분산 P2P 시스템을 활용한 안전한 파일 분산 방안

김진홍* · 김선영* · 이윤진* · 조인준*

*배재대학교 컴퓨터공학과

Secure file distribution method using distribution P2P system

Jin-Hong Kim* · Seon-Young Kim* · Yoon-Jin Lee* · In-June Jo*

*Dept. of Computer Engineering, Paichai Univ.

E-mail : {jhkim@mail.pcu.ac.kr, ksy2443@msn.com, gomyung@mail.pcu.ac.kr, injune@mail.pcu.ac.kr}

요 약

최근 들어 클라이언트/서버 컴퓨팅 환경에서 서버의 과도한 부하 문제를 해결하고자 P2P 컴퓨팅 환경이 대두되고 있다. 현재 실용화 되어 운영중인 P2P 컴퓨팅 환경은 주로 파일 출판자/송신자/수신자의 실명기반 혹은 익명기반의 P2P 시스템으로 전개되고 있다. 하지만, 어떤 환경에서건 현재의 파일 보호는 파일 단위 기반의 보안기술이 적용됨에 따라 첫째, 적의 공격목표가 특정 단일 호스트라는 취약점을 지니게 되고, 둘째, 취득된 파일에 대해 Brute Force 공격이 용이하고, 셋째, DOS공격 목표를 명료하게 하는 등의 문제점을 지니고 있다.

본 논문에서는 이러한 문제점 해결책으로 파일을 블록단위로 분리하고 각 블록을 피어들에게 안전하게 분산시켜 이를 활용하는 분산 P2P 파일분리 시스템을 새롭게 제안하였다. 제안 시스템은 파일이 블록 단위로 인코딩되어 각 피어로 분산됨에 따라 상기의 3가지 문제점을 해결할 뿐만 아니라 파일 활용의 효율성을 증진시키는 부대 효과를 얻을 수 있다.

ABSTRACT

Recently, the computing environment of P2P come out to solve the excessive load of the server in the computing environment of the client/server. Currently, operated computing environment of P2P is mainly spreading out P2P system of read name or anonymity base about a publisher, sender and receiver of the file. But, to the current file protection there is three problem. The first problem is to a host become attack target. The second is to received file loose attack of Brute Force. The third is to define target of attack of DOS.

To solve the this problem, it divide file into block unit. Each block is safely scattered peers. This paper propose the distribution P2P system of file division. Both proposing system solve the this problem and promote efficiency of file application.

키워드

File distribution, P2P network, File splitting, Small sized terminal

I. 서 론

기존의 시스템은 한 파일이 한 호스트에 저장되어 처리되는 중앙집중형 파일처리 체계이다. 따라서 기하급수적으로 증가되는 파일 때문에 과도한 부하가 증대되고, 공격자에게 분명한 단일의 공격목표를 제공함에 따라 보안상 취약성을 지니게 된다.

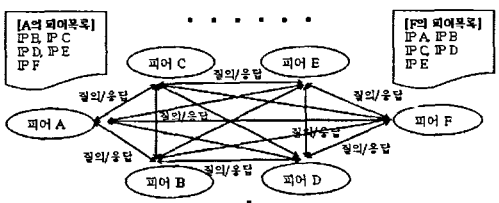
본 논문에서는 파일을 블록단위로 분리하고 이를 분산 P2P시스템에 안전하게 분산시킴으로써 이러한 문제 해결을 위한 새로운 방안을 제안하였다.

II. 제안시스템 구조

제안 시스템은 파일 분산에 동의한 피어들이

네트워크를 통해서 연결된 집합으로 구성된다. 그리고 파일 분산 및 재구성 기능의 효율성 증대를 위해 그물망 위상의 오버레이 네트워크 구조를 이룬다. 따라서 각 노드는 파일 분산에 참여하는 모든 피어들의 IP주소 목록을 저장하는 데이터 구조체를 가진다(그림1). 그리고 분산된 파일을 저장하는 디스크 영역이 할당된다.

