Lecture #24: Peer-to-Peer Networks and the Network Layer

COMS 4995-001:
The Science of Blockchains
URL: https://timroughgarden.org/s25/

Tim Roughgarden

Goals for Lecture #24

1. The Bitcoin network.
   – using a peer-to-peer network for the dissemination of txs and blocks
2. Ethereum and gossipsub.
   – post-merge ➔ how to organize communication between 1M+ validators?
3. Solana's Turbine.
   – how to disseminate a block as you're assembling it
4. Narwhal.
   – tx dissemination with data availability guarantees

Communication in a Blockchain Protocol

Categories of communication: (for validators and full nodes)

Communication in a Blockchain Protocol

Categories of communication: (for validators and full nodes)

1. Transaction dissemination.
   - user (or wallet) generally responsible for getting a tx to *some* validators, not necessarily to *all* validators (e.g., upcoming leaders)

Communication in a Blockchain Protocol

Categories of communication: (for validators and full nodes)

1. Transaction dissemination.
   – user (or wallet) generally responsible for getting a tx to *some* validators, not necessarily to *all* validators (e.g., upcoming leaders)
2. Block dissemination.
   – e.g., in Tendermint or longest-chain consensus

Communication in a Blockchain Protocol

Categories of communication: (for validators and full nodes)

1. Transaction dissemination.
   - user (or wallet) generally responsible for getting a tx to *some* validators, not necessarily to *all* validators (e.g., upcoming leaders)
2. Block dissemination.
   - e.g., in Tendermint or longest-chain consensus
3. Consensus-related data (e.g., votes).
   - for Tendermint, scales dangerously with the number of validators

Communication in a Blockchain Protocol

Categories of communication: (for validators and full nodes)

1. Transaction dissemination.
   - user (or wallet) generally responsible for getting a tx to *some* validators, not necessarily to *all* validators (e.g., upcoming leaders)
2. Block dissemination.
   - e.g., in Tendermint or longest-chain consensus
3. Consensus-related data (e.g., votes).
   - for Tendermint, scales dangerously with the number of validators
4. Data about validators (e.g., IP addresses/ports).

Communication in a Blockchain Protocol

Categories of communication: (for validators and full nodes)

1. Transaction dissemination.
2. Block dissemination.
3. Consensus-related data (e.g., votes).
4. Data about validators (e.g., IP addresses/ports).

Note: network-layer responsibilities generally outside of the scope of a blockchain protocol's specification.

- spec = tx semantics, validity for txs and blocks, etc.

Communication in a Blockchain Protocol

Categories of communication: (for validators and full nodes)

1. Transaction dissemination.
2. Block dissemination.
3. Consensus-related data (e.g., votes).
4. Data about validators (e.g., IP addresses/ports).

Note: network-layer responsibilities generally outside of the scope of a blockchain protocol's specification.

  – spec = tx semantics, validity for txs and blocks, etc.

- network layer functionality implemented in clients
  – typically one dominant client/network (but can have multiple)

The Bitcoin Network

Question: if validators unknown, how do they communicate?

The Bitcoin Network

Question: if validators unknown, how do they communicate?

Idea: txs + blocks disseminated via “peer-to-peer (P2P)” network.

- participants (“nodes”) download from each other, not a server
  - a “gossip network” (note: no key-value store functionality needed)

The Bitcoin Network

Question: if validators unknown, how do they communicate?

Idea: txs + blocks disseminated via “peer-to-peer (P2P)” network.

- participants (“nodes”) download from each other, not a server
  - a “gossip network” (note: no key-value store functionality needed)
- each node maintains two-way connections with small # of “peers”
  - e.g., connect to 8 random peers when joining the network
    - if nothing else, start from nodes hard-coded into client software
  - communicate only with peers, rely on multi-hop paths for all other nodes

The Bitcoin Network

Question: if validators unknown, how do they communicate?

