Introduction to the IPFS

170629
SFC Arch B3 gentam

目次

- IPFSの目標と全体像
- IPFSのデザイン
- IPFSのアプリケーション

目次

- **IPFSの目標と全体像**
- IPFSのデザイン
- IPFSのアプリケーション

IPFSの目標と全体像

- **IPFSのコンセプト**
- IPFSの特徴
- IPFSの仕組み

IPFSとは (1/3)

- The **I**nter**P**lanetary **F**ile **S**ystem
- 一文で言うと: "Content Addressed, Versioned, P2P File System"
- Protocol Labs < https://protocol.ai> Juan Benet を中心として，2013 12月頃に開発開始
- Goで書かれた実装がここ< https://github.com/ipfs/go-ipfs>にある

IPFSとは (2/3)

- IPFSとは，成功したP2Pシステムのアイディアをレイヤに分けて組み合わせることで実現した，汎用的な分散ファイルシステム
  - DHTs, BitTorrent, Git, SFS ...
- IPFSの基本的な考え方は，全てのデータを1つのMerkle DAGにモデル化していくこと

IPFSとは (3/3)

- 全てのデバイスを1つの分散ファイルシステムによって接続する
- Webに似ている部分もあるが，IPFSは，Gitレポジトリ内でオブジェクトを交換する単一のBitTorrent スワームとも見ることができる
- IPFSは高スループットの，コンテンツへのハイパーリンクを持ったブロックストレージを提供する
- 最終的には Permanent Web も目指す

IPFSの目的

- HTTPを取り替え，より良いWebを作ることを目的とする
- より良いWeb
  - Faster: P2P型で経由を減らすため?
  - Safer: ブロックチェーンと組み合わせることで対改竄
  - Open: あらゆるコンテンツが，共通の名前空間にオープンにアドレシングされる

IPFSとBlockchain

- Blockchain は大量のデータを保持するには向かない
  - IPFSを利用して，サイズの大きなデータに対応
  - そのイミュータブルかつパーマネントなリンクを，ブロックチェーンに載せる
    - Blockchain上のアプリケーションはIPFSを利用することでパブリックにアクセス可能なデータベースを作ることができる
- IPFSとBlockchainはとても相性が良い

“The contribution of IPFS is simplifying, evolving, and connecting proven techniques into a single cohesive system, greater than the sum of its parts.”

– Juan Benet

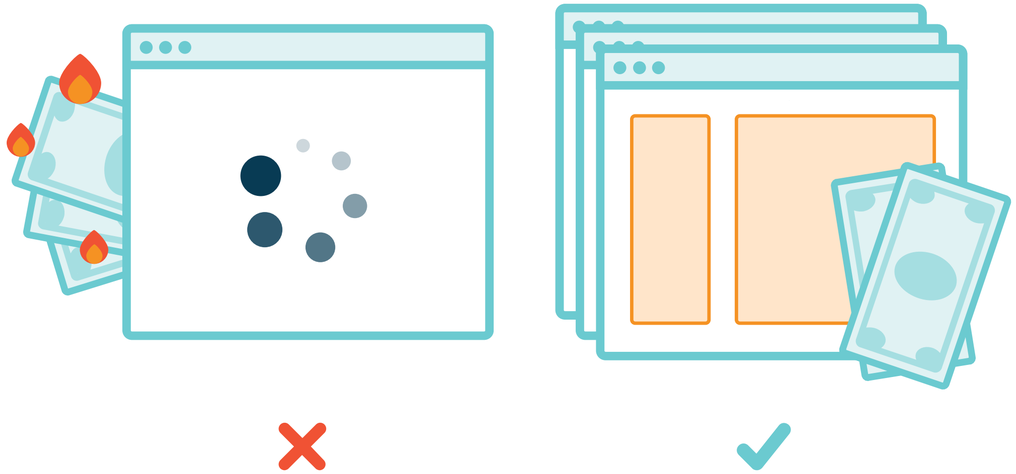
ここから次の2セクションは，IPFSのHP (ipfs.io) でされている，特徴と仕組みの概要をまとめたものです.

IPFSの目標と全体像

- IPFSのコンセプト
- **IPFSの特徴**
- IPFSの仕組み

1. HTTPの問題を解決する

- HTTPではファイルを1度に1つのマシンからしかダウンロードできない.
- IPFSでは，複数のマシンから同時にピースをダウンロード可能
  - 動画の転送においてはP2P型のアプローチにすることで60%のバンド幅を削減できる (らしい
- IPFSでは，大量のデータを重複なしに，効率良く分散させることができる

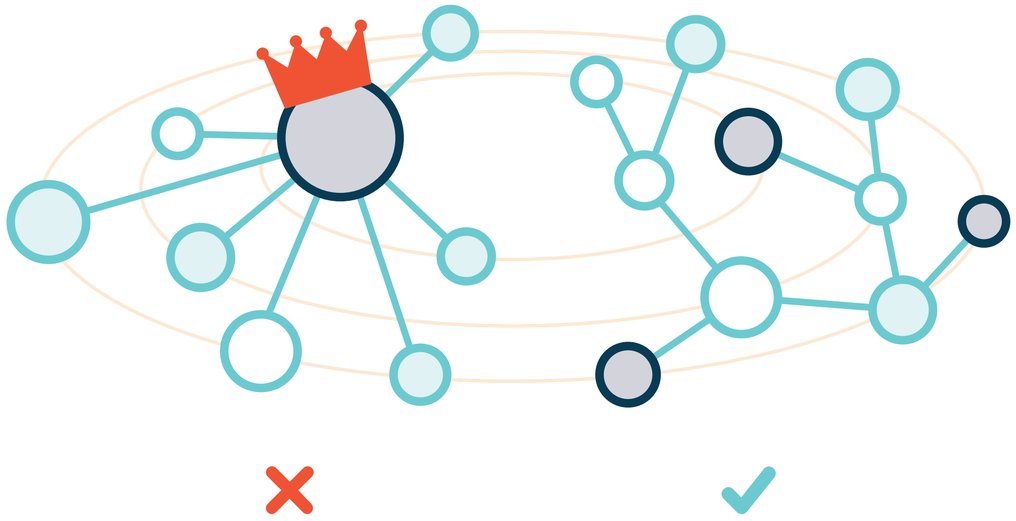


2. バージョン管理

- 日々人類の歴史は消されている
- ウェブページの平均的寿命は100日しかない．主要なメディアとして利用するにはあまりに不安定
- IPFSはgitのような，バージョン管理機能を提供し，レジリエントなネットワークを作ることをシンプルにする

