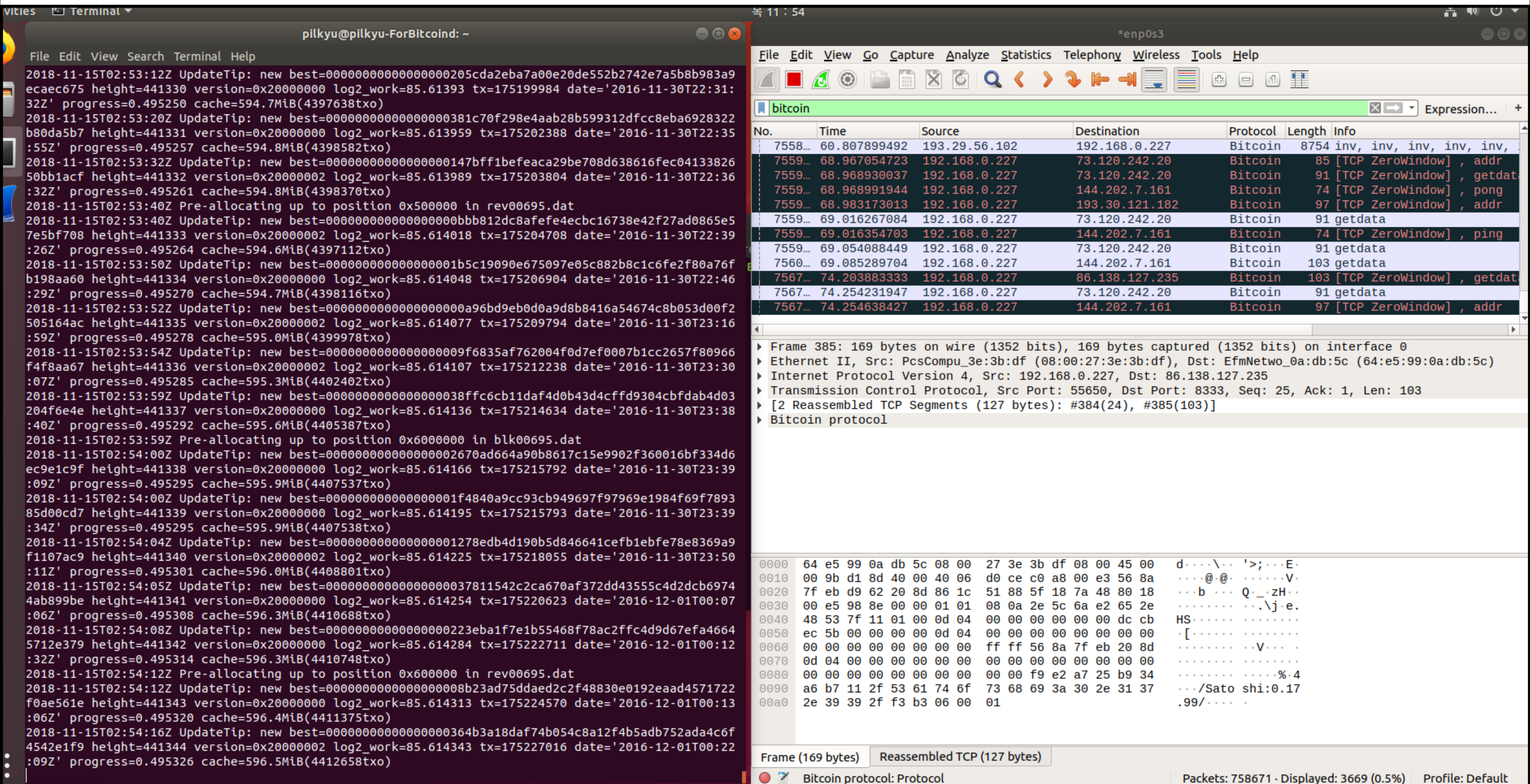
P2P Network & Synchronization

Pilkyu Jung

Index

- Peer Discovery
- Peer Connecting & Initial Block Download(IBD)
- Block Broadcasting
- Memory Pool
- Bloom filters

Downloading Data
& Analyzing Packet



Peer Discovery

- Programs don't know the IP addresses of any active full nodes.
- 제일 처음 시작할 때 client는 활성화된 full node의 IP를 전혀 모름.
- They query one or more DNS names (called DNS seeds).
- IP를 찾기 위해, hardcoding된 DNS seeds에 활성화된 full node IP를 요청함.
- The response should include one or more DNS A records with the IP addresses of full nodes.
- 요청에 대한 응답은 새로운 연결을 받아줄 full node의 IP, DNS A records를 포함함.

Peer Discovery

```
// Note that of those which support the service bits prefix, most only support a subset of
// possible options.
// This is fine at runtime as we'll fall back to using them as a oneshot if they don't support the
// service bits we want, but we should get them updated to support all service bits wanted by any
// release ASAP to avoid it where possible.
vSeeds.emplace_back("seed.bitcoin.sipa.be"); // Pieter Wuille, only supports x1, x5, x9, and xd
vSeeds.emplace_back("dnsseed.bluematt.me"); // Matt Corallo, only supports x9
vSeeds.emplace_back("dnsseed.bitcoin.dashjr.org"); // Luke Dashjr
vSeeds.emplace_back("seed.bitcoinstats.com"); // Christian Decker, supports x1 - xf
vSeeds.emplace_back("seed.bitcoin.jonasschnelli.ch"); // Jonas Schnelli, only supports x1, x5, x9, and xd
vSeeds.emplace_back("seed.btc.petertodd.org"); // Peter Todd, only supports x1, x5, x9, and xd
vSeeds.emplace_back("seed.bitcoin.sprovoost.nl"); // Sjors Provoost
```

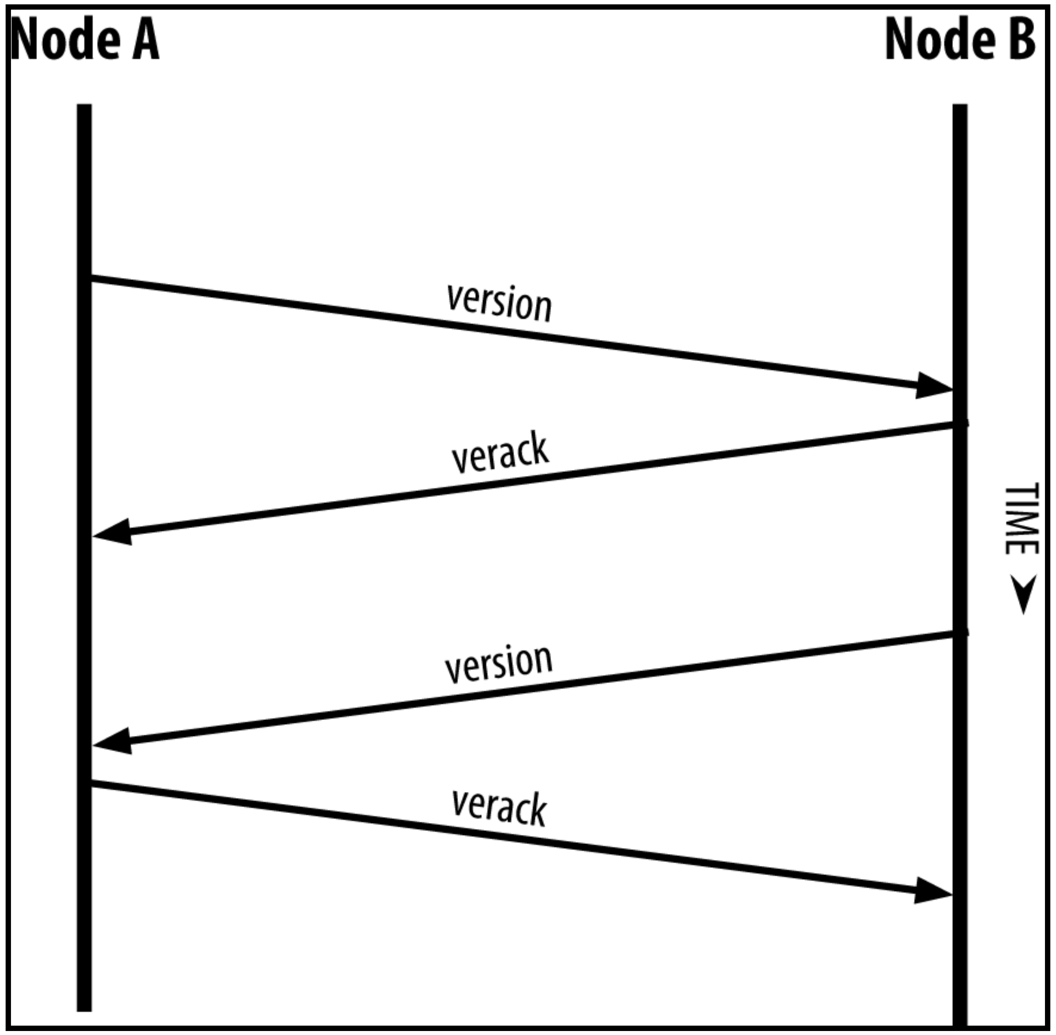
Peer Connecting

- Node A sends a protocol message ***version*** that contains various fields.

