Proof of Real-time Transfer: A Consensus Protocol for Decentralized Data Exchange

Narayanan Ramanathan
Kandola Network
nara@kandola.network

Ragul Kumar
Kandola Network
gul@kandola.network

Siddharth Banerjee
Kandola Network
sb@kandola.network

Sriram Padmanabhan
Kandola Network
sriram@kandola.network

***Abstract*—Proof of Real-Time Transfer (PoRT) consensus is a secure, highly scalable, decentralized data verification protocol that supports the exchange of data with guarantees towards privacy, data integrity and source authentication. It is built on a modular framework that serves core functionalities towards decentralized public key infrastructure (DPKI), message standardization and encryption, decentralized message verification and storage. PoRT's security guarantees stem from its adoption of cryptographic primitives and their nuanced derivatives that confer verifiability and further game theoretic constructs that prove pertinence in work. High scalability of PoRT can be attributed to network sharding and compute sharding that optimize resource usage and to parallel chains that confer instant finality. PoRT allows applications to define and implement purpose-built message-driven business solutions across industries like IoT, social media, instant messaging and gaming. Our analysis indicates that PoRT consensus can achieve throughputs upto $976K$ messages per second on today's commodity hardware.**

***Index Terms*—blockchain, consensus protocol, network sharding, compute sharding, parallel chains, decentralized data exchange**

I. Introduction

The heart of decentralization resides in the building of a trustless environment where decision making is placed in the hands of distributed peer nodes as opposed to centralized authorities. Through the adoption of decentralized identities [1], the digital signature scheme [2], and distributed ledger technology, decentralized networks confer privacy and data security by design and are designed to be highly fault tolerant. Depending upon the nature of the consensus protocol that is powering the decentralized network, transaction throughput and nature of finality vary from one to another. Proof of Work [3], [4], Proof of Stake [5], [6], Pure Proof of Stake [7], Proof of History [8] are some of the popular decentralized protocols designed and deployed in the Decentralized Finance (DeFi) space. A common theme among such protocols is that improved scalability (high transactions-per-second and low latency) is achieved at the cost of either security or the degree of decentralization or both.

While there has been considerable focus in facilitating DeFi platforms and the applications they serve, similar such traction in developing consensus protocols that can power far more encompassing decentralized data exchange platforms with steeper operational demands is notably absent. Today data exchange happens predominantly on centralized platforms that are custom-built to serve individual data domains with negligible emphasis on data standardization across domains and key data handling protocols for security and privacy of data all through its life-cycle. This paradigm has resulted in data siloing which in turn has introduced numerous challenges in data reusability, in data interoperability and largely in establishing data provenance. In the ideal setting, we need a fully decentralized data exchange platform that secures data end-to-end, guarantees data privacy, authenticates source of data and is built to support applications that demand very high throughputs and low latency, while remaining content agnostic. Such a platform restores data ownership back at the hands of the producer and unlocks the true potentials of decentralized data marketplaces.

In this paper, we are proposing a new consensus protocol called Proof of Real-time Transfer (PoRT) that powers decentralized data exchange platforms with the capabilities listed above. The protocol is designed to facilitate the secure data exchange between two authenticated entities upon running a light-weight deterministic data verification algorithm in a decentralized setting. The protocol is highly scalable and poses strong defense against many kinds of malicious attacks.

II. Relevant Work

Open Data Fabric [9] is a decentralized data exchange and transformation protocol that is designed to make multi-party data management more sustainable. Data from trusted publishers flows through a computational graph undergoing deterministic transformations and is made readily ingestible for the AI / ML community for model training and for data collaborators in general. DIoTA [10] is a decentralized ledger based authentication framework that enables low powered IoT devices with limited computational capacity to authenticate other devices without expending too much energy. Roll-DPoS [11] is a randomized scalable variant of Delegated Proof-of-Stake that powers decentralized Internet-of-Things. While retaining the scalability aspects of DPoS, Roll-DPoS adopts distributed key generation, BLS threshold signature [12] and random beacons in enhancing decentralization. [13] proposes solutions for decentralized networks serving IoT applications to defend against denial-of-service attacks using verifiable delay functions. A decentralized message exchange protocol is proposed in [14] whereby transactions between two users are executed on the Ethereum public blockchain by deploying smart contracts that perform user authentication and message verification. A consensus protocol that builds trust models on

978-8-3503-1019-1/23/$31.00 ©2023 IEEE

nodes and messages is proposed in [15] catering to communication relay in vehicular ad-hoc networks.