Idea: txs + blocks disseminated via “peer-to-peer (P2P)” network.

- participants (“nodes”) download from each other, not a server
  - a “gossip network” (note: no key-value store functionality needed)
- each node maintains two-way connections with small # of “peers”

Terminology: Bitcoin has a “public mempool.”

- i.e., any node can keep track of pending txs (just join P2P network)
  - in particular, all miners will want to do this

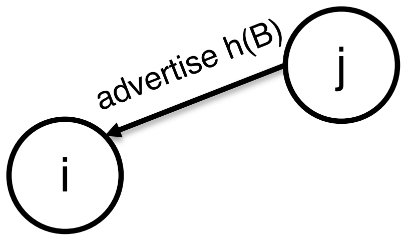
Communication in the Bitcoin Network

Idea #2: disseminate txs + blocks through P2P network via “flooding.”

Communication in the Bitcoin Network

Idea #2: disseminate txs + blocks through P2P network via “flooding.”

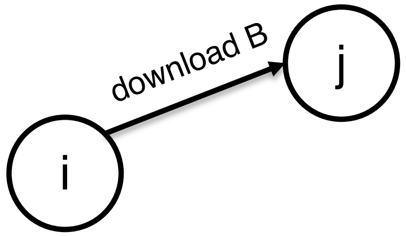
- when node i hears from a peer j about the hash of a tx (h(tx)) or block (h(B)) for the first time:



Communication in the Bitcoin Network

Idea #2: disseminate txs + blocks through P2P network via “flooding.”

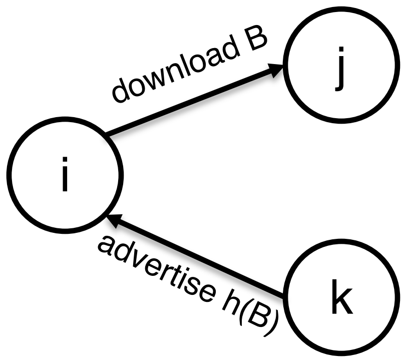
- when node i hears from a peer j about the hash of a tx (h(tx)) or block (h(B)) for the first time:
  - request from j the full download of tx or B



Communication in the Bitcoin Network

Idea #2: disseminate txs + blocks through P2P network via “flooding.”

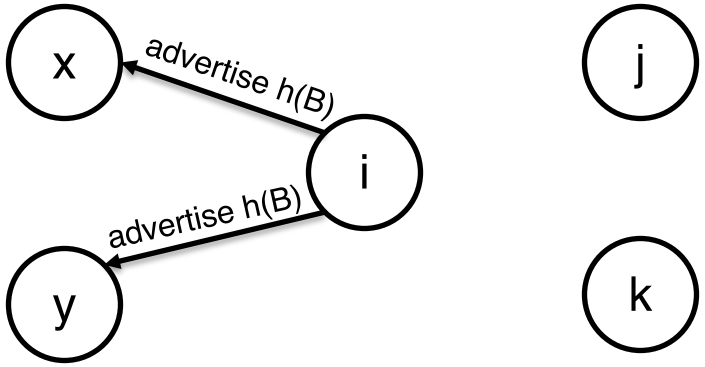
- when node i hears from a peer j about the hash of a tx (h(tx)) or block (h(B)) for the first time:
  - request from j the full download of tx or B
  - do some basic validity checks (e.g., B includes appropriate proof-of-work)



Communication in the Bitcoin Network

Idea #2: disseminate txs + blocks through P2P network via “flooding.”

- when node i hears from a peer j about the hash of a tx (h(tx)) or block (h(B)) for the first time:
  - request from j the full download of tx or B
  - do some basic validity checks (e.g., B includes appropriate proof-of-work)
  - advertise h(tx) or h(B) to all peers that might not have seen it yet



Communication in the Bitcoin Network

Idea #2: disseminate txs + blocks through P2P network via “flooding.”

