의해 가정된 디폴트 파라미터들은 확장된-원래-순서 HTTP 파일 포맷, UOSI 범위 = 0, ..., F/T, FEC OTI = 브로드캐스트 FEC OTI일 수 있고, 즉, 그것은 단지 원래의 순서에서 원래의 소스 파일이다.

[0354] 그 대신에, UOSI 범위가 시그널링되는 경우, 예를 들어, 클라이언트에 제공되는 URL이 www.example.UOSI = X_to_Y.com인 경우, 클라이언트는 UOSI 범위가 X 내지 Y인 것을 확인할 것이고 이것을 사용할 것이며, FEC OTI가 시그널링되지 않으므로, 그것은 디폴트에 의해 이 파일의 브로드캐스트 버전의 FEC OTI와 동일할 것이고, 확장된-원래-순서 HTTP 파일 포맷이 가정될 것이다.

[0355] 세 번째 예에서는, UOSI 범위 및 FEC OTI의 둘 모두가 URL, 예를 들어, www.example.FEC_OTI = (F,Al,T,Z,N).UOSI = X_to_Y.com에서 시그널링되는 경우, 디폴트의 확장된-원래-순서 HTTP 파일 포맷이 주어진 FEC OTI 및 UOSI 범위와 함께 가정될 것이다.

[0356] 네 번째 예에서는, 포맷이 URL, 즉, www.example.Format = SS.FEC_OTI = (F,Al,T,Z,N).com에서 시그널링되는 경우, 이것은 포맷이 SS-순서화된 HTTP 파일 포맷임을 클라이언트에 시그널링하고 FEC OTI를 제공한다(FEC OTI가 제공되지 않았을 경우, 그것은 이 예에서 브로드캐스트 FEC OTI와 동일한 것으로 가정될 것이다).

[0357] 따라서, URL에서 시그널링될 수 있는 이 선택적인 파라미터들을 가짐으로써, 다음을 달성할 수 있다: 파일들과 연관된 URL을 이미 가지는 파일들에 대한 수정되지 않은 URL들을 허용하거나(시그널링되지 않은 파라미터들에 대한 디폴트 값들이 파일들의 포맷과 일치하는 것을 확실히 함), 또는 파라미터들의 일부 또는 전부를 시그널링하는 URL들을 허용한다. 핵심적인 장점은, 이 경우에 시그널링되지 않은 파라미터들이 파일이 원래의 소스 파일임을 표시하는 것을 디폴트로 할 것이므로, 수정되지 않은 URL들이 인터넷 또는 오버-더-탑(Over-The-Top)을 통해 이미 현존하는 이용가능한 파일들을 위해 이용될 수 있다는 것이다. 따라서, 파라미터들을 시그널링하는 URL들을 갖는 파일들에 대하여, 이 파일들은 아마도 특수하게 형성되어야 할 것이다(예를 들어, 파일의 부분인 리페어 데이터를 생성하거나, 파일의 포맷을 재조직화함). 이 파일들은 원래의 소스 파일들이 아니기 때문에, URL에서 추가적인 파라미터들을 시그널링해야 하는 것은 이 파일들이 특수하게 형성될 때에 URL이 이 파일들에 대해 발생될 필요가 있으므로 허용 가능하다.

[0358] 확장된-원래-순서 HTTP 파일에 대하여, MBMS UE는 송신될 자원의 모든 심볼들을 요청할 때에 HTTP GET 요청을 이용한다. MBMS UE가 심볼들 또는 서브-심볼들의 서브세트의 송신을 요청하기만 할 경우에는, UE가 RFC 2626의

14.35.2에서 정의된 바와 같이 범위 요청 헤더를 갖는 HTTP 부분적인 GET 요청을 이용할 수도 있다. MBMS UE는 RFC 2616의 14.35.1에서 정의된 바와 같이 특정한 심볼들 또는 서브-심볼들을 byte-range-spec으로서 표시한다.

[0359] 예를 들어, MBMS 다운로드 세션에서는, www.example.com/news/sports.3gp로 설정된 FLUTE FDT 사례에서의 "Content-Location" 속성을 갖는 3gp 파일이 객체 크기 F = 20,000,000 바이트들, Al = 4바이트들의 정렬 인자, T = 1320 바이트들의 심볼 크기, 소스 블록들의 수 Z = 20, 및 N = 1의 소스 블록당 서브-블록들의 수의 FEC OTI로 전달되어야 한다고 가정한다. 그 예에서, 파일에서의 소스 심볼들의 총 수는 KT = 15,152이고, SBN들 0-11을 갖는 최초 12 소스 블록들은 각각 758 소스 심볼들을 가지고, SBN들 12-19를 갖는 나머지 8 소스 블록들은 각각 757 소스 심볼들을 가진다. 이 예에서, 소스 심볼들은 MBMS 다운로드 세션에서 송신되고, HTTP를 통해 이용가능한 리페어 파일과 연관된 UOSI 범위는 [15, 152, 17, 151]이며, 즉, 리페어 파일은 확장된-원래-순서 HTTP 파일 포맷의 20 소스 블록들의 각각에 대한 최초 100 리페어 심볼들을 포함한다고 가정한다.

[0360] MBMS 다운로드 세션 후에, MBMS 클라이언트는 이 소스 블록들을 디코딩하기 위하여, SBN = 5인 소스 블록에 대한 추가적인 18 심볼들, 및 SBN = 19인 최종 소스 블록에 대한 추가적인 10 심볼들을 필요로 한다는 것을 인식하였다고 가정한다. SBN = 5인 소스 블록에 대하여, ESI = 758인 파일에서의 최초 심볼의 시작 바이트 위치는 도 40에 도시된 바와 같이 계산될 수 있다. ESI = 775인 최종 심볼의 종료 바이트 위치에 대한 계산들은 도 40에 또한 도시되어 있다. SBN = 19인 소스 블록에 대하여 심볼들에 대한 시작 및 종료 바이트 위치들의 계산들은 도 40의 표의 최종 행에 또한 도시되어 있다.

[0361] 선택된 리페어 서버가 바이트 범위 요청들을 받아들이고(예를 들어, 서버 URI는 연관된 전달 절차 메타 데이터 프래그먼트에서 "byteRangeServiceURI" 엘리먼트로 식별됨) 그 서버 URI가 http://httprepair1.example.com인 경우, 리페어 서버로의 HTTP GET 요청은 다음과 같다:

```
GET /www.example.com/news/sports.3gp HTTP/1.1
Range: bytes = 5002800-5026559,19001400-19014599
Host: httprepair1.example.com
```

[0362]

[0363] 서브-블록킹이 이용되는 또 다른 예에서는, MBMS 다운로드 세션에서, www.example.com/news/international.3gp로 설정된 FLUTE FDT 사례에서의 "Content-Location" 속성을 갖는 3gp 파일이 객체 크기 F = 20,000,000 바이트들, Al = 4바이트들의 정렬 인자, T = 1320 바이트들의 심볼 크기, 소스 블록들의 수 Z = 20, 및 N = 12의 소스 블록당 서브-블록들의 수의 FEC OTI로 전달되어야 한다고 가정한다. 그 예에서, 파일에서의 소스 심볼들의 총 수는 KT = 15,152이고, SBN = 0인 제 1 소스 블록 및 SBN = 1인 제 2 소스 블록은 각각 7,576 소스 심볼들을 가진다. 그 예에서, 각각의 소스 블록은 7,576 소스 서브-심볼들을 각각 갖는 12 서브-블록들로 각각 더 구획되고, 최초 6 서브-블록들은 크기 112 바이트들의 서브-심볼들을 가지고, 나머지 6 서브-블록들은 크기 108 바이트들의 서브-심볼들을 가진다. 이 예에서, 소스 심볼들이 MBMS 다운로드 세션에서 송신되고, HTTP를 통해 이용가능한 리페어 파일과 연관된 UOSI 범위는 [15, 152, 17, 151]이며, 즉, 리페어 파일은 확장된-원래-순서 HTTP 파일 포맷의 2 소스 블록들의 각각에 대해 최초 1000 리페어 심볼들을 포함하는 것으로 가정한다.

[0364] MBMS 다운로드 세션 후에, MBMS 클라이언트는 이 소스 블록들을 디코딩하기 위하여, SBN = 0인 제 1 소스 블록에 대한 18 추가적인 심볼들을 필요로 하고, SBN = 1인 제 2 소스 블록에 대한 18 추가적인 심볼들을 필요로 한다는 것을 인식하였다고 가정한다.

[0365] SBN = 0인 제 1 소스 블록에 대하여, ESI = 7576인 심볼들(리페어 파일에서 이용가능한 제 1 심볼)을 구성하는 12 서브-심볼들의 시작 바이트 위치들은 도 41에서 도시된 바와 같이 계산될 수 있다. 도 41은 ESI = 7593인 심볼(18 심볼들 중의 최종 심볼)을 구성하는 12 서브-심볼들의 종료 바이트 위치들이 어떻게 계산될 수 있는지를 또한 도시한다. 도 41의 표에서, "SuBN"은 서브-블록 번호를 지칭한다.

[0366] 위에서 설명된 실시예들의 다수의 다른 실시예들 및 변형들이 있다. 여기에 설명된 모든 포맷들에 대한 변형의 예로서, 소스 블록들의 심볼들 및 서브-심볼들이 ESI의 역방향 순서로 순서화되어 있는 것을 제외하고는 동일한 포맷이 이용될 수도 있다. 예를 들어, 파일이 2 개의 소스 블록들로 구획되고, 각각의 소스 블록은 2 개의 서브-블록들을 가지고, 각각의 서브-블록은 3 개의 서브-심볼들을 가지는 경우, 이 파일의 원래-순서 HTTP 포맷은 (0, 0, 0), (0, 0, 1), (0, 0, 2), (0, 1, 0), (0, 1, 1), (0, 1, 2), (1, 0, 0), (1, 0, 1), (1, 0, 2), (1, 1, 0), (1, 1, 1), (1, 1, 2)로서 표현될 수 있고, 여기서, 각각의 삼중항은 각각의 서브-심볼의 SBN, SuBN, 및 ESI를 표시한다. 역방향 순서의 이 파일은 (0, 0, 2), (0, 0, 1), (0, 0, 0), (0, 1, 2), (0, 1, 1), (0,

1, 0), (1, 0, 2), (1, 0, 1), (1, 0, 0), (1, 1, 2), (1, 1, 1), (1, 1, 0)로서 표현될 수 있다. 유사한 역방향 변형들은 여기에서 설명된 모든 다른 파일 포맷들에 대해 유지된다. 파일의 역방향 변형은 예를 들어, 클라이언트가 2 개의 상이한 서버들로부터 컨텐츠를 재구성하기 위해 이용될 수 있는 데이터를 다운로딩할 경우에 유용할 수 있고, 클라이언트는 하나의 파일 포맷으로 저장된 제 1 서버로부터의 컨텐츠를 재구성하기 위해 이용될 수 있는 데이터를 다운로딩 또는 수신하고 있고, 그리고 클라이언트는 역방향 포맷으로 저장된 제 2 서버로부터의 컨텐츠를 재구성하기 위해 이용될 수 있는 파일을 다운로딩 또는 수신하고 있다. 2 개의 서버들로부터의 데이터의 전달 속도들이 상이하거나 예측불가능할 경우, 클라이언트는 컨텐츠를 재구성하기 위하여 2 개의 서버들로부터 조합하여 충분한 데이터가 도달할 때까지 간단히 대기할 수 있고, 그 다음으로, 클라이언트는 2 개의 서버들로부터의 접속을 종결하거나 추가적인 데이터의 수신을 간단히 정지할 수 있다. 하나의 포맷이 다른 포맷의 역방향이므로, 클라이언트는, 얼마나 많은 데이터를 클라이언트가 각각의 서버로부터 수신하는지에 관계없이, 클라이언트가 컨텐츠를 재구성하도록 하는 비-중복 데이터를 주로 수신한다.

