

튜링완전 암호화폐의 새로운 네트워크 보안 방법: 사용자 책임 시스템

김성일¹⁾

Abstract. 이더리움은 스마트 계약을 사용하기 위해 튜링완전 언어를 처음으로 암호화폐에 적용했고, 이 언어로부터 발생하는 무한루프 공격으로부터 자신의 네트워크를 보호하기 위해 가스 시스템(gas system)을 처음 도입했다. 하지만 이 가스 시스템은 스팸 공격에 취약하고 강한 종량제 특성 때문에 네트워크 사용량을 줄이는 치명적인 약점을 가진다. 이 문제를 해결하기 위해서, 본 논문은 새로운 네트워크 보안방법인 사용자 책임 시스템을 제안한다. 우리 시스템의 핵심은 이더리움의 가스 시스템과 달리 사용자에게 네트워크의 보안책임을 강제로 부과하는 것이다. 구체적으로 말하면, 우리 시스템에서 사용자는 자신의 행위에 책임지기 위해 계약 예치금을 먼저 예치한 후에 계약을 네트워크에서 실행시킬 수 있다. 따라서 이 계약 예치금은 암호화폐가 가진 익명성을 완화시켜 암호화폐 네트워크의 보안을 강화시켜준다. 그 이유는 만일 사용자가 암호화폐 네트워크를 공격하면, 자신의 계약 예치금 중 일부 또는 전부를 빼앗길 수 있기 때문이다. 이것이 우리 시스템이 암호화폐 네트워크의 보안을 향상시키는 기본 원리이다. 또한 우리 시스템은 스팸 공격을 막을 뿐 아니라 계약 수수료를 낮출 수 있고, 이 둘은 가스 시스템에서 서로 상반관계에 있다.

1. 서론

암호화폐(cryptocurrency)인 비트코인(Bitcoin)은 신뢰하는 중앙기관이 없이 분산된 다수의 노드들이 피어-투-피어(peer-to-peer, p2p) 네트워크를 구성하여 온라인 지불(online payments)을 하는 복제 시스템(replication system)이다.[1] 비트코인은 작업증명(proof-of-work, PoW) 알고리즘을 통해서 분산되고 공개된 블록체인을 최신 상태로 유지한다.[1] 비트코인은 loop 명령어 등을 포함하지 않은 *튜링 불완전 스크립트 언어(turing-incomplete language)*를 이용하여 온라인 지불 거래를 실행한다.[2,3] 튜링 불완전 언어는 일반적으로 반복(loop) 및 분기(branch) 명령어와 저장소를 포함하지 않지만, 반대로 만일 이들을 모두 포함하면 *튜링 완전 언어(turing-complete language)*라고 일반적으로 말한다.[3] 비트코인이 튜링 불완전 언어를 채택한 것은 이 언어가 가진 무한루프 공격(infinite loop attacks)으로부터 자체 네트워크를 보호하기 위한 의도적인 목적 때문이었다.[2] 하지만, 이 때문에 비트코인은 사용자가 복잡한 로직을 작성하지 못하는 내재적 약점을 가진다.

이런 단점을 극복하기 위해서, 비트코인의 분산된 블록체인 개념을 이어받은 이더리움(Ethereum)은 튜링 완전 언어를 고의적으로 채택했다.[4] 이더리움의 p2p 네트워크를 연결하는 노드들(nodes)은 이더리움 가상머신(Ethereum Virtual Machine, EVM)을 통해서 *스마트 계약(smart contract)*을 실행한다. 여기서, 스마트 계약이란 계약의 내용을 컴퓨터 코드로 동작시키는 거래 규약(transaction

1) Sungil Kim, Ph.D. (hanaloum@gmail.com) is a researcher studying cryptocurrency, Korea.

