Future Proofing
IPFS, Blockchains,
and other Systems
Future Proofing
Future Proofing

The process of anticipating the future and developing methods of minimizing the effects of shocks and stresses of future events.
Future Proofing

Designing interfaces and protocols with extensibility and future changes in mind, to remain operational for a long time.
• Content Addressing for the Web
• Addresses MUST be *Permanent*
• MUST work over any transports
• Routing MUST improve w/ SoA
• Encryption & Auth MUST track SoA
• MUST work across planets
IPFS

• Content Addressing for the Web
• Addresses MUST be Permanent
• MUST work over any transports
• Routing MUST improve w/ SoA
• Encryption & Auth MUST track SoA
• MUST work across planets

Blockchains

• Secure, Decentralized, Transactions
• Addresses MUST be Permanent
• MUST work over any transports
• Contracts MUST improve w/ SoA
• Encryption & Auth MUST track SoA
• MUST work across planets
How Protocols Last
Aging Problems
Aging Problems

- Protocol Ossification
- Scalability Problems
- Total Breaks — Cryptographic Breaks
- Tyranny of Past Mistakes
- Tyranny of Bad Types and Bad Notation
Principles to Engineer for the Future

- "Never going to change" considered harmful
- Interplanetary Principle
- Future Scoping ("will it work in k years?")
"Never going to change" considered harmful
Interplanetary Principle
Future Scoping

- Will this protocol work in \{10, 25, 50\} years?
- Can we **trust & rely** on this protocol in \{10, 25, 50\} years?
- What features or choices are time dependent? (time bombs)
- What sequence of events can break the protocol?
- What will computation resources be like?
- Will protocol assumptions change? How can we be certain?
multi formats

libp2p

IPLD
multiformats - self describing values
protocol agility, interop, avoid lock in

multihash - cryptographic hashes
multiaddr
multibase
multicodec
multistream
multikey
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Length</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha2-256</td>
<td>256</td>
<td>0x08e11fc41466fcd4af7dee0905605d9</td>
</tr>
<tr>
<td>sha2-512</td>
<td>256</td>
<td>0xaada4961542da952c8bb93080cc6f9</td>
</tr>
<tr>
<td>sha3-256</td>
<td>256</td>
<td>0xcaadb37a46daeda4e0d5e61574a9aaca</td>
</tr>
<tr>
<td>blake2b-256</td>
<td>256</td>
<td>0x0f8ea61a5dea457d69fe5c12575c1d</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Block Size</td>
<td>256</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>sha2 256</td>
<td>256</td>
<td>112008e11fc41466fccda0af7dee0905605d9c2e44ada4961542da952c8bb93080cc6f9</td>
</tr>
<tr>
<td>sha2 512</td>
<td>256</td>
<td>112008e11fc41466fccda0af7dee0905605d9c2e44ada4961542da952c8bb93080cc6f9</td>
</tr>
<tr>
<td>sha3</td>
<td>256</td>
<td>112008e11fc41466fccda0af7dee0905605d9c2e44ada4961542da952c8bb93080cc6f9</td>
</tr>
<tr>
<td>blake2b</td>
<td>256</td>
<td>402008fbea061a5dea457d69fe5c12575c1d9d30c49f575936f6e1c6d4ea0ab078df</td>
</tr>
</tbody>
</table>
multihash - cryptographic hashes

- self describing
- in the value itself (not out of band)
- as small as possible
- no assumptions
- no lock in
- interop of hash functions
Because aesthetically I prefer the code first. You already have to write your stream parsing code to understand that a single byte already means "a length in bytes more to skip". Reversing these doesn't buy you much.

Implementations:

- go-multihash
- node-multihash
- clj-multihash
- rust-multihash
- haskell-multihash
- python-multihash
- elixir-multihash, elixir-multihashing
- swift-multihash
- ruby-multihash
- scala-multihash

**table for Multihash v1.0.0-RC (semver)**

The current multihash table is [here](#):
multiformats - self describing values
  protocol agility, interop, avoid lock in

multihash - cryptographic hashes
multiaddr - network addresses
multibase - base encodings
multicodec - serialization codecs
multistream - stream wire protocols
multikey - cryptographic keys and artifacts
multiaddr - network addresses

/ip6/::1/tcp/80/http
/ip4/1.2.3.4/udp/5001/sctp/sip
/ip4/1.2.3.4/udp/5002/utp/bittorrent
/ip4/1.2.3.4/udp/5003/quic/ipfs
/onion/3g2upl4pq6kufc4m/80/http
multiformats - self describing values
protocol agility, interop, avoid lock in

multihash - cryptographic hashes
multiaddr - network addresses
multibase - base encodings
multicodec - serialization codecs
multistream - stream wire protocols
multikey - cryptographic keys and artifacts
multiformats - self describing values
protocol agility, interop, avoid lock in

- multihash
- multiaddr
- multibase
- multicodec
- multistream
- multikey

- In Value (not OOB)
- Small, Binary (perf)
- Human Readable
- Stable
- Starting Standard Now
- Impls in many langs
a common hash-chain format for distributed data structures
distributed data structures

authenticated data structures

hash linked data structures
IPFS is like a forest of linked merkle-trees
a collection of peer-to-peer protocols

Content Routing
- mDNS
- DNS
- pub
- sub

Peer Routing
- Kad
- DHT
- STUN
- TURN

Discovery
- Kad
- DHT
- DNS
- boot
- strap

Transports
- TCP
- uTP
- QUIC
- SCTP
- BLE
- TOR
- I2P

NAT Traversal
- ICE
- STUN
- TURN
- PEX
- PKI
Future Proofing Blockchains
Future Proofing Blockchains

• Extensible Protocols (multiformats and other techniques)
• Interlink hashchains (IPLD)
• Modularize the network stack (libp2p, devp2p, …)
• Create "halt, fix, resume" protocol (including humans)
• Adaptive Networks (Ethereum, Stellar, Tezos, …)
• Future Scope ("will it work in {10, 25, 50} years?")
Open Problems & Challenges
Open Problems & Challenges

• Fully self-describing crypto protocols (link to code, vms)
• Permanent Identifiers ( Truly. For humans, programs, things, …) 
• Interplanetary Consensus
Future Proofing
IPFS, Blockchains, and other Systems