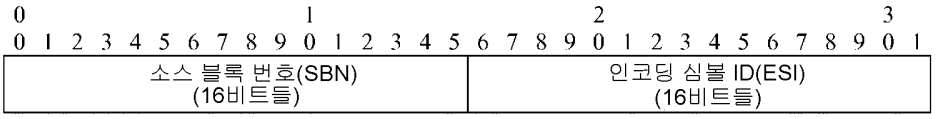
[0367] 확장된-원래-순서 HTTP 포맷 파일들의 포맷 또는 여기에서 설명된 다수의 다른 포맷들을 특정하는 다수의 다른 변형들이 있다. 예를 들어, 바이트-범위는 UOSI 범위 대신에, 확장된-원래-순서 HTTP 포맷 파일의 데이터를 특정하기 위하여 이용될 수 있다. 예를 들어, UOSI 범위가 (X, Y)이고 심볼 크기가 T인 경우, 동일한 포맷을 특정하는 바이트 범위는 바이트 X*T에서 시작하며 바이트 (Y+1)*T - 1에서 종료하는 바이트 범위일 수 있다. 포맷을 특정하는 다른 변형들은 파일의 각각의 소스 블록에 대하여 ESI 범위들의 명시적인 리스트를 특정하는 것을 포함한다. 예를 들어, 파일이 5 소스 심볼들을 가지는 경우, 파일의 포맷의 사양은 SBN = 0, ESI들 = 20-50, SBN = 1, ESI들 = 30-40, SBN = 2, ESI들 = 10-19, SBN = 3, ESI들 = 0-50, SBN = 4, ESI들 = 17-37일 수 있다. 이 예에서, 동일한 소스 블록은 ESI들의 상이한 범위들과 함께 다수 회 열거될 수 있고, 예를 들어, SBN = 4, ESI들 = 55-65는 위의 리스트에 추가될 수 있고, 그 다음으로, 포맷팅된 파일에서 SBN = 4인 소스 블록에 대하여 ESI들의 2 개의 범위들이 있을 것이다. 대안적으로, ESI 범위들은 바이트 범위들로 대체될 수 있고, 여기서, 이 경우에 각각의 바이트 범위는 소스 블록에 상대적일 것이다. 예를 들어, 위에서, 심볼 크기가 1,000 바이트들인 경우, 바이트 범위 30,000 - 40,999는 ESI 범위 30-40과 동등할 것이다. 포맷들의 사양들의 이 변형들은 여기에서 이전에 설명된 모든 포맷들과 조합될 수 있다.

[0368] 지금까지 설명된 바와 같이, 일부의 특정한 예들이 있다. 이 개시내용을 읽은 후에, 추가의 실시예들이 당업자에 의해 구상될 수 있다. 다른 실시예들에서는, 상기 개시된 발명의 조합들 또는 하위 조합들이 유리하게 행해질 수 있다. 컴포넌트들의 일 예의 배치들은 예시의 목적들로 도시되어 있고, 조합들, 추가들, 재배치들 등은 본 발명의 대안적인 실시예들에서 고려된다는 것을 이해해야 한다. 따라서, 발명은 예시적인 실시예들에 대하여 설명되었지만, 당업자는 여러 수정들이 가능하다는 것을 인식할 것이다.

[0369] 예를 들어, 여기에서 설명된 프로세스들은 하드웨어 컴포넌트들, 소프트웨어 컴포넌트들, 및/또는 그 임의의 조합을 이용하여 구현될 수도 있다. 이에 따라, 명세서 및 도면들은 한정적인 것이 아니라 예시적인 의미로 간주되어야 한다. 그러나, 청구항들에서 기재된 바와 같이 본 발명의 범위 및 더 넓은 사상으로부터 이탈하지 않으면서 다양한 수정들 및 변경들이 발명에 행해질 수도 있고, 발명은 다음의 청구항들의 범위 내에서 모든 수정예들 및 등가물들을 포괄하도록 의도된 것은 분명할 것이다.

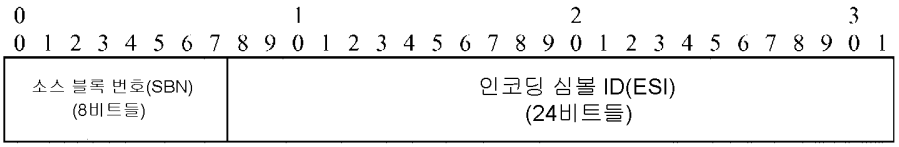
도면

도면1a



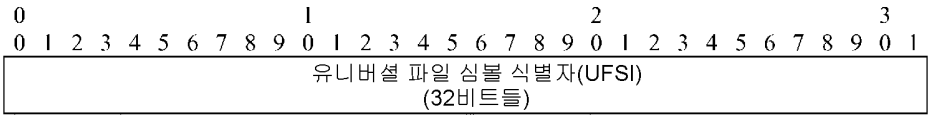
Raptor-RFC 5053 페이로드 ID(종래 기술)

도면1b

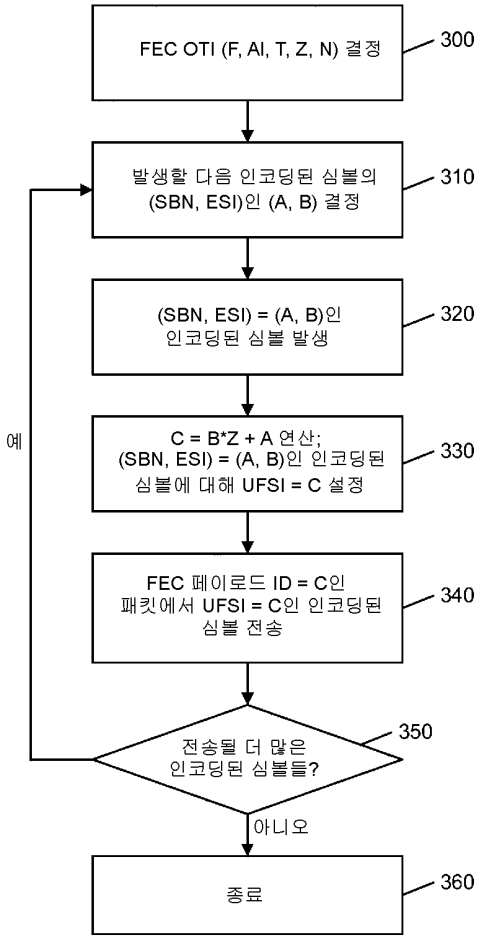


RaptroQ-Spec 페이로드 ID(종래 기술)

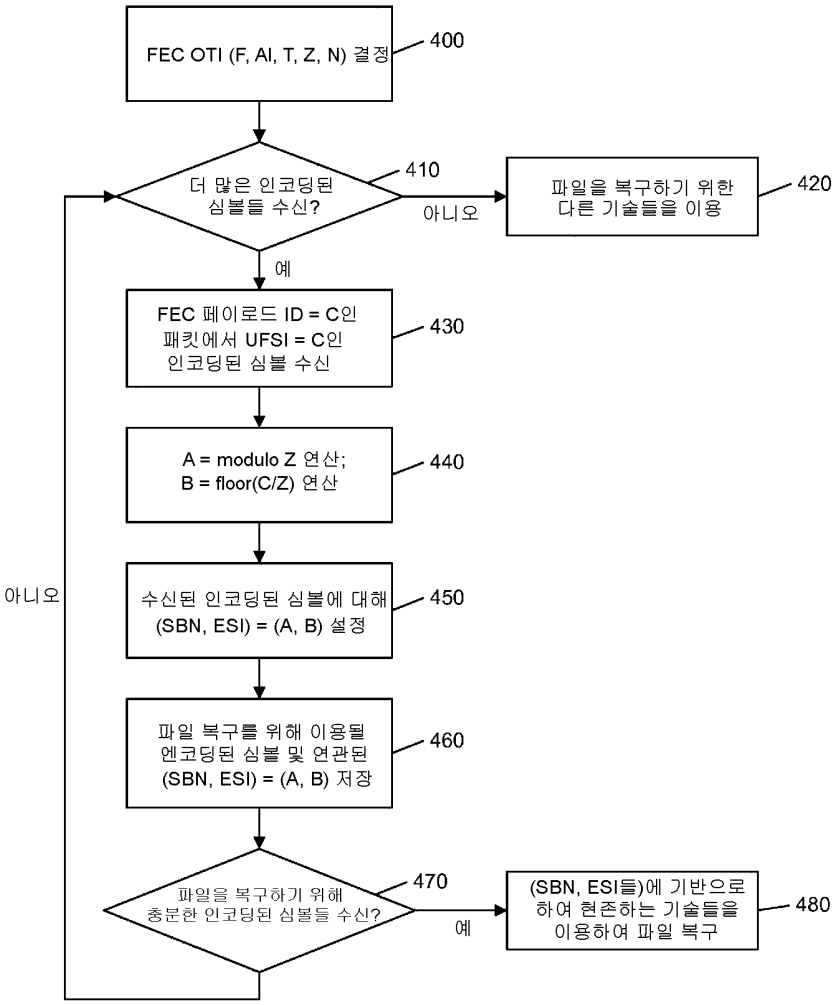
도면2



도면3



도면4



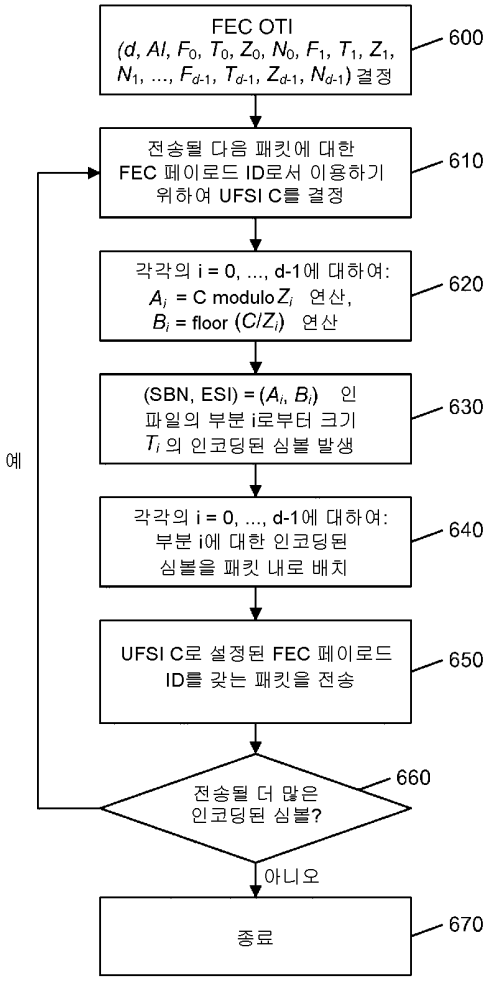
도면5a

ESI 0	ESI 1	ESI 2	ESI 3	ESI 4	ESI 5	
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	SBN 0
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	SBN 1
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	SBN 2
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)		SBN 3
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)		SBN 4

