비트코인 세그윗과 소프트포크

고혁준*, 한성수**, 전유부***, 정창성****
*고려대학교 영상정보처리협동과정
**강원대학교 자유전공학부
***순천향대학교 컴퓨터소프트웨어공학과
****고려대학교 전기전자공학부
e-mail : doltwo@hanmail.net*, sshan1@kangwon.ac.kr**, jeonyb@sch.ac.kr***, csjeong@korea.ac.kr****

Bitcoin SegWit and Softfork

Hyug-Jun Ko*, Seong-Soo Han**,You-Boo Jeon***, Chang-Sung Jeong****
*Dept. of Visual Information Processing, Korea University
**Dept. of Division of Liberal Studies, Kangwon National University
***Dept. of Computer Software Engineering, Soonchunhyang University
****Dept. of Electrical Engineering, Korea University

요 약

비트코인은 분산시스템으로 많은 노드를 가질수록 가용성 및 안정성이 유지된다. 이를 위해서는 블록 크기가 작고 많은 트랜잭션을 처리할 수 있는 구조를 가지는 것이 유리하다. 비트코인의 트랜잭션이 많아지면서 2017 년 8 월 24 일 세그윗(SegWit) 이후에 블록사이즈는 1MB 에서 2MB 로 변경되었고, 2019 년 9 월 현재 블록당 사이즈는 1MB 이상이 사용되고 있다. 이러한 추세라면 가까운 시일 내에 최대 블록사이즈에 근접하게 될 것이다. 본 논문에서는 세그윗 적용에 따른 비트코인의 변화를 조사하여 세그윗을 적용하지 않은 레거시(Legacy) 노드와의 차이점과 소프트포크(Softfork)로 알려진 호환성(Backward Compatibility)을 살펴보고, 세그윗을 통해 가단성(Malleability) 버그가 해결과 블록 사이즈 증가를 통해 TPS(Transaction Per Second)가 향상되는 것을 확인하고자 한다.

1. 서론

비트코인은 UTXO(Unspent Transaction Output)를 기반으로 하는 분산원장을 기록하는 블록체인으로 트랜잭션은 사용된 트랜잭션과 사용되지 않은 트랜잭션으로 나뉜다.

모든 트랜잭션은 블록내의 트랜잭션으로 Berkeley DB 에 저장되어 보관되고, 사용되지 않은 트랜잭션은 메모리 풀(Memory Pool)에 로딩되어 잔액조회 및 트랜잭션 생성시 이용된다.

미사용 트랜잭션의 출력은 잠금 스크립트로 보호되며 이를 사용하기 위해서는 잠금 스크립트(Locking Script)의 조건을 충족할 해제 스크립트(Unlocking Script)로 해당 비트코인 주소의 UTXO 에 대한 소유권을 입증하여 그 UTXO 를 소비하도록 한다. 그러므로 해제 스크립트는 거래에 대한 전자서명이 필요하며 이전 출력에 대한 사용을 명시적으로 표현하게 된다.

비트코인에서 채굴되기 전 해제 스크립트의 OP 코드를 조작 시 스크립트의 규칙에 어긋나지 않기 때문에 기존 트랜잭션 아이디가 변경되어 전파될 수 있다는 가단성(Malleability) 문제를 근본적으로 해결하기 위해 BIP-0141, BIP-0142, BIP-0143, BIP-0144, BIP- 0145 를 통하여 세그윗(SegWit)이 제안되어 해제 스크립트를 트랜잭션에서 분리하여 근본적으로 가단성을 해결하고, 1MB 에 해당하는 블록사이즈 제한 내에서 더 많은 트랜잭션을 처리할 수 있도록 하고 있다[1].

세그윗은 Segregated Witnesses 의 약어로 “분리된 증인”으로 변역될 수 있으나 위트니스는 암호학적으로 서명 정보라는 의미로 사용되어 “서명 정보의 분리”로 사용된다.

본 논문에서는 세그윗을 적용한 노드와 적용하지 않은 레거시 노드와의 차이점과 소프트포크(Softfork)로 알려진 호환성(Backward Compatibility)을 살펴보고, 세그윗을 통한 가단성(Malleability) 버그의 해결과 블록 사이즈 증가를 통해 TPS(Transaction Per Second)가 향상되는 것을 확인하고자 한다.

