Introduction to Error-Correction Codes Proof-of-Work


박상준
광주과학기술원


김형성
광주과학기술원


이흥노
광주과학기술원

1. Introduction

2008년 신원 미상의 사토시 나카토모는 백서[1]를 통해 탈중앙화된 – 즉 신뢰받는 기관이 없는 – 화폐 시스템을 소개하고, Peer-to-Peer (P2P) 네트워크를 통해 구현하였다. 이것이 최초의 암호화폐인 비트코인이고, P2P 네트워크에서의 이중 지불을 막기 위해 블록체인을 사용하였다.

블록체인이란 블록들이 하나의 체인으로써 연결된 것을 뜻한다. 각 블록들은 거래 내역(데이터)을 담고 있고, P2P에 존재하는 불특정 다수 노드들에 의해 검증 받는다. 검증하는 노드들을 채굴자라 일컫고, 이 검증과정을 작업증명이라 한다.

작업증명의 목적은 다수의 채굴자들이 하나의 블록을 채굴 – 즉 검증 – 하기 위해 많은 노력을 했다는 것을 입증하기 위함이다. 비트코인의 경우, SHA256 (Secure Hash Algorithm) 함수의 특정 해쉬 값을 산출하게 하는 nonce를 찾음으로써 작업증명이 완료된다. SHA256의 출력 값을 통해 역으로 입력 값을 알아내는 것이 불가능하므로, 채굴자들은 무차별적으로 nonce들을 대입해야 한다. SHA256의 출력 값인 해쉬 값은 블록 간의 연결을 위해 사용된다. 이전 블록의 해쉬 값을 현재 블록 내역에 포함시킴으로써 인접한 블록들을 연결시킨다. 이와 같이 연결하여 채굴된 블록들의 위·변조를 어렵게 만들고, 이를 통해 이중 지불을 막는다.

초창기 비트코인의 작업증명은 CPU를 통해서 이뤄졌다. 이 시기는 사토시가 백서에서 언급한 것처럼, 어느 누구나 CPU만 있다면 공

정한 채굴 경쟁이 가능하였다. 비트코인이 세상에 알려지고, 채굴 이윤이 발생함에 따라 채굴 경쟁이 시작되었고, 2010년, 2013년 각각 GPU와 ASIC (Application-Specific Integrated Circuit) 채굴 장비들이 등장하였다. ASIC 채굴의 성능은 CPU/GPU 보다 월등하였기 때문에, 채굴 난이도의 급격한 상승을 야기하였다. 결국 CPU/GPU 채굴자들은 더 이상 이윤 창출이 불가능해졌고, 오늘날의 비트코인 채굴은 ASIC을 통해 이뤄진다.

ASIC 채굴로 전환됨에 따라 일반 사람들 혹은 자본이 적은 사람들은 채굴에서 배제되고, 막대한 자본력을 가지고 있는 소수의 집단들이 채굴을 독점하였다. 이 집단들이 전 세계의 채굴 능력을 많이 점유하면 (극단적인 예로써, 51%이상), 이중 지불 등을 수행하여 악의적인 용도로 블록들을 채굴하고, 채굴된 블록들을 위·변조할 가능성이 존재하게 되는 것이다. ASIC 채굴을 억제하기 위한 새로운 작업증명들[3][4]이 제안됐지만, 결국 ASIC 채굴 장비들이 등장하였다.

오류-정정 부호 [6]는 무선 통신에서 발생하는 오류를 정정하기 위해 사용된다. 대표적인 부호들 중 하나로 LDPC [5] 부호가 있다. 문헌에 따르면, LDPC 디코더의 ASIC 구현은 구조적/비용적 문제로 인하여, 구현의 유연성이 떨어진다 [7]. LDPC 디코더와 해쉬 함수를 결합한 오류-정정 부호 기반의 작업증명 [2] (ECCPoW, Error-Correction Codes Proof-of-Work)을 제안하였다. 본 특집호의 목표는 ECCPoW의 동작과정과 ASIC 채굴 장비의 등장을 어떻게 억제하는지 설명하는 것이다.