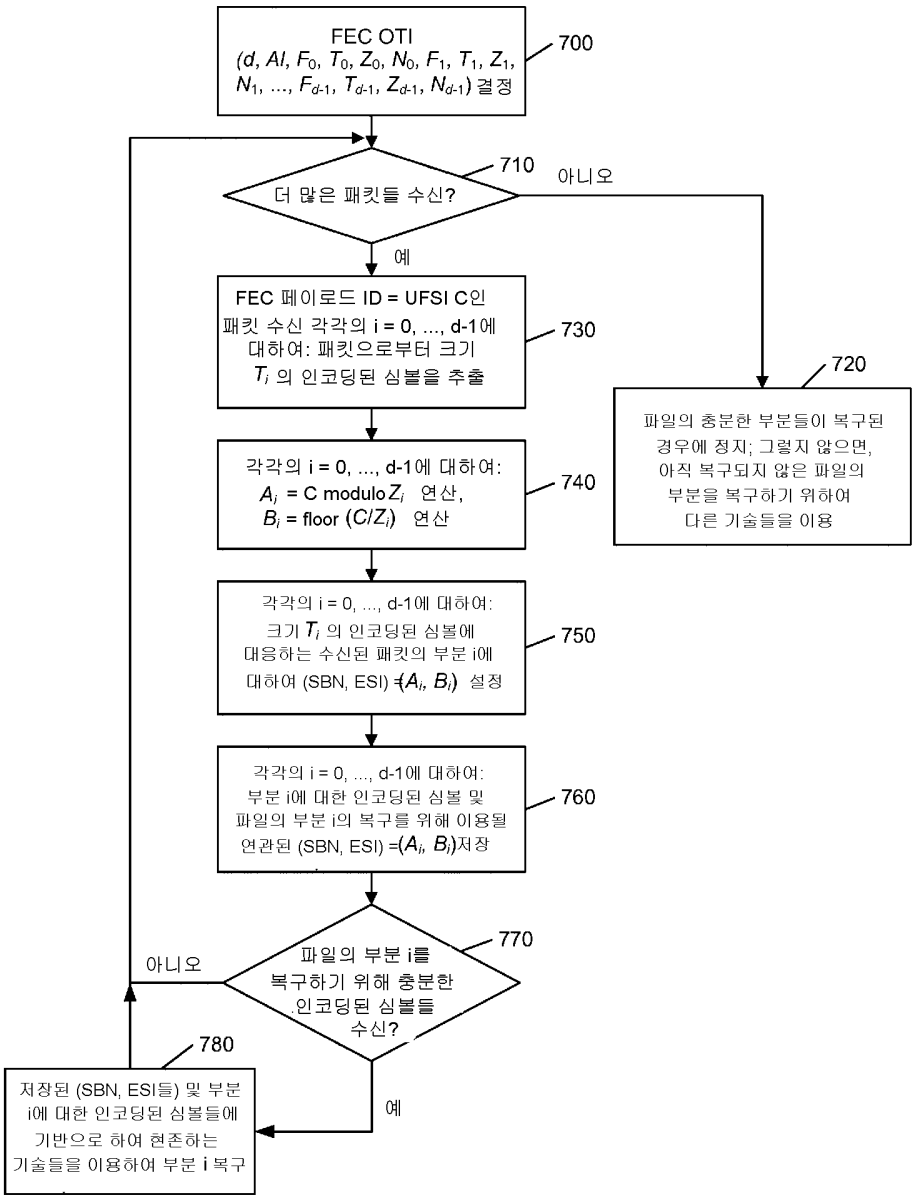
도면5b

SBN 0	UFSI 0	UFSI 5	UFSI 10	UFSI 15	UFSI 20	UFSI 25
SBN 1	UFSI 1	UFSI 6	UFSI 11	UFSI 16	UFSI 21	UFSI 26
SBN 2	UFSI 2	UFSI 7	UFSI 12	UFSI 17	UFSI 22	UFSI 27
SBN 3	UFSI 3	UFSI 8	UFSI 13	UFSI 18	UFSI 23	
SBN 4	UFSI 4	UFSI 9	UFSI 14	UFSI 19	UFSI 24	

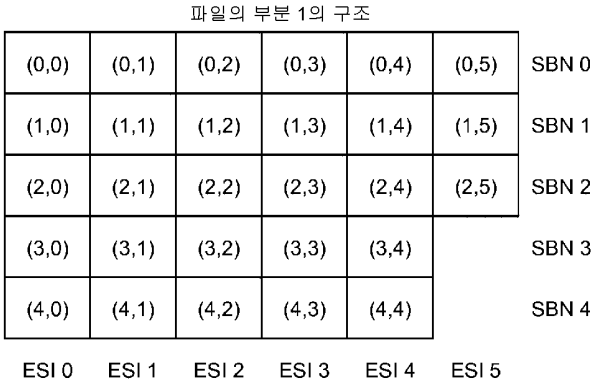
도면6



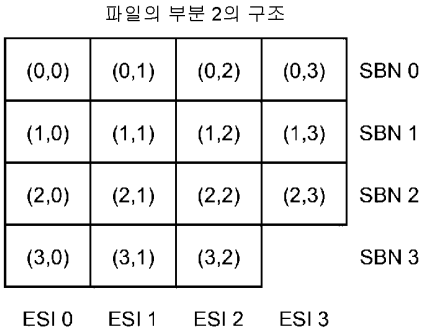
도면7


FEC OTI
(d, Al, F0, T0, Z0, N0, F1, T1, Z1, N1, ..., Fd-1, Td-1, Zd-1, Nd-1) 결정
700
더 많은 패킷들 수신?
710
아니오
예
FEC 페이로드 ID = UFSI C인 패킷 수신 각각의 i = 0, ..., d-1에 대하여: 패킷으로부터 크기 Ti 의 인코딩된 심볼을 추출
730
720
파일의 충분한 부분들이 복구된 경우에 정지; 그렇지 않으면, 아직 복구되지 않은 파일의 부분을 복구하기 위하여 다른 기술들을 이용
각각의 i = 0, ..., d-1에 대하여: Ai = C modulo Zi 연산, Bi = floor (C/Zi) 연산
740
각각의 i = 0, ..., d-1에 대하여: 크기 Ti 의 인코딩된 심볼에 대응하는 수신된 패킷의 부분 i에 대하여 (SBN, ESI) =(Ai, Bi) 설정
750
각각의 i = 0, ..., d-1에 대하여: 부분 i에 대한 인코딩된 심볼 및 파일의 부분 i의 복구를 위해 이용될 연관된 (SBN, ESI) =(Ai, Bi)저장
760
파일의 부분 i를 복구하기 위해 충분한 인코딩된 심볼들 수신?
770
아니오
예
780
저장된 (SBN, ESI들) 및 부분 i에 대한 인코딩된 심볼들에 기반으로 하여 현존하는 기술들을 이용하여 부분 i 복구

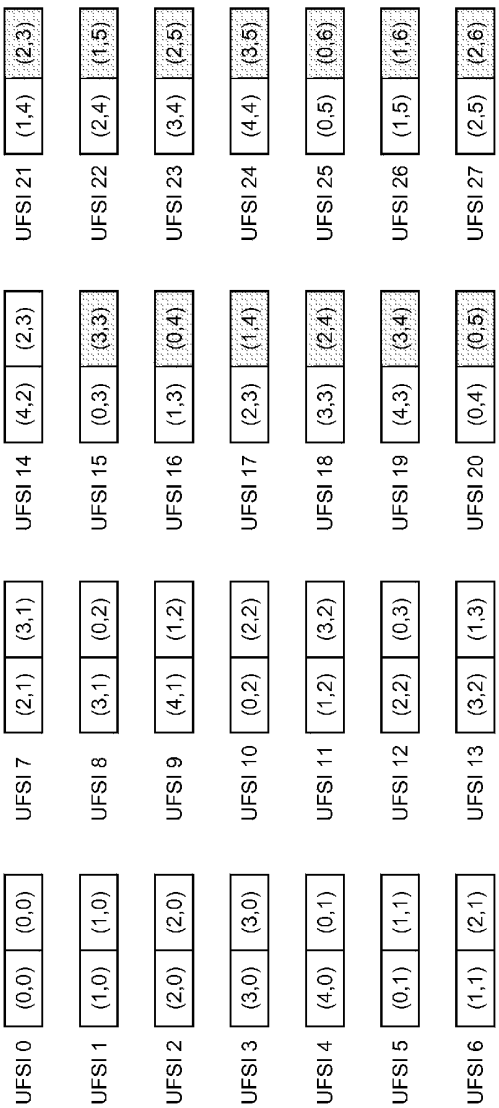
도면8a


파일의 부분 1의 구조
(0,0) (0,1) (0,2) (0,3) (0,4) (0,5) SBN 0
(1,0) (1,1) (1,2) (1,3) (1,4) (1,5) SBN 1
(2,0) (2,1) (2,2) (2,3) (2,4) (2,5) SBN 2
(3,0) (3,1) (3,2) (3,3) (3,4) SBN 3
(4,0) (4,1) (4,2) (4,3) (4,4) SBN 4
ESI 0 ESI 1 ESI 2 ESI 3 ESI 4 ESI 5

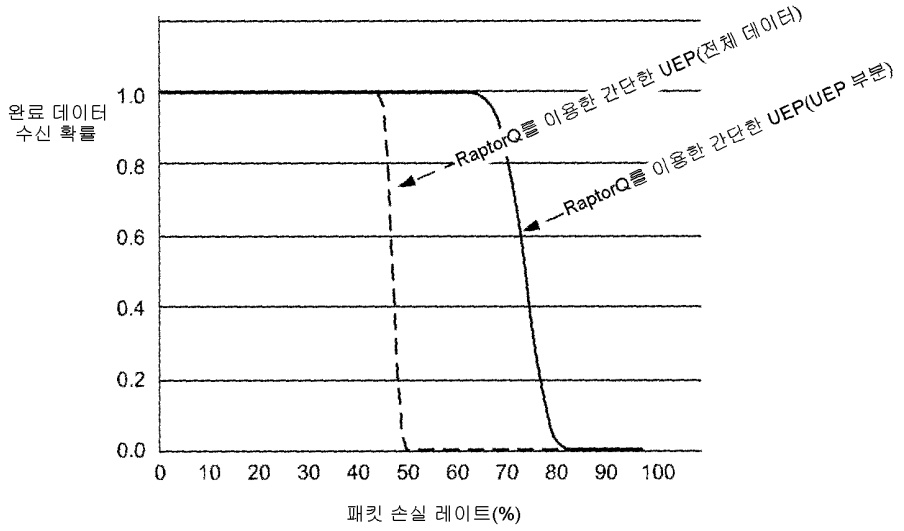
도면8b


파일의 부분 2의 구조
(0,0) (0,1) (0,2) (0,3) SBN 0
(1,0) (1,1) (1,2) (1,3) SBN 1
(2,0) (2,1) (2,2) (2,3) SBN 2
(3,0) (3,1) (3,2) SBN 3
ESI 0 ESI 1 ESI 2 ESI 3