2. 관련 연구

2.1 지갑 주소

비트코인의 주소는 비트코인의 송수신 대상으로 공개 정보이며 개인키에서 생성한 공개키를 이용하여 공개키 해시 RIPEMD160(DoubleSHA-256(공개키)) 를 생성하고 버전 프리픽스(version prefix)를 추가하여 Base58Check(버전 프리픽스+공개키 해시)를 통하여 생성한다. 결과적으로 메인넷의 P2PKH 주소용은 주소가 1 로 시작되며

P2SH 는 3 으로 시작된다[2].

세그윗용 주소 체계는 BIP-0173 로 제안되었으며, Bech32 체계를 사용한다. 메인넷은 bc1 으로 시작되고 테스트넷은 tc1 으로 시작되는 주소 체계가 추가되었다.

2.2 트랜잭션

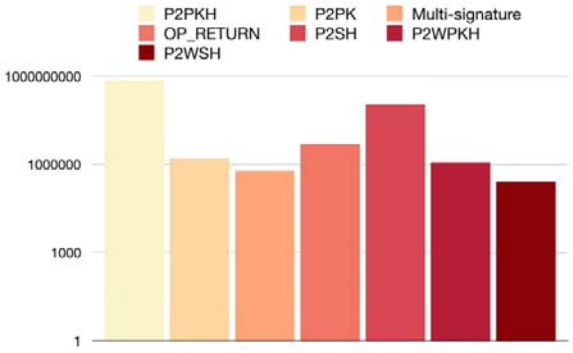
비트코인 지갑은 사용자의 개인키와 공개키 쌍들을 저장한다. 비트코인 주소는 공개키의 해시를 통해 만들어지며, 개인키는 임의의 256 비트의 숫자로써 해당 공개키는 ECDSA(Elliptic Curve Digital Signature Algorithm)를 통해 생성된다.

1 개의 트랜잭션에는 입력부와 출력부로 구성되며 입력부는 이전 트랜잭션을 통해 받은 비트코인이 scriptPubKey(locking script)로 잠겨져 있기 때문에 이에 맞는 scriptSig(unlocking script)로 잠김에 대한 소유권을 증명해야 한다, 출력부에는 송금 대상의 주소와 scriptPubKey (잠금 스크립트)를 이용하여 사용을 제한하게 된다.

2.3 비트코인 스크립트 사용통계

비트코인의 표준 스크립트는 총 7 개 이며, 기존 Pay to Public Key Hash(P2PKH), Pay to Public Key(P2PK), Multi-signature, Data output(OP RETURN), Pay to Script Hash(P2SH) 와 세그윗 이후에, Pay to Witness Public Key Hash(P2WPKH), Pay to Witness Script Hash(P2WSH)가 추가되었다.

비트코인의 제너시스(Genesis) 블록에서 #516,040 블록을 고려하면 총 837,854,035 출력을 생성하는 307,844,694 트랜잭션이 있었으며, 그 중 93.78%인 782,343,118 개의 출력이 소비 되었다. 그 중 표준 트랜잭션 출력은 전체의 99.97%인 837,632,874 개 중 782,123,115 가 소비되어 93.37 %가 사용되었다. <그림 1>은 비트코인 표준 트랜잭션 분포이다[3].



<그림 1>비트코인 표준 트랜잭션 분포[3]

<그림 2>는 세그윗 트랜잭션 점유율이며 세그윗이 시작된 2017 년 8 월에서 2019 년 9 월 기준 세그윗 트랜잭션이 50%를 넘어서고 있다[4].

<그림 2>세그윗 트랜잭션 점유율[4]

2.4 세그윗 이전 트랜잭션의 구조

<표 1>세그윗 이전의 트랜잭션 직렬화 구조에 따라 입력부는 (Previous output) + (output Index) + (Script Length) + ScriptSig) + (Sequence)로 구성되며, 출력부는 (Value)+(Script Length) + (scriptPubKey)로 구성된다[5].

