Information Theory vs Filters

Intro

Lukáš Hozda

- Marketing at Braiins

Lukáš Hozda

- Marketing at Braiins
- Rust/Lisp programmer

Lukáš Hozda

- Marketing at Braiins
- Rust/Lisp programmer
- Former embedded Rust developer

Lukáš Hozda

- Marketing at Braiins
- Rust/Lisp programmer
- Former embedded Rust developer
- Teaching Rust at MatFyz

Context

- Ongoing Core vs Knots debate

Context

- Ongoing Core vs Knots debate
- Filtering is one of the main fault lines

Context

- Ongoing Core vs Knots debate
- Filtering is one of the main fault lines
- Focus today: mempool / relay-level filtering

Context

- Ongoing Core vs Knots debate
- Filtering is one of the main fault lines
- Focus today: mempool / relay-level filtering
- `BIP-110` exists, but consensus-level filtering is out of scope today

Ordinals

- Ordinals are a convention for tracking individual satoshis

Ordinals

- Ordinals are a convention for tracking individual satoshis
- They assign each sat a stable identity/index within the total supply

Ordinals

- Ordinals are a convention for tracking individual satoshis
- They assign each sat a stable identity/index within the total supply
- That lets people say “this specific sat moved here”

Ordinals

- Ordinals are a convention for tracking individual satoshis
- They assign each sat a stable identity/index within the total supply
- That lets people say "this specific sat moved here"
- By themselves, ordinals are about identification and tracking, not content

Inscriptions

- Inscriptions are arbitrary data embedded in Bitcoin transactions (usually witness data)

Inscriptions

- Inscriptions are arbitrary data embedded in Bitcoin transactions (usually witness data)
- In the ordinals ecosystem, an inscription is associated with a specific sat

Inscriptions

- Inscriptions are arbitrary data embedded in Bitcoin transactions (usually witness data)
- In the ordinals ecosystem, an inscription is associated with a specific sat
- The payload can be images, text, HTML, or other bytes

Inscriptions

- Inscriptions are arbitrary data embedded in Bitcoin transactions (usually witness data)
- In the ordinals ecosystem, an inscription is associated with a specific sat
- The payload can be images, text, HTML, or other bytes
- This is the part that usually triggers filtering debates

Difference

- `Ordinal` = which sat

Difference

- `Ordinal` = which sat
- `Inscription` = what data is attached/associated

Difference

- `Ordinal` = which sat
- `Inscription` = what data is attached/associated
- You can discuss ordinals as tracking without discussing inscriptions

Difference

- `Ordinal` = which sat
- `Inscription` = what data is attached/associated
- You can discuss ordinals as tracking without discussing inscriptions
- In practice, the current controversy is mostly about inscriptions

Criticism

- Most criticism targets inscriptions and transaction patterns, not the numbering convention itself

Criticism

- Most criticism targets inscriptions and transaction patterns, not the numbering convention itself
- One concern is chain data / blockspace usage (especially witness payloads)

Criticism

- Most criticism targets inscriptions and transaction patterns, not the numbering convention itself
- One concern is chain data / blockspace usage (especially witness payloads)
- Another concern is UTXO set growth from certain usage patterns

Criticism

- Most criticism targets inscriptions and transaction patterns, not the numbering convention itself
- One concern is chain data / blockspace usage (especially witness payloads)
- Another concern is UTXO set growth from certain usage patterns
- These are different resource problems and should not be conflated

Network

Mempool

- There is no single global mempool

Mempool

- There is no single global mempool
- Every node has its own local mempool

Mempool

- There is no single global mempool
- Every node has its own local mempool
- Local policy decides what is accepted and relayed

Mempool

- There is no single global mempool
- Every node has its own local mempool
- Local policy decides what is accepted and relayed
- Miners build blocks from what they see and what pays

Gossip

- Nodes talk to peers, not to a central coordinator

Gossip

- Nodes talk to peers, not to a central coordinator
- Transactions propagate hop by hop

Gossip

- Nodes talk to peers, not to a central coordinator
- Transactions propagate hop by hop
- Topology matters: peers, connectivity, implementation mix

Gossip

- Nodes talk to peers, not to a central coordinator
- Transactions propagate hop by hop
- Topology matters: peers, connectivity, implementation mix
- Small relay minorities can still create viable paths