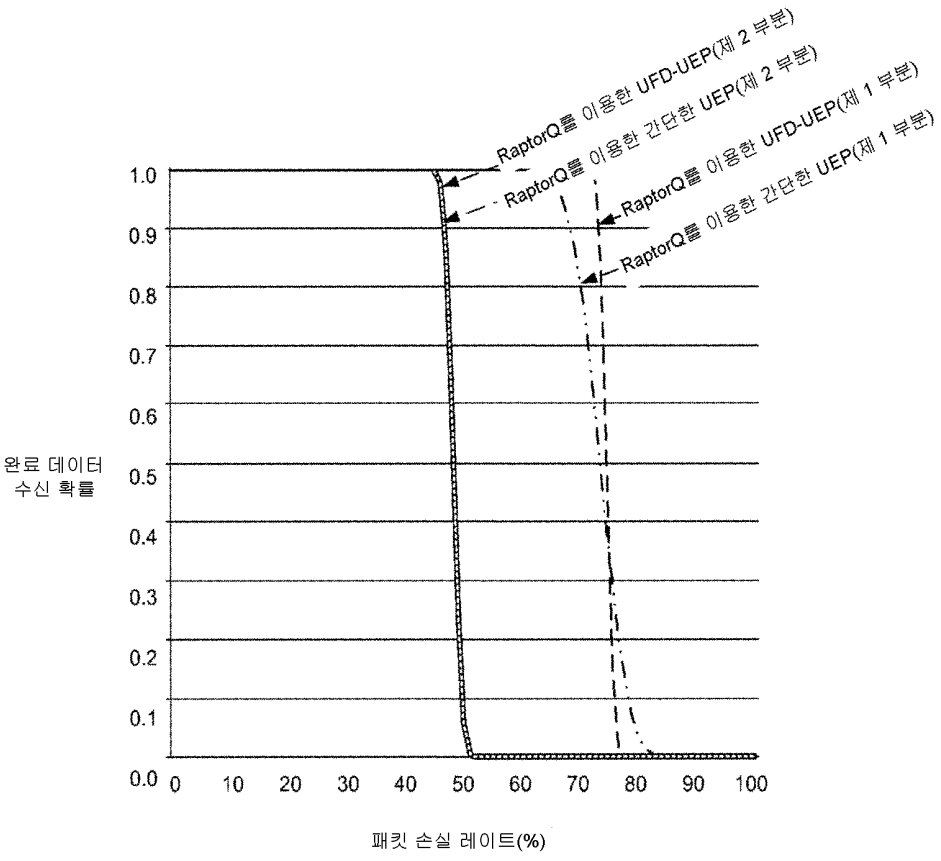
도면9


UFSI 0
UFSI 1
UFSI 2
UFSI 3
UFSI 4
UFSI 5
UFSI 6
UFSI 7
UFSI 8
UFSI 9
UFSI 10
UFSI 11
UFSI 12
UFSI 13
UFSI 14
UFSI 15
UFSI 16
UFSI 17
UFSI 18
UFSI 19
UFSI 20
UFSI 21
UFSI 22
UFSI 23
UFSI 24
UFSI 25
UFSI 26
UFSI 27
T_1
T_2
(0,0) (0,0)
(1,0) (1,0)
(2,0) (2,0)
(3,0) (3,0)
(4,0) (0,1)
(0,1) (1,1)
(1,1) (2,1)
(2,1) (3,1)
(3,1) (0,2)
(4,1) (1,2)
(0,2) (2,2)
(1,2) (3,2)
(2,2) (0,3)
(3,2) (1,3)
(4,2) (2,3)
(0,3) (3,3)
(1,3) (0,4)
(2,3) (1,4)
(3,3) (2,4)
(4,3) (3,4)
(0,4) (0,5)
(1,4) (2,3)
(2,4) (1,5)
(3,4) (2,5)
(4,4) (3,5)
(0,5) (0,6)
(1,5) (1,6)
(2,5) (2,6)

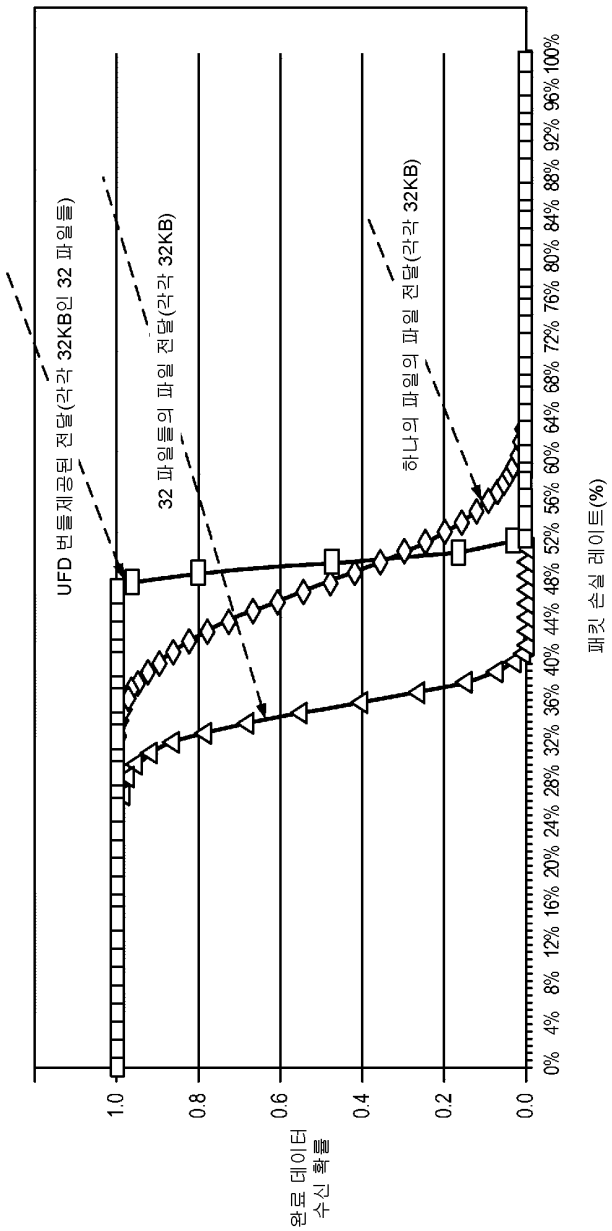
도면10


완료 데이터
수신 확률
1.0
0.8
0.6
0.4
0.2
0.0
0 10 20 30 40 50 60 70 80 90 100
패킷 손실 레이트(%)
RaptorQ를 이용한 간단한 UEP(전체 데이터)
RaptorQ를 이용한 간단한 UEP(UEP 부분)

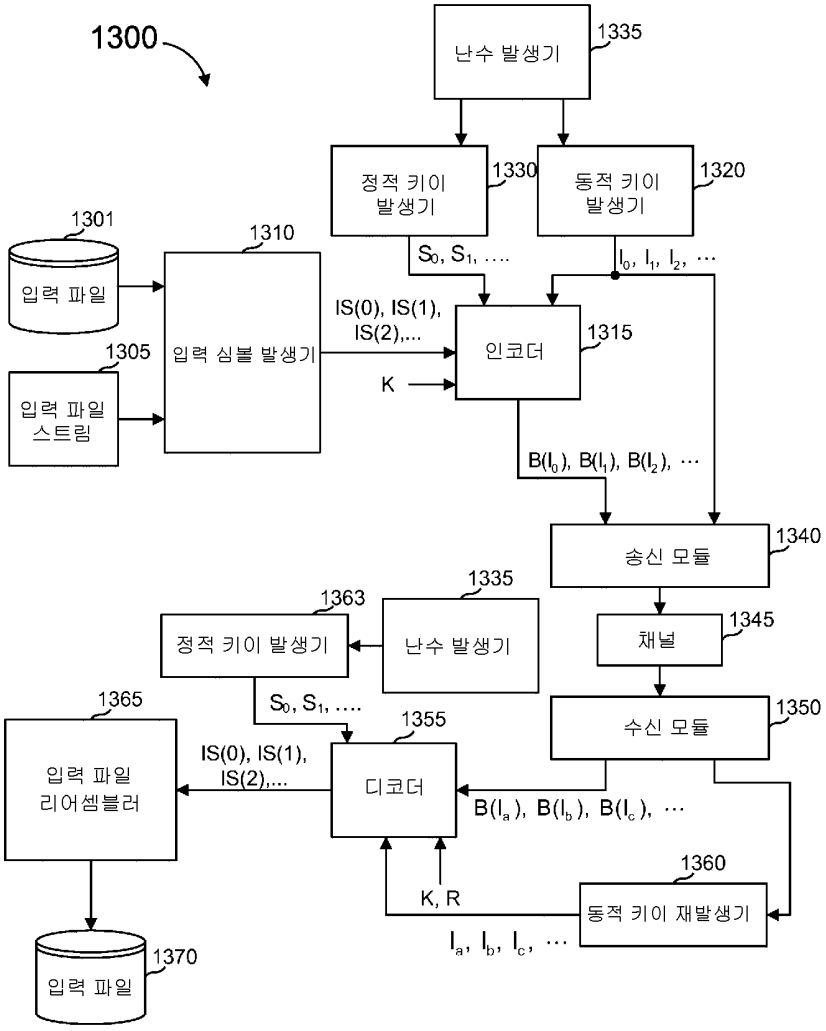
도면11


완료 데이터
수신 확률
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0 10 20 30 40 50 60 70 80 90 100
패킷 손실 레이트(%)
RaptorQ를 이용한 UFD-UEP(제 2 부분)
RaptorQ를 이용한 간단한 UEP(제 2 부분)
RaptorQ를 이용한 UFD-UEP(제 1 부분)
RaptorQ를 이용한 간단한 UEP(제 1 부분)

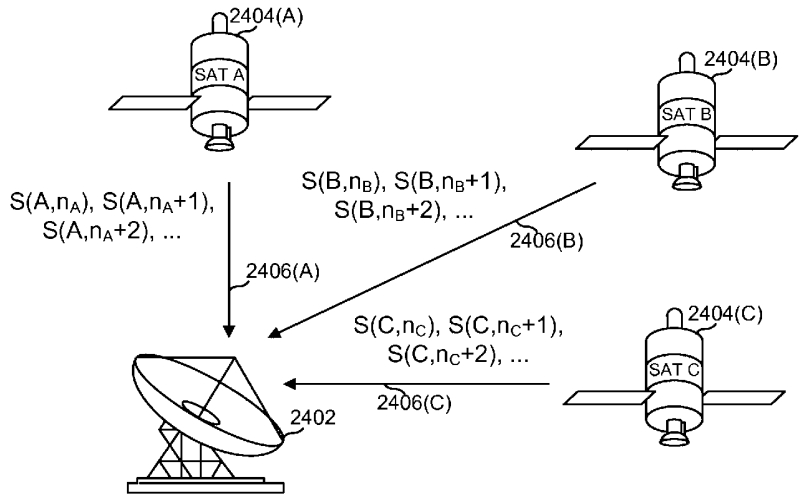
도면12


UFD 번들제공된 전달(각각 32KB인 32 파일들)
32 파일들의 파일 전달(각각 32KB)
하나의 파일의 파일 전달(각각 32KB)
완료 데이터 수신 확률
1.0
0.8
0.6
0.4
0.2
0.0
패킷 손실 레이트(%)
0% 4% 8% 12% 16% 20% 24% 28% 32% 36% 40% 44% 48% 52% 56% 60% 64% 68% 72% 76% 80% 84% 88% 92% 96% 100%