<표 1 >세그윗 이전의 raw transaction format[5]

Size	Name	Description
4	version	트랜잭션 버전
C	Input count	Must be zero
3	Previous output	이전 출력부의 트랜잭션 ID
4	Output Index	이전 출력부들 중 사용할 출력부 번호
C	Script length	스크립트 길이
V	scriptSig	해제 스크립트, 송신자 서명 (최대 10,000bytes)
4	Sequence	Lock Time or Disabled(0xffffffff)
C	Output count	출력부 개수
8	Value	송금 금액(사토시 단위)
C	Script length	스크립트 길이
V	scriptPubKey	잠금 스크립트, 수신자의 공개키 해시 값(최대 10,000bytes)
4	LockTime	5 억 이하 면 최상위 블록 높이, 그 외 Unix Timestamp

C 는 압축크기(CompactSize Unsigned Integers)이며, 가변 길이 정수(Variable Length Integer) 값으로 범위에 따라 크기가 정해지는 것으로 <표 2>와 같은 규칙을 갖는다[6].

<표 2> CompactSize Unsigned Integers[6]

Size	Range	형식
1	<=0xfd	uint8_t
2	<= 0xffff	0xfd followed by the number as uint16_t
5	<= 0xffffffff	0xfe followed by the number as uint16_t
9	<=0xffffffffffffffff	0xff followed by the number as uint16_t

<그림 3>은 scriptSig 의 구성 예로 2 개의 파트로 구성된 전자 서명 <r, s>와 복호화용 Public key 데이터와 서명 검증용 OP 코드가 존재한다.

PUSHDATA Opcode		47
Signature (DER encoded)	Header	30
	Sig Length	44
	Integer	02
	R Length	20
	R	1c 3b e7 1e 17 94 62 1c be 3a 7a de c1 af 25 f8 18 f2 38 f5 79 6d 47 15 21 37 eb a7 10 f2 17 4a
	Integer	02
	S Length	20
	S	4f 8f e6 67 b6 96 e3 00 12 ef 4e 56 ac 96 af b8 30 bd df fe e3 b1 5d 2e 47 40 66 ab 3a a3 9b ad
SigHash Code		01
PUSHDATA Opcode		21
Public Key (compressed with 03 prefix)		03 bf 35 0d 28 21 37 51 58 a6 08 b5 1e 3e 89 8e 50 7f e4 7f 2d 2e 8c 77 4d e4 a9 a7 ed ec f7 4e da

<그림 3>scriptSig 구조 및 예시

2.5 세그윗 이후 트랜잭션의 구조

세그윗 트랜잭션을 위한 직렬화 구조는 BIP-0144 에 <표 3>과 같이 정의된다.

<표 3>세그윗 트랜잭션 직렬화 포맷[7]

Size	Name	Description
4	Version	Transaction data format version
1	Marker	Must be zero
1	Flag	Must be nonzero
1+	Txin_count	Number of transaction inputs
41+	Txins	A List of one or more transaction inputs
1+	Txout_count	Number of transaction outputs
9+	Txouts	A list of one or more transaction outputs
1+	Scripts_witness	The witness structure as a serialized bytes array
4	Lock_time	The block number or timestamp until which the transaction is locked

세그윗 이후에는 기존 업그레이드하지 않은 레거시 시스템을 위한 트랜잭션 아이디(txid)와 세그윗 노드를 위한 위트니스 아이디(wtxid)로 구분된다.

-트랜잭션 아이디(txid):

```
nVersion + inputs + outputs + nLockTime
```

의 해시값을 사용하며,

-위트니스 아이디(wtxid):

```
nVersion + 0x00 + 0x01+ inputs + outputs + witness +
nLockTime
```

의 해시값을 사용한다.

기존 레거시(Legacy)와의 호환성을 위해 txid 는 트랜잭션의 입력부에 Anyone-Can-Spend 스크립트를 넣어 정상적으로 네트워크에 전파되고, 세그윗 노드는 wtxid 를 통해 네트워크에 전파된다.

2.6 서명 검증

세그윗 이후에 P2WPKH 와 P2WSH 의 서명 검증용 스크립트가 추가되었다.