- when node i hears from a peer j about the hash of a tx (h(tx)) or block (h(B)) for the first time:
  - request from j the full download of tx or B
  - do some basic validity checks (e.g., B includes appropriate proof-of-work)
  - advertise h(tx) or h(B) to all peers that might not have seen it yet

Hope: tx/block reaches all n nodes in P2P network in ≈ log n hops.

- intuition: each hop should increase # of nodes reached by constant factor
- but no guarantee that P2P network is even connected!

Overcoming Drawbacks of Flooding

Idea #2: disseminate txs + blocks through P2P network via “flooding.”

- when node i hears about tx/block for first time, download it and (if valid) advertise it to all peers that might not have seen it yet

Note: if P2P network has n nodes and m connections:

Overcoming Drawbacks of Flooding

Idea #2: disseminate txs + blocks through P2P network via “flooding.”

- when node i hears about tx/block for first time, download it and (if valid) advertise it to all peers that might not have seen it yet

Note: if P2P network has n nodes and m connections:

- each tx/block will be downloaded ≈ n times [good]

Overcoming Drawbacks of Flooding

Idea #2: disseminate txs + blocks through P2P network via “flooding.”

- when node i hears about tx/block for first time, download it and (if valid) advertise it to all peers that might not have seen it yet

Note: if P2P network has n nodes and m connections:

- each tx/block will be downloaded ≈ n times [good]
- each tx/block will be advertised ≈ m times [less good]

Overcoming Drawbacks of Flooding

Idea #2: disseminate txs + blocks through P2P network via "flooding."

- when node i hears about tx/block for first time, download it and (if valid) advertise it to all peers that might not have seen it yet

Note: if P2P network has n nodes and m connections:

- each tx/block will be downloaded ≈ n times [good]
- each tx/block will be advertised ≈ m times [less good]
- another issue: propagation delay (in the seconds)
  - multiple hops, unrelated to underlying geography

Overcoming Drawbacks of Flooding (con’d)

Flooding drawbacks: redundant advertisements, slow dissemination.

Erlay: proposal to reduce communication overhead.

Overcoming Drawbacks of Flooding (con’d)

Flooding drawbacks: redundant advertisements, slow dissemination.

Erlay: proposal to reduce communication overhead.

- use only a subset of connections for advertising
  - e.g., even if have many peers, advertise along random subset of 8 of them
  - reduces communication overhead but increases propagation delay

Overcoming Drawbacks of Flooding (con’d)

Flooding drawbacks: redundant advertisements, slow dissemination.

Erlay: proposal to reduce communication overhead.

- use only a subset of connections for advertising
  - e.g., even if have many peers, advertise along random subset of 8 of them
  - reduces communication overhead but increases propagation delay
- replace “last mile” of flooding with “set reconciliation problem”
  - two peers identify + exchange set difference of what they’ve received

Overcoming Drawbacks of Flooding (con’d)

Flooding drawbacks: redundant advertisements, slow dissemination.

Erlay: proposal to reduce communication overhead.

- use only a subset of connections for advertising
- replace “last mile” of flooding with “set reconciliation problem”

Overlay network: subset of P2P nodes maintain direct (and possibly very fast) connections with each other.

Overcoming Drawbacks of Flooding (con’d)

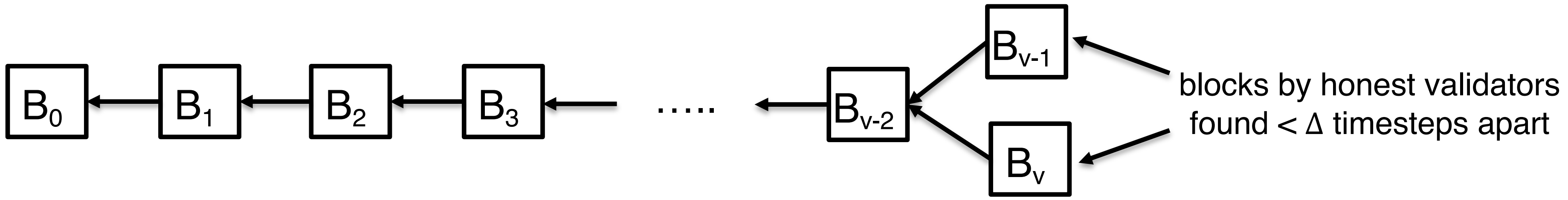
Flooding drawbacks: redundant advertisements, slow dissemination.