- 다양한 데이터(version, timestamp, nonce, start height 등)를 담고 있는 ***version*** 메세지를 보냄.

- Node B responds with its own ***version*** message followed by ***verack*** message, indicating that the connection has been established.

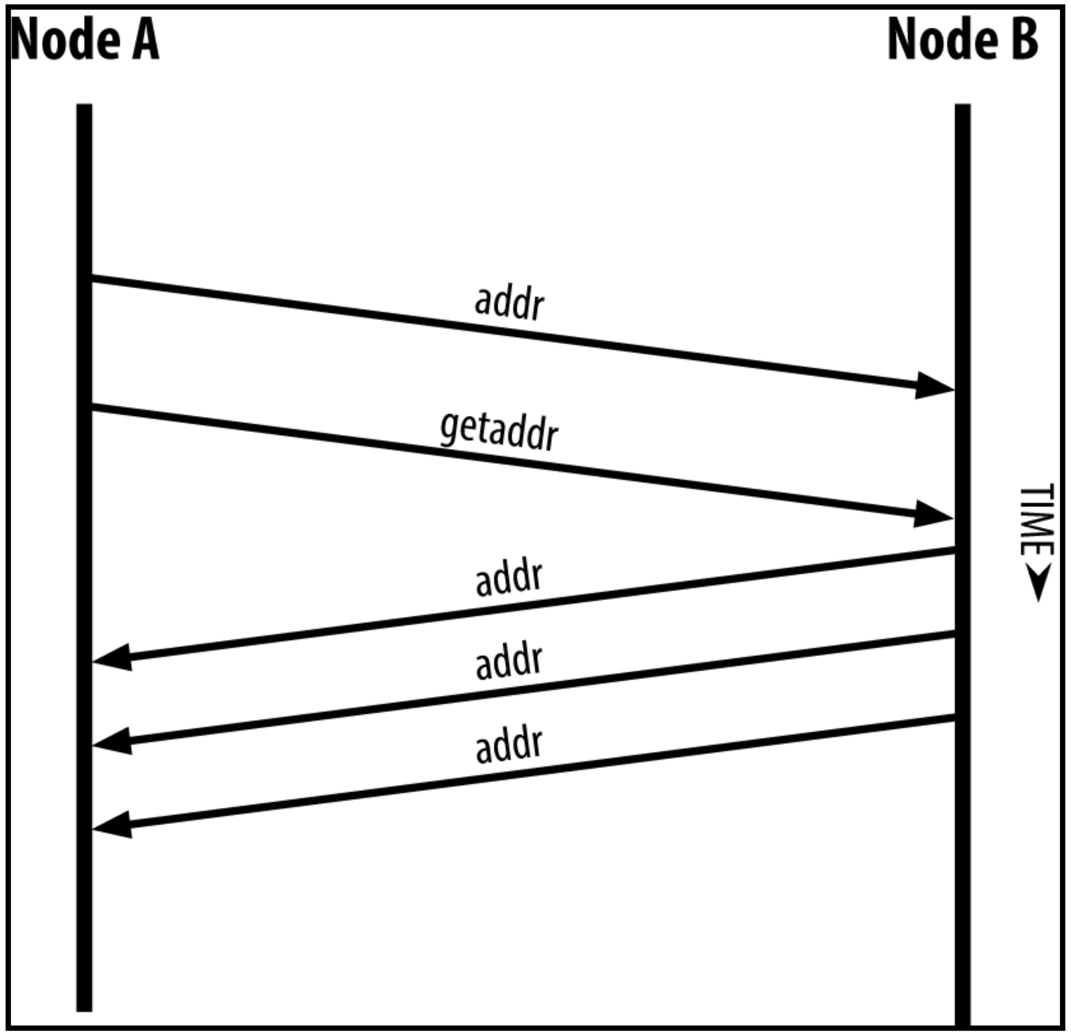
- 연결이 성립되었다는 의미의 ***verack*** 메세지를 보냄.



Peer Connecting

- After Node A received ***verack***, ***getaddr*** and ***addr*** messages are exchanged to find the peers that Node A does not know.

- It also, can send a ***ping*** message to see whether the connection is still live.



Initial Block Download (IBD, Initial sync)

- It must <u>download</u> and <u>validate all blocks</u> from block 1 (the block after the hardcoded genesis block) to the current tip of the best block chain.

- 아직 confirm 되지 않은 트랜잭션과 최근 채굴된 블록의 유효성검사를 하기전에, 현재의 마지막 블록을 알기위해서 반드시 genesis 블록 이후의 <u>모든 블록을 다운로드하고 유효성검사를 해야만 함</u>.

Initial Block Download(IBD)

- IBD can be used any time a large number of blocks need to be downloaded.
- IBD는 많은 양의 블록 다운로드가 필요한경우 언제든 사용될 수 있음.
- A node can use the IBD method to download all the blocks which were produced since the last time it was online.
- 노드가 온라인 상태였을 상태부터 이후의 모든 블록을 다운로드할 수 있음.

Blocks-First

Msg	***getblocks***	***inv***	***getdata***	***block***
From -> To	IBD -> Sync	Sync -> IBD	IBD -> Sync	Sync -> IBD

Blocks-First

- At the first time, node doesn't have any blocks.
- 제일 처음의 노드는 아무런 블록이 없는 상태임.
- It only has <u>a single block</u> in its local best block chain—the hardcoded genesis block (block 0).
- 단 하나의 블록(하드코딩된 genesis 블록)만을 가지고 있음.
- This node chooses a remote peer, which is called the sync node.
- 노드는 sync node라 불리는 peer를 선택함.
- And sends it the <u>***getblocks***</u> message.
- <u>***getblocks***</u> 메세지를 보냄.