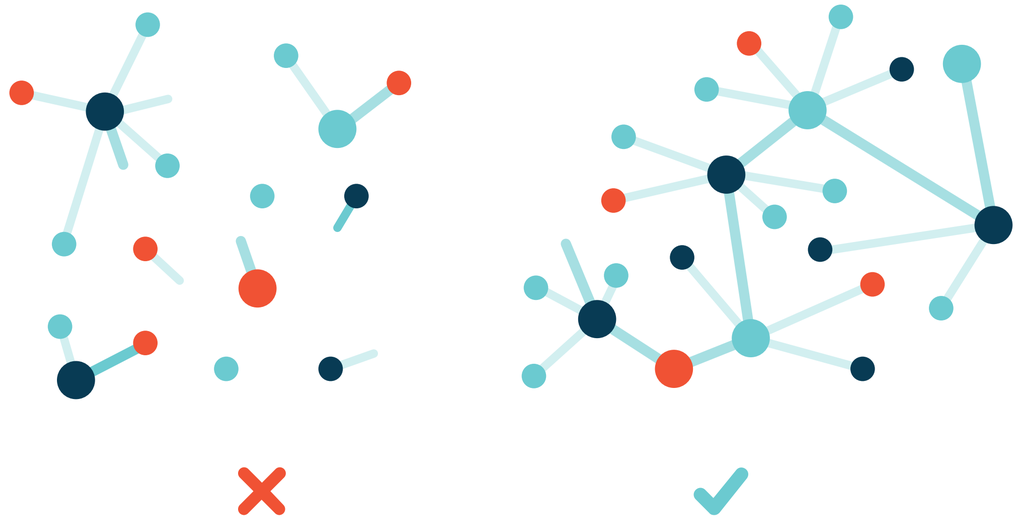
3. 脱中心化

- Webの中心化は機会を奪っている
- インターネットは歴史的に見て，人々に平等な情報アクセスを提供し，イノヴェーションを加速してきたが，近年の中心化の傾向はそれを脅かすものである
- IPFSは初期のオープンでフラットなウェブを保ちながらも，テクノロジーをアップグレードする



4. バックボーンに依存しない

- 現在のアプリケーションはバックボーンに依存している
- 途上国，災害時，不安定な接続，さらには惑星間通信などに対応するには今の設計は向いていない
- IPFSは単一のバックボーンに依存せずに，高い可用性を持った多様でレジリエントなネットワークを可能にする