Erlay: proposal to reduce communication overhead.

– use only a subset of connections for advertising
– replace “last mile” of flooding with “set reconciliation problem”

Overlay network: subset of P2P nodes maintain direct (and possibly very fast) connections with each other.

- used by large miners to avoid inadvertent forks:



Incentives for Honest Behavior?

Question: what about malicious (or just lazy) P2P nodes?

Incentives for Honest Behavior?

Question: what about malicious (or just lazy) P2P nodes? E.g.:

- don't bother to advertise new txs/blocks
- don't respond to download requests
- skip validity check, disseminate invalid txs or blocks
- make up fake hashes

Incentives for Honest Behavior?

Question: what about malicious (or just lazy) P2P nodes? E.g.:

- don’t bother to advertise new txs/blocks
- don’t respond to download requests
- skip validity check, disseminate invalid txs or blocks
- make up fake hashes

Ad hoc solution: each node tracks the “penalty points” accrued by its peers, drop peers once they’ve accumulated too many.

Incentives for Honest Behavior?

Question: what about malicious (or just lazy) P2P nodes? E.g.:

- don’t bother to advertise new txs/blocks
- don’t respond to download requests
- skip validity check, disseminate invalid txs or blocks
- make up fake hashes

Ad hoc solution: each node tracks the “penalty points” accrued by its peers, drop peers once they’ve accumulated too many.

- note: not all peer misbehaviors are easily identified
  - e.g., skipping validity checks or advertising

Ethereum and Gossipsub

Communication in the Ethereum network:

Ethereum and Gossipsub

Communication in the Ethereum network:

- historically, a public mempool (i.e., txs disseminated via gossip)
  - less and less true over time due to “MEV”/”private order flow” (next week)

Ethereum and Gossipsub

Communication in the Ethereum network:

- historically, a public mempool (i.e., txs disseminated via gossip)
  - less and less true over time due to “MEV”/”private order flow” (next week)
- historically, blocks disseminated via gossip (+ overlay networks)
  - gossip protocol = “gossipsub” (cf., “pubsub,” for “publish-subscribe”)

Ethereum and Gossipsub

Communication in the Ethereum network:

- historically, a public mempool (i.e., txs disseminated via gossip)
  - less and less true over time due to “MEV”/”private order flow” (next week)
- historically, blocks disseminated via gossip (+ overlay networks)
  - gossip protocol = “gossipsub” (cf., “pubsub,” for “publish-subscribe”)
  - each message annotated with a “topic”
  - nodes “subscribe” to topics, maintain topic-specific peers

Ethereum and Gossipsub

Communication in the Ethereum network:

- historically, a public mempool (i.e., txs disseminated via gossip)
  - less and less true over time due to "MEV"/"private order flow" (next week)
- historically, blocks disseminated via gossip (+ overlay networks)
  - gossip protocol = "gossipsub" (cf., "pubsub," for "publish-subscribe")
  - each message annotated with a "topic"
  - nodes "subscribe" to topics, maintain topic-specific peers

New challenge: dissemination of consensus-related data (e.g. votes).