각 구성요소의 기능 및 역할을 정리하면 다음과 같다. 각 피어의 역할은 첫째, 파일을 분리하여 분배하는 역할을 행한다. 이때 다음 장에서 설명하는 인코딩 법칙에 따라 파일을 블록으로 분리하고 분리된 블록을 랜덤 방식으로 선택된 피어에게 분산 시키는 기능을 한다. 둘째, 사용자가 요구한 파일 탐색기능을 한다. 이러한 파일 탐색 요구를 받은 피어는 자신의 루트블럭 저장 파일로부터 해당 루트 블럭을 추출하여 제안 시스템에 산재되어 있는 데이터 블록을 질의하여 이에 대한 응답으로 수신된 결과로 원형 파일을 복원한다.



III. 제안 시스템의 인코딩 및 분산 방안

1. 파일 분리 인코딩 및 블록 분산 절차

제안 시스템은 파일을 가변길이 블록으로 인코딩하여 이를 피어간에 전송한다. 이들 블록은 3가지 유형으로 나누었다. 즉 *D(Data)*블럭, *I(Indirection)*블럭, *R(Root)*블록이다. 이들 블록의 생성 방법은 다음과 같다.

(1) *D* 블럭 생성

step1) 파일 F 를 196개 블록으로 분리한다.
$F = \{D_1, D_2 \cdots\cdots D_{196}\}$,
D블럭 길이 = (file size)KB / 196

step2) 분리된 각 블록 D_i 를 160비트 RIPE-MD로 해쉬한다.
$\{H(D_1), H(D_2), \cdots..H(D_{196})\}$

step3) 상기 해쉬 값을 키로 하여 각 D 블록을 암호화하여 최종 D 블록 생성을 완료한다.
$\{E_{H(D1)}(D_1), E_{H(D2)}(D_2), \cdots\cdots, E_{H(Dn196}(D_{196}\}$

(2) *I* 블럭 생성 및 *D* 블럭 분산

step1) 생성된 D 블록을 160 RIPE-MD로 해쉬하여 D 블록의 저장소를 생성한다.
$\{H(E_{H(D1)}(D_1)), H(E_{H(D2)}(D_2)), \cdots\cdots$
$\cdots, H(E_{H(D196)}(D_{196}))\}$

step2) 각 D 블록에 대해 평문 D 블록의 해쉬값과 암호화된 D 블록의 해쉬 값 을 선택하고 이들이 분산될 피어의 IP주소 2개를 자신의 피어 목록에서 선택하여 *D-blocks-idx*를 생성한다.

D-blocks-idx =
$\{(H(D_1), ip\text{-}add1(D_1), ip\text{-}add2(D_1), H(E_{H(D1)}(D_1))),$
$(H(D_2), ip\text{-}add1(D_2), ip\text{-}add2(D_2), H(E_{H(D2)}(D_2)))\cdots$
$\cdots(H(D_{196}), ip\text{-}add1(D_{196}), ip\text{-}add2(D_{196}) H(E_{H(D196)}(D_{196})))\}$

step3) 'step2)'의 196개의 D 블록에 대해 선택된 피어의 IP주소로 해당 D 블록을 전송한다. 이를 수신한 각 피어는 전송 받은 암호화 D 블록을 암호화된 D 블록을 해쉬한 값을 파일이름으로 하여 저장한다.

step4) 'step2'에서 생성된 *D-blocks-idx* 를 구성하는 196개의 쌍을 대상으로 한 14개의 쌍을 한 묶음으로 14개 그룹을 생성한다.

D-blocks-idx1 =
$\{(H(D_1), ip\text{-}add1(D_1), ip\text{-}add2(D_1), H(E_{H(D1)}(D_1))),$
$(H(D_2), ip\text{-}add1(D_2), ip\text{-}add2(D_2), H(E_{H(D2)}(D_2)))\cdots$
$\cdots(H(D_{14}), ip\text{-}add1(D_{14}), ip\text{-}add2(D_{14}) H(E_{H(D14)}(D_{14})))\}$
.....

D-blocks-idx14 =
$\{(H(D_{183}), ip\text{-}add1(D_{183}), ip\text{-}add2(D_{183}), H(E_{H(D1)}(D_{183}))),$
$(H(D_{184}), ip\text{-}add1(D_{184}), ip\text{-}add2(D_{184}, H(E_{H(D184)}(D_{184})))\cdots$
$\cdots(H(D_{196}), ip\text{-}add1(D_{196}), ip\text{-}add2(D_{196}) H(E_{H(D196)}(D_{196})))\}$

step5) 'step4)'에서 생성된 14개의 *D-blocks-idx1* 의 각각에 대해 해쉬값을 계산하여 다음의 I 블록을 생성한다.