본 특집호는 다음과 같이 구성되었다. 2장에서는 ASIC 장비 등장을 억제하기 위한 작업증명들을 소개하고, 이들의 한계점을 보여준다. LDPC 부호 및 디코더에 관한 문헌 결과를 보고한다. 3장에서는 ECCPoW의 동작과정과 ASIC 채굴 장비 등장 억제의 요인을 설명한다. 4장에서는 확률적 분석을 통해 ECCPoW의 작업증명 완료가 쉽지 않다는 것을 보여준다. 5장에서 본 특집호의 결론을 제시한다.

2. Background

본 섹션에서는 ASIC 채굴 장비 등장을 억제하기 위한 작업증명들 (Ethash와 X11)과 한계점들을 소개한다. LDPC 부호 및 디코더 소개와, ASIC 디코더에 대한 문헌들을 제공한다. 이 문헌들을 제공하는 이유는 ECCPoW의 ASIC 채굴 장비 억제 기능이 LDPC 디코더에서 기인하기 때문이다.

2.1. Ethash [3] and X11 [4]

이더리움의 작업증명인 Ethash [3]는 비선형 그래프 (DAG, Directed Acyclic Graph)를 이용하여 ASIC 채굴 장비 등장을 억제하였다. 이 DAG는 30,000 블록 단위로 무작위로 생산되는 데이터들의 집합으로써, 2019년 5월 기준, DAG의 크기는 약 3기가 바이트이다.

표 1은 Ethash의 동작과정을 보여준다. Nonce와 BH (블록 헤더)를 활용하여 해쉬 값을 생산하고, 이 값은 mix0에 저장한다. 이 과정이 Step 2이다. Step 4에서는 DAG로부터 데이터 (data1)를 읽어온다. 읽어올 데이터 위치는 mix0에 의해 결정된다 Step 5에서는 data1과 mix0를 mixing 함수에 넣고 얻은 결과를 mix0에 저장한다. Step 4부터 5까지 총 63번 반복 수행하고, 최종적으로 얻은 mix0를 사용해 작업증명 완료 유무를 판단한다.

63번의 반복 수행에서의 mixing 함수는 ASIC을 통해 처리 가능하다. 데이터 읽기 연산은 ASIC와 무관하다. 더욱이, 어떤 데이터를 읽어오는지 미리 알 수 없고, DAG의 크기가 너무 크기 때문에 캐쉬를 이용해 빠르게 읽는 것 또한 불가능하다. 따라서, 데이터 읽기와 mixing 함수 실행 사이에서 병목현상이 발생한다. 이 병목으로 인해, ASIC 채굴 장비를 사용할 필요가 없던 것이었다. 하지만 병목이 해결되면, ASIC 채굴 장비를 활용해 좀 더 빠른 채굴이 가능해진다. 문헌 조사에 따르면, 2018년 7월 비트메인은 ASIC 장비를 공포하였다.

대쉬의 작업증명인 X11 [4]은 다수의 해쉬 함수들을 사

〈표 1〉 Ethash 의사 결정 코드

Inputs: BH, L and DAG

Step 1: for nonce = 0, 1, 2, … $2^{32}-1$
Step 2: $\text{mix0} = (\text{SHA3}(\text{nonce}, \text{BH}))$
Step 3: for i = 1, 2, …, 63
Step 4: $\text{data1} = \text{Fetch}(\text{DAG}, \text{mix0})$
Step 5: $\text{mix0} = \text{Mixing}(\text{mix0}, \text{data1})$
Step 6: end
Step 7: If mix0 begins with L zero bits, then break.
Step 8: end

BH는 블록 헤더, L은 주어진 난이도, 해당 의사 결정,
코드는 본 연구팀의 논문인 [2]로부터 인용

〈표 2〉 X11 의사 결정 코드

Inputs: BH and L

Step 1: for nonce = 0, 1, 2, … $2^{32}-1$
Step 2: $\mathbf{e} = \text{Blake}(\text{nonce}, \text{BH})$
Step 3: $\mathbf{e} = \text{Bmw}(\mathbf{e})$
….
Step 12: $\mathbf{e} = \text{Echo}(\mathbf{e})$
Step 13: If $\mathbf{e}$ begins with L zero bits, then break.
Step 14: end

BH는 블록 헤더, L은 주어진 난이도, 해당 의사 결정,
코드는 본 연구팀의 논문인 [2]로부터 인용.