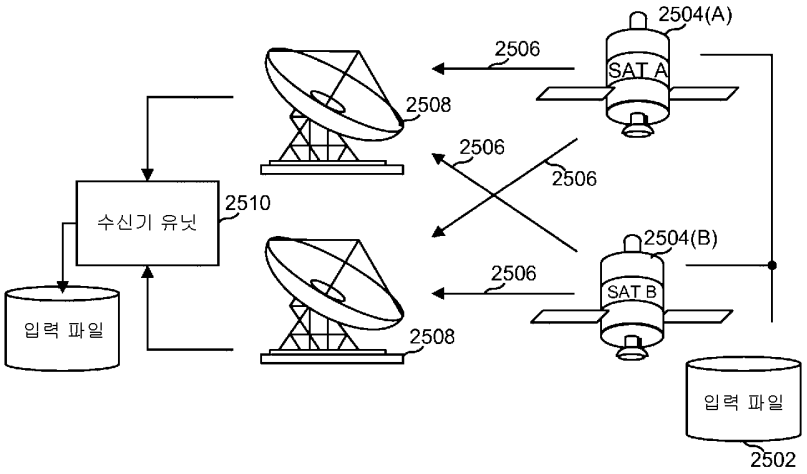
도면13


1300
난수 발생기
1335
정적 키이 발생기
1330
동적 키이 발생기
1320
1301
입력 파일
1310
입력 심볼 발생기
1305
입력 파일 스트림
$S_0, S_1, \ldots$
$I_0, I_1, I_2, \ldots$
IS(0), IS(1), IS(2),...
인코더
1315
K
$B(I_0), B(I_1), B(I_2), \cdots$
송신 모듈
1340
1335
난수 발생기
1363
정적 키이 발생기
채널
1345
수신 모듈
1350
1365
입력 파일 리어셈블러
1355
디코더
$B(I_a), B(I_b), B(I_c), \cdots$
1360
동적 키이 재발생기
K, R
$I_a, I_b, I_c, \cdots$
1370
입력 파일

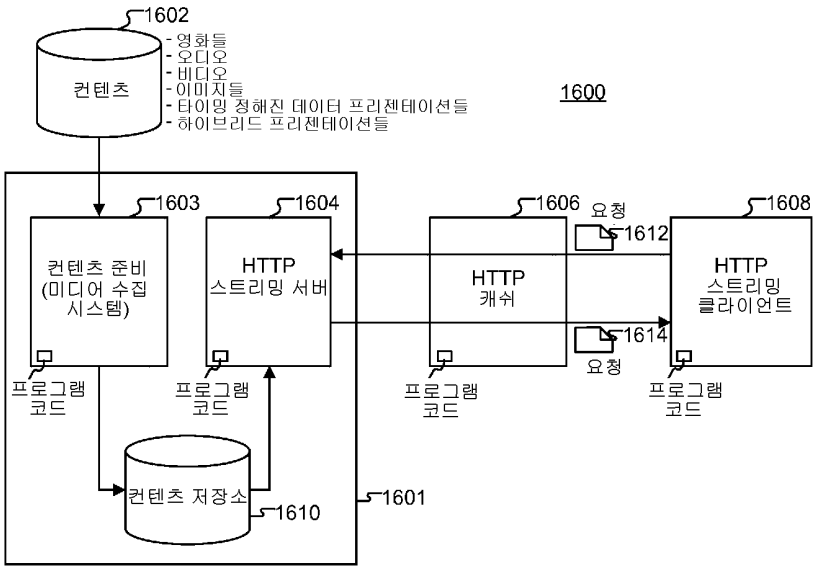
도면14


2404(A)
SAT A
2404(B)
SAT B
$S(A,n_A), S(A,n_A+1), S(A,n_A+2), \ldots$
2406(A)
$S(B,n_B), S(B,n_B+1), S(B,n_B+2), \ldots$
2406(B)
$S(C,n_C), S(C,n_C+1), S(C,n_C+2), \ldots$
2406(C)
2404(C)
SAT C
2402

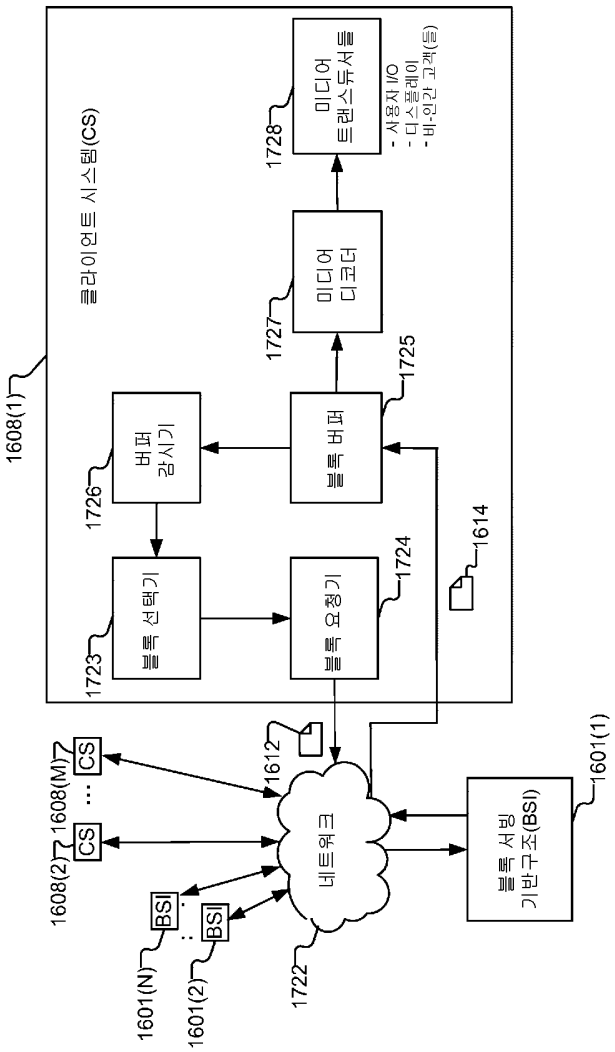
도면15


2504(A)
2506
SAT A
2508
2506
2506
수신기 유닛
2510
2504(B)
2506
SAT B
입력 파일
2508
입력 파일
2502

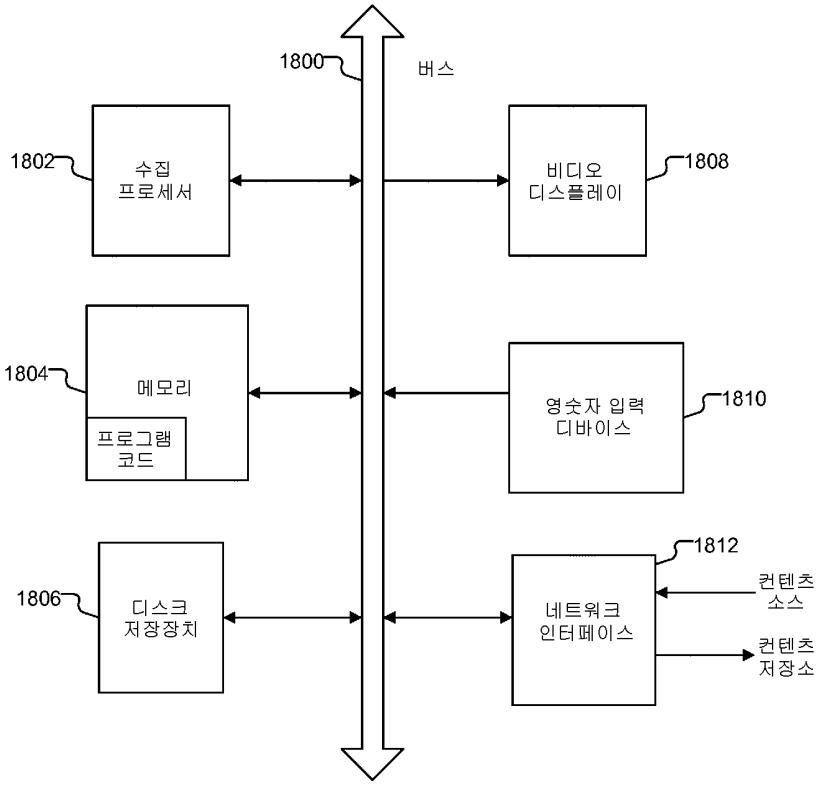
도면16


1602
- 영화들
- 오디오
- 비디오
- 이미지들
- 타이밍 정해진 데이터 프리젠테이션들
- 하이브리드 프리젠테이션들
컨텐츠
1600
1603
1604
1606
요청
1612
1608
컨텐츠 준비
(미디어 수집
시스템)
HTTP
스트리밍 서버
HTTP
캐쉬
HTTP
스트리밍
클라이언트
1614
요청
프로그램
코드
프로그램
코드
프로그램
코드
프로그램
코드
컨텐츠 저장소
1610
1601

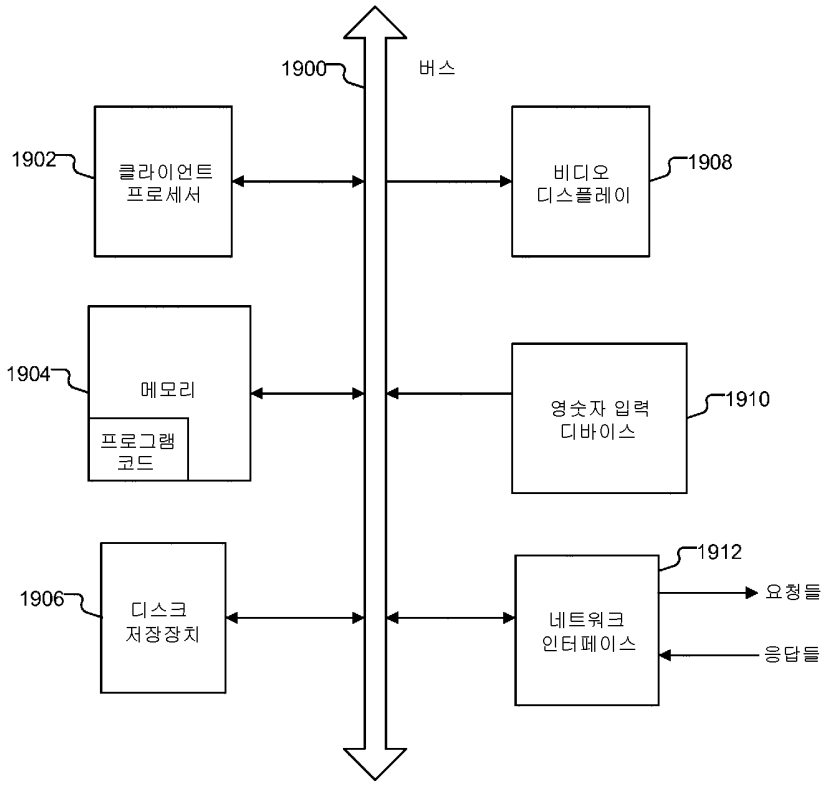
도면17


1608(1)
클라이언트 시스템(CS)
1728
미디어 트랜스듀서들
- 사용자 I/O
- 디스플레이
- 비-인간 고객(들)
1727
미디어 디코더
1725
블록 버퍼
1726
버퍼 감시기
1723
블록 선택기
1724
블록 요청기
1614
1612
네트워크
1722
1608(M)
CS
1608(2)
CS
BSI
1601(N)
BSI
1601(2)
블록 서빙 기반구조(BSI)
1601(1)