P2WPKH 의 서명 검증:

```
Witness : <signature> <pubkey>
scriptSig : (empty)
scriptPubKey : 0 <0x0014{20-bytes-key-hash}>
```

scriptPubKey 의 0 는 위트니스의 버전으로 사용되며, 길이 0x0014 는 이 트랜잭션이 P2WPKH 유형을 의미한다.

서명은 다음과 같이 검증된다.

```
<signature> <pubkey> CHECKSIG
```

해제 스크립트와 잠금 스크립트로 검증한다.

```
<20-bytes-key-hash> HASH160 <20-bytes-script-hash>
EQUAL
```

이는 서명에 관계없이 항상 true 임으로 트랜잭션을 패스하기 위한 검증으로 Anyone-Can-Spend 라 한다[8].

P2WSH 서명 검증:

```
Witness : 0 <signature1> <1 <pubkey1> <pubkey2> 2
CHECKMULTISIG>
scriptSig : <empty>
scriptPubKey : 0 <32-bytes-hash> (0x0020{32-byte-hash})
```

scriptPubKey 의 0 는 위트니스의 버전으로 사용되며, 길이 0x0020 는 이 트랜잭션이 P2WSH 유형임을 의미한다.

2.7 세그윗 이후 머클 트리(Merkle Tree)

머클 트리(Merkle Tree)에는 위트니스가 없으므로 위트니스의 머클 루트를 저장하기 위하여 코인 베이스 트랜잭션의 출력부에 채굴자의 잠금 스크립트 외에 출력 부를 한 개 더 두어 여기에 위트니스의 머클 루트(Merkle Root) 값을 저장한다.

2.8 세그윗 이후의 블록 사이즈

세그윗으로 인해 비트코인의 블록 사이즈를 측정하는 방식에 변화는 1MB(1,000,000Bytes)의 제한에서 4,000,000 weight 로 변경되었다.

트랜잭션의 무게(weight)는 다음과 같이 정의하며, 이는 위트니스 데이터가 트랜잭션의 약 75% 미만에 해당되어 위트니스 데이터에 3 배가 사용된다.

```
(tx size with witness data stripped) * 3 + (tx size)
```

예를 들어, 레거시 tx 가 500bytes 인 경우 위트니스 데이터를 제거할 수 없으므로 3*500+500= 2000 weight 가 된다.

이를 부분별로 나누면 다음과 같다.

```
legacy_block_size = ∑(size_of_non-segwit-
data_of_each_transaction)

segwit_block_size = legacy_block_size + ∑(size_of_segwit-
data_of_each_transaction)
```

하나의 블록에 대해서는 다음과 같이 정의할 수 있다.

non_segwit_weight = 3 * ∑ (size_of_non-segwit-data_of_each_transaction)

segwit_weight = 1 * ∑ (size_of_segwit-data_of_each_transaction)

block_weight = non_segwit_weight + segwit_weight

세그윗 노드를 실행하면 1MB 이상이 될 수 있는 블록을 만들거나 수신할 수 있지만 기존 레거시(Legacy) 노드를 실행 시 모든 위트니스 데이터가 제거된 1MB 미만의 레거시 블록이 되어 하위 호환성을 가지게 된다.

2.9 채굴 API 변경

채굴에서 기존 채굴은 더 이상 진행되지 않으며, 신규 채굴을 위해서는 반드시 세그윗 노드에서 진행된다. 채굴을 위해 사용하는 API 인 getblocktemplate 은 세그윗으로 인한 블록구조의 변경으로 인해 BIP-0145 에 의해 변경되어 무게(weight) 제한 계산 방법이 추가되었다.

3. 세그윗을 이용한 TPS 계산

세그윗의 Anyone-Can-Spend 를 이용하면 비트코인의 이론적인 최대 TPS 를 구할 수 있다.

3.1 실제 트랜잭션 크기

하나의 블록에서 트랜잭션을 담을 수 있는 공간은 다음과 같이 계산된다.

Tx space = 1MB(blockSize) – 80Bytes(blockHeader)
- 4bytes(blocklength) – 3bytes(transaction counter)
= 999,913bytes

3.2 이론적인 최대 TPS

아래의 Anyone-can-spend 출력 스크립트를 이용하여 이론적인 최대 TPS 를 구할 수 있다.

scriptSig : <OP_TRUE>
scriptPubKey : (empty)

이 트랜잭션은 세그윗 트랜잭션 검증 시 패스되는 비표준 스크립트이다.