용하여 ASIC 채굴 장비 등장을 억제하였다. 이 X11은 다음 함수들이 순차적으로 사용한다:

Blake, Bmw, Groestl, Jh, Keccak, Skein, Luffa, Cubehash, Shavite, Simd and Echo

표 2는 X11의 동작과정을 보여준다. Nonce와 BH를 이용하여 Blake의 해쉬 값을 얻는다. 얻어진 값을 Bmw의 입력 값으로 사용한다. 이 과정을 반복하여 최종적으로 Echo의 해쉬 값을 얻고, 이 해쉬 값을 통해 작업증명 완료 유무를 판단한다.

X11에서 사용되는 해쉬 함수들의 순서는 고정이다. 따라서 ASIC 채굴 장비를 개발하려면, 함수들을 구현하고, 연결 하면 된다. 2014년과 2016년 사이에는 하드웨어 생산 비용 문제로, ASIC 채굴 장비 개발이 억제되었다. 하지만, 공정 기술 발달로 저 비용 생산이 가능해짐에 따라, ASIC 채굴 장비가 2016년부터 판매되고 있다.

X11을 확장하여 X13, X14, X15 그리고 X17 작업증명들이 제안되었다. 이름에서 알 수 있듯이, 별도의 함수들을 추가적으로 사용하여 ASIC 채굴 장비 등장을 억제하는 것이다. 2019년, X17을 제외한 작업증명들의 ASIC 채굴 장비가 판매되고 있다.

2.2. LDPC 부호와 디코더

대표적인 오류 정정 부호 중 하나인 LDPC [5] 부호는 대부분의 원소 값이 1인 패리티 체크 행렬 $\mathbf{H} \in \{0,1\}^{m \times n}$ (PCM, parity check matrix)를 이용해 정의된다. 구체적으로, PCM이 주어졌을 때, 다음 조건을 만족시키는

$$C := \left\{ \mathbf{c} \middle| \mathbf{Hc} = \mathbf{0} \cap \mathbf{c} \in \{0,1\}^{n \times 1} \right\}$$

벡터들 $\mathbf{c} \in \{0,1\}^{n \times 1}$ 의 집합이 LDPC 부호이다.

PCM을 이용해 LDPC 부호를 이분 그래프로 표현 할 수 있다. 이 그래프는, 변수 (variable) 및 체크 (check) 노드들과 이들을 연결하는 선으로 구성된다. 변수/체크 노드들은 PCM의 열/행에 각각 대응한다. PCM의 (i, j)번째 원소 값이 1이면 i번째 변수 노드와 j번째 체크 노드가 연결 된 것을 뜻한다.

LDPC 부호의 성능 – 얼마나 많은 오류들을 고치는지 – 은 PCM의 최소 해밍 거리 d (minimum hamming distance)에 의해 결정된다. 이 값은 PCM을 통해 생성할 수 있는 0 벡터를 제외한 모든 부호들 중에 가장 적은 해밍 값이다:

$$d = \min_{\mathbf{u} \in C, \mathbf{u} \neq \mathbf{0}} \|\mathbf{u}\|_h$$

여기서 $k = n - m$, $\mathbf{c}_i$ 는 i번째 부호, 부호의 개수는 총 2^k, 그리고 벡터 x의 해밍 값은 다음과 같다:

$\|\mathbf{x}\|_h :=$ $\mathbf{x}$에 포함된 1의 개수.

PCM의 최소 해밍 거리 d가 주어지면, LDPC 부호를 활용해 정정할 수 있는 bits 오류들의 숫자는 다음과 같이 결정된다:

$$t = \lfloor (d-1)/2 \rfloor \quad (1)$$

여기서 $\lfloor x \rfloor$는 x의 정수를 표시한다. 위의 결과의 유도 과정은 [6]에 있다.