*14QUhtKW329SqbfkNdNhtZgpWUoUorQdV (Bitcoin)

*0x008C9A583e7E9E982ca4b5a851e23633fFe96Ac0 (Ethereum)

protocol)이다.[12] 구체적으로, 이것은 블록체인에 읽기/쓰기가 가능한 바이트코드(bytecodes)의 집합으로 이루어지고, 이 코드로 코딩된 명령어에 따라 계약을 실행한다.[13] 따라서, 스마트 계약(계약이라 부르자)은 내부 토큰을 보내고, 다른 스마트 계약을 읽고, 결정을 내리고, 다른 계약을 실행할 수 있다.[13] 사용자는 이더리움에서 제공하는 튜링 완전 언어를 사용하여 스마트 계약을 작성하고, 이를 블록체인과 연동하여 실행할 수 있다. 결국, 이더리움은 사용자가 작성한 다양한 계약들을 네트워크에서 실행시킬 수 있는 분산 플랫폼(distributed platform)을 제공한다.[4] 그리고 이더리움에서 사용자 계정은 계약의 바이트코드(bytecodes) 및 데이터를 기록하는 내부 저장소(storage)를 가지고 있다.[8] 본 논문에서 언급하는 튜링 완전 암호화폐는 이더리움을 가리킨다.

본 논문의 2장은 이더리움의 네트워크 보안 방법인 개스 시스템과 문제점을 살펴본다. 3장은 사용자 책임 시스템과 이를 구현하기 위한 계약 예치금 방법을 제안한다. 4장은 사용자 책임 시스템의 장점을 토론하고, 5장은 결론을 낸다.

2. 이더리움의 네트워크 보안 방법: 개스 시스템(the gas system)

이더리움 네트워크는 튜링 완전 언어에 의한 무한루프 공격 때문에 네트워크가 불안정해질 수 있다. 즉, 만일 공격자가 이 네트워크에서 무한루프 공격을 경제적 부담이 거의 없이 수행할 수 있다면, 그가 계약의 계산 및 저장 등을 급속히 증가시켜 네트워크를 상당히 느리게 만들거나 최악의 경우 멈추게 할 수 있다. 이런 이유 때문에, 이더리움은 사용자의 악의적인 공격 또는 남용으로부터 네트워크를 보호하기 위해서 계약의 각 실행 단계마다 소비되는 수수료인 개스(gas)를 도입하였다.[5,9] 이것을 *계약 수수료(contract fee)*라고 부르자. 즉 이더리움에서 개스는 계약 수수료이고, 이것은 암호화폐의 지불 거래에 부과되는 거래 수수료(transaction fee)와 다르다. 여기서, 중요한 것은 이더리움에서 개스는 악의적인 공격 또는 남용으로부터 네트워크를 보호하는 핵심적인 기능이라는 것이다. 이것을 *개스 시스템(gas system)*, 또는 *사용량 기반 요금 시스템(usage-based charging system)*으로 부르겠다. 쉽게 설명하면, 개스 시스템은 사용량 기반의 요금제를 사용하여 이더리움 네트워크를 보호한다.

구체적으로, 개스 시스템에서 계약이 EVM에 의해 실행될 때, 이더리움의 개스는 계약의 각 계산 단계마다 소비된다. 즉, 계약을 구성하는 각각의 명령어(bytecodes)가 실행될 때마다 미리 정해진 개스가 소비된다.[9] 다시 말하면, 각각의 바이트코드들은 네트워크에 부담이 되는 정도 등에 따라 요금이 서로 다르다. 예로, 더하기인 ADD 명령어가 실행될 때 3 개스가 소비한다. 개스는 계약이 실행될 때 노드의 자원 사용량에 비례하는 소비되는 요금이기 때문에 이를 *연료(fuel)*라고 부르기도 한다.

2.1 이더리움 개스 시스템의 문제점

비트코인과 이더리움과 같은 암호화폐는 사용자가 개인키에 의해서 오직 구별되고, 그가 수많은 새로운 비밀키(private key)를 아무 제한 없이 즉시 만들 수 있기 때문에 강한 익명성을 가진다. 이런 강한 익명성은 암호화폐에서 사용자의 거래를 비공개로 유지하는 좋은 특징을 가진다. 하지만 컴퓨터 네트워크에 대한 공격은 이런 익명성 때문에 발생된다. 왜냐하면 암호화폐가 지닌 익명성은 공격자가 암호화폐 네트워크를 심하게 공격하더라도 그가 이에 대한 책임에서 완전히 자유롭게 만들어준다. 따라서 이더리움의 개스 시스템은 암호화폐가 지닌 익명성 때문에 스팸 공격(spam attacks)을 근본적으로 해

결할 수 없다.