Handshake

- `version`: announces protocol version, services, user agent, and chain height

Handshake

- `version`: announces protocol version, services, user agent, and chain height
- `verack`: confirms the handshake

Handshake

- `version`: announces protocol version, services, user agent, and chain height
- `verack`: confirms the handshake
- `ping`: keepalive / liveness check

Handshake

- `version`: announces protocol version, services, user agent, and chain height
- `verack`: confirms the handshake
- `ping`: keepalive / liveness check
- `pong`: response to `ping`

Relay

- `inv`: “I have object(s) with these hashes” (announcement only)

Relay

- `inv`: “I have object(s) with these hashes” (announcement only)
- `getdata`: “send me the full object for these hashes”

Relay

- `inv`: “I have object(s) with these hashes” (announcement only)
- `getdata`: “send me the full object for these hashes”
- `tx`: full transaction payload

Discovery

- `getaddr`: “send me peer addresses”

Discovery

- `getaddr`: “send me peer addresses”
- `addr`: list of peer addresses

Relay Flow

- Node A sends `inv` with tx hash(es)

Relay Flow

- Node A sends `inv` with tx hash(es)
- Node B decides what it wants

Relay Flow

- Node A sends `inv` with tx hash(es)
- Node B decides what it wants
- Node B sends `getdata` for selected txs

Relay Flow

- Node A sends `inv` with tx hash(es)
- Node B decides what it wants
- Node B sends `getdata` for selected txs
- Node A sends `tx` with full bytes

Relay Flow

- Node A sends `inv` with tx hash(es)
- Node B decides what it wants
- Node B sends `getdata` for selected txs
- Node A sends `tx` with full bytes
- This reduces bandwidth compared to pushing full txs to everyone

Filtering

- Mempool filtering is local policy, not consensus

Filtering

- Mempool filtering is local policy, not consensus
- It can block relay through that node

Filtering

- Mempool filtering is local policy, not consensus
- It can block relay through that node
- It cannot directly control the whole network topology

Filtering

- Mempool filtering is local policy, not consensus
- It can block relay through that node
- It cannot directly control the whole network topology
- The real question is network-wide effectiveness

Limits

Topology

- Relay is path-based

Topology

- Relay is path-based
- Transactions need some permissive paths, not universal approval

Topology

- Relay is path-based
- Transactions need some permissive paths, not universal approval
- More connectivity makes bypass easier

Topology

- Relay is path-based
- Transactions need some permissive paths, not universal approval
- More connectivity makes bypass easier
- Strong suppression needs near-universal adoption

Sub-1 sat/vB

- We already have a concrete precedent

Sub-1 sat/vB

- We already have a concrete precedent
- Many nodes reject or deprioritize sub-1 sat/vB transactions

Sub-1 sat/vB

- We already have a concrete precedent
- Many nodes reject or deprioritize sub-1 sat/vB transactions
- They still propagate and still get mined

Sub-1 sat/vB

- We already have a concrete precedent
- Many nodes reject or deprioritize sub-1 sat/vB transactions
- They still propagate and still get mined
- Relay policy does not equal miner/pool inclusion policy

Sub-1 sat/vB

- We already have a concrete precedent
- Many nodes reject or deprioritize sub-1 sat/vB transactions
- They still propagate and still get mined
- Relay policy does not equal miner/pool inclusion policy
- Partial filtering is not network-wide suppression

Steganography

- Useful protocols have encoding capacity

Steganography

- Useful protocols have encoding capacity
- Bitcoin transactions have a large valid design space

Steganography

- Useful protocols have encoding capacity
- Bitcoin transactions have a large valid design space
- Data can be embedded in forms that look economically ordinary

Steganography

- Useful protocols have encoding capacity
- Bitcoin transactions have a large valid design space
- Data can be embedded in forms that look economically ordinary
- Filters must infer intent from valid transactions

Examples

- Explicit payload fields (for example `OP_RETURN`)

Examples

- Explicit payload fields (for example `OP_RETURN`)
- Witness data payloads (the inscriptions case)

Examples

- Explicit payload fields (for example `OP_RETURN`)
- Witness data payloads (the inscriptions case)
- Data encoded indirectly via transaction structure/pattern choices