LDPC 부호가 무선 채널을 통해 전송되면, 채널에 존재하는 잡음으로 인하여 오류가 발생한다. 잡음에 의해 왜곡된 부호 y는 다음과 같이 표현 할 수 있다:

$$y = c+e$$

여기서 $\mathbf{e} \in \{0,1\}^n$는 잡음에 의한 오류 벡터이고 그리고 c는 전송된 부호이다. LDPC 디코더의 목적은 오류를 정정하여, 원 부호 c를 찾는 것이다. 이 디코더는 일반적으로 메시지 전달 (message passing) [6] 알고리즘을 사용한다. 이 알고리즘은 변수/체크 노드들이 서로 메시지를 반복적으로 주고 받으며 원 부호를 찾기 위해 노력한다.

일반적으로, 알고리즘들을 빠른 속도 및 저전력으로 실행시키기 위해, ASIC 장치를 사용한다. 따라서, LDPC 디코더 또한 ASIC을 사용해 구현된다. ASIC-LDPC 디코더에서는 변수 및 체크노드들이 PCM에 따라 물리적으로 연결된다. 따라서, 부호의 길이의 변화에 따라 노드들을 늘리거나 혹은 PCM 변화에 따라 유동적으로 노드들의 연결을 재 설정하는 것이 쉽지가 않기 때문에, 다수의 PCM들을 지원하는 ASIC-LDPC 디코더 구현은 어렵다.

최신 리뷰 논문[7]에 따르면, 추가적인 하드웨어 장치들을 이용하면 다수의 PCM들을 지원하는 ASIC-LDPC 디코더를 구현 할 수 있다고 보고 하였다. 하지만, 그로 인해 디코더 면적 혹은 생산 비용이 증가되는 문제가 발생한다고 보고했고, 그 사례로써 [8]에서 구현된 ASIC-LDPC 디코더를 소개했다. 이 디코더는 약 100 개의 PCM들을 지원하지만, 추가적인 장치들이 디코더 면적의 약 75%를 점유하는 문제를 가지고 있다. 더 많은 PCM들을 지원하려면 더 많은 장치들이 사용되고, 그로 인해 ASIC-LDPC 구현이 매우 비효율적이다.

마지막으로, [7]에는 ASIC-LDPC 디코더의 구현 사례들이 표로 제시되어있다. 이 표에서 부호의 길이가 가장 긴 경우는 n = 64,800로써, 해당 디코더는 [9]에 구현되어 있다.

3. 오류-정정 부호 기반의 작업증명

본 섹션에서 ECCPoW 동작과정을 간단히 소개하고, ASIC 채굴 장비 등장의 억제 요인을 설명한다. ECCPoW에 관한 자세한 설명 및 이론적 분석 결과들은 [2]에 있다. 원활한 설명을 위해 용어들을 다음과 같이 축약한다. 현재 블록 헤더와 이전 블록 헤더를 각각 CBH (current block header)와 PBH (previous block header)로 사용한다.

먼저, ECCPoW에 포함된 LDPC 디코더와 그 입력 값을 다음과 같이 각각 정의한다.

정의 1. 크기가 $m \times n$인 PCM H와 길이가 n인 해쉬 벡터 r이 주어졌다 가정한다. LDPC 디코더는 H와 r을 취득하고, 메시지 전달 알고리즘을 사용해 길이가 n인 벡터 $\hat{\mathbf{c}}$을 산출한다:

$$D_{MP} : \{\mathbf{r}, \mathbf{H}\} \mapsto \hat{\mathbf{c}} \in \{0,1\}^{n\times 1}. \quad (2)$$

정의 2. 길이가 n인 해쉬 벡터 r은 다음과 같이 정의된다:

$$\mathbf{r} := \begin{cases} \mathbf{s}_1[1:n] & \text{if } n \le 256 \\ [\mathbf{s}_1 \ \cdots \ \mathbf{s}_l \ \ \mathbf{s}_{l+1}[1:j]] & \text{if } n > 256 \end{cases} \quad (3)$$

여기서 $l = \lfloor n/256 \rfloor$, $j = n - 256 \times l$,

$$\mathbf{s}_1 := \mathrm{SHA256}(\mathrm{nonce}, \mathrm{CBH}) \in \{0,1\}^{256} \quad (4)$$