실제로, 2016년 11월에, 개스 시스템의 단점을 이용한 스팸 공격(spam attacks)이 이더리움의 네트워크에서 발생했다.[6] 당시 스팸 공격으로 이더리움의 블록 생성속도가 약 2~3배 느려질 정도로 이 공격은 이더리움 네트워크에 큰 부담을 주었다.[6] 이 공격을 해결하기 위해서, 이더리움 개발자들은 1) 스팸공격에 사용된 명령어의 개스 값을 인상하고, 2) 스팸공격에 사용된 빈 계정을 없애는 2번의 하드 포크(hard fork)를 진행했다.[7] 여기서, 하드포크는 이더리움의 프로토콜(protocol)을 변경하는 것으로 이전에 유효하지 않은 블록 또는 거래를 유효하게 만드는 방법이고,[14] 만일 당시 프로토콜(프로그램)을 업그레이드하지 않은 노드가 있다면 블록체인은 두개로 갈라진다.

하지만 이더리움 개발자들의 이런 대응은 임시방편적이다. 왜냐하면 개스 시스템에서 현재 스팸 공격에 대한 대응 수단은 해당하는 개스 값을 올리는 것 외에 다른 대응 방법이 없기 때문이다. 그렇지만, 이런 방법은 계약 수수료를 인상시키는 부작용 때문에 사용자들의 네트워크 이용을 감소시킬 가능성이 있다. 사실, 이런 이유 때문에 개스 시스템은 스팸 공격 등으로부터 네트워크를 방어하는데 한계를 가질 수밖에 없다. 2016년 11월의 스팸 공격 이후에, 이더리움 커뮤니티에서는 공격에 사용된 특정 명령어의 개스 가격을 조정하는 방법 등에 대한 다양한 토론이 있었다.[10]

개스 시스템의 다른 단점은 비용이 네트워크의 사용량에 비례하여 증가하는 *강한 종량제 특성 (strong usage-based characteristics)*지닌다는 것이다. 이 종량제 특성 때문에, 개스 시스템에서 스팸 공격 가능성과 저렴한 개스 값은 서로 상반관계에 있다. 따라서 사용자는 이런 종량제 특성 때문에 계약 (contracts)의 코드 길이가 너무 길어지는 것을 두려워할 가능성이 있고, 그 이유는 계약에 더욱 많은 명령어를 포함시킬수록, 더 많은 비용을 지불해야 한다는 것 때문이다. 하지만 이더리움은 다양한 사용자가 자신의 계약들을 쉽게 실행할 수 있는 플랫폼을 제공하려 노력하고 있다. 따라서 개스 시스템이 가진 이런 강한 종량제 특성은 플랫폼으로서 이더리움의 역할을 제한할 수 있다. 플랫폼은 가능한 많은 사용자가 적은 부담 또는 거의 부담이 없이 이를 이용해야 더욱 활성화될 수 있다. 또한 이런 문제는 이더리움의 작업증명(PoW) 알고리즘이 앞으로 지분증명(proof-of-stake, PoS) 알고리즘으로 바뀌면, 계약이 노드들의 컴퓨팅 자원을 더욱 많이 사용할 수 있게 되면 더욱 문제가 될 수 있다.

3. 발상의 전환: 사용자 책임 시스템 (User accountability system)

본 논문은 개스 시스템의 이런 문제점을 해결하기 위한 방법으로 *사용자 책임 시스템 (우리 시스템)*을 제안한다. 우리 시스템의 핵심은 네트워크 보안의 책임을 사용자에게 강제로 부과시키는 것이다. 만일 사용자가 자신에게 주어진 책임, 즉 *규칙(rules)*을 지키면, 그는 계약 수수료를 개스 시스템 대비 훨씬 저렴한 비용을 지불하여 경제적 이익을 얻는다. 반대로, 만일 사용자가 이 규칙을 어기면, 그는 경제적 손해로 불이익을 받게 된다. 따라서 우리 시스템에서 네트워크 보안은 경제 원리로 유지되는 것이다. 다른 말로, 우리 시스템은 사용자가 부주의 또는 악의로 네트워크에 부담을 주면, 이를 유발한 사용자가 경제적 손해를 입게 해서 네트워크의 보안을 유지하는 새로운 네트워크 보안 방법을 제공한다. 여기서, 강조하고 싶은 것은 우리 시스템은 스팸 공격을 막고, 계약 수수료를 낮출 수 있다는 것이고, 이 둘은 개스 시스템에서 서로 상반관계에 있는 것이다. 이제, 우리 시스템을 구현하기 위한 새로운 계약 예치금 방법을 소개하자.