An alternate paradigm of solutions in the name of *Remote Attestation* [16] exist for nodes in a decentralized network to reason about the state of its untrusted peers. A node (called the *Verifier*) can attest to the state of another node or device (called the *Prover*) through hardware-based attestation (Trusted Platform Modules [17]) or software-based attestation or hybrid approaches. While hardware based attestation leverages specialized hardware modules that enable secure storage, secure computation and dedicated processor architectures (Intel SGX, ARM TrustZone), software based attestation assess the software state of its remote peers through programs that assess the memory reads and writes. Such approaches are bound to play a critical role in the evolution of Blockchain solutions.

III. Decentralized Data Exchange Platform

A. Network model

The decentralized data exchange platform comprises a set of nodes, each that meet a minimum criteria identified for compute, memory and network speed. Each node serves as a full-node catering to functionalities spanning message handling, message verification and storage on-chain and get rewarded for the roles served. They are subject to outages and attacks just as public nodes typically are. They can communicate with one another either directly (port-to-port) or through gossip protocol. PoRT protocol is designed to function in a partially synchronous network setting, whereby nodes share a global clock and communication delays can be arbitrary up to an unknown global stabilization time (GST). PoRT protocol is designed for safety (always) and liveness (eventually, post-GST).

B. Decentralized Public Key Infrastructure

Every actor (producer, consumer, node) that sends or receives or processes data on the decentralized data exchange platform is conferred a globally unique decentralized identifier (DID URI) that follows W3C DID specifications [1]. Decentralized identifiers yield verifiable credentials to actors such that they are cryptographically secure, respects privacy and are systematically verifiable, thereby building *trust* in a trustless environment. Verifiable credentials play a critical role in zero-knowledge authentication and authorization. PoRT sets up a decentralized public key infrastructure that maintains a key-value database comprising the DIDs and DID documents, which comprises cryptographic material (public keys), cryptographic protocols and set of service endpoints that describe authorization, capability delegation and modes of interactions among DID subjects.

C. Standardized Message Schema

PoRT identifies domain-specific message schema for actors from a wide variety of domains to exchange data with one another. Message standardization is paramount for effective information exchange, for invoking value added services on data streams and for building decentralized data marketplaces promoting data availability and data accessibility across domains. Let PbK_C and PvK_P be the public key of the consumer and the private key of the producer respectively. PoRT protocol encrypts payload at source and adopts digital signature scheme to guarantee data privacy, data integrity and authenticity, as illustrated here: $Sign(HashfEncrypt(data; PbK_C)g; PvK_P)$. The producer either sends the message to the entire network with ack set to 0 (receives no acknowledgement) or sends it to any one node with ack set to 1 or 2 (receive acknowledgement after the designated node or a majority of the network received the message). In the latter setting, the node that first received the message broadcasts the message to the rest of the network. Table I illustrates PoRT's standardized data schema identified.

TABLE I
Standardized Data Schema

Header	
Message ID	A globally unique identifier (GUID)
From address	DID URI of the producer
To address	DID URI of the consumer or group or self
Nonce	Number incremented for every new message
Schema	social-v-0.1 / iot-v-0.3 / other
Timestamp	Dispatch timestamp
Producer Acks	ack = 0 or 1 or 2
Chain ID	DID URI of the chain the producer is tied to
Signature	Digital signature of the message
Body	
Data	Encrypted payload

Algorithm 1 Verification Function F_v

```
1: function VERIFY_MESSAGE(msg)
2:     if not validSource(msg.from) then
         return false
3:
4:     if not validTarget(msg.to) then
         return false
5:
6:     if not authorizedTarget(msg.to) then
         return false
7:
8:     if not verifySignature(signature) then
         return false
9:
   return true
```

D. Message Verification Function

A transaction on the decentralized data platform is a fully self-contained exchange of data (messages) between parties that bear no dependencies to prior such transactions. As a result, transaction verification comprises a set of atomic operations that entail (a) signature verification (source authentication) (b) a check on data privacy and integrity (c) a check on source authorization. The verification function F_v (Algorithm

