Verifiable bandwidth sampling in P2P networks

Liam Zebedee

Dappnet

Abstract. Decentralized data hosting is blocked by the ability to guarantee speed of access. We propose a mechanism called verifiable bandwidth sampling, wherein a node samples another node's speed for solving a continuous off-chain data availability challenge. The challenge is designed to fit within the timekeeping of a verifiable clock - a blockchain consensus protocol. And in doing so, the bandwidth of a connection is securely measured. Building upon this result, we can design an automated market maker for data hosting (storage and bandwidth) which can scale horizontally in capacity.

Keywords: decentralized frontends · Mechanism design · Data availability sampling · P2P networks.

1 Defining bandwidth

What is bandwidth?

$$\text{Bandwidth} = \frac{\text{Amount of Data}}{\text{Time}}$$

For the bandwidth in the path A->B for nodes A and B, the bandwidth is defined as:

$$\text{Bandwidth} = \frac{\text{Amount of Data}}{T_{\text{received}} - T_{\text{sent}}}$$

Our goal is to come up with a way to measure this. To begin, let's look at hash-lock time contracts.

1.1 Atomic swaps using HLTC's

Say we have a blockchain where we can deploy smart contracts.

A hashlock time contract (HLTC) [4] is a type of smart contract where a reward is locked inside a contract, and can only be claimed if you reveal a preimage to a specific hash.

What's interesting about a HLTC is how they are used for cross-chain atomic swaps.

There are two nodes, and each one has coins they want to atomically swap. Alice has bitcoins, Bob has ether.

1. Alice generates a random secret s, and computes the hash of the secret, hash(s) = h. Alice sends h to Bob.

2. Alice and Bob both lock their asset into a smart contract with the following rules (Alice locks first, Bob locks after seeing Alice's asset successfully locked). On Alice's side, if the secret is provided within 2X seconds, then the asset is transferred to Bob, otherwise it is sent back to Alice. On Bob's side, if the correct secret (ie. the value whose hash is h) is provided within X seconds, then the asset is transferred to Alice, otherwise it is sent back to Bob.

3. Alice reveals the secret within X seconds in order to claim the asset from Bob's contract. However, this also ensures that Bob learns the secret allowing Bob to claim the asset from Alice's contract.

This atomic swap is demonstrably causal. By Alice claiming the coins, and revealing her secret, she causes Bob to be able to claim his coins.

Can we use this casual chain to our advantage?

1.2 Onion routing

In onion routing [5], a node encrypts a data packet to be sent through a specific network path. For a packet that follows a path of A->B->C->D->E, the packet is encrypted as $P = enc(enc(enc(enc, A), B), C), D), E)$. Each node "unwraps" the packet by decrypting each layer and sending it to the next node.

The causality here forces a packet to be proxied through multiple nodes, which provides some privacy in terms of the origins of the packet.

Logically, the packet can **only** be read by E if it has passed through A, B, C and D for decryption.

What's interesting is that this enforces a form of verifiable transmission. Since a node must send it to the next node, in order for the next node to receive it, etcetera.

Using onion routing, we can construct a minimal proof of data transfer. Each node records the hash of the layer they decrypted to a blockchain like Ethereum [3]. The time between two recordings is at minimum the time it took to send the packet from the 1st to the 2nd node, since logically, they could only receive the packet if it has proceeded through this onion path.

1.3 A naive proof-of-data-transfer protocol

We could construct a contract for proof-of-data-transfer, by blending an onion-route style mechanism with the incentive design of a hash-lock time contract.

We commit to a hash of the data upfront, and lock a reward. We then post the packet from node A to node B. Node B races to download the packet, and then decrypt it with their keypair. As soon as they do, they submit a transaction to reveal and claim their reward.

In doing this, we inadvertently measure the **time** it takes for node B to receive this data from node A.

To recap the phases of this protocol:

1. **Setup**. Node A sets up the data transfer contract, committing a reward that can be claimed by revealing the preimage - which is a secret contained in the packet. The contract records the start time.
2. **Send**. Node A constructs the packet containing the secret, and encrypts it with Node B's key so only they can decrypt it. They send the packet.
3. **Receive**. Node B receives the packet, decrypts it to get the secret. They submit the secret on-chain to claim the reward.
4. **Measure speed**. The contract records the receipt time, and calculates the latency as
$$latency = T_{\text{received}} - T_{\text{start}}$$
.

But there are problems:

1. **Collusion**. Node A could preshare the secret with node B, and collude against the protocol.
2. **Inaccurate timekeeping**. Using a blockchain means using the block as a verifiable tick for a clock. But if a block comes every 12s, that means our timing can only be as accurate as the minimum tick - every 12s.

Collusion is solved by using an unpredictable challenge value in place of the secret. Instead of a secret, Alice commits to a challenge predicate - can node B provide a random chunk of data according to a random oracle index at a specific time in future? This is similar to a data availability challenge [2] [1]. The random oracle could be defined as the block hash in 10 blocks, and this could be modulo the data string as to be a random data availability challenge. When the node receives the packet B, they must provide the hashes of some data leaf at a random index plus some seed value, thus ensuring neither A or B can collude ahead of time.

Solving inaccurate timekeeping is detailed below.

1.4 Accurate timekeeping in the verifiable clock model

This protocol would suffice to measure latency, but our verifiable clock, the blockchain, is not granular enough in its measurement. Keep in mind - the block time of Ethereum proof-of-stake is 12s, the block time of Solana is at least 250ms. We can only measure time in increments as small as the block time - for Ethereum this is 12s increments. The latency of sending a simple ping packet is somewhere in the range of 5- 500ms. Steps 1 and 4 of this protocol would occur within a single block! And the latency would be 0.

