Alma Mater Studiorum · Università di Bologna

Department of computer science and engineering - DISI

IPFS and Filecoin

Lorello Luca Salvatore Tsiotas Giorgio

Introduction

Definitions (1/4)

IPFS Global versioned peer-to-peer file system with no single-point-of-failure

Filecoin Incentive layer which rewards IPFS users with a cryptocurrency (FIL)

Introduction

Definitions (1/4)

IPFS Global versioned peer-to-peer file system with no single-point-of-failure $\rightsquigarrow$ Aims to offer a good Internet experience without relying on a central server (eg. it can deployed to other planets with no link to Earth) and avoiding censorship

Filecoin Incentive layer which rewards IPFS users with a cryptocurrency (FIL)

Introduction

Definitions (1/4)

IPFS Global versioned peer-to-peer file system with no single-point-of-failure $\rightsquigarrow$ Aims to offer a good Internet experience without relying on a central server (eg. it can deployed to other planets with no link to Earth) and avoiding censorship

Filecoin Incentive layer which rewards IPFS users with a cryptocurrency (FIL) $\rightsquigarrow$ Aims to compete with traditional file storage businesses by offering extremely low fees

Introduction

IPFS (2/4)

- The file is split into blocks, each of which (along with the entire file) is associated with its cryptographic hash
- Each node stores only blocks it's interested in
- A request is resolved by finding which nodes store the blocks composing the file and retrieving them
- IPNS (Interplanetary Naming System) allows to associate filenames with hashes
- IPLD (Interplanetary Linked Data) is an unified namespace for any hash-based data structure

Introduction

IPFS (2/4)

- The file is split into blocks, each of which (along with the entire file) is associated with its cryptographic hash $\rightsquigarrow$ Blocks avoid duplicating parts which are shared among different files
- Each node stores only blocks it's interested in
- A request is resolved by finding which nodes store the blocks composing the file and retrieving them
- IPNS (Interplanetary Naming System) allows to associate filenames with hashes
- IPLD (Interplanetary Linked Data) is an unified namespace for any hash-based data structure

Introduction

IPFS (2/4)

- The file is split into blocks, each of which (along with the entire file) is associated with its cryptographic hash $\rightsquigarrow$ Blocks avoid duplicating parts which are shared among different files
- Each node stores only blocks it's interested in $\rightsquigarrow$ Additionally each node stores indexing informations for retrieval
- A request is resolved by finding which nodes store the blocks composing the file and retrieving them

- IPNS (Interplanetary Naming System) allows to associate filenames with hashes
- IPLD (Interplanetary Linked Data) is an unified namespace for any hash-based data structure

Introduction

IPFS (2/4)

- The file is split into blocks, each of which (along with the entire file) is associated with its cryptographic hash $\rightsquigarrow$ Blocks avoid duplicating parts which are shared among different files
- Each node stores only blocks it's interested in $\rightsquigarrow$ Additionally each node stores indexing informations for retrieval
- A request is resolved by finding which nodes store the blocks composing the file and retrieving them $\rightsquigarrow$ A node may decide to cache content for faster retrieval and let the garbage collector free space periodically
- IPNS (Interplanetary Naming System) allows to associate filenames with hashes
- IPLD (Interplanetary Linked Data) is an unified namespace for any hash-based data structure

Introduction

IPFS (2/4)

- The file is split into blocks, each of which (along with the entire file) is associated with its cryptographic hash $\rightsquigarrow$ Blocks avoid duplicating parts which are shared among different files
- Each node stores only blocks it's interested in $\rightsquigarrow$ Additionally each node stores indexing informations for retrieval
- A request is resolved by finding which nodes store the blocks composing the file and retrieving them $\rightsquigarrow$ A node may decide to cache content for faster retrieval and let the garbage collector free space periodically
- IPNS (Interplanetary Naming System) allows to associate filenames with hashes
- IPLD (Interplanetary Linked Data) is an unified namespace for any hash-based data structure $\rightsquigarrow$ Can be used to make an IPFS-based network stack

Introduction

IPFS (2/4)

- The file is split into blocks, each of which (along with the entire file) is associated with its cryptographic hash $\rightsquigarrow$ Blocks avoid duplicating parts which are shared among different files
- Each node stores only blocks it's interested in $\rightsquigarrow$ Additionally each node stores indexing informations for retrieval
- A request is resolved by finding which nodes store the blocks composing the file and retrieving them $\rightsquigarrow$ A node may decide to cache content for faster retrieval and let the garbage collector free space periodically
- IPNS (Interplanetary Naming System) allows to associate filenames with hashes
- IPLD (Interplanetary Linked Data) is an unified namespace for any hash-based data structure $\rightsquigarrow$ Can be used to make an IPFS-based network stack

Filecoin nodes are IPFS nodes as well, therefore IPLD-formatted data can be retrieved seamlessly