IPFSの目標と全体像

- IPFSのコンセプト
- IPFSの特徴
- **IPFSの仕組み**

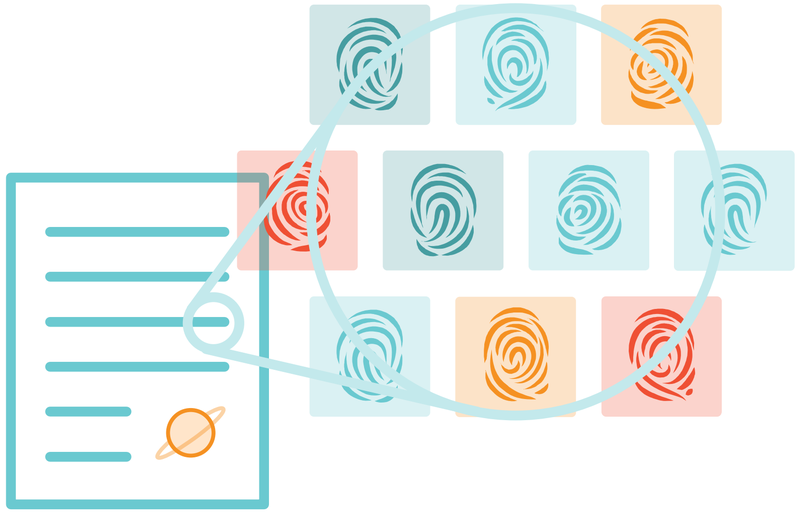
IPFSの仕組み

- 新しいファイルをIPFSネットワークに追加したとき
- ファイルをIPFSネットワークから探すとき

に起こること

1. ハッシュ値の付与

全てのファイルと，それを構成するブロックは，ネットワーク内でユニークな識別子として，暗号学的ハッシュ値が与えられる

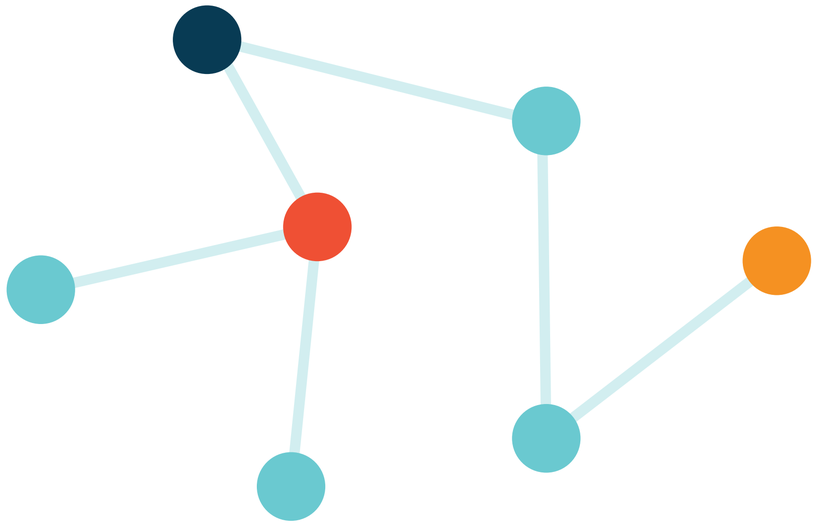


2. 重複回避とバージョン管理

- ハッシュ値を識別子とするため，ネットワーク内での，重複するコンテンツの存在は防がれる
- さらに全てのファイルはバージョン履歴が保存される

3. 情報保存先の分散

各ノードは，自分が欲しいコンテンツ自体と，誰が何を持っているか発見するために必要なインデックス情報を持つ

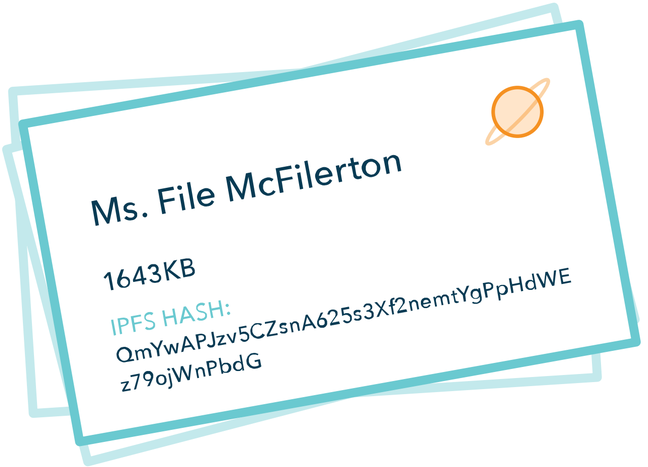


4. ハッシュ値からのファイル探査

ファイルを探す際には，ユニークなハッシュ値を利用して，ネットワークにそのデータを持つノードを尋ねる

5. 脱中心的な名前管理

全てのファイルは，IPNSという脱中心的な名前管理システムを用いて，人間が読める形の名前で探すことができる



目次

- IPFSの目標と全体像
- **IPFSのデザイン**
- IPFSのアプリケーション

ここまではIPFSというプロジェクトの背景や目標，その大まかなアプローチ方法に触れてきました.

ここからは，もう少し具体的かつ技術的な部分をまとめます.

IPFSのデザイン

- **IPFSを構成する技術とその特徴**
- IPFSのプロトコル

IPFSを構成する技術とその特徴

- IPFSは様々な成功した技術とそのアイディアを組み合わせることで，ファイルシステムを実現しています
- ここでは各技術の簡単な紹介と，IPFSが特に利用する特徴を説明します

IPFSとは (1/3)

- 一文で言うと: "Content Addressed, Versioned, P2P File System"
- Protocol Labs <https://protocol.ai> Juan Benet を中心として，2013 12月頃に開発開始
- Goで書かれた実装がここ<https://github.com/ipfs/go-ipfs>にある

IPFSの目標と全体像→IPFSのコンセプト

IPFSとは (1/3)

Git (+Blockchain)

- 一文で言うと: "**Content Addressed, Versioned, P2P File System**"
- Protocol Labs < h DHT, SFS .ai> Juan Benet を中心と SFS 2013 12月頃に開発開始
- Goで書かれた実装がここ<https://github.com/ipfs/go-ipfs>にある

IPFSを構成する技術とその特徴

- Routing - DHT
- Block Exchanges - BitTorrent
- Version Control Systems - Git
- Self-Certified Filesystems - SFS

IPFSを構成する技術とその特徴

- **Routing - DHT**
- Block Exchanges - BitTorrent
- Version Control Systems - Git
- Self-Certified Filesystems - SFS

Distributed Hash Table

- 分散ハッシュテーブルは，メタデータやピアの情報を調整/維持するためにP2Pシステムにおいて広く利用される
- コンテンツとノードにうまくIDをつけて，中央となるインデックスサーバの存在なしに，しかもフラッディングも起こさず情報を発見することが可能なオーバーレイネットワークを作る

DHTの例 (1/3)

Kademlia

- IDのXORを距離関数において木構造を作る
- log(n) の効率的なlookup (e.g. 10,000,000 ノードで 約20ホップ)
- コントロールメッセージを減らし，クエリーのオーバーヘッド減少
- 長生きなノードを優先する設計によりDoS攻撃に強い

ちなみに自分はKademlia DHTのシミュレーションをErlangで作ろうとしてます

DHTの例 (2/3)

S/Kademlia DHT

- Kademlia をさらに悪意のある攻撃に強くセキュアに
- PKIキーやPoWを導入し，Sybil 攻撃を難しく
- 役半数のノードが攻撃者の場合でも 0.85 の成功率を実現

DHTの例 (3/3)

Coral DSHT

- Distributed Sloppy Hash Table
- KademliaではコンテンツのIDに"近い"ノードが欲しくない情報まで持つ必要があった.
  - → 代わりに"ポインタ" (実際にデータを持っているノードのアドレス)を保持することで効率化
- 地域と規模によって階層化された"クラスタ"を作ることで，近いノードから探索を始め，低いレイテンシーを実現

IPFSを構成する技術とその特徴

- Routing - DHT
- **Block Exchanges - BitTorrent**
- Version Control Systems - Git
- Self-Certified Filesystems - SFS

BitTorrent

- BitTorrentは広く実用されているP2Pのファイル共有システム
- Bram Cohen が2001年から開発
- 信頼できない "スワーム*"内で，ファイルの一部分を互いに配布し合う仕組みを実現

---

* 同じトレントファイルにより，同じファイルを提供／ダウンロード中のピアグループ全体

BitTorrentの特徴:

1. ネットワークに貢献する(多くをアップロード)ほど早くダウンロードでき，逆に他者からもらうだけで，自分はアップロードしない"リーチャー"は罰せられる仕組み

2. ファイルピースの入手しやすさを確認し，希少なピースを優先的に送ることで，まだファイルの一部しか持っていないピア間でも交換が起こりやすくする仕組み

3. BitTorrent標準の，"tit-for-tat"戦略 (しっぺ返し=1の仕組み) には特定の搾取的なピアに対する脆弱性がある．そこを改善したPropShareという高パフォーマンスの戦略もある

IPFSを構成する技術とその特徴

- Routing - DHT
- Block Exchanges - BitTorrent
- **Version Control Systems - Git**
- Self-Certified Filesystems - SFS