Blocks-First

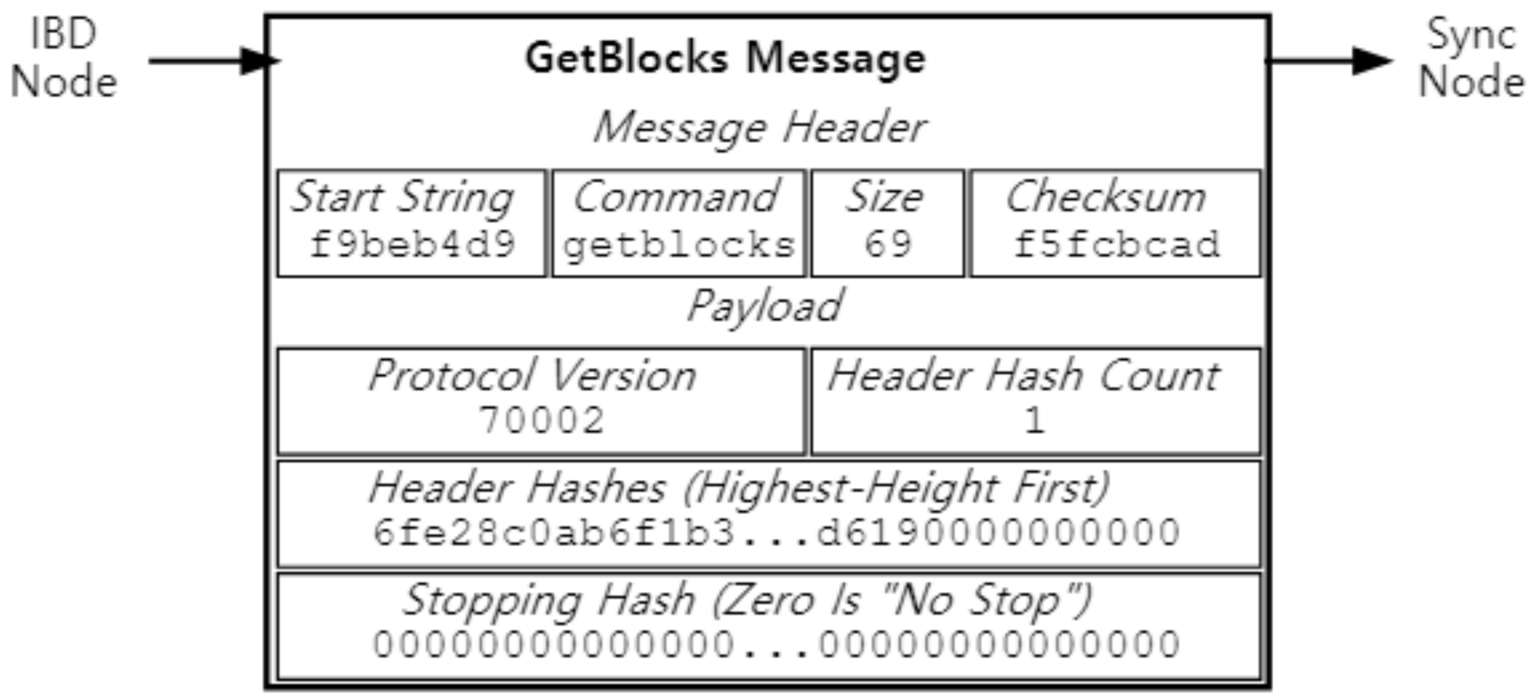
```
/**
 * Build the genesis block. Note that the output of its generation
 * transaction cannot be spent since it did not originally exist in the
 * database.
 *
 * CBlock(hash=00000ffd590b14, ver=1, hashPrevBlock=00000000000000,
 hashMerkleRoot=e0028e, nTime=1390095618, nBits=1e0ffff0, nNonce=28917698, vtx=1)
 *   CTransaction(hash=e0028e, ver=1, vin.size=1, vout.size=1, nLockTime=0)
 *     CTxIn(COutPoint(000000, -1), coinbase 04ffff001d01044c5957697265642030392
 f4a616e2f32303134205468652047726616e642045787065726696d656e7420476f6573204c697665
 3a204f76657273746f636b2e636f6d204973204e6f7720416363657074696e6720426974636f696
 e73)
 *     CTxOut(nValue=50.00000000, scriptPubKey=0xA9037BAC7050C479B121CF)
 *   vMerkleTree: e0028e
 */
static CBlock CreateGenesisBlock(uint32_t nTime, uint32_t nNonce, uint32_t nBits,
 int32_t nVersion, const CAmount& genesisReward)
{
    const char* pszTimestamp = "On 17/Jul/2018 Larry Fink confirmed to Reuters
    that BlackRock have a bitcoin working group ";
    const CScript genesisOutputScript = CScript() << ParseHex("04195a6bd56beda45
    d2a2cd5bc0ca329cb05362664a6ad81f3ce683db09596e1f58f33e7da0e265b4f569179fe6eb
    1d5ec9dc246514e911f5d7c917fb9a31eb30e") << OP_CHECKSIG;
    return CreateGenesisBlock(pszTimestamp, genesisOutputScript, nTime, nNonce,
     nBits, nVersion, genesisReward);
}
```

Blocks-First

```
static CBlock CreateGenesisBlock(const char* pszTimestamp, const CScript&
genesisOutputScript, uint32_t nTime, uint32_t nNonce, uint32_t nBits, int32_t
nVersion, const CAmount& genesisReward)
{
    CMutableTransaction txNew;
    txNew.nVersion = 1;
    txNew.vin.resize(1);
    txNew.vout.resize(1);
    txNew.vin[0].scriptSig = CScript() << 486604799 << CScriptNum(4)
     << std::vector<unsigned char>((const unsigned char*)pszTimestamp, (const
     unsigned char*)pszTimestamp + strlen(pszTimestamp));
    txNew.vout[0].nValue = genesisReward;
    txNew.vout[0].scriptPubKey = genesisOutputScript;

    CBlock genesis;
    genesis.nTime    = nTime;
    genesis.nBits    = nBits;
    genesis.nNonce   = nNonce;
    genesis.nVersion = nVersion;
    genesis.vtx.push_back(txNew);
    genesis.hashPrevBlock.SetNull();
    genesis.hashMerkleRoot = BlockMerkleRoot(genesis);
    return genesis;
}
```

Blocks-First



First getblocks message sent from Initial Blocks Download (IBD) node

Blocks-First

- New node sends the header hash of the only block it has, the genesis block and the stop hash field (to all zeroes to request a maximum-size response).
- 새로운 노드는 헤더(하드코딩된 genesis 블록)의 해시값과 마지막 해시 값을 보냄.

Blocks-First

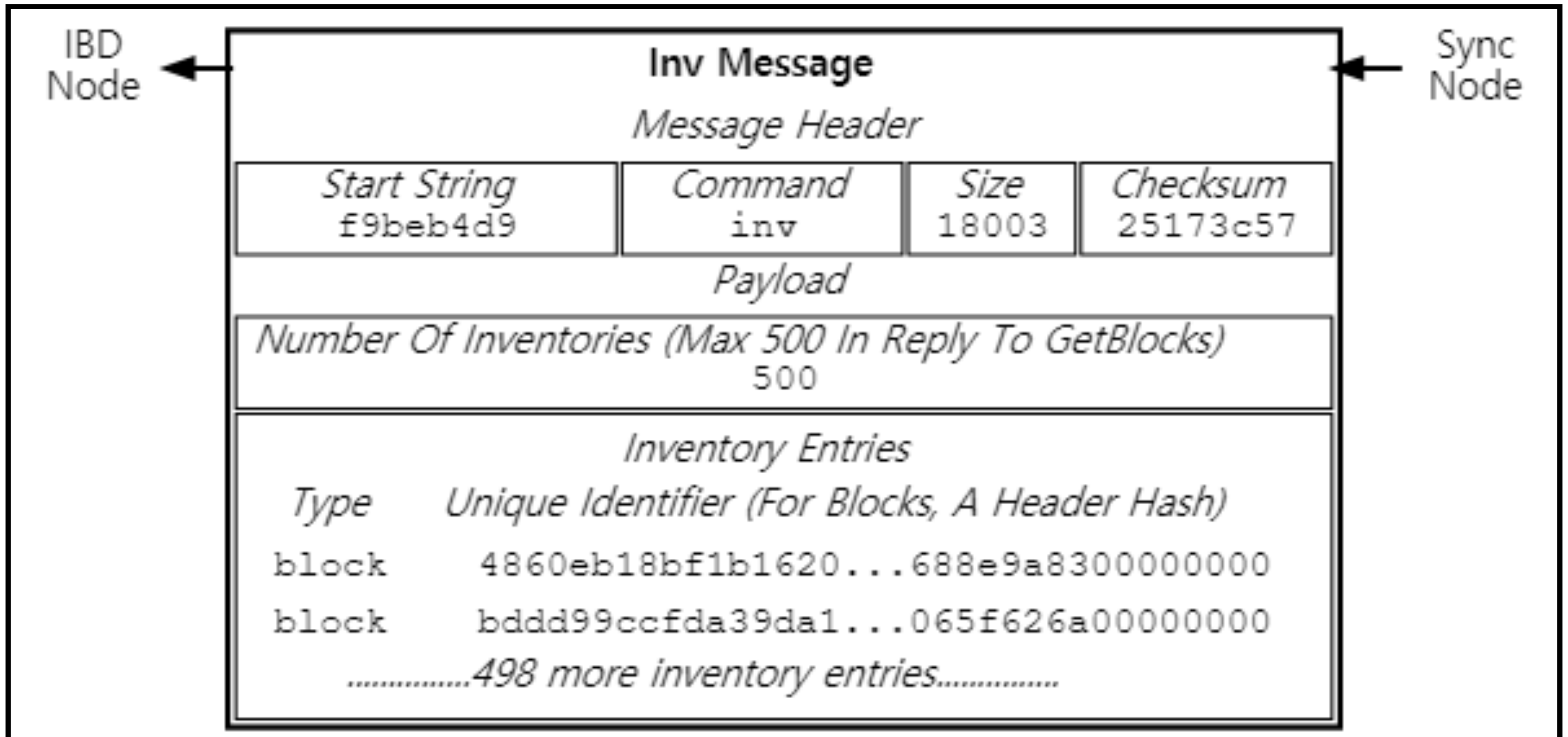
- The sync node takes the first (and only) header hash and searches.

- sync 노드는 자신의 저장소에 저장된 블록들에서 node로부터 받은 헤더 해시값을 찾음.

- It finds that block 0 matches, so it replies with 500 block inventories (the maximum response to a ***getblocks*** message) starting from block 1 that is the ***inv*** message.