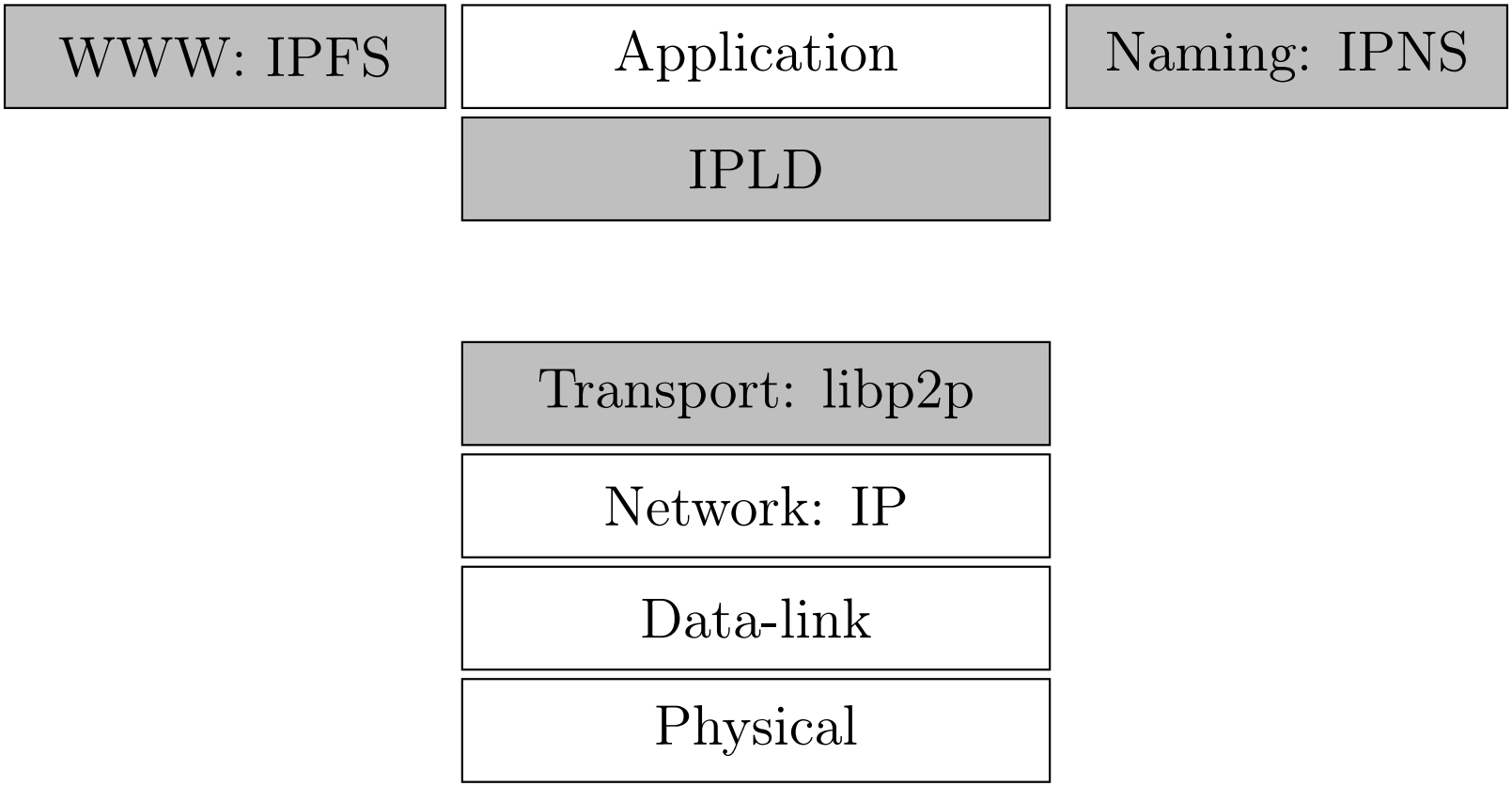
Introduction

IPFS-based Stack (3/4)

WWW: HTTP	Application	Naming: DNS
	Presentation	
	Session	
	Transport: TCP/UDP	
	Network: IP	
	Data-link	
	Physical	

Introduction

IPFS-based Stack (3/4)



Introduction

Filecoin (4/4)

- File storage is treated as an algorithmic market
- Reliable because it's built on top of mature technologies
- Supports any format

Introduction

Filecoin (4/4)

- File storage is treated as an algorithmic market $\rightsquigarrow$ Available storage and prices are not controlled by a single company, instead they depend on a supply-demand model
- Reliable because it's built on top of mature technologies
- Supports any format

Introduction

Filecoin (4/4)

- File storage is treated as an algorithmic market ⇝ Available storage and prices are not controlled by a single company, instead they depend on a supply-demand model
- Reliable because it's built on top of mature technologies ⇝ libp2p, IPLD and IPFS
- Supports any format

Introduction

Filecoin (4/4)

- File storage is treated as an algorithmic market ⇝ Available storage and prices are not controlled by a single company, instead they depend on a supply-demand model
- Reliable because it's built on top of mature technologies ⇝ libp2p, IPLD and IPFS
- Supports any format ⇝ Future-proof data types

Introduction

Filecoin (4/4)

- File storage is treated as an algorithmic market ⇝ Available storage and prices are not controlled by a single company, instead they depend on a supply-demand model
- Reliable because it's built on top of mature technologies ⇝ libp2p, IPLD and IPFS
- Supports any format ⇝ Future-proof data types

User

- Pays miners to store and retrieve files
- Can auction storage to miners who satisfy its requirements
- Has guarantees thanks to proofs on the blockchain