그리고, $u = 2, 3, \ldots, l + 1$에 대해

$$\mathbf{s}_u := \mathrm{SHA256}(\mathbf{s}_1) \in \{0,1\}^{256}. \quad (5)$$

하나의 블록을 채굴하기 위한 작업증명에서는, CBH와 PCM은 모두 상수로써 취급된다. Nonce가 변경되면, 정의 2에 나온 것처럼 해쉬 벡터가 재 생성되고, 그로 인해 디코더의 출력 값이 변경된다. Nonce와 디코더의 입력 값인 해쉬 벡터의 관계는 SHA256 함수들에 결정된다. 따라서 nonce만 보고 디코더의 출력 값이 무엇인지 미리 예측하는 것은 불가능하다. 특정 조건을 만족하는 디코더의 출력 값을 찾으려면 무수히 많은 nonce들을 대입해야 한다. 결론적으로 ECCPoW의 작업증명은 디코더의 출력 값 $\hat{\mathbf{c}}$이 다음 조건이 다음 조건

$$\mathbf{H}\hat{\mathbf{c}} = \mathbf{0} \quad (6)$$

을 만족하게 하는 nonce를 찾으면 완료된다.

ECCPoW가 어떻게 동작하는지 살펴보았다. 이제 LDPC 디코더의 입력 값으로써 활용되는 PCM에 대해 논의한다. 블록들을 채굴 할 때, 하나의 PCM 사용을 가정한다. 섹션 2에서 이야기 한 것처럼, 이 가정에서는 ASIC-LDPC 디코더 구현에 아무런 문제가 없다. 하지만, 이 가정하에서는, ECCPoW를 위한 ASIC 채굴 장비가 등장 할 수 있다.

이제, 매 블록 채굴 할 때 마다 무작위로 생성된 PCM 사용을 가정한다. 섹션 2에서 언급한 것처럼, 다수의 PCM을 위한 ASIC-LDPC 디코더 구현에는 추가적인 하드웨어 장치들이 필요하다. 더욱이, 각 PCM들이 무작위로 생성된다. 따라서 어떤 형태로 생성될지 예측 할 수가 없다. 매 블록마다 변하는 PCM을 사용하면, 결국 ASIC-LDPC 디코더 구현을 억제할 수 있다. 이것이 ECCPoW에서 ASIC 채굴 장비 등장을 억제하는 이유이다.

이제 부호의 길이에 대해 논의를 해보자. 부호의 길이인 n인 경우, ASIC-LDPC 구현을 위해 필요한 computing fabric의 총량은 n의 제곱에 비례한다. 가령 n을 2배 키우면, 필요한 총량은 4배이다. 또한, n은 가변적으로 변하는 값이다. 무수히 많은 PCM을 지원하는 ASIC-LDPC 구현에 성공하더라도, n을 크게 키움으로써 구현된 ASIC-LDPC 디코더의 사용을 막을 수 있다.

어떻게 하면 매 블록마다 무작위로 변하는 PCM을 생성 할 수 있을까? 우리의 해답은 PBH의 해쉬 값과 Gallager의 PCM 생성 방법 [5]을 동시에 이용하는 것이다. 이 방법은 사용하면, 임의의 seed 값이 주어졌을 때, PCM을 결정적으로 생성할 수 있다. 즉, 여러 사람들이 동일한 seed 값을 가지고 있으면, 동일한 PCM을 생성 할 수 있는 것이다. 그리고, PBH의 해쉬 값은 매 블록 마다 바뀌고, 이미 채굴된 블록이므로, 알려진 값이다. 따라서, 이 값을 seed로 활용함으로써, 매 블록 마다 무작위로 PCM을 생성 할 수 있다. [2]에 우리의 PCM 생성 방법의 의사 결정 코드 및 좀 더 자세한 설명들을 기록하였다.

4. 해쉬 사이클 분석

이 섹션에서는 ECCPoW 작업증명을 푸는 것이 쉽지 않다는 것을 보인다. 이를 다음 아래에 해쉬 사이클을 정의한다.

정의 3. 하나의 nonce을 이용하여, LDPC 디코더의 출력 값을 산출하고, 이 값을 이용해 작업증명의 완료 유무를 검증하는 것까지 포함한 과정을 해쉬 사이클로 정의한다.

이제 작업증명을 완료하기 위해서 총 몇 번의 해쉬 사이클이 필요한지 고려한다. 이 섹션의 분석 결과들은 [2]에서 일부 발췌된 것이다.