- 일치하는 블록을 찾으면 ***getblocks*** 메세지에 응답하는 ***inv*** 메세지를 보냄.

Blocks-First



First inv message reply sent to Initial Blocks Download (IBD) node

Blocks-First

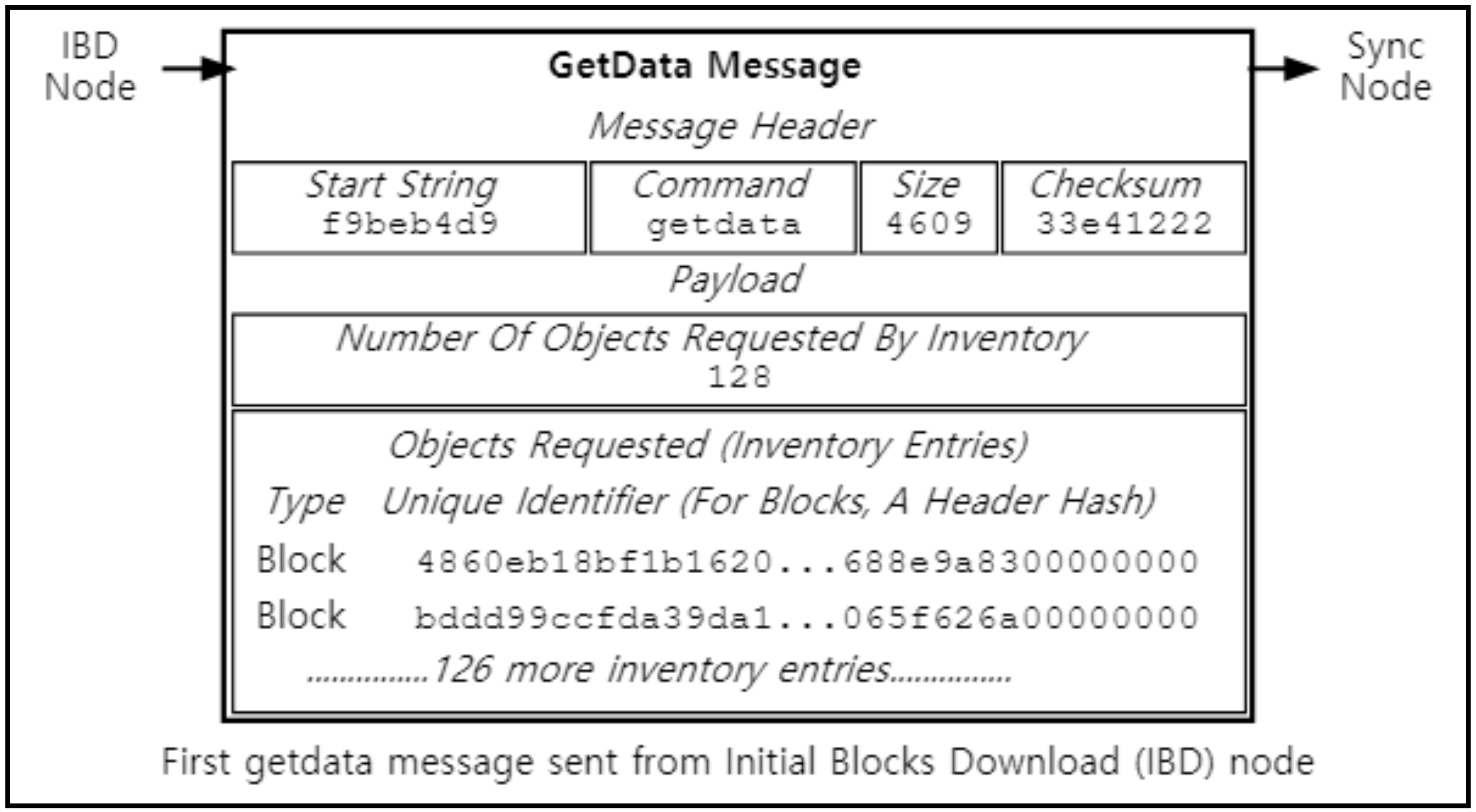
- Each inventory contains a type field and the unique identifier for an instance of the object. For blocks, the unique identifier is a hash of the block's header.

- Inventory들은 네트워크 상의 정보로써의 특별한 식별자들이며, 각각의 inventory는 type 필드와 각 객체의 instance로써의 특별한 식별자를 포함하고 있음. 블록의 경우 블록의 헤더가 특별한 식별자임.

- And the block inventories appear in the ***inv*** message in the same order they appear in the block chain.

- ***inv*** message에 나타나는 블록 inventory들의 순서는 실제 블록체인 내의 블록 순서와 같음.

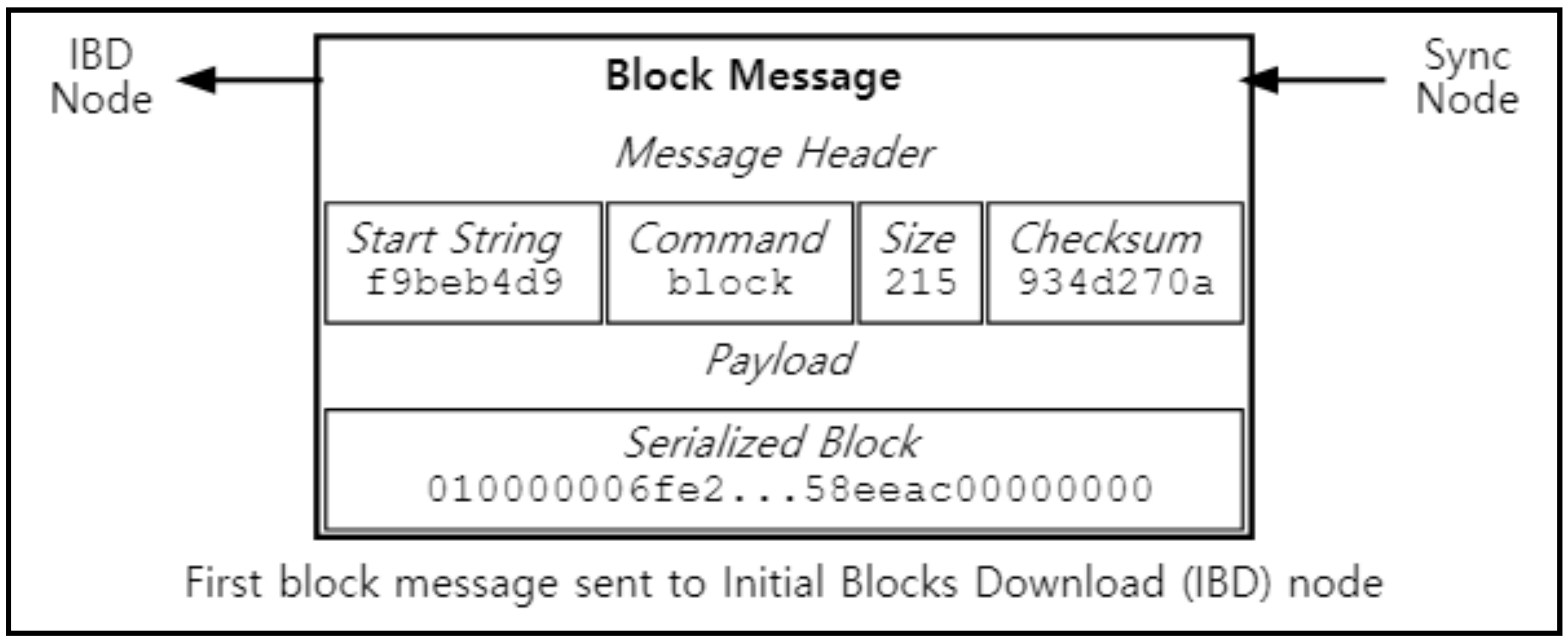
Blocks-First



- The IBD node uses the received inventories to request 128 blocks from the sync node in the ***getdata*** message.

- IBD 노드가 ***getdata*** 메세지에 128블록의 요청을 담아 전송함.