Ethereum and Gossipsub

Communication in the Ethereum network:

- historically, a public mempool (i.e., txs disseminated via gossip)
  - less and less true over time due to “MEV”/”private order flow” (next week)
- historically, blocks disseminated via gossip (+ overlay networks)
  - gossip protocol = “gossipsub” (cf., “pubsub,” for “publish-subscribe”)
  - each message annotated with a “topic”
  - nodes “subscribe” to topics, maintain topic-specific peers

New challenge: dissemination of consensus-related data (e.g. votes).

- source of challenge: Ethereum has > 1 million validators!

Gossiping Attestations

Challenge: Ethereum has > 1 million validators (32 ETH each).

- roughly 25% of all ETH in circulation

Gossiping Attestations

Challenge: Ethereum has > 1 million validators (32 ETH each).

- – roughly 25% of all ETH in circulation

- cannot run a Tendermint-type protocol with 12-second block times
  - – especially with relatively low validator hardware requirements

Gossiping Attestations

Challenge: Ethereum has > 1 million validators (32 ETH each).

- – roughly 25% of all ETH in circulation

- cannot run a Tendermint-type protocol with 12-second block times
  - – especially with relatively low validator hardware requirements
  - – too much communication (votes), too much computation (sig verification)

Gossiping Attestations

Challenge: Ethereum has > 1 million validators (32 ETH each).

- roughly 25% of all ETH in circulation

- cannot run a Tendermint-type protocol with 12-second block times
  - especially with relatively low validator hardware requirements
  - too much communication (votes), too much computation (sig verification)
  - but if want slashing guarantees ➔ need to use such a protocol

Gossiping Attestations

Challenge: Ethereum has > 1 million validators (32 ETH each).

- cannot run a Tendermint-type protocol with 12-second block times

Ethereum's solution:

Gossiping Attestations

Challenge: Ethereum has > 1 million validators (32 ETH each).

- cannot run a Tendermint-type protocol with 12-second block times

Ethereum's solution:

- blocks produced every 12 seconds via longest-chain-type protocol
  - with idiosyncratic fork-choice rule ("GHOST")

Gossiping Attestations

Challenge: Ethereum has > 1 million validators (32 ETH each).

- cannot run a Tendermint-type protocol with 12-second block times

Ethereum's solution:

- blocks produced every 12 seconds via longest-chain-type protocol
  - with idiosyncratic fork-choice rule ("GHOST")
- one "checkpoint" finalized each "epoch" (= 32 views = 6.4 minutes)
  - Tendermint-style votes ("attestations") accumulate over these views

Gossiping Attestations

Challenge: Ethereum has > 1 million validators (32 ETH each).

- cannot run a Tendermint-type protocol with 12-second block times

Ethereum's solution:

- blocks produced every 12 seconds via longest-chain-type protocol
  - with idiosyncratic fork-choice rule ("GHOST")
- one "checkpoint" finalized each "epoch" (= 32 views = 6.4 minutes)
  - Tendermint-style votes ("attestations") accumulate over these views
- within a slot, validators split into 64 committees
  - currently ≈ 30K validators per slot, ≈ 500 members per committee

Gossiping Attestations

Challenge: Ethereum has > 1 million validators (32 ETH each).

- – cannot run a Tendermint-type protocol with 12-second block times

Ethereum's solution:

- blocks produced every 12 seconds via longest-chain-type protocol
- one "checkpoint" finalized each "epoch" (= 32 views = 6.4 minutes)
- within a slot, validators split into 64 committees
    - – aggregation of (BLS) signatures within each committee (in parallel)

Gossiping Attestations

Challenge: Ethereum has > 1 million validators (32 ETH each).