3.1 계약 예치금 방법 (contract deposit method)

사용자 책임 시스템은 사용자가 자신의 행위에 책임지기 위해서 계약 예치금을 미리 예치하도록 강제한다. 즉 만약 사용자가 자신의 계약 예치금을 미리 예치되지 않으면, 그는 자신의 계약을 네트워크에서 실행할 수 없다. 이것을 *계약 예치금 방법(contract deposit method)*이라 부르겠다. 사용자는 계약을 실행하기 전에 자신의 계약 예치금을 특별한 잠금 계좌에 먼저 옮긴 후에 출금을 못하도록 이를 잠근다. 즉, 사용자가 자신의 지갑에서 계약 예치금을 직접 잠근다. 결국, 계약 예치금은 사용자가 네트워크를 공격할 의도가 없음을 확인하는 용도이다. 따라서 두 가지 네트워크 보안 방법은 완전히 다르다. 즉, 튜링완전 암호화폐의 네트워크 보안을 위해서, 우리 시스템은 계약 예치금 방법을 이용하지만, 개스 시스템은 강한 종량제 특성(usage-based characteristics)을 이용한다.

현실 세계에서, 행위자가 규칙을 어기면, 그는 어느 형태로든지 대부분 손해를 입게 되고, 이것이 현실 세계의 가장 독특한 특징이다. 이와 마찬가지로, 사용자가 우리 시스템의 규칙을 어기면, 계약 예치금 때문에 자신이 경제적 손해를 입는다. 이것이 우리 시스템에서 네트워크 보안을 향상시키는 기본 원리이다. 이 때문에 계약 예치금 방법은 암호화폐가 가진 익명성을 완화시킬 수 있는 것이다. 다시 설명하면, 사용자 책임 시스템은 사용자의 잠겨있는 계약 예치금으로 암호화폐 네트워크를 강하게 보호할 뿐 아니라 저렴한 계약 수수료를 부담하여 경제적인 이익을 얻을 수 있다.

또한 예치금으로 보안을 유지하는 방법은 텐더민트 (tendermint) 등과 같이 암호화폐의 지분증명(PoS) 합의 알고리즘에서 사용하고 있다. 이를 *PoS 예치금(PoS deposit)*이라고 부르자. 하지만, 이들의 목적은 완전히 다르다. 즉, 계약 예치금 방법은 스팸 공격 등으로부터 암호화폐 네트워크를 보호하지만, 이와 달리 PoS 예치금은 오직 합의 알고리즘의 보안을 유지하는 기능만 가진다.

이제, 우리는 구체적인 계약 예치금 방법으로 1) 개스 제한을 규칙으로 이용한 방법 및 2) 사후 검증 방법을 소개한다. 우리 시스템은 이 방법들 외에도 다양한 다른 방법으로 구현될 수 있다.

3.1.1 개스기반 방법(gas-based method)

그림 1은 계약이 실행될 때 (a) 개스 시스템과 사용자 (b) 책임 시스템이 동작하는 구체적인 예를 보여준다. 단, 그림에서 개스는 모든 블록에서 시간에 따라 일정하게 증가한다고 가정했다. 물론 각 명령어마다 개스 값이 다르고, 계약이 일반적으로 모든 블록에서 동일하게 실행되지 않기 때문에, 이 그림은 실제 상황을 보여주지는 않는다.

먼저 이더리움의 개스 시스템 (그림 1a)을 살펴보자. 개스 시스템에서, 개스는 강한 종량제 특성을 반드시 가지고 있어야 한다. 이것은 개스가 이 시스템의 네트워크 보안을 전적으로 책임지고 있기 때문이다. 다른 말로 표현하면, 개스 시스템은 계약에서 더 많이 실행하면 더 많은 수수료를 소비하게 만들어 네트워크를 보호한다. 그림 1a에서 실선은 개스이고, 이것은 계약 실행 동안 사용자가 지불한 총 계약 수수료를 나타낸다. 그리고 이더리움의 개스 시스템에서 사용자는 자신이 소비할 수 있는 최대 개스인 개스 제한(gas limit)을 설정한 후에 계약을 실행해야 한다. 만약 계약의 실행 중에 개스 제한을 초과하면, 수행했던 모든 작업은 초기로 되돌아가(롤백 기능)고 이때 소모된 개스는 돌려주지 않는다.