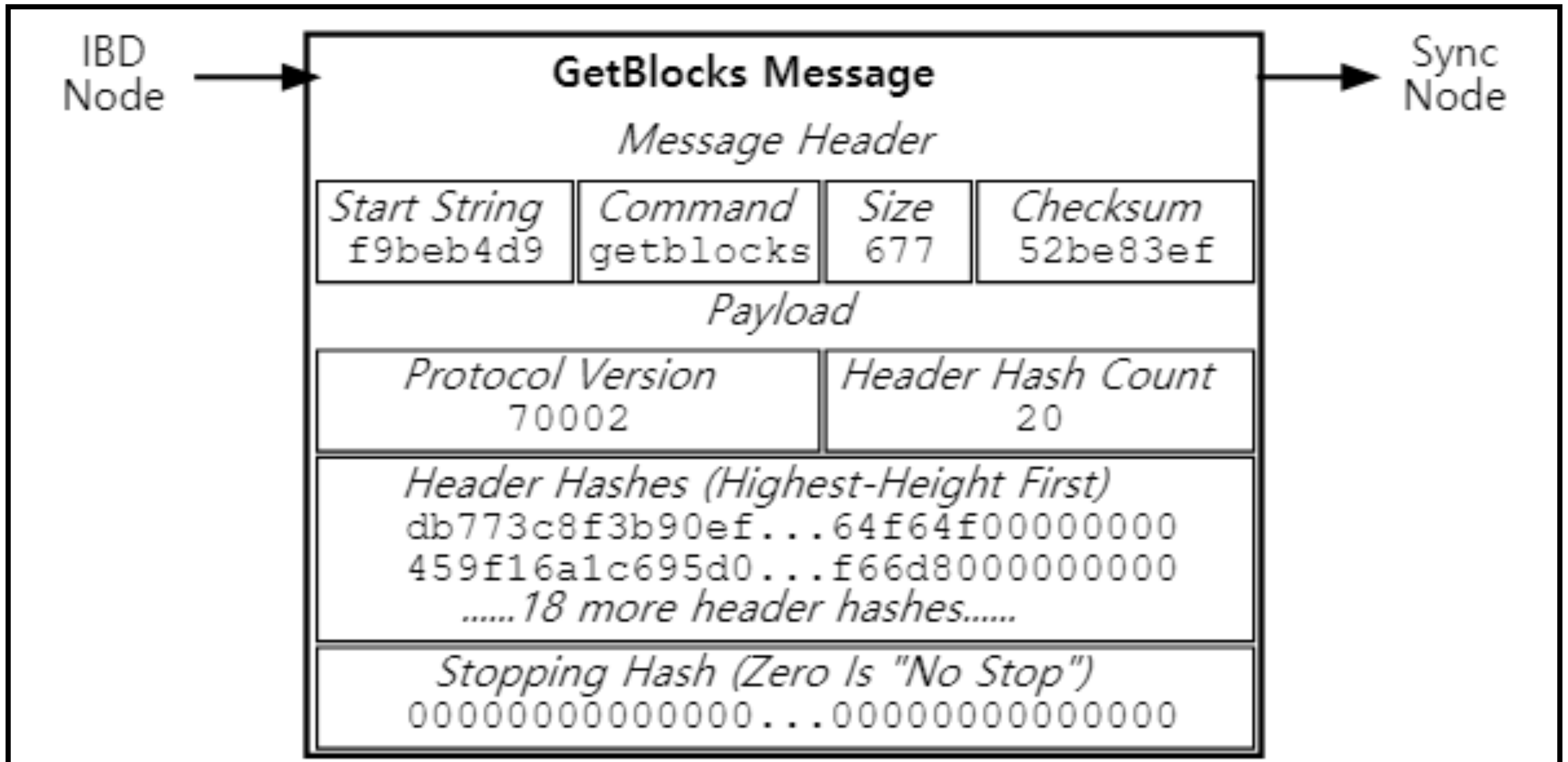
Blocks-First



- Upon receipt of the ***getdata*** message, the sync node replies with each of the blocks requested.

- 상위 ***getdata*** 메세지에서 sync 노드는 요청된 블록에 대해 각각의 답변을 ***block*** 메세지에 담아서 보냄.

Blocks-First



Second getblocks message sent from Initial Blocks Download (IBD) node

- ***getblock***, ***inv***, ***getdata***, ***block***을 반복함.

Blocks-First

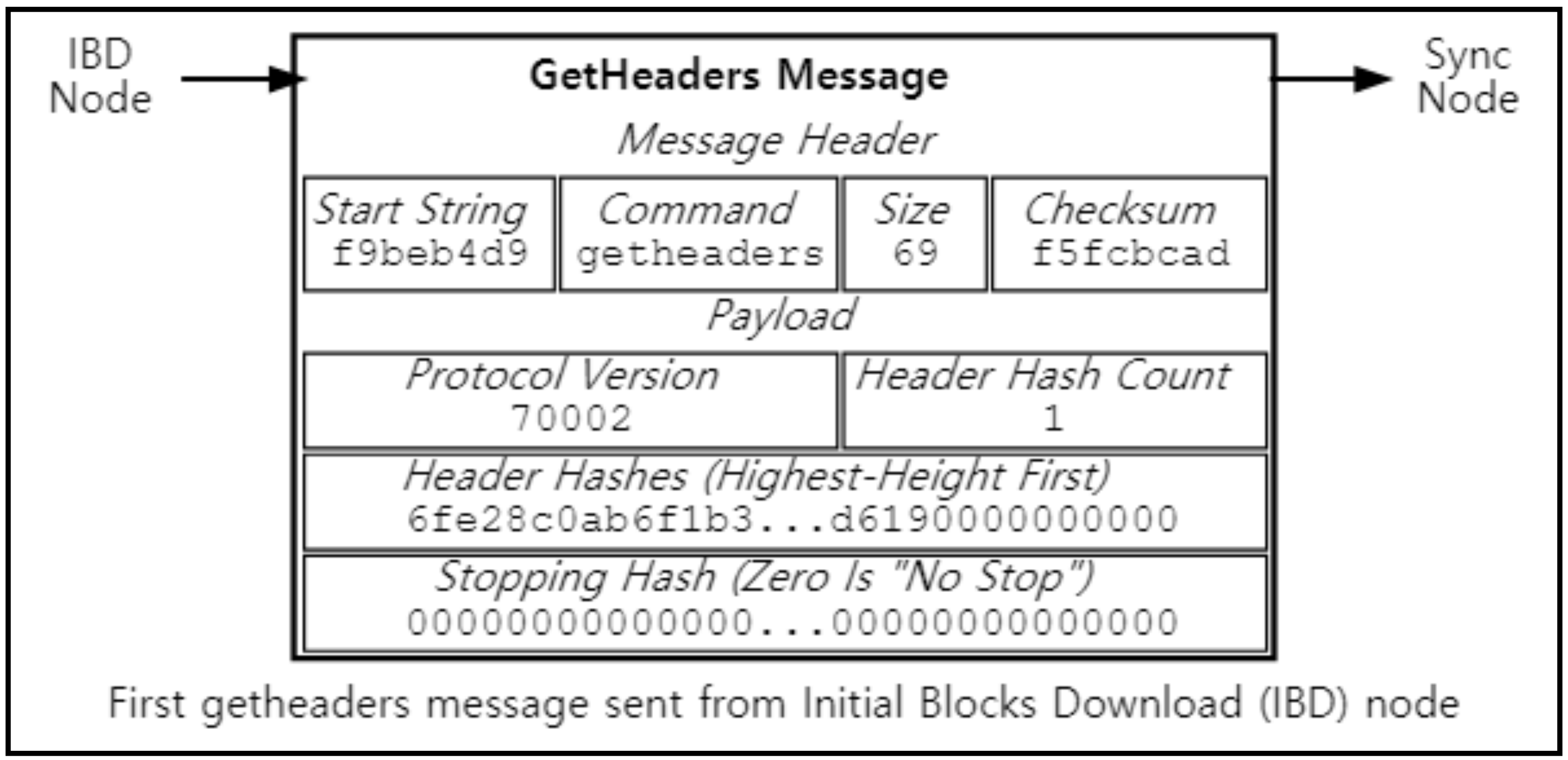
- The IBD node will request more inventories with another ***getblocks*** message—and the cycle will repeat until the IBD node is synced to the tip of the block chain.

- IBD 노드는 ***inv*** 메세지를 받은 후 ***getdata*** 메세지로써 블록을 요청함. 아울러 sync 노드는 ***block*** 메세지로써 응답을 함. 이 후 IBD 노드는 또다른 ***getblocks*** 메세지를 통해 더 많은 inventory를 요청함. 그리고 이 사이클은 IDB 노드가 동기화를 마칠 때까지 반복함.