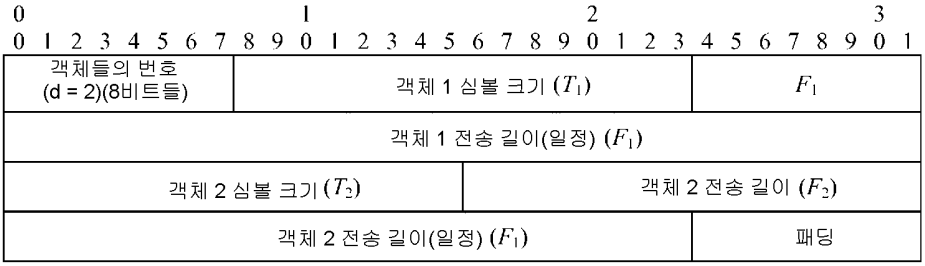
도면18


1800
버스
1802
수집
프로세서
비디오
디스플레이
1808
1804
메모리
프로그램
코드
영숫자 입력
디바이스
1810
1806
디스크
저장장치
1812
네트워크
인터페이스
컨텐츠
소스
컨텐츠
저장소

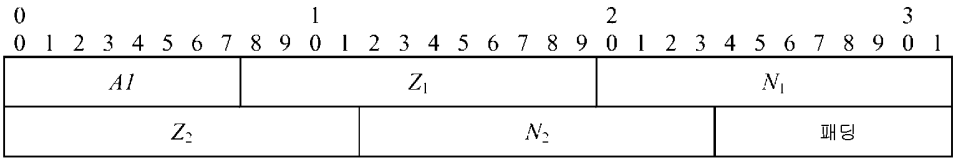
도면19


1900
버스
1902
클라이언트
프로세서
비디오
디스플레이
1908
1904
메모리
프로그램
코드
영숫자 입력
디바이스
1910
1906
디스크
저장장치
1912
네트워크
인터페이스
요청들
응답들

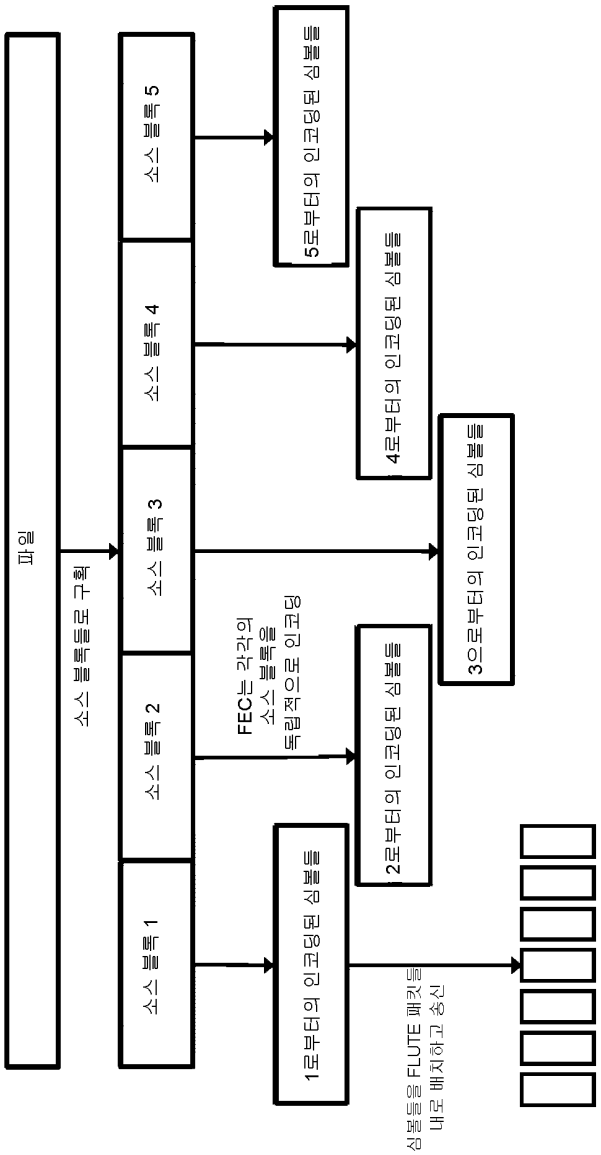
도면20


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
객체들의 번호 (d = 2)(8비트들)
객체 1 심볼 크기 (T_1)
F_1
객체 1 전송 길이(일정) (F_1)
객체 2 심볼 크기 (T_2)
객체 2 전송 길이 (F_2)
객체 2 전송 길이(일정) (F_1)
패딩

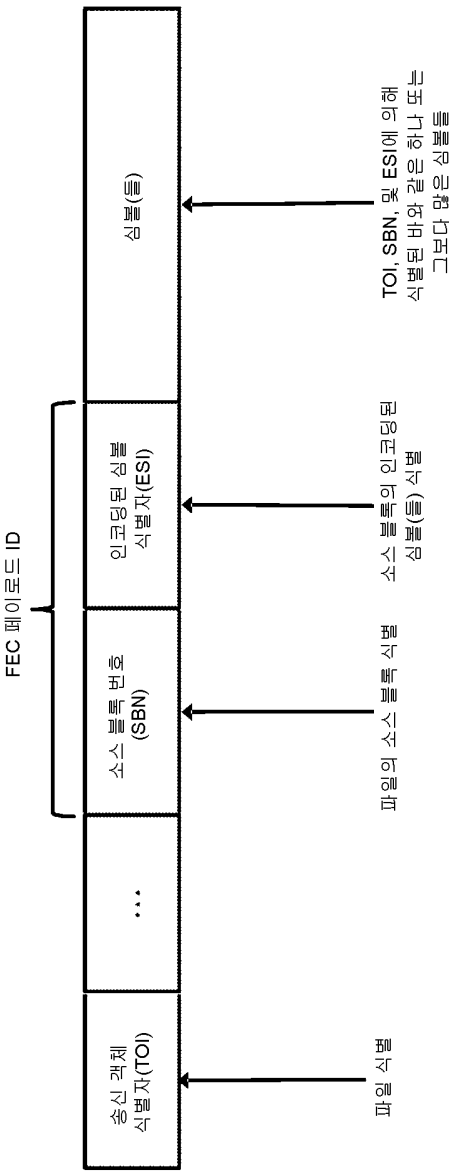
도면21


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
$A1$
Z_1
N_1
Z_2
N_2
패딩

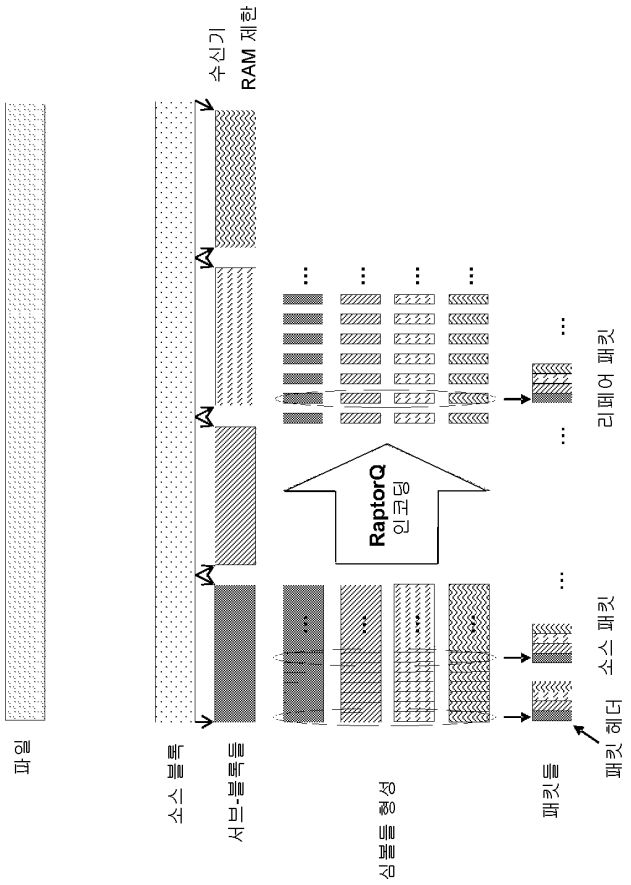
도면22


파일
소스 블록들로 구획
소스 블록 1
소스 블록 2
소스 블록 3
소스 블록 4
소스 블록 5
FEC는 각각의
소스 블록을
독립적으로 인코딩
1로부터의 인코딩된 심볼들
2로부터의 인코딩된 심볼들
3으로부터의 인코딩된 심볼들
4로부터의 인코딩된 심볼들
5로부터의 인코딩된 심볼들
심볼들을 FLUTE 패킷들
내로 배치하고 송신

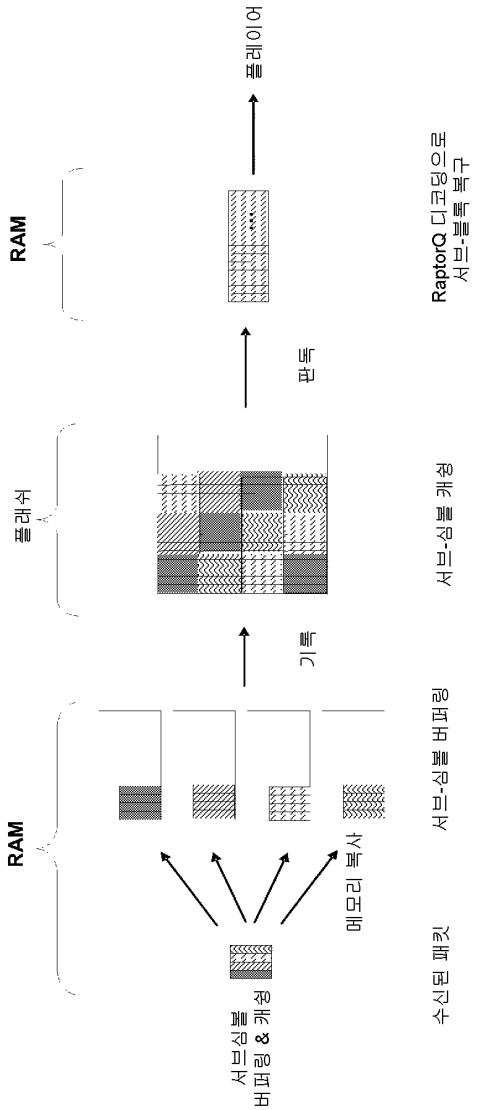
도면23


FEC 페이로드 ID
송신 객체 식별자(TOI)
...
소스 블록 번호 (SBN)
인코딩된 심볼 식별자(ESI)
심볼(들)
파일 식별
파일의 소스 블록 식별
소스 블록의 인코딩된 심볼(들) 식별
TOI, SBN, 및 ESI에 의해 식별된 바와 같은 하나 또는 그보다 많은 심볼들