Storage miner

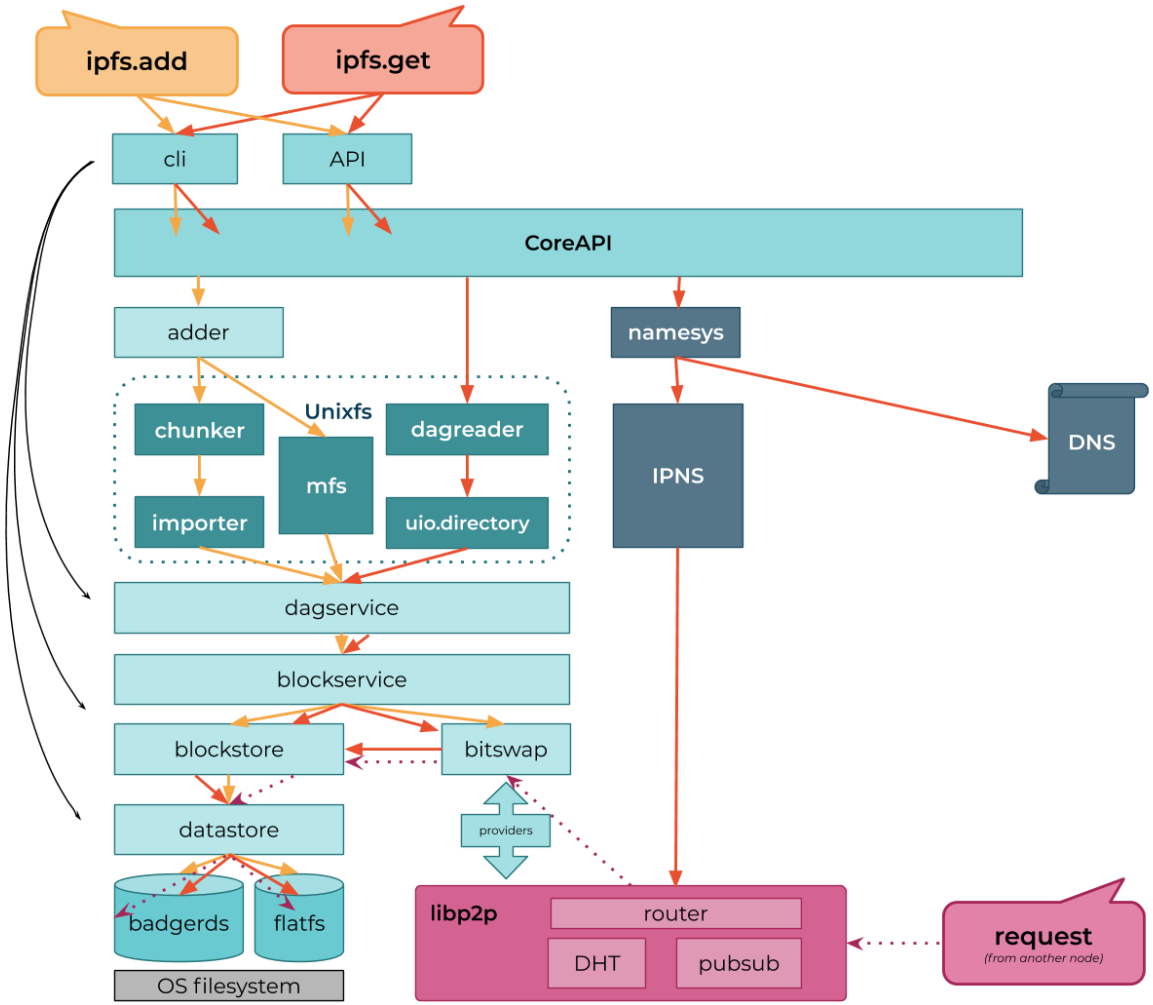
- Stores files for users
- Mines by providing storage
- Mines additional FIL by performing proofs

Retrieval miner

- Retrieves files for users
- May be a different entity other than a storage miner
- Unlike storage deals, retrieval deals are handled off-chain and their value may depend on retrieval speed

IPFS

Architecture (1/4)



IPFS

Routing and retrieval (2/4)

- A file is identified by its CID (content identifier) and each of its blocks form a Merkle DAG (directed acyclic graph) where nodes store blocks and edges link each block together using their CIDs
- Each block content is stored on a DHT (distributed hash table)

- Peers are assigned a peer ID and possess a public/private key pair for secure communication
- Block retrieval based on bitswap

IPFS

Routing and retrieval (2/4)

- A file is identified by its CID (content identifier) and each of its blocks form a Merkle DAG (directed acyclic graph) where nodes store blocks and edges link each block together using their CIDs $\rightsquigarrow$ similar to inodes in ext and NTFS
- Each block content is stored on a DHT (distributed hash table)

- Peers are assigned a peer ID and possess a public/private key pair for secure communication
- Block retrieval based on bitswap

IPFS

Routing and retrieval (2/4)

- A file is identified by its CID (content identifier) and each of its blocks form a Merkle DAG (directed acyclic graph) where nodes store blocks and edges link each block together using their CIDs $\rightsquigarrow$ similar to inodes in ext and NTFS
- Each block content is stored on a DHT (distributed hash table) $\rightsquigarrow$ uses a variant of S/Kademlia and Coral's Sloppy hashing
- Peers are assigned a peer ID and possess a public/private key pair for secure communication
- Block retrieval based on bitswap

IPFS

Routing and retrieval (2/4)

- A file is identified by its CID (content identifier) and each of its blocks form a Merkle DAG (directed acyclic graph) where nodes store blocks and edges link each block together using their CIDs $\rightsquigarrow$ similar to inodes in ext and NTFS
- Each block content is stored on a DHT (distributed hash table) $\rightsquigarrow$ uses a variant of S/Kademlia and Coral's Sloppy hashing
- Peers are assigned a peer ID and possess a public/private key pair for secure communication $\rightsquigarrow$ IDs are in the form `/ip4/7.7.7.7/tcp/4242/ipfs/QmYyQSohx5N`
- Block retrieval based on bitswap

IPFS

Routing and retrieval (2/4)

- A file is identified by its CID (content identifier) and each of its blocks form a Merkle DAG (directed acyclic graph) where nodes store blocks and edges link each block together using their CIDs $\rightsquigarrow$ similar to inodes in ext and NTFS
- Each block content is stored on a DHT (distributed hash table) $\rightsquigarrow$ uses a variant of S/Kademlia and Coral's Sloppy hashing
- Peers are assigned a peer ID and possess a public/private key pair for secure communication $\rightsquigarrow$ IDs are in the form `/ip4/7.7.7.7/tcp/4242/ipfs/QmYyQSohx5N`
- Block retrieval based on bitswap $\rightsquigarrow$ blocks are **exchanged** downloading the rarest first, the probability of sending a block to a specific peer depends on its **share ratio** effectively choking leeches

IPFS

Versioning and IPNS (3/4)

- Hashes in IPFS are immutable **content-based** references to files and are **not human-readable**
- When a file is modified, it's unlikely that its entire content will change, therefore many blocks will remain unchanged

IPFS

Versioning and IPNS (3/4)

- Hashes in IPFS are immutable **content-based** references to files and are **not human-readable**
- When a file is modified, it's unlikely that its entire content will change, therefore many blocks will remain unchanged $\rightsquigarrow$ Similarly to git, each version links to the previous one and only the modified blocks are added to the DAG

IPFS

Versioning and IPNS (3/4)