용이한 분석을 위하여 다음 2가지를 가정한다. 첫 번째로, 디코더는 이상적(optimal)이다. 즉, 섹션 2에서 나온 것처럼 t개 이하의 오류가 발생시 항상 정정 가능한 것을 가정한다. 두 번째로, 해쉬 벡터 와 nonce는 서로 1:1 관

계로 가정한다. 즉, 다른 nonce들은 각각 다른 해쉬 벡터를 생성한다.

이제 i번째 부호가 주어졌을 때, 해당 부호와의 해밍 거리가 t보다 작은 벡터들의 집합을

$$\mathrm{A}(\mathbf{c}_i,t) \coloneqq \{\mathbf{r} : \|\mathbf{r}-\mathbf{c}_i\|_h \le t\} \tag{7}$$

로 정의한다. 이 집합의 크기는 다음과 같다

$$|\mathrm{A}(\mathbf{c}_i,t)| = 1+\binom{n}{1}+\binom{n}{2}+\cdots+\binom{n}{t} = \sum_{l=0}^{t}\binom{n}{l}$$

여기서 $i = 1, 2, \ldots, 2^k$.

첫 번째 가정으로 인하여, 해쉬 벡터가 $\mathrm{A}(\mathbf{c}_i,t)$의 원소라면, 출력 값이 항상 i번째 부호이다. 즉,

$$\mathrm{D}_{\substack{MP \\ \mathbf{r}\in\mathrm{A}(\mathbf{c}_i,t)}} : \{\mathbf{r},\mathbf{H}\} \mapsto \mathbf{c}_i.$$

따라서, i번째 부호를 산출할 확률은 다음과 같다

$$\begin{aligned}\Pr\{\hat{\mathbf{c}}=\mathbf{c}_i\} &= 2^{-n}|\mathrm{A}(\mathbf{c}_i,t)| \\ &= 2^{-n}\sum_{l=0}^{t}\binom{n}{l} \\ &= 2^{-n}\sum_{l=0}^{\lfloor\frac{d-1}{2}\rfloor}\binom{n}{l}\end{aligned}$$

여기서 마지막 등호는 (1)에서 기인한다. 디코더의 출력 값이 임의의 부호일 확률은 다음과 같다:

$$\begin{aligned}p &\coloneqq \Pr\{\mathbf{H}\hat{\mathbf{c}}=\mathbf{0}_m\} \\ &= \sum_{i=1}^{2^k}\Pr\{\hat{\mathbf{c}}=\mathbf{c}_i\} \\ &= 2^{k-n}\sum_{l=0}^{\lfloor\frac{d-1}{2}\rfloor}\binom{n}{l}.\end{aligned} \tag{8}$$

LDCP 디코더가 부호를 출력할 확률이 p이므로, 이것의 역수는 부호를 출력하기 위해 필요한 평균 해쉬 사이클이다. 따라서, p를 알면 어느 정도 해쉬 사이클이 필요한지 계산 가능하다. P를 계산하려면, d를 알아야 한다. 임의의 주어진 PCM의 d를 찾는 것은 어렵다. 우리는 편의를 위해 d를 n의 10%로 가정한다.

첫 번째로, n = 64, m = 32라 하자. 이 경우 p는 약 2×10^{-10}이다. 이 작업증명을 완료하기 위해 필요한 해쉬 사이클은 p의 역수인 4×10^{9}이다. 두 번째로, n과 m을 각각 128과 64로 하면, p는 5×10^{-20} 이다. 따라서, 2×10^{19} 해쉬 사이클이 필요하다. 이것들은 작업증명 완료를 위해 많은 해쉬 사이클이 필요한 것을 보여준다.

마지막으로, LDPC 디코더의 총 연산량은

$$Iter \times O(n\log n)$$

여기서 n은 부호의 길이이고, $Iter$은 메시지 전달 알고리즘의 반복 횟수이다. 각 nonce마다 디코더를 실행하므로, 평균적으로 작업증명을 완료하기 위해 사용되는 연산량은 다음과 같다:

$$Iter \times O(n\log n) \times p^{-1}$$

이것을 ECCPoW의 채굴 난이도로 여길 수 있다.