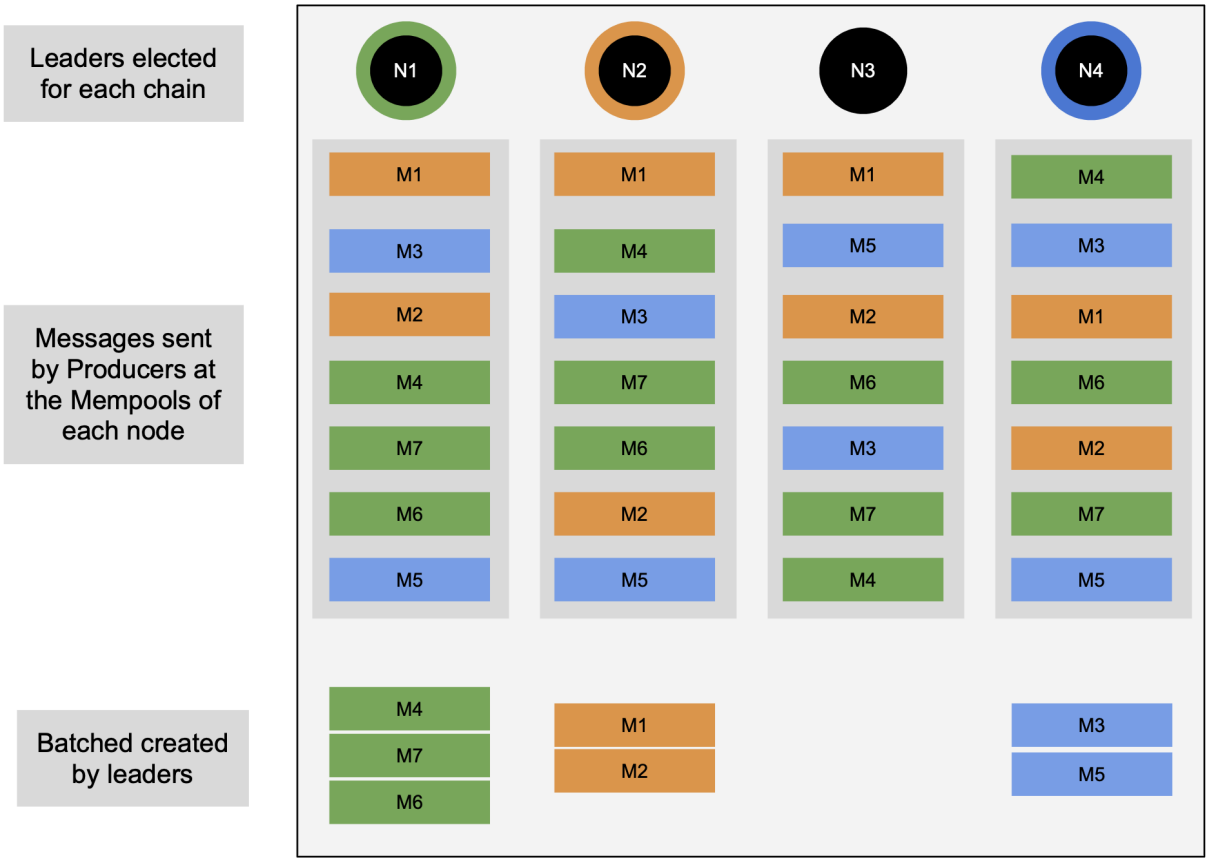

Fig. 1. Mempool data structure for four nodes are illustrated alongside chain based leader designations for each of the nodes. Messages of the same color belong the the same chain. $N1$, $N2$, $N3$ are leaders for the chain 1 (green), chain 2 (orange), chain 3 (blue) respectively

1) is computationally light and is highly parallelizable and hence paves way for massive throughput support.

E. Mempool Data Structure

Mempool is an organized queue where messages reside before being dispatched for verification, the status of which can be queried upon using public APIs. Every message that was sent to the network either directly by the producer or by the first node that received the message reaches the mempools maintained by each node. Network topology and transmission latency can result in messages making it to the respective mempools in different order. Mempools across the network taken collectively form a powerful data structure that captures incoming message rate, latency in message handling, network-wide message dispatch designations and availability status of nodes. PoRT protocol provides Network Discovery Service APIs that runs queries on mempools, that producers can avail to select message dispatch leaders and that the individual nodes can avail to check operational status of the network.