- Hashes in IPFS are immutable **content-based** references to files and are **not human-readable**
- When a file is modified, it's unlikely that its entire content will change, therefore many blocks will remain unchanged ⇝ Similarly to git, each version links to the previous one and only the modified blocks are added to the DAG
- A stable **Node ID** is used to find all the versions of the file through routing on DHT:
  1. A public/private key pair is generated to sign the file
  2. The hash of the public key is the node ID associated with that file
  3. The node ID points to the signed CID of the file
  4. Whenever the file is modified, a new CID is generated and signed using the private key

IPFS

Versioning and IPNS (3/4)

- Hashes in IPFS are immutable **content-based** references to files and are **not human-readable**
- When a file is modified, it's unlikely that its entire content will change, therefore many blocks will remain unchanged $\rightsquigarrow$ Similarly to git, each version links to the previous one and only the modified blocks are added to the DAG
- A stable **Node ID** is used to find all the versions of the file through routing on DHT:
  1. A public/private key pair is generated to sign the file
  2. The hash of the public key is the node ID associated with that file
  3. The node ID points to the signed CID of the file
  4. Whenever the file is modified, a new CID is generated and signed using the private key
- IPNS is used to lookup the node IDs of the requested files

IPFS

Versioning and IPNS (3/4)

- Hashes in IPFS are immutable **content-based** references to files and are **not human-readable**
- When a file is modified, it's unlikely that its entire content will change, therefore many blocks will remain unchanged $\rightsquigarrow$ Similarly to git, each version links to the previous one and only the modified blocks are added to the DAG
- A stable **Node ID** is used to find all the versions of the file through routing on DHT:
  1. A public/private key pair is generated to sign the file
  2. The hash of the public key is the node ID associated with that file
  3. The node ID points to the signed CID of the file
  4. Whenever the file is modified, a new CID is generated and signed using the private key
- IPNS is used to lookup the node IDs of the requested files $\rightsquigarrow$ Links are in the form `/ipfs/QmW98pJrc6FZ6/my/directory/structure/file.pdf`

IPFS

Setting up an IPFS node (4/4)

1. Install
2. Initialize the repository
3. Put the node online
4. Configure the node
5. Live demo!

IPFS

Setting up an IPFS node (4/4)

1. Install ⇝ `sudo snap install ipfs` (See http://snapcraft.io/ipfs for other OSes)
2. Initialize the repository
3. Put the node online
4. Configure the node
5. Live demo!

IPFS

Setting up an IPFS node (4/4)

1. Install ⇝ `sudo snap install ipfs` (See http://snapcraft.io/ipfs for other OSes)
2. Initialize the repository ⇝ `ipfs init`
3. Put the node online
4. Configure the node
5. Live demo!

IPFS

Setting up an IPFS node (4/4)

1. Install ⇝ `sudo snap install ipfs` (See http://snapcraft.io/ipfs for other OSes)
2. Initialize the repository ⇝ `ipfs init`
3. Put the node online ⇝ `ipfs daemon --profile server`
4. Configure the node
5. Live demo!

IPFS

Setting up an IPFS node (4/4)

1. Install ⇝ `sudo snap install ipfs` (See http://snapcraft.io/ipfs for other OSes)
2. Initialize the repository ⇝ `ipfs init`
3. Put the node online ⇝ `ipfs daemon --profile server`
4. Configure the node ⇝ Edit `ipfs/common/config`
5. Live demo!