Git

- バージョン管理システムは，ファイルの変更をモデル化し別のバージョンを効率的に配布する機能を提供

- Gitは，分散環境に適した，Merkle DAG* オブジェクトモデルにフィルツリーへの変更をキャプチャする

---

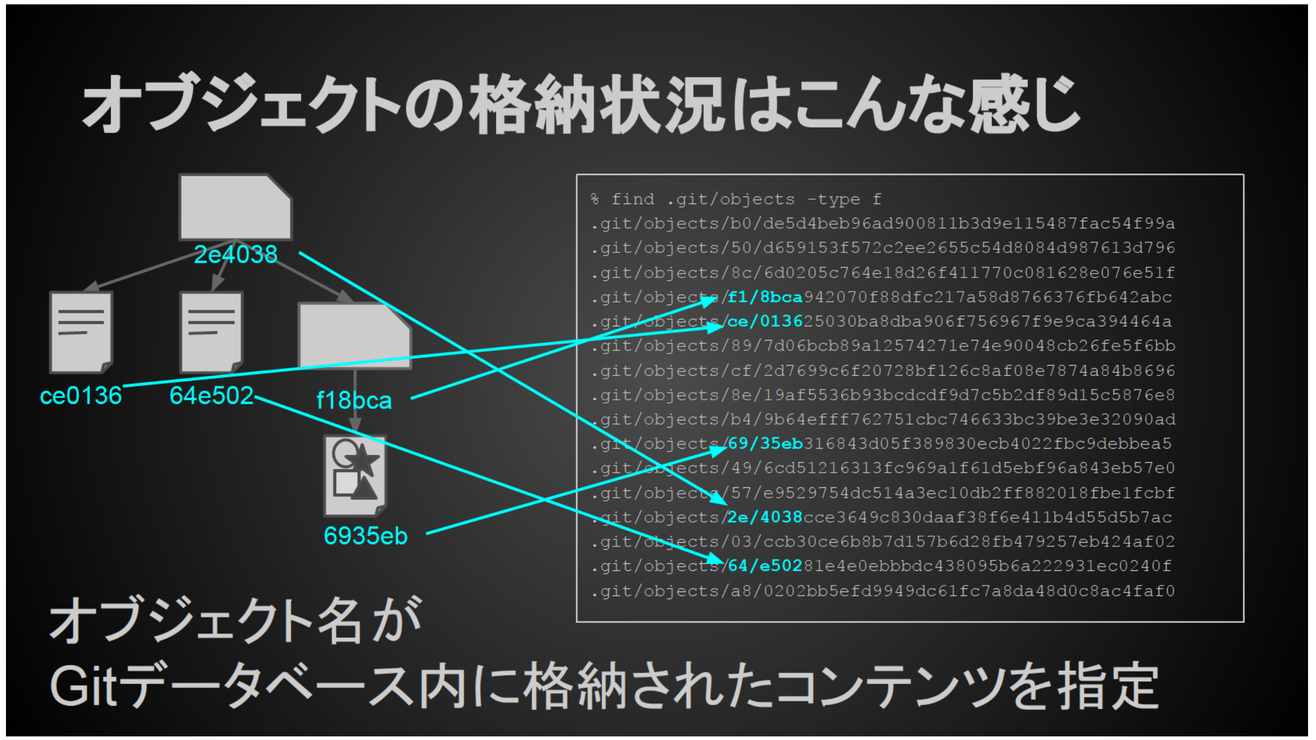
* Merkle Directed Acyclic Graph (ハッシュ木 有向非循環グラフ?) – マークルツリーと似ているが，より汎用的な構造．重複をバランスする必要がなく，葉でないノードもデータを持つ

Git

1. イミュータブルなオブジェクトが，ファイル (blob), ディレクトリ (tree), 変更 (commit)を表す
2. オブジェクトは，コンテンツのハッシュ値によってアドレスがつけられる
3. 他のオブジェクトへのリンクは埋め込まれ，Merkle DAGを形成する
4. ほとんどのメタデータ(ブランチ，タグ，etc)は単にオブジェクトへの参照ポインタでしかないため，作成や更新の処理は軽い
5. バージョンの変更は，参照の更新かオブジェクトの追加
6. バージョンの変更を他のユーザへ配布するには，単にオブジェクトを転送し，そこの参照を更新するだけ

参考資料

この辺りがとても分かりやすい資料: コンンセプトから理解するGitコマンド <https://www.slideshare.net/ktateish/git-concept1>



IPFSを構成する技術とその特徴

- Routing - DHT
- Block Exchanges - BitTorrent
- Version Control Systems - Git
- **Self-Certified Filesystems - SFS**

SFS: Self-Certified Filesystems

- SFS は，分散型トラストチェーンと，平等でグローバルな名前空間を以下のようなスキーマで実現する

```
/sfs/<Location>:<HostID>
 Location = サーバのネットワークアドレス
 HostID = hash(public_key || Location)
```

SFS

```
/sfs/<Location>:<HostID>
```

- つまり，ユーザはSFSファイルシステムの名前それ自体からサーバの公開鍵の正しさを確認でき，全ての通信を暗号化できる

- 全てのSFSインスタンスは，グローバルな名前空間を共有し，中心となる管理者なしに，暗号学的に名前が割り当てられる

IPFSのデザイン

- IPFSを構成する技術とその特徴
- **IPFSのプロトコル**

IPFSプロトコルの構成

- IPFSのプロトコルはサブプロトコルのスタックに分けられ，それぞれが異なる機能をカバーします

Identities, Network, Routing, Exchange, Objects, Files, Naming

IPFSプロトコルの構成

1. **Identities** - ノードのID生成や認証を管理
2. **Network** - 他のピアとの接続を管理
3. **Routing** - 特定のピアやオブジェクトを探すための情報を管理
4. **Exchange** - ブロック交換プロトコル(BitSwap)を利用して，効率的なブロック配布
5. **Objects** - 任意のデータをアドレス付きイミュータブルなオブジェクトのMerkle DAG へ
6. **Files** - Gitに影響されたバージョン管理ファイルシステム
7. **Naming** - self-certifying で変更可能な名前