4. 결론

본 특집호에서는 블록체인 커뮤니티에서 많이 주목 받고 있는 ASIC 채굴 장비 등장으로 인한 중앙화 문제를 고찰하였다. 이 문제를 해결하기 위해 제안된 오류-정정 부호 기반의 작업증명 [2] (ECCPoW, Error-Correction Codes Proof-of-Work)를 소개하였다. 이 방법의 핵심은 기존 SHA256함수와 LDPC 디코더를 연결한 것이다. SHA256의 출력 값이 디코더의 입력 값이 되고, 이 디코더의 출력 값을 이용해 작업증명의 완료 유무를 판단하였다.

ASIC 채굴 장비 등장의 억제는 LDPC 디코더의 사용에서 기인한다. 매 블록 마다 새로운 패리티 체크 행렬을 무작위로 생성함으로써, ASIC-LDPC 디코더의 구현을 현실적으로 매우 어렵게 만든다. 그로 인해 디코더의 실행을 CPU/GPU에 의해서만 처리되도록 설계한 것이다.

Acknowledgement

이 논문은 2019년도 광주과학기술원의 재원으로 GRI(GIST연구원) 사업의 지원을 받아 수행된 연구임. 이 논문 은 2019년도 광주과학기술원의 재원으로 "과학기술 응용연구 단의 실용화 연구개발사업"의 지원을 받아 수행된 연구임.

참고문헌

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System." [Online]. Available: https://bitcoin. org/bitcoin.pdf.

[2] Sangjun Park, Haeung Choi and Heung-No Lee, "Time-variant proof-of-work using error-correction codes," in preparation.

[3] V. Buterin. Dagger: A memory-hard to compute, memory-easy to verify scrypt alternative. Tech Report, hashcash.org website, 2013.

[4] E. Duffield and D. Diaz, Dash: A payments-focused cryptocurrency, [Online]. Available: https: //github.com/dashpay/dash/wiki/Whitepaper

[5] R. G. Gallager, Low Density Parity Check Codes, Monograph, M.I.T. Press, 1963

[6] W. E. Ryan and S. Lin, Channel Codes Classical and modern, Cam¬bridge

[7] S. Shao, P. Hailes, T-Y. Wang, J-Y. Wu, R. G. Maunder, B. M Al-Hashimi and L. Hanzo, "Sur-vey of Turbo, LDPC and Polar De¬coder ASIC Im-plementation", IEEE Communications Surveys & Tutori¬als 2019

[8] Y. L. Ueng, B. J. Yang, C. J. Yang, H. C. Lee, and J. D. Yang, "An Efficient Multi-Standard LDPC Decoder Design Using Hard¬ware-Friendly Shuffled Decoding," IEEE Trans. Circuits Syst. I, vol. 60, no. 3, pp. 743 - 756, March 2013.

[9] T. Brack, M. Alles, T. Lehnigk-Emden et al., "Low complexity LDPC code decoders for next generation standards," in Proceedings of the De-sign, Automation and Test in Europe Conference and Exhibition (DATE '07), pp. 331 - 336, April 2007

박상준

- 2009년 8월 충남대학교 컴퓨터과 학사
- 2009년 2월~현재 광주과학기술원 전기전자컴퓨터공학부 통합
- 2018년 5월~2019년 2월 광주과학기술원 블록체인 인터넷 경제 센터 연구원

〈관심 분야〉
정보이론, 신호처리, 블록체인, 수치최적화 이론, 압축센싱

김형성

- 2019년 2월 전남대학교 전자컴퓨터공학부 학사
- 2019년 3월~현재 광주과학기술원 전기전자컴퓨터공학부 석사

〈관심 분야〉
블록체인

이흥노

- 1993년 5월 UCLA 전기공학과 학사
- 1994년 10월 UCLA 전기공학과 석사
- 1999년 12월 UCLA 전기공학과 박사
- 1999년~2002년 HRL Laboratories Researcher
- 2002년~2008년 University of Pittsburgh, Assistant Professor
- 2009년~현재 GIST Professor

〈관심 분야〉
정보이론, 신호처리, 블록체인, 수치최적화 이론, 압축센싱