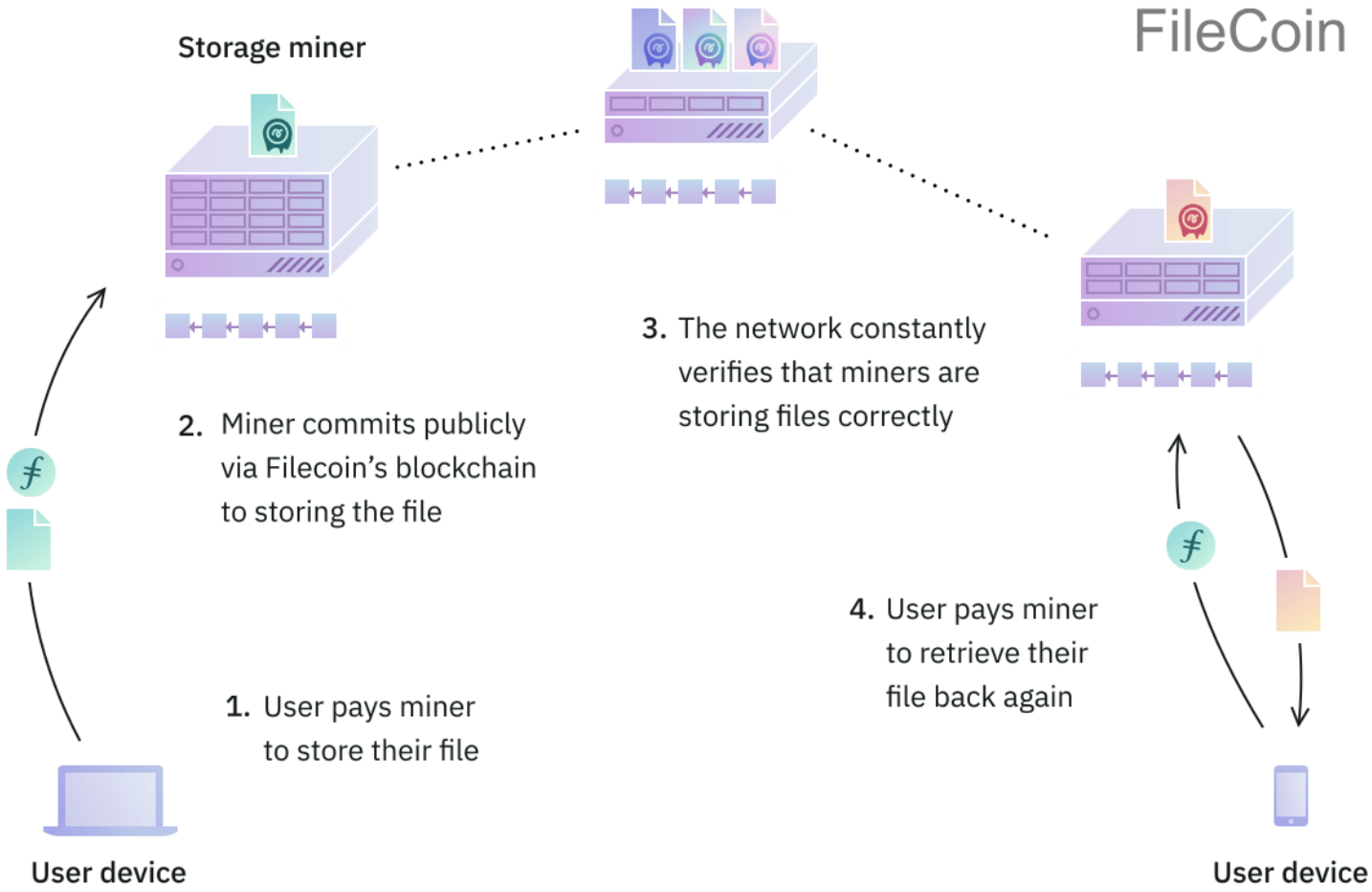
IPFS

Setting up an IPFS node (4/4)

1. Install ⇝ `sudo snap install ipfs` (See http://snapcraft.io/ipfs for other OSes)
2. Initialize the repository ⇝ `ipfs init`
3. Put the node online ⇝ `ipfs daemon --profile server`
4. Configure the node ⇝ Edit `ipfs/common/config`
5. Live demo! ⇝ http://ipfs.tsiotas.com:5001/webui

Filecoin

Storage and retrieval (1/7)



Filecoin

Deals (2/7)

Storage deal

Agreement between user and storage miner. The user pays in advance and the miner must periodically prove that it's still storing the file up to the expiration time

Retrieval deal

Agreement between user and retrieval miner. Payment is off-chain, incremental and typically based on retrieval reliability or speed

Filecoin

Deals (2/7)

Storage deal

Agreement between user and storage miner. The user pays in advance and the miner must periodically prove that it's still storing the file up to the expiration time

Retrieval deal

Agreement between user and retrieval miner. Payment is off-chain, incremental and typically based on retrieval reliability or speed

Gas fees

- Gas is spent by a message sender to pay for block mining and the overall network infrastructure
- $BaseFee$ is dynamically determined by the network congestion and burned (sent to an unspendable address, also useful for deflation over time)
- $GasPremium$ priority fee which incentives miners to add the transaction in the block
- $GasLimit$ estimate (made by the sender) of the maximum gas consumable by a single message
- $OverEstimation$ percentage of $GasLimit - GasUsage$ burned to punish overestimation of $GasLimit$ (miners privilege transactions with an high $GasLimit$)

Filecoin

Deals (2/7)

Storage deal

Agreement between user and storage miner. The user pays in advance and the miner must periodically prove that it's still storing the file up to the expiration time

Retrieval deal

Agreement between user and retrieval miner. Payment is off-chain, incremental and typically based on retrieval reliability or speed

Gas fees

- Gas is spent by a message sender to pay for block mining and the overall network infrastructure
- $BaseFee$ is dynamically determined by the network congestion and burned (sent to an unspendable address, also useful for deflation over time)
- $GasPremium$ priority fee which incentives miners to add the transaction in the block
- $GasLimit$ estimate (made by the sender) of the maximum gas consumable by a single message
- $OverEstimation$ percentage of $GasLimit - GasUsage$ burned to punish overestimation of $GasLimit$ (miners privilege transactions with an high $GasLimit$)

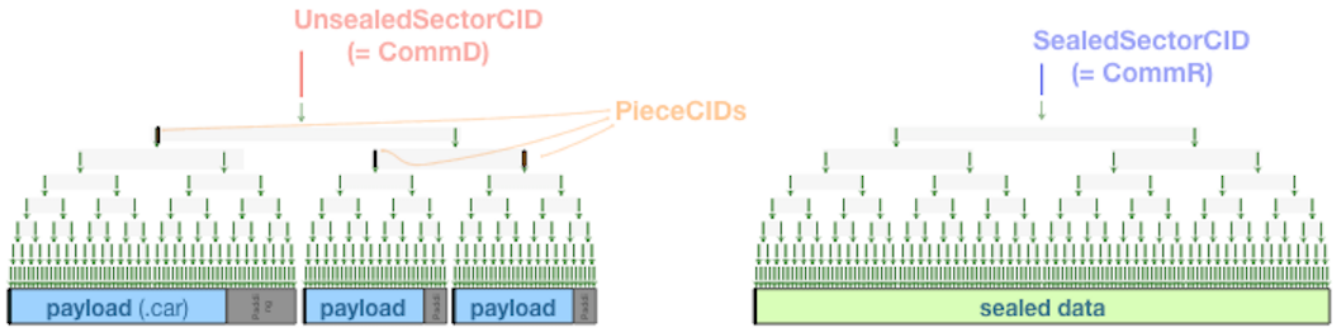
$$\begin{aligned} TotalGas =& (GasUsage + OverEstimation) \cdot BaseFee \\ &+ GasLimit \cdot GasPremium \end{aligned}$$

Filecoin

Sectors (3/7)

Sector

A sector is the basic storage unit on Filecoin. Size and time-increments for commitments are standardized and are chosen as a tradeoff between security and usability, while a sector's lifetime depends on the storage market.