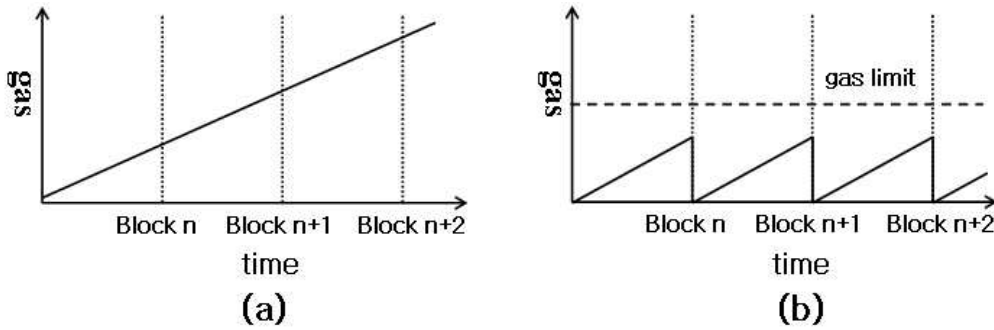


그림 1. (a) 이더리움의 가스 시스템: 실선(가스)은 사용자가 소비한 총 계약 수수료를 나타낸다. (b) 사용자 책임 시스템: 실선(가스)은 사용자가 규칙을 지켰는지를 알려주고, 여기서 규칙은 가스 제한을 의미한다. 그리고 그림에 표시되어 있지 않지만, 계약 수수료는 계약이 실행되는 각 블록마다 한 번씩 지불된다.

이제 계약 예치금을 이용하여 네트워크 보안을 유지하는 사용자 책임 시스템의 예를 살펴보자. 먼저, 우리는 가스 시스템의 가스를 이용한 *가스기반 방법(gas-based method)*을 제안한다. 그림 1b는 가스기반 방법의 한 가지 예이고, 이에 대한 동작방법을 보여준다. 그림 1b에서 가스는 각 블록마다 초기화된 후에 적산된다. 즉 이 그림에서 실선은 가스를 나타내고, 또한 이 실선은 사용자가 규칙을 지켰는지를 알려준다. 다시 말하면, 그림 1b에서 가스는 우리 시스템의 규칙을 지키는지 여부를 판단하는 지표로 사용된다. 결국, 우리의 가스기반 방법과 가스 시스템에서 가스의 역할은 완전히 다르다. 또한, 그림에 표시되어 있지 않지만, 계약 수수료는 계약이 실행되는 각 블록마다 한 번씩 지불된다. 따라서 만일 그림 1b와 같이 가스의 피크가 가스 제한(gas limit)을 초과하지 않으면, 사용자는 가스 시스템에 비해서 저렴한 계약 수수료를 각 블록마다 지불한다. 그리고 이 예에서 우리 시스템의 *규칙*은 *가스 제한(gas limit)*을 초과하지 않는 것이다. 그리고 가스 제한은 사용자가 예치한 계약 예치금의 금액에 따라 자동적으로 계산되어 변한다. 결국, 계약 예치금이 많아질수록, 가스 제한은 더 높아지게 된다. 우리 시스템에서 가스 제한과 가스의 피크는 계약이 실행될 동안 각 블록마다 계속 비교한다. 그리고 우리 시스템에서 최소 계약 예치금은 미리 정해져야 한다. 여기서, 최소 계약 예치금은 사용자가 네트워크를 이용하기 전에 인출을 못하도록 잠긴 가장 작은 예치금을 말한다. 그래서 이 최소 예치금은 여러 가지 문제를 종합적으로 고려하여 결정해야 한다.

그림 1b의 예에서 가스는 각 블록마다 초기화되지만, 이와 달리 블록의 묶음마다 초기화될 수도 있다. 이에 따라, 우리 시스템에서 계약 수수료는 각 블록 또는 각 블록의 묶음마다 한 번씩 부과될 수 있다. 따라서 우리 시스템에서 계약 수수료는 가스 시스템과 달리 각 블록 또는 각 블록의 묶음의 명령어의 첫 실행에서 한번 지불된다.