Headers-First

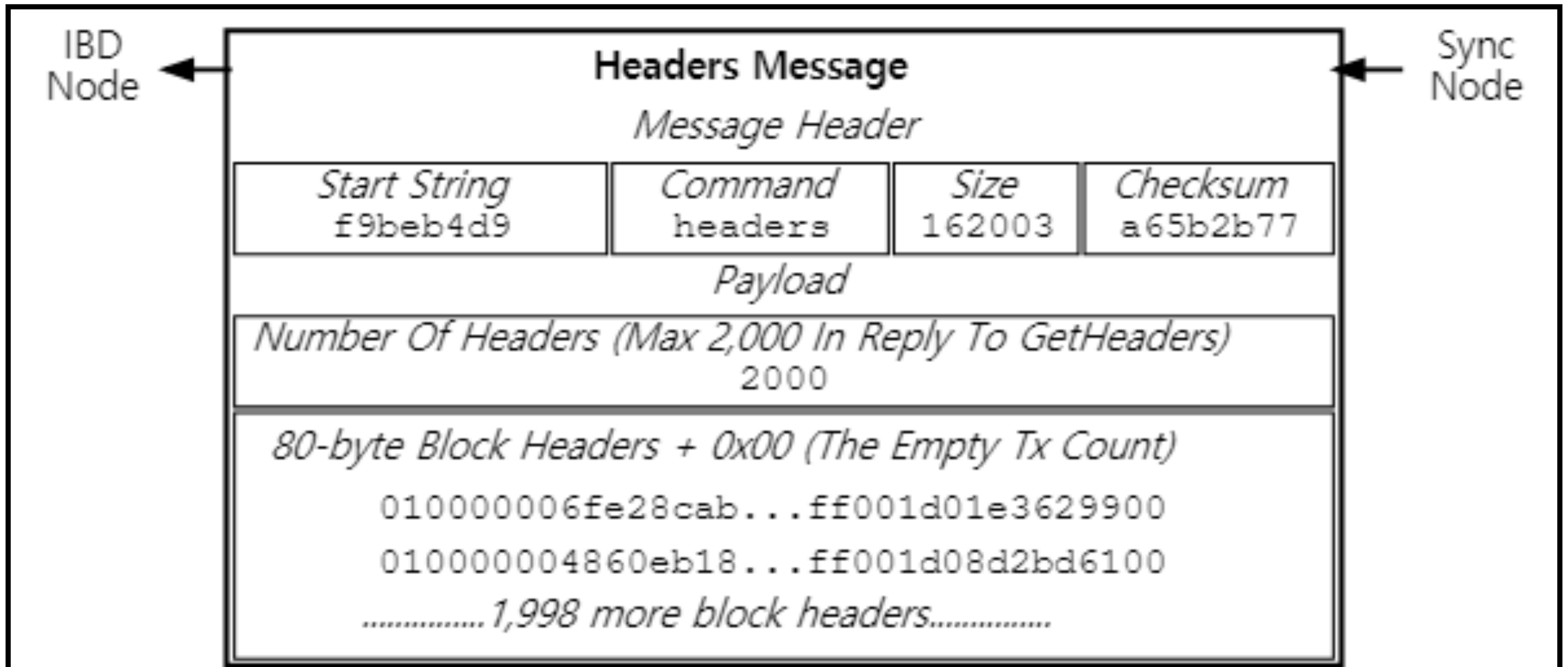
Msg	***getheaders***	***headers***	***getdata***	***block***
From -> To	IBD -> Sync	Sync -> IBD	IBD -> *Many*	*Many* -> IBD

Headers-First



- The first time a node is started, it only has a single block in its local best block chain and it sends the ***getheaders*** message to Sync Node.

Headers-First



First headers message reply sent to Initial Blocks Download (IBD) node

- The sync node takes the first (and only) header hash and searches. It finds that block 0 matches, so it replies with <u>2,000 header</u> (the maximum response) starting from block 1. It sends these <u>header hashes</u> in the headers message.

Headers-First

- The IBD node can partially validate these block headers by ensuring that all fields follow consensus rules. The hash of the header is below the target threshold according to the nBits field.

- IBD 노드는 consensus rules과 헤더의 해시값이 target threshold보다 작아야한다는 것을 통해서 블록헤더들을 부분적으로 나눠서 유효성검사를 할 수 있음.

- After the IBD node has partially validated the block headers, it can do two things in parallel:

- IBD 노드가 부분적으로 유효성 검사를 할 수 있게됨으로써, 동시에 두가지 액션을 취할 수 있음.

1. Download More Headers:

- The IBD node can send another ***getheaders*** message to the sync node to request the next 2,000 headers on the best header chain.

- IBD는 best header chain에서 다음 2,000개의 헤더를 받기 위해 또다른 ***getheaders*** 메세지를 보낼 수 있음.

- Those headers can be immediately validated and another batch requested repeatedly until a headers message is received from the sync node with fewer than 2,000 headers.

- 이러한 헤더들은 즉시 유효성검사를 해볼 수 있고, 2,000개 보다 더 적은 양으로써 다 받아질 때까지 반복적으로 요청될 수 있음. 이 것은 더 이상 제공할 헤더가 없다는 것을 의미함.

1. Download More Headers:

- Once the IBD node receives a headers message with fewer than 2,000 headers from the sync node, it sends a ***getheaders*** message to each of its outbound peers to get their view of best header chain.

- 2,000개 보다 더 적은양의 헤더를 받으면, 노드는 최종헤더임을 확인하기 위해 각 다른 노드들에게 ***getheaders*** 메세지를 보냄.

- By comparing the responses, it can easily determine if the headers it has downloaded belong to the best header chain reported by any of its outbound peers.

- 응답을 비교해 봄으로써, 다른 노드들에 의해 보고된 최종에 속함의 유무를 쉽게 결정 할 수 있음. 이 것은 checkpoints를 사용하지 않는다 하더라도 불온한 sync 노드를 빠르게 찾을 수 있다는 것을 의미함.

2. Download Blocks:

- While the IBD node continues downloading headers, and after the headers finish downloading, the IBD node will request and download each block.
- IBD 노드가 헤더를 다운로드하는 중이거나 끝난 상황일 동안에, IBD 노드는 각 블록 다운로드를 요청함.
- This allows it to fetch blocks in parallel and avoid having its download speed constrained to the upload speed of a single sync node.
- 이 것은 블록들을 동시에 패치하는 것과 다운로드 속도가 sync 노드의 업로드 속도를 방해하는 것을 피하게 함.

Block Broadcasting

- A relay node may skip the round trip overhead of an inv message followed by getheaders by instead immediately sending a headers message containing the full header of the new block.

- 릴레이노드는 새로운 블록의 full 헤더를 포함한 headers message (***sendheaders***)를 즉시 보냄.

- A HF peer receiving this message will partially validate the block header as it would during headers-first IBD, then request the full block contents with a ***getdata*** message if the header is valid.

- HF(Header First) 피어는 이 메세지를 받고 headers-first IBD를 할 때처럼, 부분적인 유효성 검사를 실행함.

- The relay node then responds to the ***getdata*** request with the full or filtered block data in a ***block*** or ***merkleblock*** message, respectively.

- 만약 헤더가 유효하면 ***getdata*** 메세지로써 full 블록을 요청함. 이 후 relay 노드는 요청에 맞는 메세지를 보냄.

WireShark

- (ip.src == *SourceIP* and ip.dst == *DestinationIP* and bitcoin) or (ip.src == *SourceIP* and ip.dst == *DestinationIP* and bitcoin)

`(ip.src == 192.168.0.227 and ip.dst == 185.25.48.160 and bitcoin) or (ip.src == 185.25.48.160 and ip.dst == 192.168.0.227 and bitcoin)`

No.	Time	Source	Destination	Protocol	Length	Info
467	96.878837228	192.168.0.227	185.25.48.160	Bitcoin	169	version
470	97.180425083	185.25.48.160	192.168.0.227	Bitcoin	192	version
472	97.180444695	185.25.48.160	192.168.0.227	Bitcoin	90	verack
474	97.180454508	185.25.48.160	192.168.0.227	Bitcoin	98	ping
476	97.180462673	185.25.48.160	192.168.0.227	Bitcoin	121	addr
478	97.181072867	185.25.48.160	192.168.0.227	Bitcoin	1119	getheaders

(ip.src == 192.168.0.227 and ip.dst == 90.171.117.252 and bitcoin) or (ip.src == 90.171.117.252 and ip.dst == 192.168.0.227 and bitcoin)