– cannot run a Tendermint-type protocol with 12-second block times

Ethereum's solution:

- blocks produced every 12 seconds via longest-chain-type protocol
- one "checkpoint" finalized each "epoch" (= 32 views = 6.4 minutes)
- within a slot, validators split into 64 committees
  - aggregation of (BLS) signatures within each committee (in parallel)
- in gossipsub, use one topic per committee
  - publish attestations to that topic; sig aggregators subscribe to that topic

More on Gossipsub

Gossipsub innovations:

More on Gossipsub

Gossipsub innovations:

- allow any number of peers

More on Gossipsub

Gossipsub innovations:

- allow any number of peers
- “eager push” of new data to small subset of peers (“mesh”)
  - get fast dissemination if mesh peers form well-connected network

More on Gossipsub

Gossipsub innovations:

- allow any number of peers
- “eager push” of new data to small subset of peers (“mesh”)
  - get fast dissemination if mesh peers form well-connected network
- periodic gossiping of metadata to all peers (a la Bitcoin network)
  - backup to ensure that all messages eventually propagate

More on Gossipsub

Gossipsub innovations:

- allow any number of peers
- “eager push” of new data to small subset of peers (“mesh”)
  - get fast dissemination if mesh peers form well-connected network
- periodic gossiping of metadata to all peers (a la Bitcoin network)
  - backup to ensure that all messages eventually propagate
- each node maintains “score” for each of its peers

More on Gossipsub

Gossipsub innovations:

- allow any number of peers
- "eager push" of new data to small subset of peers ("mesh")
  - get fast dissemination if mesh peers form well-connected network
- periodic gossiping of metadata to all peers (a la Bitcoin network)
  - backup to ensure that all messages eventually propagate
- each node maintains "score" for each of its peers
  - increases e.g. if first to report a msg, decreases e.g. if send invalid msgs
  - "mesh peers" generally the peers with the highest scores

More on Gossipsub

Gossipsub innovations:

- allow any number of peers
- "eager push" of new data to small subset of peers ("mesh")
  - get fast dissemination if mesh peers form well-connected network
- periodic gossiping of metadata to all peers (a la Bitcoin network)
  - backup to ensure that all messages eventually propagate
- each node maintains "score" for each of its peers
  - increases e.g. if first to report a msg, decreases e.g. if send invalid msgs
  - "mesh peers" generally the peers with the highest scores
  - issues: can be gamed, non-trivial barriers to becoming mesh peer

Solana’s Network Layer

Recall: target block rate = 1 block/400ms.

Solana's Network Layer

Recall: target block rate = 1 block/400ms.

- gossip used primarily to keep validator info up-to-date

Solana's Network Layer

Recall: target block rate = 1 block/400ms.

- gossip used primarily to keep validator info up-to-date
- tx dissemination: send txs to upcoming leaders (known in advance)

Solana's Network Layer

Recall: target block rate = 1 block/400ms.

- gossip used primarily to keep validator info up-to-date
- tx dissemination: send txs to upcoming leaders (known in advance)
- Tendermint-style votes treated as a form of transaction
  - accumulate over multiple blocks (somewhat reminiscent of Ethereum)

Solana's Network Layer

Recall: target block rate = 1 block/400ms.

- gossip used primarily to keep validator info up-to-date
- tx dissemination: send txs to upcoming leaders (known in advance)
- Tendermint-style votes treated as a form of transaction
  - accumulate over multiple blocks (somewhat reminiscent of Ethereum)
- block dissemination done via "Turbine"
  - important component of Solana's performance

Solana's Turbine

High-level idea:

- leader of view disseminates block as it assembles it

Solana’s Turbine

High-level idea:

- leader of view disseminates block as it assembles it
- basic unit = one “shred” (fits in UDP packet, say 8 txs)

Solana’s Turbine

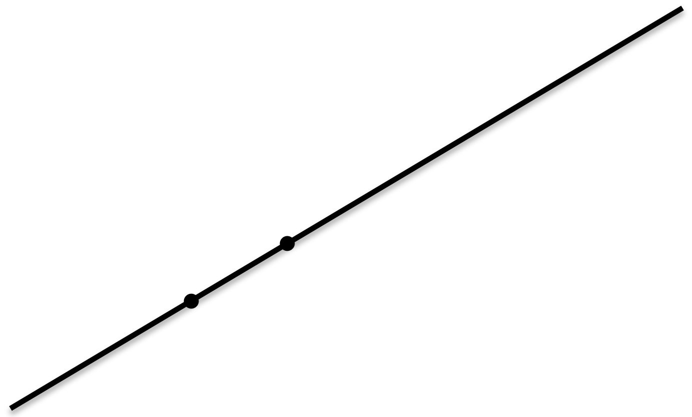
High-level idea:

- leader of view disseminates block as it assembles it
- basic unit = one “shred” (fits in UDP packet, say 8 txs)
- takes k shreds, use Reed-Solomon erasure code to produce (say) k/2 “recovery shreds” (cf., parity bits)

Reed-Solomon Codes

Recall: any two points determine a (unique) line.

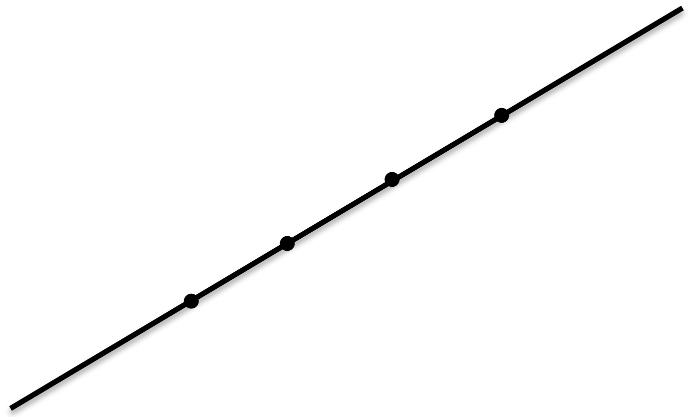
- example: points (1,4) and (2,9) ➔ line y = 5x-1



Reed-Solomon Codes

Recall: any two points determine a (unique) line.

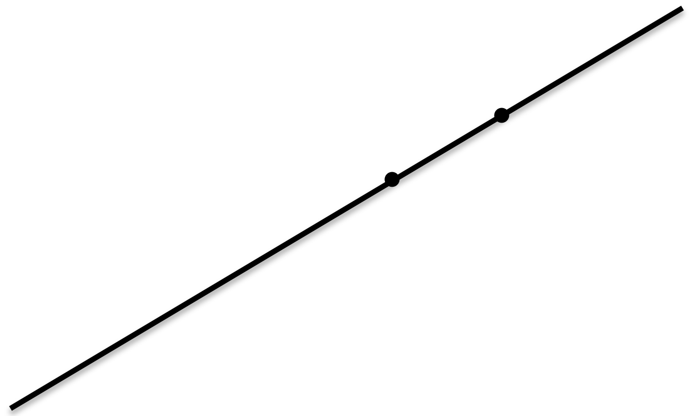
- example: points (1,4) and (2,9) ➔ line y = 5x-1
- note: could redundantly encode line y = 5x -1 via the points (1,4), (2,9), (3,14), and (4,19)



Reed-Solomon Codes

Recall: any two points determine a (unique) line.

- example: points (1,4) and (2,9) ➔ line y = 5x-1
- note: could redundantly encode line y = 5x -1 via the points (1,4), (2,9), (3,14), and (4,19)
  - – if later remember only (3,14) and (4,19), can still recover line 5x-1



Solana’s Turbine

High-level idea:

- leader of view disseminates block as it assembles it
- basic unit = one “shred” (fits in UDP packet, say 8 txs)
- takes k shreds, use Reed-Solomon erasure code to produce (say) k/2 “recovery shreds” (cf., parity bits)

Solana's Turbine

High-level idea:

- leader of view disseminates block as it assembles it
- basic unit = one "shred" (fits in UDP packet, say 8 txs)
- takes k shreds, use Reed-Solomon erasure code to produce (say) k/2 "recovery shreds" (cf., parity bits)
- send 3k/2 shreds to 3k/2 validators, who forward shreds to all validators
  - point: leader no longer a communication bottleneck