IPFSプロトコルの構成

1. **Identities**
2. Network
3. Routing
4. Exchange
5. Objects
6. Files
7. Naming

1. Identities

- ノードはNodeIDによって識別される
  - これは，S/Kademliaにおける暗号パズルと共に作られた公開鍵のハッシュ値

- ユーザは自由に新たなNodeIDを発行でき，毎回異なるIDを利用することもできる(が，そうするとネットワーク的に不利になるため同じIDを使うことが動機付けられる

```
type NodeId Multihash
type Multihash []byte
// self-describing cryptographic hash digest
type PublicKey []byte
type PrivateKey []byte
// self-describing keys
type Node struct {
  NodeId NodeID
  PubKey PublicKey
  PriKey PrivateKey
}
```

IDの生成

- S/Kademlia ベースの IPFS におけるID生成:

```
difficulty = <integer parameter>
n = Node{}
do {
    n.PubKey, n.PrivKey = PKI.genKeyPair()
    n.NodeId = hash(n.PubKey)
    p = count_preceding_zero_bits(hash(n.NodeId))
} while (p < difficulty)
```

- 最初の接続時にピアは公開鍵を交換し，`hash(other.PublicKey)` が`other.NodeID`と一致するか確認する．しなかったら接続を切る.

multihash フォーマット

- IPFSは将来性を考え，特定のハッシュ関数に依存しないように設計されている

- 具体的に，multihash フォーマットは以下のようになっている:

```
<function code><digest length><digest bytes>
```

- こうすることで，セキュリティと速度のどちらを重視するかという場合ごとに最適なハッシュ関数を選ぶこともできる

IPFSプロトコルの構成

1. Identities
2. **Network**
3. Routing
4. Exchange
5. Objects
6. Files
7. Naming

2. Network

- IPFSのノードは，インターネット上に広く分散しているかもしれない何百ものノードと定期的に通信をする
- ネットワークは以下のスタックに分割できる
  - Transport, Reliability, Connectivity, Integrity, Authenticity

Network Stack (1/2)

- **Transport**: IPFSのトランスポートレイヤにはどんなプロトコルでも使うことができる．特に WebRTC DataChannl (ブラウザとの接続時)や，uTP (LEDBAT) が適している

- **Reliability**: もし下位のネットワークが信頼性を提供しない場合は，IPFSがuTPや，SCTPを利用して信頼性を確保できる

Network Stack (2/2)

- **Connectivity**: ICE (Interactive Connectivity Establishment) NAT越えの技術を利用する
- **Integrity**: ハッシュのチェックサムを利用してメッセージの完全性をチェックできる(任意)
- **Authenticity**: HMACと送信者の公開鍵を利用して，メッセージの信頼性をチェックできる(任意)

multiaddr フォーマット

- multihashフォーマットと同じように，ネットワークアドレスは multiaddrというフォーマットで表現

- よってIPやそのバージョンにも依存しない

```
# an SCTP/IPv4 connection
/ip4/10.20.30.40/sctp/1234/
# an SCTP/IPv4 connection proxied over TCP/IPv4
/ip4/5.6.7.8/tcp/5678/ip4/1.2.3.4/sctp/1234/
```

IPFSプロトコルの構成

1. Identities
2. Network
3. **Routing**
4. Exchange
5. Objects
6. Files
7. Naming

3. Routing

- IPFSノードは，以下の条件を満たすルーティングシステムが必要:
  - 他のピアのネットワークアドレスを探すことができる
  - 特定のオブジェクトを持っているピアを探すことができる
- IPFSでは，S/Kademlia と Coral をベースとしたDSHTを利用して，これを実現
- 1KB以下の小さなデータはDHT上に直接，それより大きいデータは参照(実体を持つNodeID)を保存する

Routing Interface

```
type IPFSRouting interface {
    FindPeer(node NodeId)
    // gets a particular peer's network address
    SetValue(key []bytes, value []bytes)
    // stores a small metadata value in DHT
    GetValue(key []bytes)
    // retrieves small metadata value from DHT
    ProvideValue(key Multihash)
    // announces this node can serve a large value
    FindValuePeers(key Multihash, min int)
    // gets a number of peers serving a large value
}
```

- このインターフェイスに従っていさえすれば，全く別のルーティングシステムと入れ替えても問題はない

IPFSプロトコルの構成

1. Identities
2. Network
3. Routing
4. **Exchange**
5. Objects
6. Files
7. Naming

4. Block Exchange - BitSwap Protocol

- **BitSwap**: BitTorrentにインスパイアされたブロック交換プロトコル
  - `have_list` にあるブロックを交換にオファーしながら
  - `want_list` のブロックを探していく

- BitTorrentとは違い，この交換は1つのトレントファイルに制限されない
  - どのファイルを構成するブロックかということは関係なく，自由に(ブロックの)物々交換をするマーケットプレイス

BitSwap Credit

- BitSwapは，単純なクレジットシステムにより以下の効果を持つ:
  - Seeder (ノードにが全てのデータをダウンロードし終わった状態)に，アップロードを続けるインセンティブを与える
  - Leecher (自分はアップロードしないただ乗りノード)を防ぐ

- これは，ピア達が互いにクレジットのバランスを記録し，"負債"が増えるほどにブロックを送信する確率が下がってくることで実現

- さらに，上の仕組みによって送らないことにしたピアは，`ignore_cooldown` (`10 sec`)の時間だけ無視することで，確率的ゲームをすることを防ぐ

BitSwap Strategy (1/2)

- BitTorrentにおいてデフォルトの戦略は "tit-for-tat"だが，他にも様々な戦略が存在する:
  - BitTyrant: 速度を重視してSelfishにPeerを選択していく戦略
  - BitThief: 自分は全くアップロードせずにただのりする戦略
  - PropShare: オークションのモデルに基づき比率的にシェア
- 交換のパフォーマンスを最大化させつつ，ただのりノードや未知の戦略に対して耐性を持たなければいけない.