No.	Time	Source	Destination	Protocol	Length	Info
7248	166.554620140	192.168.0.227	90.171.117.252	Bitcoin	157	version
7276	166.884857671	90.171.117.252	192.168.0.227	Bitcoin	156	version
7278	166.886779732	90.171.117.252	192.168.0.227	Bitcoin	78	verack
7664	169.251565173	192.168.0.227	90.171.117.252	Bitcoin	78	verack
7665	169.251595640	192.168.0.227	90.171.117.252	Bitcoin	78	getaddr
7773	169.580895797	90.171.117.252	192.168.0.227	Bitcoin	78	sendheaders
7779	169.580946039	90.171.117.252	192.168.0.227	Bitcoin	63	sendcmpct
7783	169.580960863	90.171.117.252	192.168.0.227	Bitcoin	63	sendcmpct
7787	169.580973304	90.171.117.252	192.168.0.227	Bitcoin	62	ping
7791	169.580995163	90.171.117.252	192.168.0.227	Bitcoin	85	addr
7795	169.581866233	90.171.117.252	192.168.0.227	Bitcoin	1083	getheaders
7799	169.581883361	90.171.117.252	192.168.0.227	Bitcoin	62	feefilter
8199	171.849865568	192.168.0.227	90.171.117.252	Bitcoin	78	sendheaders
8201	171.849923619	192.168.0.227	90.171.117.252	Bitcoin	63	sendcmpct
8203	171.849974813	192.168.0.227	90.171.117.252	Bitcoin	63	sendcmpct
8205	171.850032520	192.168.0.227	90.171.117.252	Bitcoin	62	ping
8207	171.850104279	192.168.0.227	90.171.117.252	Bitcoin	1083	getheaders
8273	172.176960512	192.168.0.227	90.171.117.252	Bitcoin	62	feefilter
8274	172.177602157	90.171.117.252	192.168.0.227	Bitcoin	62	pong
8280	172.179607793	90.171.117.252	192.168.0.227	Bitcoin	217	headers
8825	175.530072096	90.171.117.252	192.168.0.227	Bitcoin	1027	inv
9760	182.711324485	90.171.117.252	192.168.0.227	Bitcoin	775	inv
9844	183.285014264	192.168.0.227	90.171.117.252	Bitcoin	62	pong
10064	184.034705505	90.171.117.252	192.168.0.227	Bitcoin	857	addr
11028	185.343100008	192.168.0.227	90.171.117.252	Bitcoin	631	getdata
41803	196.567550508	192.168.0.227	90.171.117.252	Bitcoin	91	getdata
50390	200.734509279	90.171.117.252	192.168.0.227	Bitcoin	4370	block [TCP segment of a reassembled PDU]
50480	200.753187182	192.168.0.227	90.171.117.252	Bitcoin	91	getdata
65788	204.465870241	192.168.0.227	90.171.117.252	Bitcoin	91	getdata
73246	206.289631519	90.171.117.252	192.168.0.227	Bitcoin	4138	block [TCP segment of a reassembled PDU]
74649	206.666912743	90.171.117.252	192.168.0.227	Bitcoin	1315	inv
79763	209.023913625	90.171.117.252	192.168.0.227	Bitcoin	2594	block [TCP segment of a reassembled PDU]

> Frame 8273: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
> Ethernet II, Src: PcsCompu_3e:3b:df (08:00:27:3e:3b:df), Dst: EfmNetwo_0a:db:5c (64:e5:99:0a:db:5c)
> Internet Protocol Version 4, Src: 192.168.0.227, Dst: 90.171.117.252
> Transmission Control Protocol, Src Port: 54050, Dst Port: 8333, Seq: 1375, Ack: 1437, Len: 8
> [2 Reassembled TCP Segments (32 bytes): #8208(24), #8273(8)]
Bitcoin protocol
Packet magic: 0xf9beb4d9
Command name: feefilter

Memory(Mem) Pool

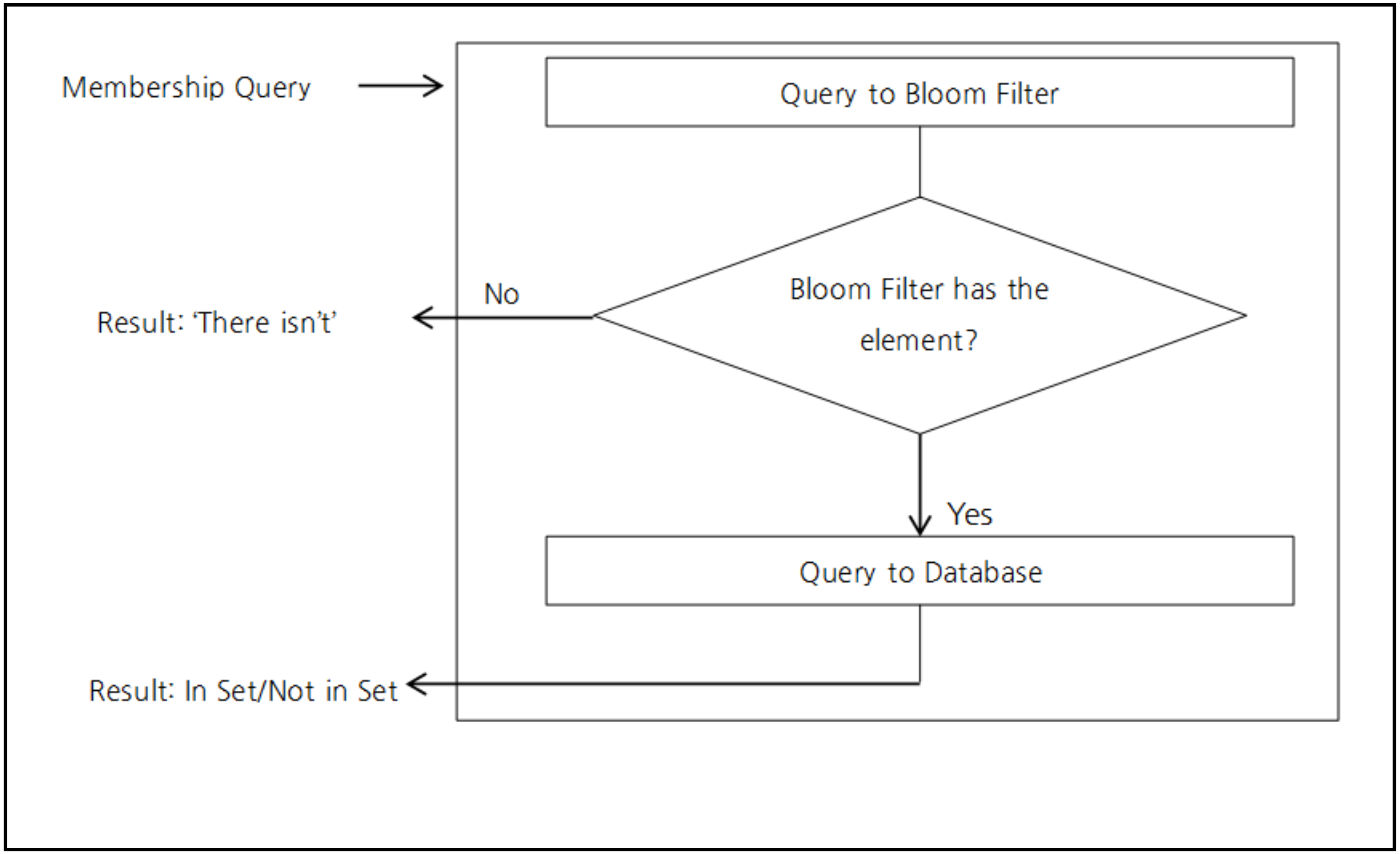
- Full peers may keep track of unconfirmed transactions which are eligible to be included in the next block. This is essential for miners who will actually mine some or all of those transactions.

- Full 노드는 다음 블록에 포함될 아직 confirm 되지 않은 트랜잭션을 가지고 있음. 이 것은 실제 채굴을 하기 위한 기초가 됨.