Size of non-coinbase transaction = (Version + Tx_in count + Tx_in + Tx_out count + Tx_out + Lock_time)
= 4 + 1 + (36 + 1 + 1 + 4) + 1 + (8 + 1 + 0) + 4
= 4 + 1 + 42 + 1 + 9 + 4
= 61 bytes.

Size of coinbase transaction = (Version + Hash + Index + Tx_in count + Tx_in
+ Tx_out count + Tx_out + Height + Sequence
+ Lock_time)
= 4 + 32 + 4 + 1 + (1 + 1) + 1 + (8 + 1 + 0) + 4 + 4 + 4
= 65 bytes.

한 블록 내에 담을 수 있는 최대 트랜잭션 수는 다음과 같다.

(999913 − 65)/61 + 1 = 16391

그러므로 비트코인의 이론적인 최대 TPS 는 아래와 같이 27TPS 가 된다[9].

16391/(10 × 60) = 27TPS

이는 세그윗 이전이나 세그윗 이후나 같은 최대 TPS 가 된다. 이때 사용된 실제 블록 사이즈는 1MB 미만이 된다.

3.3 실제 P2WPKH 를 사용한 TPS

<그림 4>와 같은 샘플 트랜잭션에서 트랜잭션 사이즈가 218bytes 이며 weight 는 542bytes 로 가정하면 한 블록 내에 담을 수 있는 최대 트랜잭션 수는 다음과 같다.

(999913*4 − 542)/542 + 1*4 = 7,382

010000000000010115e180dc28a2327e687facc33f10f2a20da717e5548406f7ae8b4c8
11072f856030000000171600141d7cd6c75c2e86f4cbf98eaed221b30bd9a0b928ffff
ffff019caef505000000001976a9141d7cd6c75c2e86f4cbf98eaed221b30bd9a0b92
888ac02483045022100f764287d3e99b1474da9bec7f7ed236d6c81e793b20c4b5aa1
f3051b9a7daa63022016a198031d5554dbb855bdbe8534776a4be6958bd8d530dc001
c32b828f6f0ab0121038262a6c6cec93c2d3ecd6c6072efea86d02ff8e3328bbd0242
b20af3425990ac00000000

<그림 4>비트코인 세그윗 샘플 트랜잭션

그러므로 비트코인의 최대 TPS 는

7,382/(10 × 60) = 12.3TPS

이며, 이때 사용된 실제 블록 사이즈는 약 1.8MB 가 된다.

4. 결론 및 향후 연구

세그윗은 기존 블록체인의 개념으로 쓰였던 비트코인의 전 부분에 변화를 가져왔다. 특히 원장(Ledger)로 쓰인 UTXO 의 변경은 블록구조와 트랜잭션 구조 및 머클 트리의 변화를 가져왔고, 비트코인의 가단성 버그는 세그윗 노드의 채굴로 인해 근본적으로 해결이 되었으며, 레거시 노드들은 채굴을 제외한 모든 호환성(Backward Compatibility)를 가지게 되었다. 또한 블록 크기에 대한 weight 개념이 도입되어 기존 7TPS 에서 최대 12.3TPS 로 향상되었다. 향후에는 비트코인의 세그윗 이후 구현 가능해진 기술 중 슈노 시그니처(Schnorr signature)를 연구할 예정이다.

참고문헌

[1] https://www.blockchain.com/ko/charts/avg-block-size
[2] https://en.bitcoin.it/wiki/List_of_address_prefixes
[3] Stefano Bistarelli, “An Analysis of Non-standard Bitcoin Transactions”, CVCBT, 2018
[4] https://transactionfee.info/charts/payments/segwit
[5] https://btcinformation.org/en/developer-reference#raw-transaction-format
[6] https://btcinformation.org/en/developer-reference#compactsize-unsigned-integers
[7] http://github.com/bitcoin/bips/blob/master/bip-0144.mediawiki
[8] https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki
[9] Evangelos Georgiadis, “How many transactions per second can bitcoin really handle”, 24 Apr 2019