Unsealed Sector

Sealed Sector

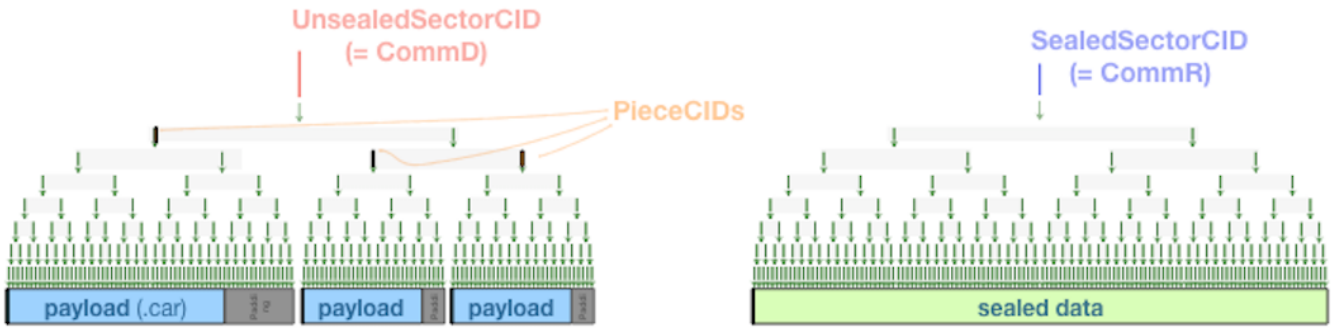
Filecoin

Sectors (3/7)

Sector

A sector is the basic storage unit on Filecoin. Size and time-increments for commitments are standardized and are chosen as a tradeoff between security and usability, while a sector's lifetime depends on the storage market.

1. A sector is fully occupied by deals and needs to be sealed before proofs ⇝ unused space is considered committed by self-deals (from the miner to itself)
2. A tree is built to compute the unsealed CID for the entire sector
3. The sector and its CID are sealed (ie. encoded in a way which can be used for proofs) into a replica
4. Proof of replication is performed on the replica



Unsealed Sector Sealed Sector

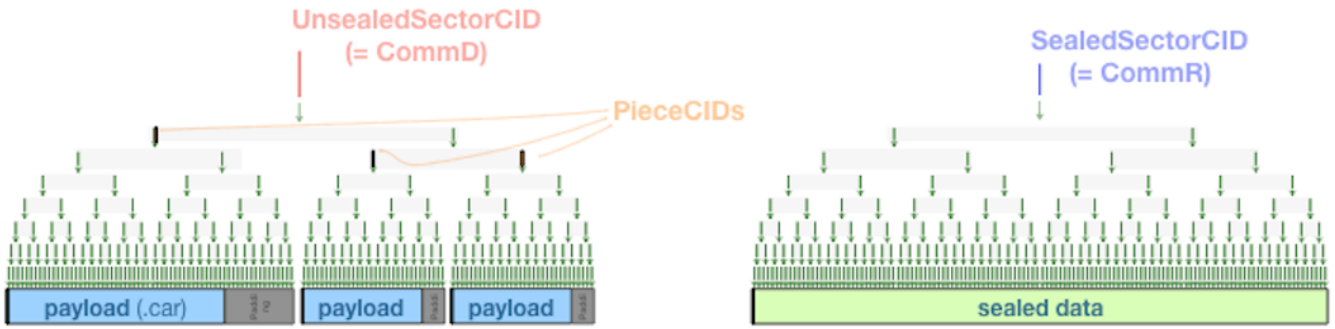
Filecoin

Sectors (3/7)

Sector

A sector is the basic storage unit on Filecoin. Size and time-increments for commitments are standardized and are chosen as a tradeoff between security and usability, while a sector's lifetime depends on the storage market.

1. A sector is fully occupied by deals and needs to be sealed before proofs ⇝ unused space is considered committed by self-deals (from the miner to itself)
2. A tree is built to compute the unsealed CID for the entire sector ⇝ the tree is only required to compute the CID and can be discarded afterwards
3. The sector and its CID are sealed (ie. encoded in a way which can be used for proofs) into a replica
4. Proof of replication is performed on the replica



Unsealed Sector Sealed Sector

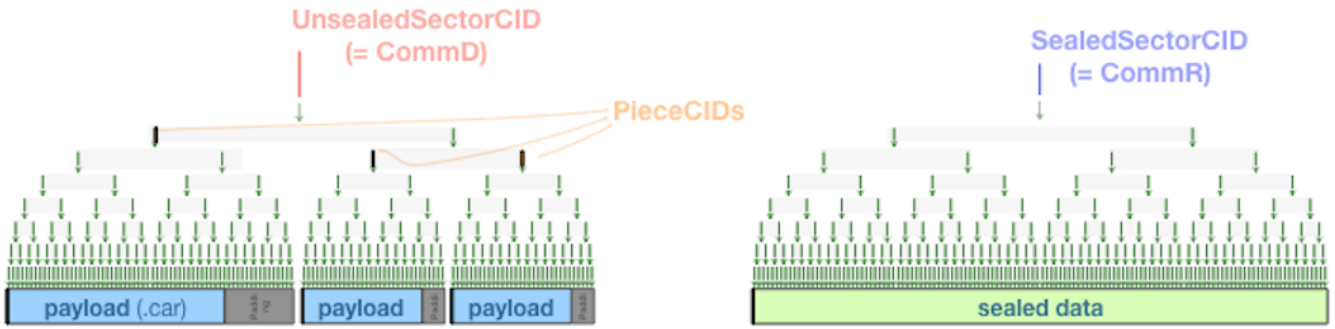
Filecoin

Sectors (3/7)

Sector

A sector is the basic storage unit on Filecoin. Size and time-increments for commitments are standardized and are chosen as a tradeoff between security and usability, while a sector's lifetime depends on the storage market.