BitSwap Strategy (2/2)

- BitSwapにおける現状の選択はdebt ratio: r でスケールさせたシグモイド関数 (負債者に対して送信する確率)

$$P(send \mid r) = \frac{1}{1 + e^{6-3r}} \qquad r = \frac{\texttt{bytes\_send}}{\texttt{bytes\_recv} + 1}$$

- rが信頼の尺度となり，シビル攻撃への耐性も持つ: 過去に多くのデータを交換した信頼できるノードには寛大で，新入りには厳しい

BitSwap Ledger

- ノードはledgerに，他のノードとの通信を記録する

- コネクションを作る際には，このledgerを交換する

```
type Ledger struct {
  owner       NodeId
  partner     NodeId
  bytes_sent  int
  bytes_recv  int
  timestamp   Timestamp
}
```

- 交換したledgerが一致しなければ，初期化される．あるいは不正だと判断して交換を拒否することもできる

BitSwap の接続例

1. **Open:** ピアは `ledgers` を合意が取れるまで送り合う
2. **Sending:** `want_lists` と ブロックを交換し合う
3. **Close:** ピアが接続を切る
4. **Ignored**(例外): ピアはタイムアウトや，不正な戦略によって無視する

IPFSプロトコルの構成

1. Identities
2. Network
3. Routing
4. Exchange
5. **Objects**
6. Files
7. Naming

5. Object – Merkle DAG

- DHTとBitSwapによって，データの保存と配布を巨大なP2Pシステムで行うことが可能になった
- IPFSはこの上に，Gitのデータ構造を一般化した Merkle DAG を形成する

Merkle DAG の特徴

1. **Content Addressing:** 全てのコンテンツとリンクはユニークなmultihashチェックサムによって特定される

2. **Tamper resistance:** もしデータが改ざんされたり，壊れたりした場合はチェックサムによって検知できる

3. **Deduplication:** 内容が同じデータは，全く同じハッシュになるため重複して保存されることはない

IPFS Object

IPFS リンクとオブジェクトのフォーマット:

```
type IPFSLink struct {
    Name string
    // name or alias of this link
    Hash Multihash
    // cryptographic hash of target
    Size int
    // total size of target
}
type IPFSObject struct {
    links []IPFSLink
    // array of links
    data []byte
    // opaque content data
}
```

ipfsコマンドとオブジェクト

- ipfs コマンドで色々な操作ができる(brew install ipfs)

- 例: リファレンスを全てリストする

```
> ipfs ls /XLZ1625Jjn7SubMDgEyeaynFuR84ginqvzb
        XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x  189458 less
        XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5  19441  script
        XLF4hwVHsVuZ78FZK6fozf8Jj9WEURMbCX4  5286   template
        <object multihash> <object size> <link name>
```

- 全てのオブジェクトがハッシュ値を持つことがわかる

Paths (1/2)

- 伝統的なUNIXファイルシステムやWebと同じパスのモデルを使う

```
# format
/ipfs/<hash-of-object>/<name-path-to-object>

# example
/ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/foo.txt
```

Paths (2/2)

- `/ipfs` プレフィックスによって，既存のシステムにマウントできる
- グローバルな"ルート"は存在しないため，/ipfs/ の後には必ずオブジェクトのハッシュが来る
  - 全てのリンクがルートになりうる:

```
/ipfs/<hash-of-foo>/bar/baz
/ipfs/<hash-of-bar>/baz
/ipfs/<hash-of-baz>
```

その他

- (当然だけど) 全てのオブジェクトはどこかのノードの物理的なストレージに存在する
  - アクセスしたものはローカルにキャッシュされる (期間は設定可

- 特定のオブジェクトをローカルに留めておきたい場合，は"ピン"することができる

- オブジェクトを世界に公開するには，キーをDHTに追加するだけ
  - バージョンを更新した場合は別のオブジェクトとなるため，それをトラッキングするにはまた別のオブジェクトが必要

オブジェクトの暗号化

- IPFSではオブジェクトレベルで暗号化が行われる

- ユーザのキーチェーンで自動的に認証や署名を行う

```
type EncryptedObject struct {
  Object []bytes
  // raw object data encrypted
  Tag []bytes
  // optional tag for encryption groups
}

type SignedObject struct {
  Object []bytes
  // raw object data signed
  Signature []bytes
  // hmac signature
  PublicKey []multihash
  // multihash identifying key
}
```

IPFSプロトコルの構成

1. Identities
2. Network
3. Routing
4. Exchange
5. Objects
6. **Files**
7. Naming

6. Files

- IPFS は以下のオブジェクトの種類を定義する:

1. **block**: a variable-size block of data.
2. **list**: a collection of blocks or other lists.
3. **tree**: a collection of blocks, lists, or other trees.
4. **commit**: a snapshot in the version history of a tree.