또한 우리 시스템의 벌금에 누진 특성을 적용하면, 네트워크 보안은 더 강화시킬 수 있다. 즉, 공격자가 네트워크의 공격을 계속 지속하면, 그의 계약 예치금은 이 누진 특성 때문에 더 빨리 빼앗기게 된다. 이 때문에, 만일 이 방법이 적용하려면, 사용자가 자신의 계약의 코드에 가스 제한을 넘지 않을 수 있는 명령어 또는 함수 등이 필요할 수 있다. 이에 대한 두 가지 구체적인 방법을 살펴보자. 첫 번째 방법으로, 만약 계약이 가스 제한에 도착하면, 계약의 실행을 일시적으로 멈춘 후 다음 블록에서 나머지 부분을 실행하도록 사용자가 계약의 코드를 작성하는 방법이다. 다른 방법은 계약이 가스 제한

에 도착했을 때 계약 예치금이 자동적으로 추가 예치되고 그 후에 나머지 계약이 실행되도록 사용자가 계약의 코드를 작성하는 방법이다.

또한 우리 시스템에서 개스 제한을 초과할 가능성이 있다. 이 경우, 사용자가 규칙을 어기었기 때문에 그의 계약 예치금 중 일정비율을 벌금으로 빼앗는다. 하지만, 이것이 사용자의 부주의 때문이라면, 사용자가 추가 예치금을 더 예치하는 방법으로 이 문제를 해결할 수 있다. 여기서, 추가 예치금은 1) 초과한 개스 제한에 해당하는 계약 예치금과 2) 벌금으로 추가된 벌금 예치금을 말한다. 이 경우, 만일 사용자가 규칙을 어긴 후, 지정된 시간 안에 해당하는 추가 예치금을 예치한다면, 벌금을 면제해 주고 계속 낮은 계약 수수료로 네트워크를 이용하는 혜택을 준다. 여기서, 벌금 예치금은 계약 예치금과 목적은 같지만, 사용자의 실수 때문에 추가로 부담하는 예치금이다. 그러나 만일 사용자가 네트워크를 공격한다면, 그에게 이런 혜택은 주어지지 않고, 그의 계약 예치금의 일정비율을 벌금으로 빼앗는다.

스팸 공격은 일부 명령어를 사용하여 네트워크를 공격한다는 특징을 가진다. 실제, 2016년 11월의 이더리움 네트워크에 대한 스팸 공격은 EXTCODESIZE 명령어를 이용했다.[6] 이런 공격 특성 때문에 네트워크의 부하를 측정하여 부하가 급격히 증가하는 특정 명령어의 개스 값을 부하에 따라 변동시키는 방법이 가능하다. 이것은 개스기반 방법으로 쉽게 구현할 수 있다. 그리고 개스기반 방법은 이더리움에서 채택한 개스를 이용하여 쉽게 구현할 수 있다. 이 경우 계약의 컴파일 시 사용하는 개스를 정확히 계산하는 것이 큰 도움이 된다. 이더리움 개발자들은 현재 컴파일 단계에서 개스를 정확히 계산하는 파이썬과 문법이 비슷한 viper 프로그래밍 언어를 개발하고 있다.[15] 또한 그림 1b에서, 계약 예치금은 소비되지 않고 계약 수수료가 소비된다. 이런 방법과 달리, 계약 예치금의 미리 정해진 일부가 수수료로 소비되도록 설계될 수 있다.

또한 만일 이더리움에 지분 증명(PoS) 알고리즘이 미래에 적용되면, 현재의 작업증명(PoW) 알고리즘과 달리, 노드들은 네트워크를 유지하기 위해서 계산 자원을 거의 사용하지 않는다. 이 경우, 사용자들은 노드의 자원을 많이 사용하는 계약을 실행할 수 있게 되고, 이 때문에 우리의 개스기반 방법이 쉽게 적용될 수 있을 것이다.

3.1.2 사후 검증방법

두 번째로, 우리는 네트워크 공격 후에 공격자의 계약 예치금을 빼앗는 *사후 검증 방법 (post-verification method)*을 제안한다. 이 방법에서, 만일 공격자가 네트워크를 악의적으로 공격을 한다면, 일정한 조건을 가진 참여자들은 이를 사후에 검증하여 미리 정해진 절차에 따라 공격자의 계약 예치금의 일부를 빼앗는다. 예를 들어, 네트워크 공격에 대한 처벌은 참여자들의 투표에 의해서 결정될 수 있다. 여기서, 참여자는 검증을 위한 그룹 또는 개발자들 또는 일정한 조건을 가진 참여자 등이 될 수 있다.