1. A sector is fully occupied by deals and needs to be sealed before proofs ⇝ unused space is considered committed by self-deals (from the miner to itself)
2. A tree is built to compute the unsealed CID for the entire sector ⇝ the tree is only required to compute the CID and can be discarded afterwards
3. The sector and its CID are sealed (ie. encoded in a way which can be used for proofs) into a replica
4. Proof of replication is performed on the replica



Unsealed Sector Sealed Sector

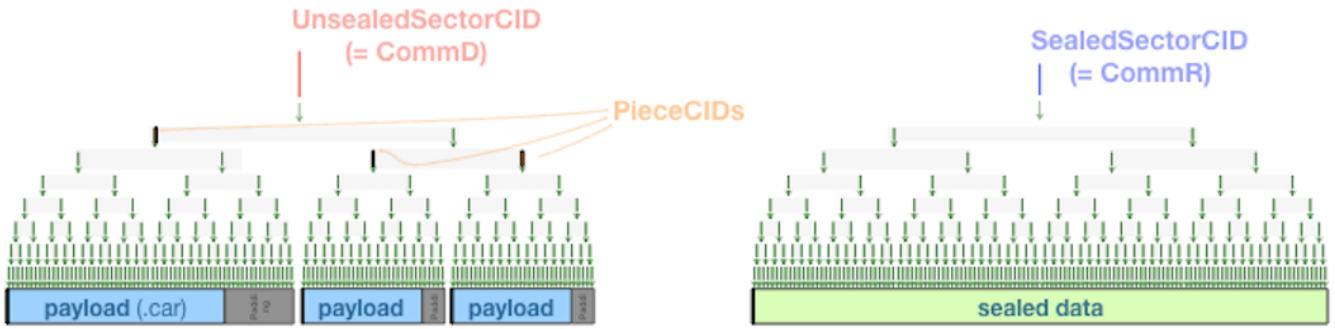
Filecoin

Sectors (3/7)

Sector

A sector is the basic storage unit on Filecoin. Size and time-increments for commitments are standardized and are chosen as a tradeoff between security and usability, while a sector's lifetime depends on the storage market.

1. A sector is fully occupied by deals and needs to be sealed before proofs ⇝ unused space is considered committed by self-deals (from the miner to itself)
2. A tree is built to compute the unsealed CID for the entire sector ⇝ the tree is only required to compute the CID and can be discarded afterwards
3. The sector and its CID are sealed (ie. encoded in a way which can be used for proofs) into a replica
4. Proof of replication is performed on the replica



Unsealed Sector Sealed Sector

Filecoin

Mining (4/7)

- Storage miners require powerful hardware
- The blockchain acts as a ledger for proofs, deals and FIL transfer
- Lotus is the most advanced miner implementation as of today

Filecoin

Mining (4/7)

- Storage miners require powerful hardware $\rightsquigarrow$ other than storage, they have to create new blocks (every 30 seconds) and perform proofs
- The blockchain acts as a ledger for proofs, deals and FIL transfer
- Lotus is the most advanced miner implementation as of today

Filecoin

Mining (4/7)

- Storage miners require powerful hardware $\rightsquigarrow$ other than storage, they have to create new blocks (every 30 seconds) and perform proofs
- The blockchain acts as a ledger for proofs, deals and FIL transfer $\rightsquigarrow$ proofs are publicly visible at proofs.filecoin.io
- Lotus is the most advanced miner implementation as of today

Proof of Replication (PoRep)

- Sectors are sealed (a time consuming process since sectors are 32Gb or 64Gb) using a key which depends on the hardware
- If a user wants its files to be stored as multiple copies on different miners, each copy will have a different seal and a malicious attacker would need to download each block of the file and reseal them
- The proof is based on the fact that an honest miner already has the sealed blocks and can reply quickly before a timeout

Filecoin

Mining (4/7)

- Storage miners require powerful hardware $\rightsquigarrow$ other than storage, they have to create new blocks (every 30 seconds) and perform proofs
- The blockchain acts as a ledger for proofs, deals and FIL transfer $\rightsquigarrow$ proofs are publicly visible at proofs.filecoin.io
- Lotus is the most advanced miner implementation as of today

Proof of Replication (PoRep)

- Sectors are sealed (a time consuming process since sectors are 32Gb or 64Gb) using a key which depends on the hardware
- If a user wants its files to be stored as multiple copies on different miners, each copy will have a different seal and a malicious attacker would need to download each block of the file and reseal them
- The proof is based on the fact that an honest miner already has the sealed blocks and can reply quickly before a timeout

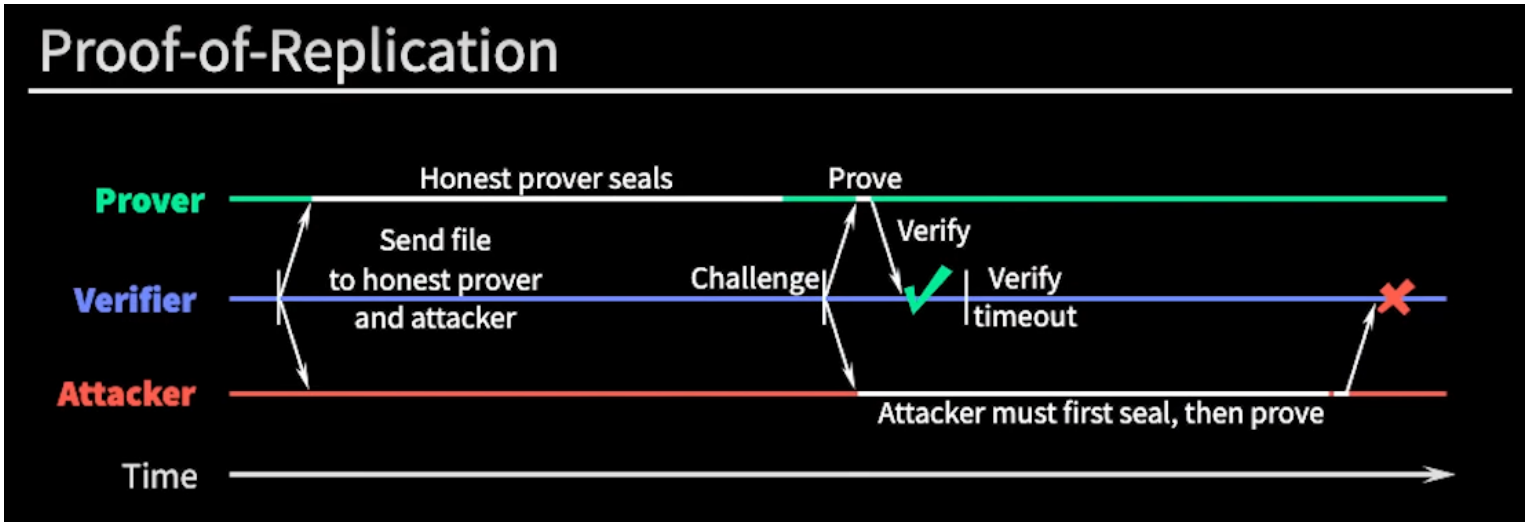
Proof of SpaceTime (PoSt)