経緯の補足

- Gitの構造をそのまま利用したかったが，以下の理由によって改変したものを利用している:
  1. 高速なサイズルックアップ (→objectsにサイズの収集機能を追加)
  2. 巨大ファイルの複製 (→ `list`オブジェクトを追加)
  3. `commits`の`tree`への埋め込み

  - それでもIPFSとGitの構造は十分に近いため変換することができる

- ファイルオブジェクトはJSONで表記されるが，実際にはprotobufでバイナリエンコードされている

File Object: blob

- blobはアドレスを持つデータ単位で，ファイルを表現する
- IPFSのblobはGitのそれや，一般的ファイルシステムにおけるデータブロックと同じようなもの.

```
{
  "data": "some data here",
  // blobs have no links
}
```

File Object: list

- list は複数のblobから構成される巨大なファイルなどを意味する

```
{
  "data": ["blob", "list", "blob"],
  // lists have an array of object types as data
  "links": [
    { "hash": "XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x",
      "size": 189458 },
    { "hash": "XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5",
      "size": 19441 },
    { "hash": "XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z",
      "size": 5286 }
    // lists have no names in links
  ]
}
```

File Object: tree

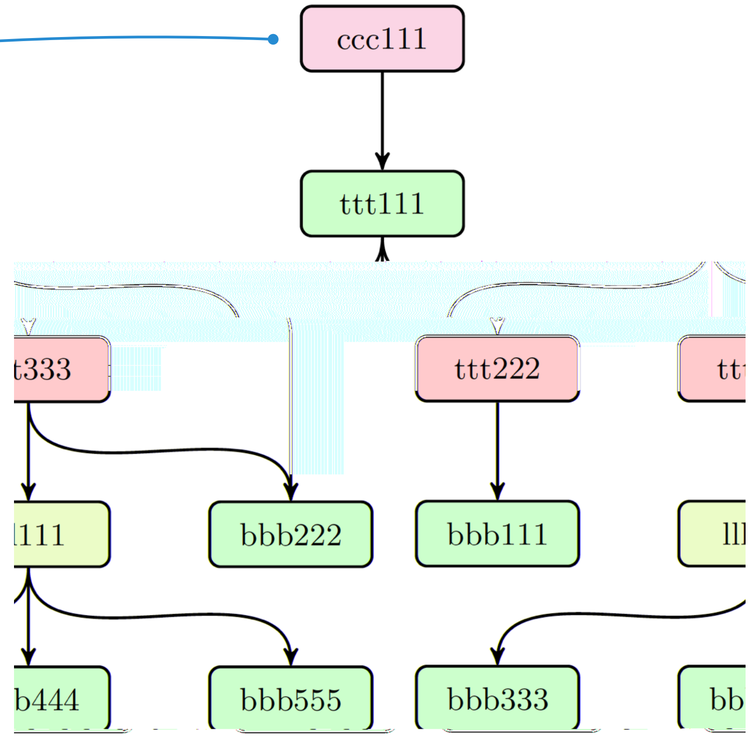
- tree はディレクトリ: 名前からハッシュへのマップで，Gitと似ている

```
{
  "data": ["blob", "list", "blob"],
  // trees have an array of object types as data
  "links": [
    { "hash": "XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x",
      "name": "less", "size": 189458 },
    { "hash": "XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5",
      "name": "script", "size": 19441 },
    { "hash": "XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z",
      "name": "template", "size": 5286 }
    // trees do have names
  ]
}
```

File Object: commit (1/2)

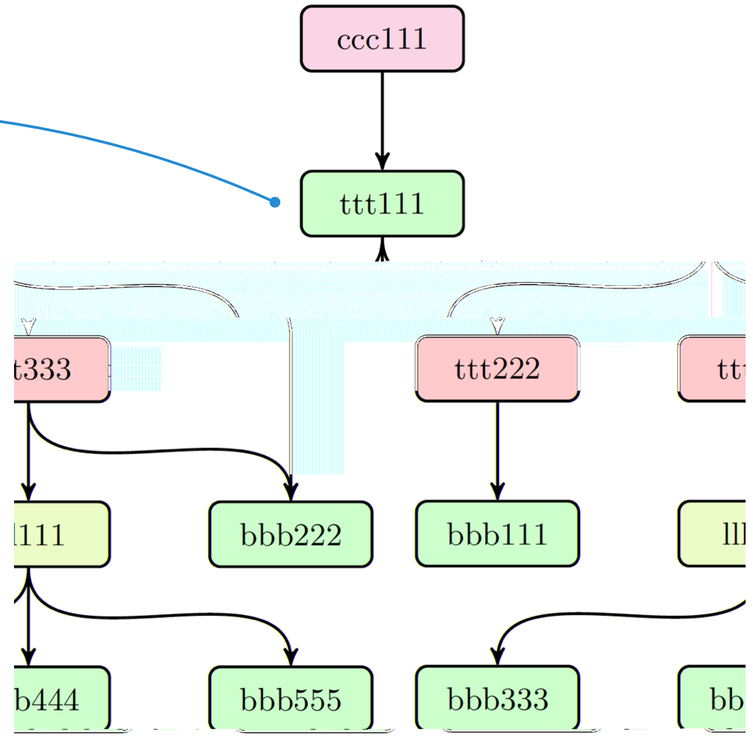
- commitは任意のオブジェクトのある時間におけるスナップショット

```
> ipfs file-cat <ccc111-hash> --json
{
  "data": {
    "type": "tree",
    "date": "2014-09-20 12:44:06Z",
    "message": "This is a commit message."
  },
  "links": [
    { "hash": "<ccc000-hash>",
      "name": "parent", "size": 25309 },
    { "hash": "<ttt111-hash>",
     "name": "object", "size": 5198 },
    { "hash": "<aaa111-hash>",
      "name": "author", "size": 109 }
  ]
}
```



File Object: commit (2/2)

```
> ipfs file-cat <ttt111-hash> --json
{
  "data": ["tree", "tree", "blob"],
  "links": [
    { "hash": "<ttt222-hash>",
      "name": "ttt222-name", "size": 1234 },
    { "hash": "<ttt333-hash>",
      "name": "ttt333-name", "size": 3456 },
    { "hash": "<bbb222-hash>",
      "name": "bbb222-name", "size": 22 }
  ]
}
```



IPFSプロトコルの構成

1. Identities
2. Network
3. Routing
4. Exchange
5. Objects
6. Files
7. **Naming**

7. IPNS: Naming and Mutable State

- ここまで作り上げてきた「イミュータブルなオブジェクト」は「名前の変更」と相性が悪い

- しかしこのままでは，新しいコンテンツに関するコミュニケーションは全て，IPFS外部でリンクを送信する必要が出てくる
  - 例: ブログの記事を修正した
    - 投稿のハッシュ(= ID)が変わってしまう
    - 記事一覧ページが古い記事にリンクしてしまっているので修正する
    - 一覧ページのハッシュ(= ID)が変わってしまう
    - あー...