F. Messages as Entropy Source

The proposed message standardization schema implicitly introduces entropy in message creation. With fields such as *timestamp*, *nonce*, *message ID* being part of the proposed structure, messages sent to the network by different producers provide sufficiently as stores of entropy that the network can tap into to derive randomness in the system. The network dissuades duplicate messages from being sent for delivery by levying penalties on the producers for such actions. The network checks for duplicate messages being sent to the network by periodically comparing the hash of the message that was sent to the network matched that of an earlier message. If such messages do arrive, the network flags the producers and applies strict penalties.

G. Parallel Chains and Chain-based Leader Election

In the typical deterministic block finality settings, a leader proposes a block comprising verified transactions and once a majority (typically, over 50% of the validators) of the committee of validators certify the block, the block is added to blockchain and the transactions comprised are deemed finalized immediately. To prevent forking, protocols are carefully designed such that the validators will never certify two or more blocks simultaneously. This constraint results in deterministic block finality falling short of *instant* block finality. PoRT protocol achieves instant block finality, while preventing any possibility of forking through the adoption of parallel chains and chain-based leader election.

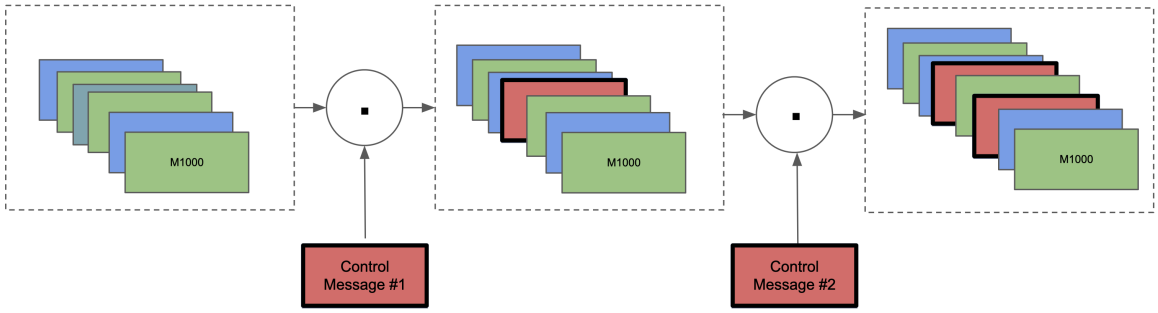

Fig. 2. Batch derivatives are created by inserting control messages (doctored messages that are designed to fail verification) on the original batch at random indices while preserving the relative order of the original messages

Parallel chains constitute a set of independent chains that are each dedicated to set of producers saving all of their verified transactions on the same chain. From the context of replicated finite state machines, parallel chains capture all of the transitions in a producer's state and accurately describe their current state in self-contained fashion. The producer - chain association is determined as follows: Let $fp_1; p_2; :::p_P g$ and $fc_1; c_2; :::c_C g$ correspond to the decentralized identifiers of P producers and C chains. Let h and $k:k_d$ correspond to sha256 hashing function and the hamming distance. The chain index k for producer p_i is determined as

$$k = \arg \min_{j\,2(1::C)} kh(p_i) \quad h(c_j)k_d \tag{1}$$

The chain index for each producer is saved in their respective DID documents. Next, to prevent forking, PoRT protocol elects one leader per chain and deems that only the elected leader is authorized to propose a certain number of blocks on the respective chain, following which a new leader is elected for the said chain. Leader election is based on a weighted combination of the node's stake and a verifiable random number [18] that the node generated.

Let $n_1; n_2; :::n_N$ be the decentralized identifiers of the N nodes of the network and let $(s_1; s_2; :::; s_N)$ correspond to their respective stakes (expressed as a proportion in the range (0,1)). h is the $sha256$ hashing function as above. Let the case be that a leader needs to be elected for chain c_j. Node n_i generates a random number $r_i = h(< (h(c_j) \quad h(n_i)); PvK_i >)$, where is a XOR operator, $< : >$ is a digital signature function and PvK_i is the private key of node n_i. Let the ordered set of bytes that make the random number r_i be r_0, r_1, $r2$, :::, r_{31}. r_i is mapped to a number in the range (0,1) as follows:

$$f_i = (r_0 \quad r_1 \quad ::: \quad r_{31})=255 \tag{2}$$

. Node n_i submits the proof $_i =< (h(c_j) \quad h(n_i)); PvK_i >$ to rest of the network. Verifying the random number r_i entails checking if $< \ _i; PbK_i >$ equals $h(c_j) \quad h(n_i)$, where PbK_i is the public key of node n_i. The weighted combination of stake and the random number is computed as $w_i = a\ s_i + b\ f_i$, where the weights $0 < a < 1$, $0 < b < 1$ and $a + b = 1$ and shared with the entire network. The index l of the leader for chain c_j is chosen as:

$$l = \arg \max_{i\,2(1::N)} a \quad s_i + b \quad f_i \tag{3}$$

The leader determines the number of blocks n_B that they are authorized to propose on the chain as $n_B = r_i \mod K$ where K is a PoRT identified threshold. The nodes with the second and third highest weighted scores are selected as back-up leaders, to step-in and serve the leader role for the given chain, if the leader were to go down.

H. Chain-based Batches

Next, the leader who was elected to propose blocks for chain c_j, gathers messages residing on their mempool and creates batches with each comprising those messages whose chain ID designation was that of the same chain. Each batch is created upon adding messages from the mempool in a first-in first-out manner, while remaining cognizant of PoRT's thresholds for batch size (in MB), for maximum number of messages per batch and maximum wait time for a message (in milliseconds). These batches will subsequently be sent to the network for verification. Should the entire network receive the same batch of messages for verification ? The next subsection details why not.

I. Batch derivatives

Here, we introduce a game theoretic construct called batch derivatives that attests for a node's pertinent work and its thoroughness in running verification function on a batch of messages. In the idealized setting, when a batch B is dispatched to the network for verification, every node shall independently run verification on the batch and shall return their verdicts to the leader. But in a real-world setting, if every node in the network had the same identical batch to run verification on, the dishonest nodes may establish out-of-band communications with fellow nodes, derive the answer key for the batch and thereby gain unfair computational advantage over honest nodes. Batch derivatives are formulated specifically to disincentivize collusion within the network. A *batch derivative* B^0 is obtained by inserting one or more non-conforming messages called the *control messages* at random