Examples

- Explicit payload fields (for example `OP_RETURN`)
- Witness data payloads (the inscriptions case)
- Data encoded indirectly via transaction structure/pattern choices
- The same information can be moved between multiple valid representations

Asymmetry

- Filters remove known patterns

Asymmetry

- Filters remove known patterns
- Encoders move to new patterns

Asymmetry

- Filters remove known patterns
- Encoders move to new patterns
- Encoders have more degrees of freedom than filters

Asymmetry

- Filters remove known patterns
- Encoders move to new patterns
- Encoders have more degrees of freedom than filters
- This is a structural limit

Incentives

- “Spam” definitions do not remove the limits above

Incentives

- “Spam” definitions do not remove the limits above
- Filtering can push embedding into worse forms

Incentives

- “Spam” definitions do not remove the limits above
- Filtering can push embedding into worse forms
- `OP_RETURN` is prunable; fake UTXO growth is worse

Incentives

- “Spam” definitions do not remove the limits above
- Filtering can push embedding into worse forms
- `OP_RETURN` is prunable; fake UTXO growth is worse
- Incentives and harm reduction matter more than prohibition

Miners

- Miners (more precisely pools) run nodes too

Miners

- Miners (more precisely pools) run nodes too
- Their incentive is fee revenue and reliable block production

Miners

- Miners (more precisely pools) run nodes too
- Their incentive is fee revenue and reliable block production
- They do not automatically share relay-policy filtering goals

Miners

- Miners (more precisely pools) run nodes too
- Their incentive is fee revenue and reliable block production
- They do not automatically share relay-policy filtering goals
- Even strong relay filtering can fail if mining incentives remain permissive

Rust

Rust at Braiins

- Rust across embedded systems

Rust at Braiins

- Rust across embedded systems
- Rust in infrastructure components

Rust at Braiins

- Rust across embedded systems
- Rust in infrastructure components
- Rust in high-level, low-latency network services

Async in Rust

- `async fn` lets us write non-blocking I/O code in direct style

Async in Rust

- `async fn` lets us write non-blocking I/O code in direct style
- Futures represent work that can make progress over time

Async in Rust

- `async fn` lets us write non-blocking I/O code in direct style
- Futures represent work that can make progress over time
- `.await` yields control while waiting for I/O

Async in Rust

- `async fn` lets us write non-blocking I/O code in direct style
- Futures represent work that can make progress over time
- `.await` yields control while waiting for I/O
- This is a good fit when one process manages many peer connections

Tokio

- Network software is mostly waiting: sockets, timeouts, backpressure

Tokio

- Network software is mostly waiting: sockets, timeouts, backpressure
- Async tasks make large peer sets practical

Tokio

- Network software is mostly waiting: sockets, timeouts, backpressure
- Async tasks make large peer sets practical
- Rust gives memory safety under concurrency

Tokio

- Network software is mostly waiting: sockets, timeouts, backpressure
- Async tasks make large peer sets practical
- Rust gives memory safety under concurrency
- Metrics/logging compose well in the same process

```
loop {
    tokio::select! {
        result = stream.read(&mut buf) => { /* parse */ }
        Some(msg) = outbound_rx.recv() => { /* send */ }
        _ = keepalive.tick() => { /* ping */ }
```

```
        }
}
```

Demo Tool

- `crab-router` is a demo instrument supporting the argument

Demo Tool

- `crab-router` is a demo instrument supporting the argument
- It connects to many mainnet peers and classifies implementations

Demo Tool

- `crab-router` is a demo instrument supporting the argument
- It connects to many mainnet peers and classifies implementations
- It observes relay, discovery, and transaction flow metrics

Demo Tool

- `crab-router` is a demo instrument supporting the argument
- It connects to many mainnet peers and classifies implementations
- It observes relay, discovery, and transaction flow metrics
- The point is measurement and demonstration

Demo

Demo

- Live `crab-router` run on mainnet

Demo

- Live `crab-router` run on mainnet
- Peer mix and transaction flow in Grafana

Demo

- Live `crab-router` run on mainnet
- Peer mix and transaction flow in Grafana
- Discovery traffic (`addr` / `getaddr`)

Demo

- Live `crab-router` run on mainnet
- Peer mix and transaction flow in Grafana
- Discovery traffic (`addr` / `getaddr`)
- Use observations to support the topology argument

End

Q&A