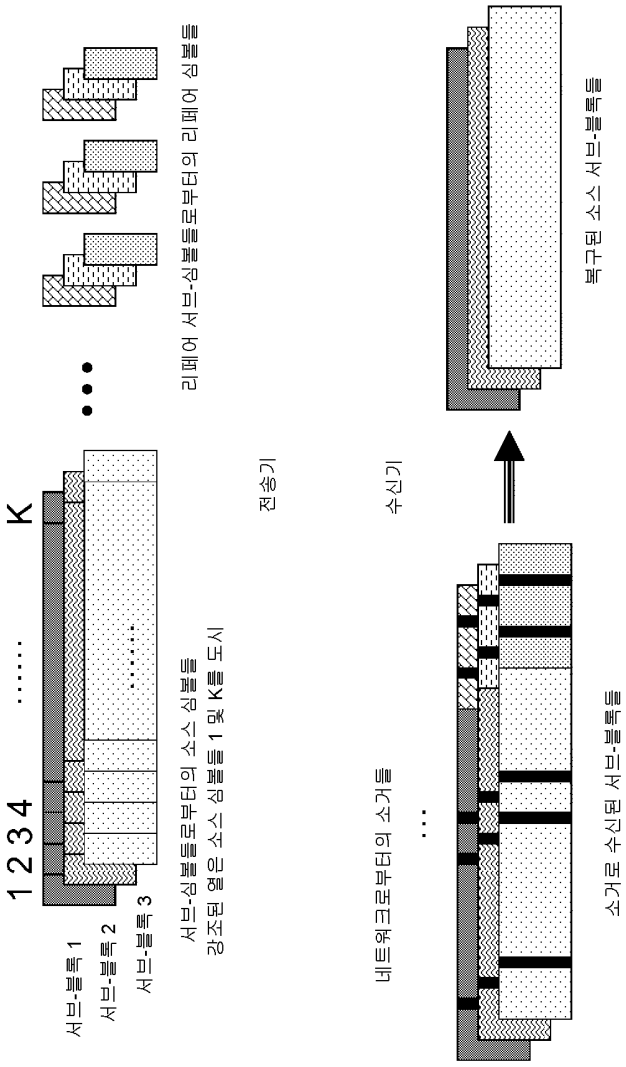
도면24


파일
소스 블록
서브-블록들
수신기
RAM 제한
심볼들 형성
RaptorQ
인코딩
패킷들
패킷 헤더
소스 패킷
리페어 패킷

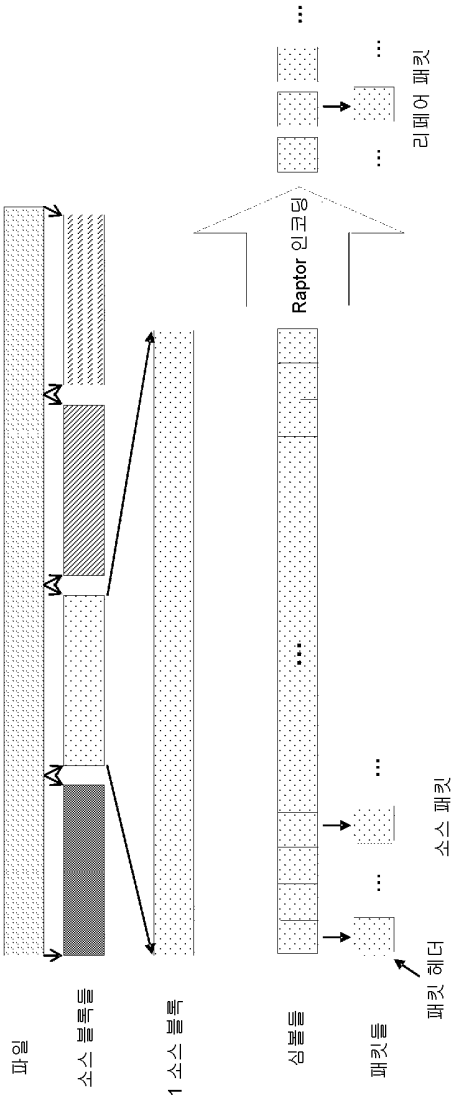
도면25


RAM
서브심볼
버퍼링 & 캐싱
메모리 복사
기록
판독
플래쉬
RAM
플레이어
수신된 패킷
서브-심볼 버퍼링
서브-심볼 캐싱
RaptorQ 디코딩으로
서브-블록 복구

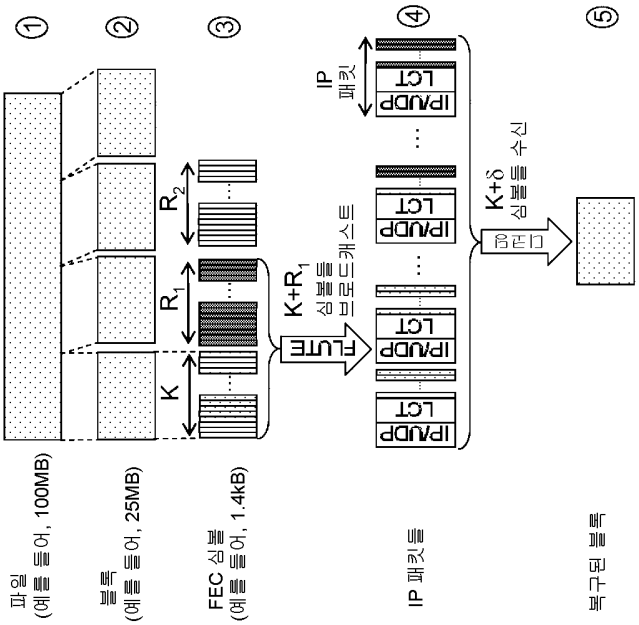
도면26


1 2 3 4 ..... K
서브-블록 1
서브-블록 2
서브-블록 3
서브-심볼들로부터의 소스 심볼들
강조된 영은 소스 심볼들 1 및 K를 도시
전송기
리페어 서브-심볼들로부터의 리페어 심볼들
네트워크로부터의 소거들
수신기
소거로 수신된 서브-블록들
복구된 소스 서브-블록들

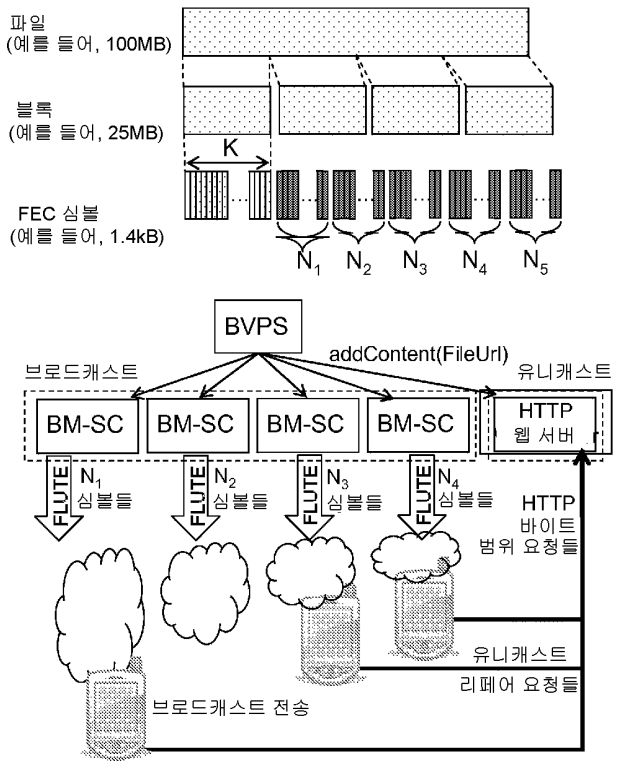
도면27



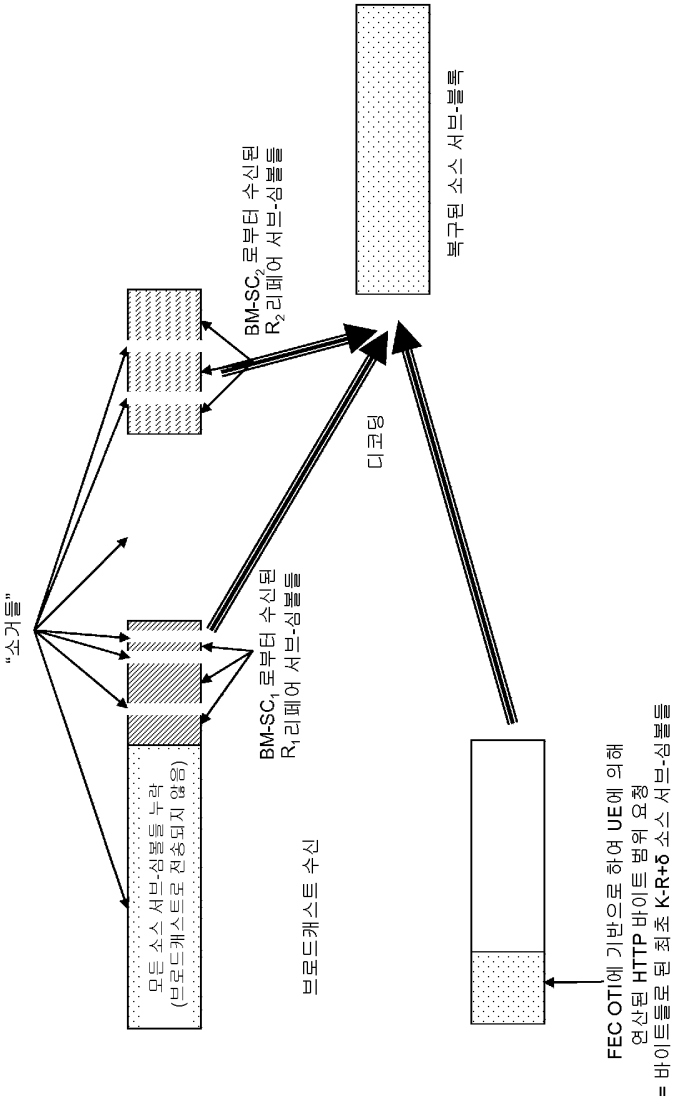
도면28


①
②
③
④
⑤
파일
(예를 들어, 100MB)
블록
(예를 들어, 25MB)
FEC 심볼
(예를 들어, 1.4kB)
IP 패킷들
복구된 블록
K
R1
R2
K+R1
심볼들
브로드캐스트
FLUTE
IP 패킷
IP/UDP
LCT
K+δ
심볼들 수신
디코딩

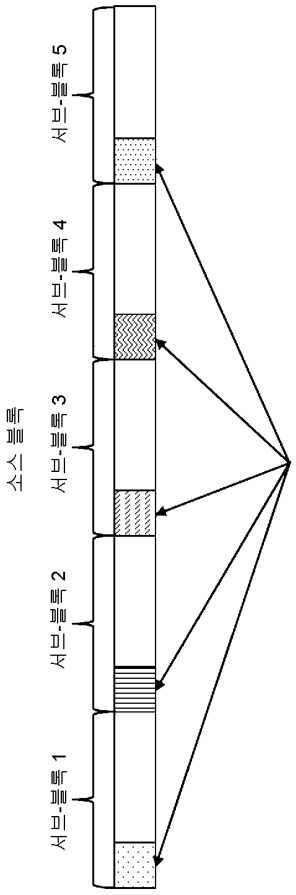
도면29


파일
(예를 들어, 100MB)
블록
(예를 들어, 25MB)
K
FEC 심볼
(예를 들어, 1.4kB)
N1
N2
N3
N4
N5
BVPS
addContent(FileUrl)
브로드캐스트
유니캐스트
BM-SC
BM-SC
BM-SC
BM-SC
HTTP
웹 서버
FLUTE
N1
심볼들
FLUTE
N2
심볼들
FLUTE
N3
심볼들
FLUTE
N4
심볼들
HTTP
바이트
범위 요청들
유니캐스트
리페어 요청들
브로드캐스트 전송