- Both users (upon request) and miners (periodically) check that blocks are valid
- Miners are punished if blocks are unavailable before a deal expires
- All miners are asked to perform a PoRep on a random sector of their storage
- Two variants of PoSt:

WinningPoSt	When mining blocks
WindowPoSt	Every day

Filecoin

Proofs (5/7)



Proof of Spacetime

(PoSt)

Over time, randomly selected miners have random sectors challenged, from which data is read for verifications and compressed into a PoSt proof.

random Proofs of Space-time

Miners provide public proof that a given encoding of the data ***existed in physical storage continuously over a period of time.***

Filecoin

Rewards (6/7)

- Storage reward: miners receive payment for their deals with users
- Block reward: at the end of each epoch, a limited number of miners is chosen (with probability dependent on the quantity of storage they provide, ie. their **power**) and the first to complete a proof of spacetime in a given random sector (WinningPoSt) appends a block, winning the reward and a percentage on every deal inside the block
- Retrieval reward: miners receive payment for serving files to users

Filecoin

Rewards (6/7)

- Storage reward: miners receive payment for their deals with users $\rightsquigarrow$ FIL are transferred from users to miners
- Block reward: at the end of each epoch, a limited number of miners is chosen (with probability dependent on the quantity of storage they provide, ie. their **power**) and the first to complete a proof of spacetime in a given random sector (WinningPoSt) appends a block, winning the reward and a percentage on every deal inside the block
- Retrieval reward: miners receive payment for serving files to users

Filecoin

Rewards (6/7)

- Storage reward: miners receive payment for their deals with users $\rightsquigarrow$ FIL are transferred from users to miners
- Block reward: at the end of each epoch, a limited number of miners is chosen (with probability dependent on the quantity of storage they provide, ie. their **power**) and the first to complete a proof of spacetime in a given random sector (WinningPoSt) appends a block, winning the reward and a percentage on every deal inside the block $\rightsquigarrow$ FIL are mined from scratch
- Retrieval reward: miners receive payment for serving files to users

Filecoin

Rewards (6/7)

- Storage reward: miners receive payment for their deals with users $\rightsquigarrow$ FIL are transferred from users to miners
- Block reward: at the end of each epoch, a limited number of miners is chosen (with probability dependent on the quantity of storage they provide, ie. their **power**) and the first to complete a proof of spacetime in a given random sector (WinningPoSt) appends a block, winning the reward and a percentage on every deal inside the block $\rightsquigarrow$ FIL are mined from scratch
- Retrieval reward: miners receive payment for serving files to users $\rightsquigarrow$ payment is off-chain, may also use other currencies

Filecoin

Rewards (6/7)

- Storage reward: miners receive payment for their deals with users ⇝ FIL are transferred from users to miners
- Block reward: at the end of each epoch, a limited number of miners is chosen (with probability dependent on the quantity of storage they provide, ie. their **power**) and the first to complete a proof of spacetime in a given random sector (WinningPoSt) appends a block, winning the reward and a percentage on every deal inside the block ⇝ FIL are mined from scratch
- Retrieval reward: miners receive payment for serving files to users ⇝ payment is off-chain, may also use other currencies

Verified clients

- Special users certified with respect to their intent to store "useful" data
- They tend to offer better deals
- Miners accepting their deals have a multiplier applied to their power (and therefore their chance of winning a block reward increases)

Filecoin

Slashing (7/7)

WindowPoSt

- A period of 24 hours is split into windows
- All the used sectors of a miner are partitioned randomly into one subset for each window
- The miner must create a PoSt for the currently active subset and add it to the blockchain within a short timeout
- A sector can be declared faulty (eg. during maintenance) **before** a WindowPoSt starts

Filecoin

Slashing (7/7)

WindowPoSt

- A period of 24 hours is split into windows
- All the used sectors of a miner are partitioned randomly into one subset for each window
- The miner must create a PoSt for the currently active subset and add it to the blockchain within a short timeout
- A sector can be declared faulty (eg. during maintenance) **before** a WindowPoSt starts

- Storage fault slashing
  - Fault fee: penalty received daily if a WindowPoSt is not submitted
  - Sector penalty: penalty received for each sector which doesn't pass the WindowPoSt, the penalty is reduced if the sector was declared faulty
  - Termination fee: penalty received for every sector which is removed from the network (eg. the miner decides to reduce its storage or an hard drive fails)
- Consensus fault slashing: penalty received if the miner acts maliciously against the consensus mechanism

Filecoin

Slashing (7/7)

WindowPoSt

- A period of 24 hours is split into windows
- All the used sectors of a miner are partitioned randomly into one subset for each window
- The miner must create a PoSt for the currently active subset and add it to the blockchain within a short timeout
- A sector can be declared faulty (eg. during maintenance) **before** a WindowPoSt starts

- Storage fault slashing
  - Fault fee: penalty received daily if a WindowPoSt is not submitted
  - Sector penalty: penalty received for each sector which doesn't pass the WindowPoSt, the penalty is reduced if the sector was declared faulty
  - Termination fee: penalty received for every sector which is removed from the network (eg. the miner decides to reduce its storage or an hard drive fails)
- Consensus fault slashing: penalty received if the miner acts maliciously against the consensus mechanism

Miner's balance

- In order to make the slashing mechanism work, a miner can enter the Filecoin network only if it has invested some FIL
- If the miner wallet becomes empty, it will be banned from the network because it's no longer capable of covering potential slashing