Solana’s Turbine

High-level idea:

- leader of view disseminates block as it assembles it
- basic unit = one “shred” (fits in UDP packet, say 8 txs)
- takes k shreds, use Reed-Solomon erasure code to produce (say) k/2 “recovery shreds” (cf., parity bits)
- send 3k/2 shreds to 3k/2 validators, who forward shreds to all validators
  - point: leader no longer a communication bottleneck
- validators reconstruct block (or portion of block) from shreds, proceed to executing those txs

Narwhal

Starting point: with a public mempool, every tx disseminated twice (once initially, as second time as part of a block proposal).

Narwhal

Starting point: with a public mempool, every tx disseminated twice (once initially, as second time as part of a block proposal).

Idea: have block dissemination reuse work from tx dissemination.

- to illustrate, assume permissioned setting with n validators

Narwhal

Starting point: with a public mempool, every tx disseminated twice (once initially, as second time as part of a block proposal).

Idea: have block dissemination reuse work from tx dissemination.

- – to illustrate, assume permissioned setting with n validators

- each "round" r, each validator i sends list L_{ir} of txs to all validators
  - – along with > 2n/3 round-(r-1) "certificates" to justify being in round r

Narwhal

Starting point: with a public mempool, every tx disseminated twice (once initially, as second time as part of a block proposal).

Idea: have block dissemination reuse work from tx dissemination.

  – to illustrate, assume permissioned setting with n validators

- each "round" r, each validator i sends list L_{ir} of txs to all validators
  – along with > 2n/3 round-(r-1) "certificates" to justify being in round r
- if L_{ir} is valid, validators send back (signed) ack messages

Narwhal

Starting point: with a public mempool, every tx disseminated twice (once initially, as second time as part of a block proposal).

Idea: have block dissemination reuse work from tx dissemination.

  – to illustrate, assume permissioned setting with n validators

- each "round" r, each validator i sends list L_{ir} of txs to all validators
  – along with > 2n/3 round-(r-1) "certificates" to justify being in round r
- if L_{ir} is valid, validators send back (signed) ack messages
- "certificate" for L_{ir} = signed acks by > 2n/3 validators

Narwhal

Starting point: with a public mempool, every tx disseminated twice (once initially, as second time as part of a block proposal).

Idea: have block dissemination reuse work from tx dissemination.

- – to illustrate, assume permissioned setting with n validators

- each "round" r, each validator i sends list L_{ir} of txs to all validators
  - – along with > 2n/3 round-(r-1) "certificates" to justify being in round r
- if L_{ir} is valid, validators send back (signed) ack messages
- "certificate" for L_{ir} = signed acks by > 2n/3 validators
- i sends (round-r) certificate for L_{ir} to all validators

Narwhal (con’d)

Transaction dissemination:

- each “round” r, each validator i sends list L_{ir} of txs to all validators
  - along with > 2n/3 round-(r-1) “certificates” to justify being in round r
- if L_{ir} is valid, validators send back (signed) ack messages
- “certificate” for L_{ir} = signed acks by > 2n/3 validators
- i sends (round-r) certificate for L_{ir} to all validators

Block proposing: pack block with certificates, not txs.

Narwhal (con’d)

Transaction dissemination:

- each “round” r, each validator i sends list L_{ir} of txs to all validators
  - along with > 2n/3 round-(r-1) “certificates” to justify being in round r
- if L_{ir} is valid, validators send back (signed) ack messages
- “certificate” for L_{ir} = signed acks by > 2n/3 validators
- i sends (round-r) certificate for L_{ir} to all validators

Block proposing: pack block with certificates, not txs.

- < n/3 Byzantine validators ➔ data availability for all certificates
  - validators can download missing txs from other as needed