도면30


"소거들"
모든 소스 서브-심볼들 누락
(브로드캐스트로 전송되지 않음)
BM-SC1 로부터 수신된
R1 리페어 서브-심볼들
BM-SC2 로부터 수신된
R2 리페어 서브-심볼들
브로드캐스트 수신
디코딩
복구된 소스 서브-블록
FEC OTI에 기반으로 하여 UE에 의해
연산된 HTTP 바이트 범위 요청
= 바이트들로 된 최초 K·R+δ 소스 서브-심볼들

도면31



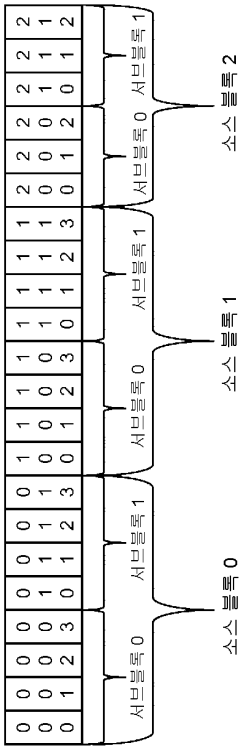
도면32

서브-블록 번호 (SuBN)	최초 ESI	시작 바이트(공식)	시작 바이트 (값)	최종 ESI	종료 바이트(공식)	종료 바이트(값)
0	12	12 * 112	1,344	29	1,344 + 18 * 112 - 1	3,359
1	12	7,576 * 12 + 12 * 112	849,856	29	849,856 + 18 * 112 - 1	851,871
2	12	7,576 * 2 * 112 + 12 * 112	1,698,368	29	1,698,368 + 18 * 112 - 1	1,700,383
3	12	7,576 * 3 * 112 + 12 * 112	2,546,880	29	2,546,880 + 18 * 112 - 1	2,548,895
4	12	7,576 * 4 * 112 + 12 * 112	3,395,392	29	3,395,392 + 18 * 112 - 1	3,397,407
5	12	7,576 * 5 * 112 + 12 * 112	4,243,904	29	4,243,904 + 18 * 112 - 1	4,245,919
6	12	7,576 * 6 * 112 + 12 * 108	5,092,368	29	5,092,368 + 18 * 108 - 1	5,094,311
7	12	7,576 * (6 * 112 + 108) + 12 * 108	5,910,576	29	5,910,576 + 18 * 108 - 1	5,912,519
8	12	7,576 * (6 * 112 + 2 * 108) + 12 * 108	6,728,784	29	6,728,784 + 18 * 108 - 1	6,730,727
9	12	7,576 * (6 * 112 + 3 * 108) + 12 * 108	7,546,992	29	7,546,992 + 18 * 108 - 1	7,548,935
10	12	7,576 * (6 * 112 + 4 * 108) + 12 * 108	8,365,200	29	8,365,200 + 18 * 108 - 1	8,367,143
11	12	7,576 * (6 * 112 + 5 * 108) + 12 * 108	9,183,408	29	9,183,408 + 18 * 108 - 1	9,185,351

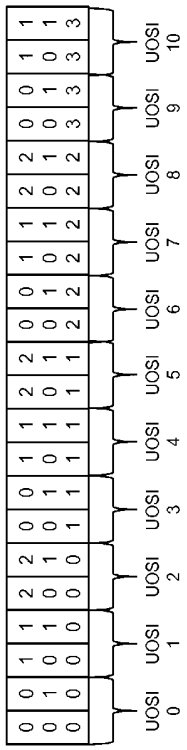
도면33

서브-블록 번호 (SuBN)	최초 ESI	시작 바이트(공식)	시작 바이트 (값)	최종 ESI	종료 바이트(공식)	종료 바이트(값)
0	12	7,576 * 1320 + 12 * 112	10,001,664	29	10,001,664 + 18 * 112 - 1	10,003,679
1	12	7,576 * (1320 + 112) + 12 * 112	10,850,176	29	10,850,176 + 18 * 112 - 1	10,852,191
2	12	7,576 * (1320 + 2 * 112) + 12 * 112	11,698,688	29	11,698,688 + 18 * 112 - 1	11,700,703
3	12	7,576 * (1320 + 3 * 112) + 12 * 112	12,547,200	29	12,547,200 + 18 * 112 - 1	12,549,215
4	12	7,576 * (1320 + 4 * 112) + 12 * 112	13,395,712	29	13,395,712 + 18 * 112 - 1	13,397,727
5	12	7,576 * (1320 + 5 * 112) + 12 * 112	14,244,224	29	14,244,224 + 18 * 112 - 1	14,246,239
6	12	7,576 * (1320 + 6 * 112) + 12 * 108	15,092,688	29	15,092,688 + 18 * 108 - 1	15,094,631
7	12	7,576 * (1320 + 6 * 112 + 108) + 12 * 108	15,910,896	29	15,910,896 + 18 * 108 - 1	15,912,839
8	12	7,576 * (1320 + 6 * 112 + 2 * 108) + 12 * 108	16,729,104	29	16,729,104 + 18 * 108 - 1	16,731,047
9	12	7,576 * (1320 + 6 * 112 + 3 * 108) + 12 * 108	17,547,312	29	17,547,312 + 18 * 108 - 1	17,549,255
10	12	7,576 * (1320 + 6 * 112 + 4 * 108) + 12 * 108	18,365,520	29	18,365,520 + 18 * 108 - 1	18,367,463
11	12	7,576 * (1320 + 6 * 112 + 5 * 108) + 12 * 108	19,183,728	29	19,183,728 + 18 * 108 - 1	19,185,671

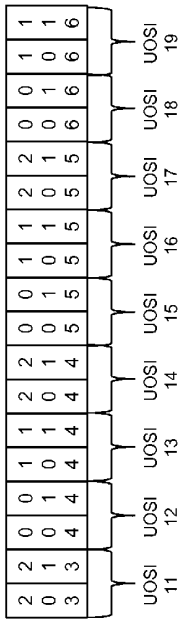
도면34


서브블록 0
서브블록 1
서브블록 0
서브블록 1
서브블록 0
서브블록 1
소스 블록 0
소스 블록 1
소스 블록 2

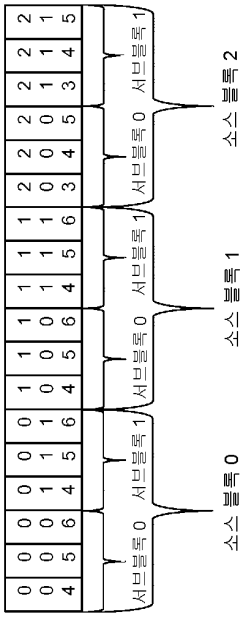
도면35


UOSI 0
UOSI 1
UOSI 2
UOSI 3
UOSI 4
UOSI 5
UOSI 6
UOSI 7
UOSI 8
UOSI 9
UOSI 10

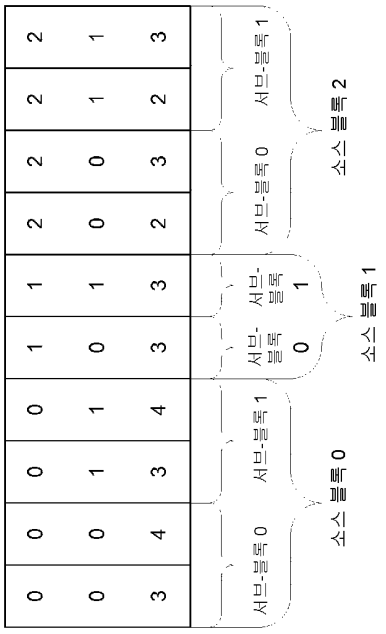
도면36


UOSI 11
UOSI 12
UOSI 13
UOSI 14
UOSI 15
UOSI 16
UOSI 17
UOSI 18
UOSI 19

도면37


서브블록 0
서브블록 1
서브블록 0
서브블록 1
서브블록 0
서브블록 1
소스 블록 0
소스 블록 1
소스 블록 2

도면38


0 0 3
0 0 4
0 0 3
0 1 4
1 0 3
1 1 3
2 0 2
2 0 3
2 1 2
2 1 3
서브-블록 0
서브-블록 1
서브-블록 0
서브-블록 1
서브-블록 0
서브-블록 1
소스 블록 0
소스 블록 1
소스 블록 2

도면39

서브-블록 번호 (SuBN)	최초 ESI	시작 바이트(공식)	시작 바이트(값)	최종 ESI	종료 바이트(공식)	종료 바이트(값)
5	12	(5 * 758 + 12) * 1320	5,018,640	29	5,018,640 + 1320 * 18 - 1	5,042,399
19	27	(12 * 758 + 7 * 757 + 27) * 1320	19,037,040	36	19,037,040 + 1320 * 10 - 1	19,050,239

도면40

서브-블록 번호 (SuBN)	최초 ESI	시작 바이트(공식)	시작 바이트(값)	최종 ESI	종료 바이트(공식)	종료 바이트(값)
5	758	5 * 758 * 1320	5,002,800	775	5,002,800 + 1320 * 18 - 1	5,026,559
19	757	(12 * 758 + 7 * 757) * 1320	19,001,400	766	19,001,400 + 1320 * 10 - 1	19,014,599

도면41

서브블록 번호 (SuBN)	최초 ESI	시작 바이트(공식)	시작 바이트 (값)	최종 ESI	종료 바이트(공식)	종료 바이트 (값)
0	7576	0	0	7593	0 + 18 * 112 - 1	2,015
1	7576	7,576 * 112	848,512	7593	848,512 + 18 * 112 - 1	850,527
2	7576	7,576 * 2 * 112	1,697,204	7593	1,697,204 + 18 * 112 - 1	1,699,219
3	7576	7,576 * 3 * 112	2,545,536	7593	2,545,536 + 18 * 112 - 1	2,547,551
4	7576	7,576 * 4 * 112	3,394,048	7593	3,394,048 + 18 * 112 - 1	3,396,063
5	7576	7,576 * 5 * 112	4,242,560	7593	4,242,560 + 18 * 112 - 1	4,244,575
6	7576	7,576 * 6 * 112	5,091,072	7593	5,091,072 + 18 * 108 - 1	5,093,015
7	7576	7,576 * (6 * 112 + 108)	5,909,280	7593	5,909,280 + 18 * 108 - 1	5,911,223
8	7576	7,576 * (6 * 112 + 2 * 108)	6,727,488	7593	6,727,488 + 18 * 108 - 1	6,729,431
9	7576	7,576 * (6 * 112 + 3 * 108)	7,545,696	7593	7,545,696 + 18 * 108 - 1	7,547,639
10	7576	7,576 * (6 * 112 + 4 * 108)	8,363,904	7593	8,363,904 + 18 * 108 - 1	8,365,847
11	7576	7,576 * (6 * 112 + 5 * 108)	9,182,112	7593	9,182,112 + 18 * 108 - 1	9,184,055

【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 제65항

【변경전】

상기 HTTP 서버는

【변경후】

HTTP 서버는