Let's consider a different frame of mind - a very large data packet that couldn't be transferred in a single block. Consider the time it takes to transfer a packet the size of 3GB over a link of 50 MB/s. This would take $3000/50 = 60$ seconds. Now considering that our verifiable clock, the blockchain, has a tick of roughly 12s (the consensus rate) - this means the transmission would take place over many blocks. This means we could measure the latency to some accuracy.

By sampling enough data such that it exceeds the minimum clock tick of one block (12s), we can properly measure the bandwidth of a node. We can use two forms of timekeeping - real time (in terms of Unix clock time to millisecond accuracy) and block time (in terms of block height). In the previous example, the transmission latency in real time is $3000/50 = 60$ seconds. $60/12 = 5$ blocks. The data transmission speed was 3GB/5 blocks, or 0.6GB/block.

For a home internet connection of 10Mbits/s, or 1.2MB/s, this would entail transferring 3MB's of data. Perhaps instead of fixing the amount of data, we can fix the time frame.

1.5 Sending a verifiable rate of data

In the previous section, we managed to verify bandwidth by sending a verifiable amount of data and measuring the timeframe. But the data size is quite big - 3GB. For a large network of nodes being sampled regularly, this would be a lot of used bandwidth. Is there a more efficient mechanism?

$$\text{Bandwidth} = \frac{\text{Amount of Data}}{\text{Time}}$$

Recall the definition of bandwidth. Instead of fixing the amount of data variable, what if we fixed the time? Our protocol would test a node by measuring how much data it can send in 20 blocks?

Imagine we have an entropy generator, which we can use to perform a continuous data availability sample.

2 Verifiable bandwidth sampling

We call this technique **verifiable bandwidth sampling**, in reference to data availability sampling popularised by Celestia and Ethereum [2] [1] [3].

Verifiable bandwidth sampling is a technique to establish a minimum bound on the transfer speed between two nodes, in a byzantine security model. The data is transmitted in an onion route, which ensures a form of verifiable transmission. Once the node receives the packet, they record the time to a blockchain. The blockchain acts as a verifiable clock that cannot be manipulated. The node receives a reward for recording the packet receipt, according to a hash-lock time contract -like mechanism. To prevent collusion, the sampler locks the reward according to an unpredictable challenge - a data availability sample based on a future blockhash.

2.1 Implications

We can use verifiable bandwidth sampling as a way to build an automated market maker (AMM) for bandwidth, and ultimately, a cryptonetwork for data hosting and retrieval.

3 A fast network for hosting decentralized frontends

Consider the problem of decentralized frontends - how do we pay for frontend hosting in a way which guarantees speed of access?

A blockchain is not a scalable solution. Storing data in blockchains is expensive, and they do not scale storage capacity by adding more nodes. They are replicated state machines.

Filecoin and Arweave are demonstrably not solutions either, since they do not guarantee download speeds (bandwidth). They also do not feature an economic model which allows the network to grow/shrink hosting capacity in response to demand.

What is needed is a network that has measured capacity in terms of two resources: storage and bandwidth. That can dynamically incent more capacity when there is high download demand.

We propose creating this using the verifiable bandwidth sampling mechanism:

1. Nodes join the network, and contribute storage and bandwidth.
2. Their capacity is regularly verified by means of verified bandwidth sampling.
3. The network sells hosting (storage + bandwidth) using an automated market maker. The market maker prices bandwidth and storage using a constant product curve.

Using this network, it should be possible for anyone to buy hosting for data as easily as they buy blockspace on Ethereum.

Imagine this for decentralized frontends. The average size of a simple frontend (HTML, JS, CSS, assets) is 5MB. Consider a network of one node with 5MB/s

upload and 5MB of storage. The network's hosting capacity can support 1 user downloading a dapp per second. Now imagine that there are 10 users who want to download a dapp. The supply is 1 and the demand is 10. What if the network were able to dynamically increase the reward for contributing hosting capacity to incent new nodes to come online, the same way Uber surge pricing incentivises more drivers to come online?

Beyond decentralized frontends, we believe this service is valuable for autonomous AI agents. We believe human consciousness will be uploaded into the machine soon - in fact, this is already possible today. Designing a simple prompt in the style of "your name is Liam, you are an Australian idiot with a LaTeX editor" is enough to clone 1 bit of personality into a digital replica of yourself, and this will increase in detail in the decades to come. Soon too will the AI models will be able to perform economically-meaningful work, and where will they domicile? Traditional cloud service providers like AWS are garrisoned by jurisdiction and law - we believe the AI's will naturally choose to reside in cyberspace, where they are unencumbered by KYC and other mechanisms. They will use blockchains to store their wealth, and to store their machine body (data)? We believe they will need an elastic substrate, which this protocol is designed to offer.

References

1. Al-Bassam, M. Lazyledger: A distributed data availability ledger with client-side smart contracts, 2019.
2. Al-Bassam, M., Sonnino, A., and Buterin, V. Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities. *CoRR abs/1809.09044* (2018).
3. Buterin, V. A next generation smart contract & decentralized application platform.
4. Buterin, V. Chain interoperability. Tech. rep., 9 2016.
5. Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In *In Proceedings of the 13th Usenix Security Symposium* (2004).