Name in IPFS

- SFSのネーミングスキーマを利用することで，グローバルな名前空間に，self-certifiedで，「ミュータブル」な名前を作ることができる

Self-Certified Names

- IPFSの名前は以下のスキーマを使う

1. まず，`NodeId = hash(node.PubKey)` である

2. 全てのユーザに，NodeIdをプレフィックスとした"ミュータブル"な名前空間を与える:

   ```
   /ipns/<NodeId>
   ```

3. ユーザはこのパスに自分のプライベートキーでサインしたオブジェクトを公開できる
   例えば:

   ```
   /ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/docs/hello
   ```

4. 他のユーザがオブジェクトを得たときには，署名が公開鍵とNodeIdに一致するか確かめることで，ミュータブルなパスを実現

Self-Certified Names

- IPFSの名前は以下のスキーマを使う

1. まず，`NodeId = hash(node.PubKey)` である

2. 全てのユーザに，NodeIdをプレフィックスとした"ミュータブル"な名前空間を与える:

   `/ipns/<NodeId>` fじゃなくてn

3. ユーザはこのパスに自分のプライベートキーでサインしたオブジェクトを公開できる

   例えば:

   `/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/docs/hello`

4. 他のユーザがオブジェクトを得たときには，署名が公開鍵とNodeIdに一致するか確かめることで，ミュータブルなパスを実現

Paths (1/2)

- 伝統的なUNIXファイルシステムやWebと同じパスのモデルを使う

```
# format
/ipfs/<hash-of-object>/<name-path-to-object>

# example
/ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/foo.txt
```

Paths (1/2)

一方ipfsのパスは"オブジェクト"のハッシュ

- 伝 Webと同じパスのモデルを使う

```
# format
/ipfs/<hash-of-object>/<name-path-to-object>

# example
/ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/foo.txt
```

InterPlanetary Name Space

- "ipns"と"ipfs", はミュータブルかイミュータブルか簡単に識別できるように別のプレフィックスとして利用する

- IPNSの名前はコンテンツのアドレスではないため, 公開するにはルーティングシステムを利用する:
  1. 普通のイミュータブルなオブジェクトとしてIPFSで公開
  2. そのハッシュ値をメタデータとしてルーティングシステムに公開

```
routing.setValue(NodeId, <ns-object-hash>)
```

Human Friendly Names

- IPNS は以下を利用して，長くて覚えられないハッシュをヒューマンフレンドリーな形にすることができる

1. Peer Links
2. DNS TXT IPNS Records
3. Proquint Pronounceable Identifiers
4. Name Shortening Services

1. Peer Links

- ユーザは他のユーザのオブジェクトへのリンクを自分のオブジェクトに直接入れることができる

- これは，web of trustを作ることにも役立つ

```
# Alice links to bob Bob
ipfs link /<alice-pk-hash>/friends/bob /<bob-pk-hash>

# Eve links to Alice
ipfs link /<eve-pk-hash/friends/alice /<alice-pk-hash>

# Eve also has access to Bob
/<eve-pk-hash/friends/alice/friends/bob

# access Verisign certified domains
/<verisign-pk-hash>/foo.com
```

2. DNS TXT IPNS Records

- /ipns/<domain> が有効なドメイン名であった場合，IPFSは，DNS TXT レコードをルックアップする:

- IPFSは，値を他のオブジェクトのハッシュ値かIPNSのパスと解釈する:

```
# this DNS TXT record
ipfs.benet.ai. TXT "ipfs=XLF2ipQ4jD3U ..."

# behaves as symlink
ln -s /ipns/XLF2ipQ4jD3U /ipns/fs.benet.ai
```

3. Proquint Pronounceable Identifiers

- バイナリを発音可能な文字列にするスキーマが存在していて，IPFSはProquintをサポートしている．例えば:

```
# this proquint phrase
/ipns/dahih-dolij-sozuk-vosah-luvar-fuluh

# will resolve to corresponding
/ipns/KhAwNprxYVxKqpDZ
```

4. Name Shortening Service

- 名前のハッシュ値を短縮して読みやすい形にするサービスを作ることができる

- これは，今のDNSとWeb URLの関係に近い:

```
# User can get a link from
/ipns/shorten.er/foobar

# To her own namespace
/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm
```

目次

- IPFSの目標と全体像
- IPFSのデザイン
- **IPFSのアプリケーション**

IPFSの発展可能性

1. As a mounted global filesystem, under /ipfs and /ipns.
2. As a mounted personal sync folder that automatically versions, publishes, and backs up any writes.
3. As an encrypted file or data sharing system.
4. As a versioned package manager for all software.
5. As the root filesystem of a Virtual Machine.
6. As the boot filesystem of a VM (under a hypervisor).
7. As a database: applications can write directly to the Merkle DAG data model and get all the versioning, caching, and distribution IPFS provides.
8. As a linked (and encrypted) communications platform.
9. As an integrity checked CDN for large files (without SSL).
10. As an encrypted CDN.
11. On webpages, as a web CDN.
12. As a new Permanent Web where links do not die.

存在するIPFSアプリケーション

- まとめてあるレポジトリ <https://github.com/ipfs/awesome-ipfs>
  - IPFSBin - PastebinのIPFS版
  - Interplanetary Wiki - IPFS上のWiki
  - ...

関連しそうなプロジェクト

- Storj <https://storj.io/>
- Gitchain <http://gitchain.org/>
- ZeroTier One <https://www.zerotier.com/>
- MaidSafe <https://maidsafe.net>
- SIA < http://sia.tech/>
- Upspin <https://upspin.io>

References

- IPFS HP <https://ipfs.io/>
- White Paper <https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf>
- the morning paper<https://blog.acolyer.org/2015/10/05/ipfs-content-addressed-versioned-p2p-file-system/>