$I_1 = H(D\text{-}blocks\text{-}idx1) \mid D\text{-}blocks\text{-}idx1$
$I_2 = H(D\text{-}blocks\text{-}idx2) \mid D\text{-}blocks\text{-}idx2$
.....
$I_{14} = H(D\text{-}blocks\text{-}idx14) \mid D\text{-}blocks\text{-}idx14$

step6) 생성된 14개의 I 블록 암호화하여 최종 I

블럭을 완성한다.

I 블록 = $\{I_1, I_2, I_3, \cdots\cdots I_{14},\}$

- 각 블록 I_i 를 160비트 RIPE-MD로 해쉬한다.
  $\{H(I_1), H(I_2), \cdots..H(I_{14})\}$
- 상기 해쉬 값을 키로 하여 각 I 블록을 암호화한다.
  $\{E_{H(I1)}(I_1), E_{H(I2)}(I_2), \cdots\cdots E_{H(I14)}(I_{14})\}$

(3) 루트 I 블럭 생성 및 I 블럭 분산

step1) 14개의 암호화 I 블록을 160 RIPE-MD로 해쉬하여 암호화 I 블록의 저장소를 생성한다.
$\{H(E_{H(I1)}(I_1)), H(E_{H(I2)}(I_2)),.. H(E_{H(I14)}(I_{14}))\}$

Step2) 각 I 블록에 대해 평문 I 블록의 해쉬값과 암호화된 I 블록의 해쉬 값 을 선택하고 이들이 분산될 피어의 IP주소 2개를 자신의 피어 목록에서 선택하여 *I-blocks-idx*를 생성한다.

I-blocks-idx =
$\{(H(I_1), ip\text{-}add1(I_1), ip\text{-}add2(I_1), H(E_{H(I1)}(I_1))),$
$(H(I_2), ip\text{-}add1(I_2), ip\text{-}add2(I_2), H(E_{H(I2)}(I_2)))\cdots$
$(H(I_{14}), ip\text{-}add1(I_{14}), ip\text{-}add2(I_{14})$
$H(E_{H(I14)}(I_{14})))\}$

step3) 'step2)'의 14개의 I 블록에 대해 선택된 피어의 IP주소로 해당 I 블록을 전송한다. 이를 수신한 각 피어는 전송 받은 암호화 I 블록을 암호화된 I 블록을 해쉬한 값을 파일이름으로 하여 저장한다.

Step4) 'step2)'에서생성된 *I-blocks-idx* 의 해쉬값을 계산하여 다음의 루트 I 블록을 생성한다.

루트 I = $H(I\text{-}blocks\text{-}idx) \mid I\text{-}blocks\text{-}idx$

Step5) 생성된 루트 I 블록 암호화하여 최종 루트 I 블럭을 완성한다.

- 루트 I 블록을 160비트 RIPE-MD로 해쉬한다.
  $\{H(I)\}$
- 상기 해쉬값을 키로 하여 루트 I 블록을 암호화한다.
  $\{E_{H(루트I블럭)}(루트\ I블럭)\}$

(4) R 블럭 생성 및 루트I 블럭 분산

R 블록의 구성요소는 파일기술 내용, 파일 총 길이 정보, 루트 I 블록을 탐색하기 위한 인덱스(즉, *I-root-idx*)로 구성된다.

step1) 파일기술 내용 및 파일 ID를 생성한다.