이 방법에서, 우리 시스템의 규칙은 *네트워크를 공격하지 않는 것*이다. 물론 실시간으로 네트워크에 대한 공격을 자동적으로 발견한다면 좋겠지만, 이것은 구현하기 힘들 수 있다. 이 때문에 네트워크 공격이 발생한 후 공격자를 찾아 그에게 경제적인 피해를 주는 것이 현재로서 좋은 방법일 수 있다. 그리고 악의적인 네트워크 공격일수록 공격자 자신의 계약 예치금 중에서 더 많은 부분을 벌금으

로 빼앗기어서, 이 때문에 그가 큰 두려움을 느끼도록 하면 좋다. 왜냐하면, 이 방법은 공격자가 자신의 손실을 예측하여 그가 네트워크를 공격하는 것을 두려워하게 만드는 것이 제일 큰 목적이기 때문이다. 따라서 우리 시스템에서 벌금은 공격의 특성에 따라서 차등적으로 부과해야 한다. 또한 네트워크 공격에 대한 처벌은 미리 결정되고 이를 공개적으로 공지하여 공격자들이 이를 잘 알 수 있도록 하는 것이 중요하다. 또한 개스 시스템과 사후 검증방법을 함께 사용하는 것도 좋은 방법일 수 있다. 이 경우 개스 시스템의 개스 값을 내릴 수 있을 것이다. 또한 처벌을 위해서, 계약 예치금의 출금은 출금 요청 후 예로, 3주일 또는 2달 이상이 걸리도록 만들어야 한다.

4. 사용자 책임 시스템의 이점

사용자 책임 시스템은 경제 원리를 이용하여 네트워크의 보안을 유지하기 때문에 상당한 강점을 가진다. 아래는 개스 시스템에 비해서 본 논문의 사용자 책임 시스템이 가진 장점을 알려준다.

1) *공격자의 악의적 공격 감소*—공격자는 튜링 완전 암호화페 네트워크의 약점을 공격을 하려고 끊임없이 노력한다. 하지만, 본 논문의 사용자 책임 시스템에서 사용자가 네트워크를 고의적으로 공격을 하면, 자신이 경제적인 손해를 입는다. 이 때문에 네트워크에 대한 악의적인 공격은 상당히 줄어들 수밖에 없다. 따라서 우리 시스템은 경제 원리를 이용하여 네트워크 공격자의 공격 의지를 감소시킬 수 있는 효과적인 방법이다.

2) *네트워크의 사용량 증가로 플랫폼 역할 강화*—우리 시스템에서 사용자는 저렴한 계약 수수료를 부담하고 네트워크를 이용할 수 있다. 이 때문에, 우리 시스템은 코드 길이가 긴 계약을 실행하는 것에 대한 사용자의 거부감을 크게 줄여주고, 이것은 사용자의 네트워크의 사용량을 크게 증가시킬 수 있다. 즉, 네트워크의 전체 사용량이 많아질수록, 이 네트워크에서 플랫폼의 역할이 강화된다.

3) *내부 토큰 가격의 안정*—암호화폐에서 내부 토큰(internal token)의 가격을 지지하는 요소 중 하나는 토큰의 사용성이다. 즉, 토큰의 사용이 증가하면, 토큰의 가격은 더욱 안정될 수 있다. 본 논문의 사용자 책임 시스템에서 사용자는 계약 예치금을 미리 예치한 경우에만 자신의 계약을 암호화페 네트워크에서 실행할 수 있다. 이 때문에 네트워크에서 내부 토큰의 사용량은 크게 증가할 수 있고 이것이 토큰의 가격을 안정화시킬 수 있다.