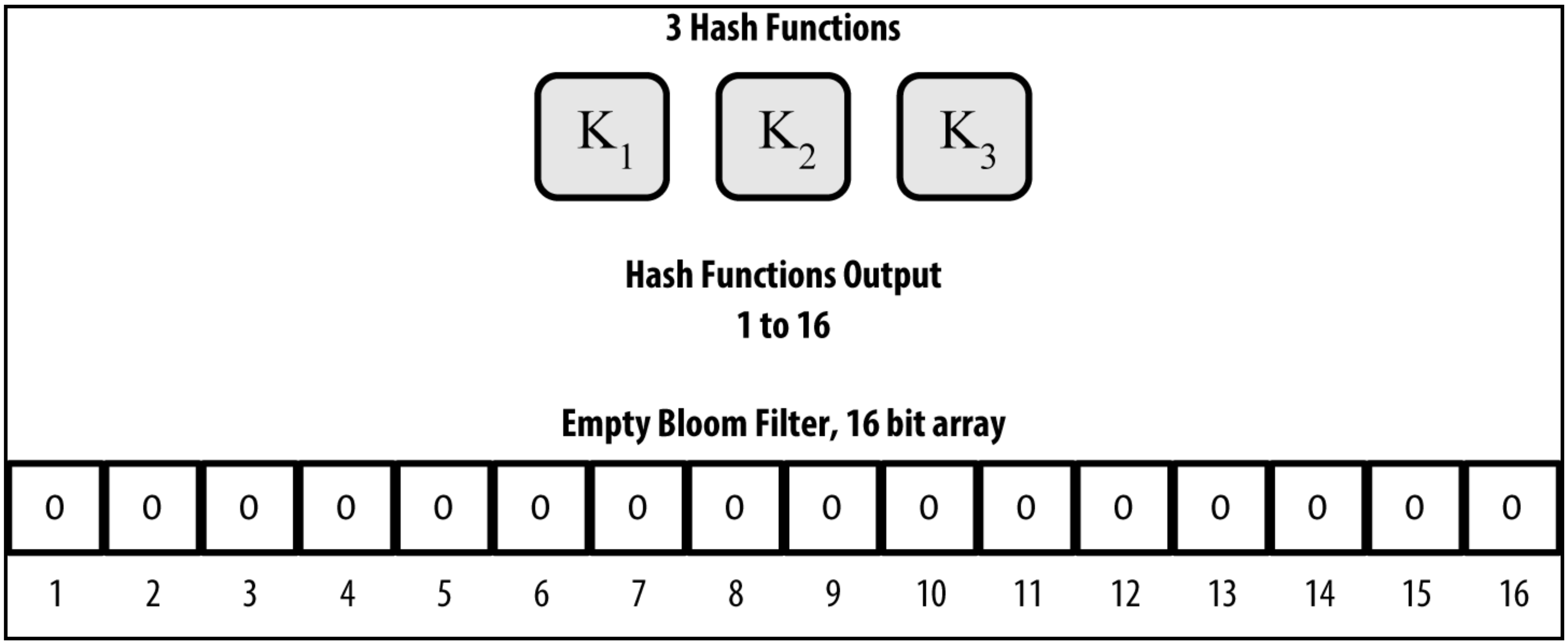
- SPV clients can’t independently verify that a transaction hasn’t yet been included in a block, so they can’t know which transactions are eligible to be included in the next block.

- SPV 노드는 독립적으로 다음 블록에 포함될 트랜잭션의 유효성검사를 할 수 없기때문에, 다음 블록에 포함될 트랜잭션을 알 수가 없음.

Bloom Filter



Bloom Filter



False positive probability

- After all members of S have been hashed to a Bloom filter, the probability that a specific bit is still 0 is

$$p' = (1 - \frac{1}{m})^{kn} \simeq e^{-kn/m} = p$$

- For a non member, it may be found to be a member of S (all of its k bits are nonzero) with *false positive probability*

$$(1 - p')^k \simeq (1 - p)^k$$

False positive probability (cont.)

- Define

$$f' = (1-p')^k = (1-(1-\frac{1}{m})^{kn})^k$$

$$f = (1-p)^k = (1-e^{-kn/m})^k$$

- Two competing forces as k increases
  - Larger k -> $(1-p')^k$ is smaller for a fixed p'
  - Larger k -> p'= $(1-1/m)^{kn}$ is smaller -> 1-p' larger

Optimal number k from derivative

Rewrite f as

$$f = \exp(\ln(1-e^{-kn/m})^k) = \exp(k\ln(1-e^{-kn/m}))$$

Let $\boxed{g = k\ln(1-e^{-kn/m})}$

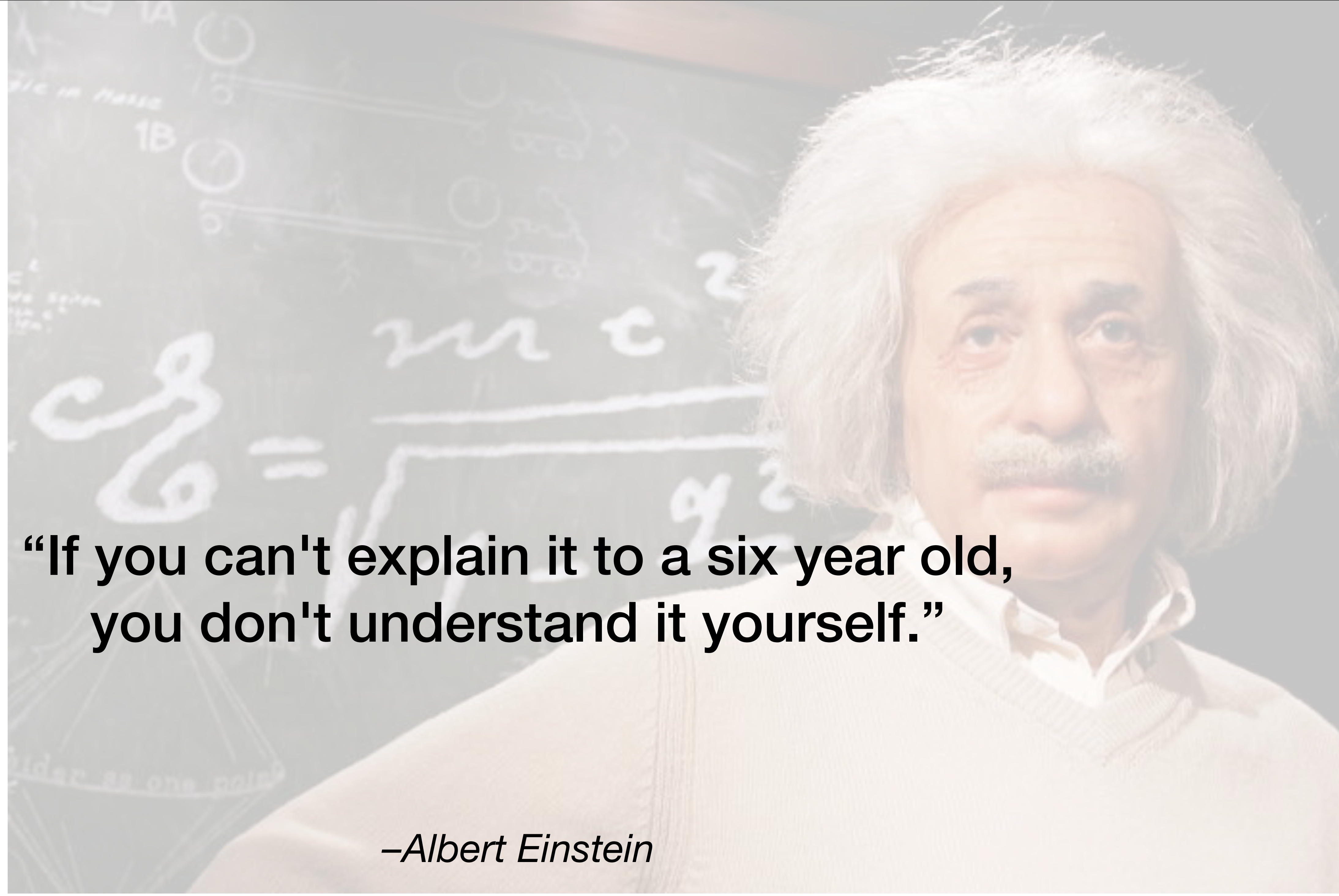
Minimizing g will minimize $f = \exp(g)$

$$\frac{\partial g}{\partial k} = \ln(1-e^{-kn/m}) + \frac{k}{1-e^{-kn/m}} \frac{\partial(1-e^{-kn/m})}{\partial k}$$

$$= \ln(1-e^{-kn/m}) + \frac{k}{1-e^{-kn/m}} \frac{n}{m} e^{-kn/m} = -\ln(2) + \ln(2) = 0$$

if we plug $\boxed{k = (m/n)\ln 2}$ which is optimal

(It is in fact a global optimum)



"If you can't explain it to a six year old,
you don't understand it yourself."

–*Albert Einstein*

References

- Andreas Antonopoulos, 2017, Mastering Bitcoin (2nd Edition) – Programming the open blockchain
- Imran Bashir, 2017, Mastering Blockchain (1st Edition)
- https://bitcoin.org/en/developer-guide#peer-discovery
- https://bitcoin.org/en/developer-reference#constants-and-defaults
- https://d2.naver.com/helloworld/749531
- http://www.cs.utexas.edu/users/lam/396m/slides/Bloom_filters.pdf