Step2) 1개의 암호화 루트 I 블록을 160 RIPE-MD로 해쉬하여 암호화 루트 I 블록의 저장소를 생성한다.
$\{H(E_{H(루트I블럭)}(루트I\ 블록))\}$

Step3) 루트 I블록에 대해 평문 루트 I 블록의 해쉬값과 암호화된 루트 I 블록의 해쉬 값을 선택하고 이들이 분산될 피어의 IP주소 2개를 자신의 피어 목록에서 선택하여 *I-root-idx*를 생성한다.
I-root-idx =
$\{(H(I_{root}), ip\text{-}add1(I_1), ip\text{-}add2(I_1),$
$H(E_{H(Iroot)}(I_{root}))\}$

Step4) 'step3)'의 루트 I 블록에 대해 선택된 피어의 IP주소로 해당 루트 I 블록을 전송한다. 이를 수신한 각 피어는 전송 받은 암호화 루트 I 블록을 암호화된 루트 I 블록을 해쉬한 값을 파일이름으로 하여 저장한다.

Step5) 'step3)'에서생성된 *I-root-idx* 의 해쉬값을 계산하여 다음의 R 블록을 생성한다.
$R = H(I\text{-}root\text{-}idx) \mid I\text{-}root\text{-}idx$

Step6) 생성된 루트 R블록을 암호화하여 최종 루트 R 블럭을 완성한다.

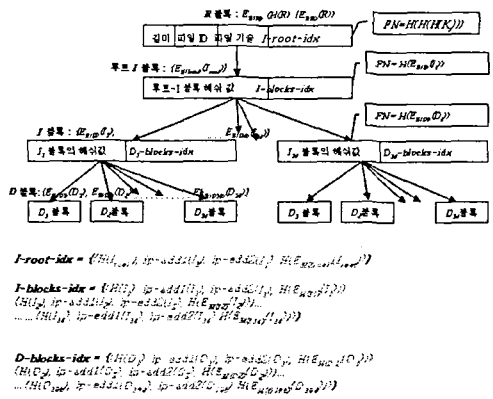
- R 블록을 160비트 RIPE-MD로 해쉬한다.
  $\{H(R)\}$
- 상기 해쉬값을 키로 하여 루트 R 블록을 암호화한다.
  $\{E_{H(R)}(R)\}$

(5) R 블록 저장 및 탐색 키워드

Step1) 기밀 키워드 K_j 를 선택하여 160비트 RIPE-MD로 해쉬하여 다음의 해쉬값을 계산한다.
$H(K_j), H(H(K_j))$

Step2) 첫번째 해쉬 값 $H(K_j)$으로 다음을암호화한다.
$E_{H(Kj)}\ (H(R)\ \mid E_{H(R)}(R))$

Step3) 두번째 해쉬값 $H(H(K_j))$을 파일이름으로 하여 'step 2)'에서 생성된 암호하 R블록을 저장한다.



IV. 결론

본 논문에서는 파일이 한 호스트를 중심으로 저장되고 처리되는 기존의 파일처리체계를 여러 호스트에 파일의 조각을 분리 분산시키는 파일체계를 제안하였다. 제안 시스템은 파일을 블록단위로 분리하고 분리된 블록들을 모아서 원형파일로 복구하는데 그물망 구조의 P2P네트워크를 활용하였다. 이는 지역적으로 분산된 호스트들을 대상으로 분리와 복원의 효율성을 위한 것이다. 이러한 방안을 통해서 한 호스트에 파일의 집중화 문제와 보안상의 취약성 완화를 기대할 수 있다.

참고문헌

[1] CW I, Amsterdam. RIPE Integrity Primitives – Final Report of RACE Integrity Primitives Evaluation (R1040), June 1992.

[2] Hans Dobbertin, Antoon Bosselaers, Bart Preneel "RIPEMD-160: A Strengthened Version of RIPEMD", 1996

[3] I. Clarke, O. Sandberg, B.Wiley, and T. Hong. "Freenet: A Distributed Anonymous Information Storage and Retrieval System", 2001

[4] Dennis Kugler, "An Analysis of GNUnet and the Implications for Anonymous, Censorship-Resistant Networks", 2003

[5] B. Wilcox-O'Hearn, Experiences Deploying a Large-Scale Emergent Network IPTPS02 LNCS 2429

[6] Jianning Yang, "APTPFS: Anonymous Peer-to-Peer File Sharing", April, 2005.

[7] Frank Stajano, "Security for Ubiquitous Computing", Wiley, 2002

[8] 팀 오라이리 외 24인, Peer-to-Peer 차세대 인터넷 P2P, 한빛미디어 출판사, 2001