5. 결론

본 논문은 튜링완전 암호화페에서 새로운 네트워크 보안 방법으로 사용자 책임 시스템을 제안했다. 우리 시스템의 가장 큰 특징은 사용자가 네트워크의 안전에 대한 책임을 부담한다는 것이다. 이와 달리, 이더리움의 개스 시스템은 계약의 각 계산 단계마다 수수료를 부과하는 강한 사용량 기반 특성을 사용하여 네트워크를 보호한다. 하지만 우리 시스템은 사용자가 계약 예치금을 먼저 예치하고 잠금 후 사용자가 네트워크에서 스마트 계약을 실행하도록 강제한다. 만일 사용자가 우리 시스템에서 규칙을 어기면, 그의 계약 예치금의 일부를 강제로 빼앗긴다. 따라서 우리 시스템은 경제 원리를 이용하여 공격자의 공격 의지를 감소시키는 효과적인 방법을 제공한다. 이것은 스팸 공격을 막을 뿐 아니라 계약 수수료를 낮출 수 있고, 이 둘은 개스 시스템에서 서로 상반관계에 있는 것이다.

우리는 사용자 책임 시스템을 구현할 수 있는 두 가지 구체적인 방법을 제시했다. 즉 1) 개스

제한을 규칙으로 사용하는 개스기반 방법과 2) 사후검증 방법이다. 또한 이더리움에 지분증명(PoS) 알고리즘이 조만간 적용이 되면, 노드의 컴퓨팅 자원을 많이 사용하는 스마트 계약을 실행할 수 있게 되고, 따라서 우리의 개스기반 방법이 쉽게 적용될 수 있을 것이다.

이제 시빌 공격(sybil attacks)에 대해서 생각해보자. 암호화폐는 익명 시스템이라서 시빌 공격에 근본적으로 취약하다. 우리 시스템은 심각한 스팸 공격에는 매우 강한 특성을 지닌다. 그러나 우리 시스템도 시빌 공격에는 취약할 수 있다. 시빌 공격을 막기 위해서, 우리는 이더리움에서 토론되고 있는 것과 같이, 네트워크의 평균 부하에 따라 계약 수수료를 변화시키는 것이 필요하다고 생각한다. 이 계약 수수료는 모든 명령어를 일정비율로 조정하거나 또는 공격에 이용된 바이트코드별로 조정하는 방법이 사용될 수 있다.

References

- [1] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System." <https://bitcoin.org/bitcoin.pdf> (2009)
- [2] No Author. "Script." *Wikipedia* <https://en.bitcoin.it/wiki/Script> (accessed 5 April 2017)
- [3] No Author. "Turing_completeness." *Wikipedia* https://en.wikipedia.org/wiki/Turing_completeness (accessed 5 April 2017)
- [4] No Author. "Ethereum." *Wikipedia* <https://en.wikipedia.org/wiki/Ethereum> (5 April 2017)
- [5] Gavin Wood. "Ethereum: A secure decentralized generalized transaction ledger" *No Publisher* (2014) <http://gavwood.com/paper.pdf>
- [6] Vitalik Buterin. "Transaction spam attack: Next Steps." *Ethereum blog* <https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/> (2016)
- [7] Martin Swende. "Announcement of imminent hard fork for EIP150 gas cost changes." *Ethereum blog* <https://blog.ethereum.org/2016/10/13/announcement-imminent-hard-fork-eip150-gas-cost-changes/> (2016)
- [8] No Author. "Ethereum white paper." *Github* <https://github.com/ethereum/wiki/wiki/White-Paper> (accessed 5 April 2017)
- [9] Ethereum. "Gas Economics." *Github* <https://github.com/LeastAuthority/ethereum-analyses/blob/master/GasEcon.md> (accessed 5 April 2017)
- [10] Vbuterin. "On Gas Price Markets." *r/Ethereum* (accessed 5 April 2017) https://www.reddit.com/r/ethereum/comments/572n4q/on_gas_price_markets/
- [11] Jae Kwon. "Tendermint: Consensus without Mining." (2014) <https://tendermint.com/static/docs/tendermint.pdf>
- [12] No Author. "Smart contract." *Wikipedia* (accessed 5 April 2017) https://en.wikipedia.org/wiki/Smart_contract
- [13] No Author. "Contract tutorial." *Github* (accessed 5 April 2017) <https://github.com/ethereum/go-ethereum/wiki/Contract-Tutorial>
- [14] No Author. "Hard fork." *Bitcoin wiki* (accessed 5 April 2017) <https://en.bitcoin.it/wiki/Hardfork>
- [15] No Author. "Viper." *Github* (accessed 5 April 2017) <https://github.com/ethereum/viper>