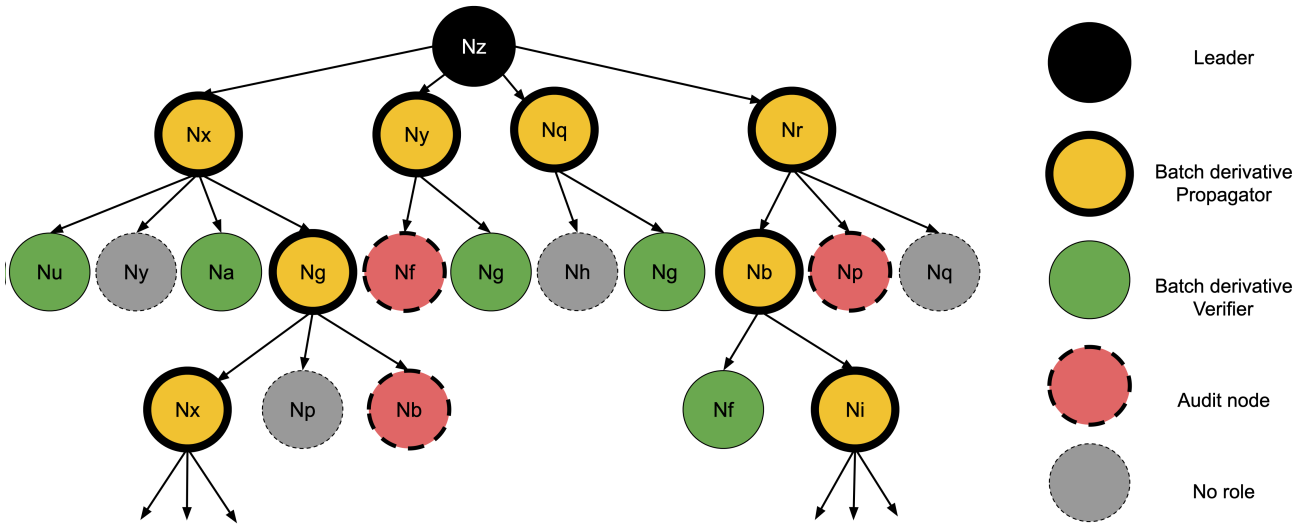

Fig. 3. An illustration of the multi-stage batch verification process

indices onto B. A control message is similar to a regular message in structure, but is designed to fail verification either due to source authentication problems or due to message integrity issues. Let $C = fc_1; c_2; :::c_y g$ be the set of y control messages. $B^0 = B \quad C$ is a batch derivative constructed upon inserting these y control messages at random locations onto B, such that the original messages in B still appear in the same relative order after the insertion of the control messages. The operator signifies control message ingestion. Figure 3 illustrates the creation of B^0, given B and C. Next subsection details how batch derivative constructs fit into the decentralized batch verification schema.

J. Dynamic Subnets for Batch Verification

PoRT deploys a multi-stage batch verification process whereby verifiably random subsets of the network are tasked to serve any one of four roles in verifying the batch depending on the stage of verification namely (a) creating and propagating batch derivatives (b) running verification on batch derivatives (c) running performance audits on a subset of the network (d) play the role of a bystander who expends no computation in verifying the batch. Through cryptographic sortition, nodes that are invoked to play a role in batch verification decipher their respective roles. Their roles are not known apriori to the rest of the network, but are verifiable once they submit their respective proofs. The above design renders adaptive attacks on the network infrastructure ineffective in slowing down batch verification and further disincentivizes collusion among fellow Byzantines seeking undue advantage over their honest peers.

Let the leader elected for chain c_j be node n_z. Let B correspond to the batch that n_z created, one comprising transactions whose chain-IDs corresponded to chain c_j. n_z generates a random number $r_z = h(< h(B); PvK_z) >)$ where h and $< : >$ are the $sha256$ hashing function and the digital signature function as before and PvK_z is the private key of node n_z. As discussed before r_z is verifiable when proof $_z =< h(B); PvK_z) >$ is submitted and is a number that can only be generated by n_z. Representing the random number r_z as 32 ordered bytes in $r_0, r_1, :::, r_{31}$, n_z determines the number of nodes to which it needs to send the batch B to as $p = \max_{i2(0;31)} r_i \mod N$ where N corresponds to the number of nodes in the network. Next p node indices are generated from r_z applying bit shift followed by a mod operation on r_z p times as $r_z >> i \mod N$ where $i\ 2\ (1; p)$. The p nodes identified by the leader form the first-stage of batch verification. Each of the p nodes, by design, shall serve the role of batch derivative creation and propagation as that results in maximum reach across the network and also separate batch verifiers from the batch leader by at least one hop thereby rendering collusion less effective.

TABLE II
SUMMARY FROM BATCH DERIVATIVE PROPAGATOR

Result Header	
Signature	Signed hash of [Header j Result Body]
Batch ID	h(B): Hash of the original batch
From address	DID URI of Node N_x
Result Body	
Role	Batch derivative creator and propagator
Proof $_x$	$< h(B); PvK_x >$
Recipient node indices	44, 23, 69, 121, 5
Hash of batch derivatives	$h(B^0_1)$, $h(B^0_2)$, $h(B^0_3)$, $h(B^0_4)$, $h(B^0_5)$

1) Batch Derivative Creation and Propagation: Let node n_x be one of the p recipients serving as a batch derivative creator and propagator in the first stage of verification. Node n_x repurposes r_x in identifying the recipients for the batch derivatives that it creates (following a process identical to that of the leader in identifying recipients). n_x generates different

batch derivatives B'_1, B'_2, :::, B'_k for the k recipients (k is determined from r_x) upon inserting control messages at random locations onto the original batch B and dispatches them to the respective recipient. Node n_x broadcasts summary of its work as illustrated in Table II.

Each of the k recipients are part of the second stage of verification. Batch derivative recipients who are part of the second stage (or any subsequent stage) identify their respective roles using cryptographic sortition. Let n_y be one such recipient. n_y generates a verifiable random number r_y upon hashing the digitally signed hash of the received batch derivative and maps r_y to $\hat{r}_y$, a float in the range (0,1), as illustrated in equation (2). Preset ranges are identified (as illustrated below) for different roles that node n_y shall serve.

$$\begin{aligned} 0 \quad \hat{r}_y < 0{:}2 \quad &=) \quad n_y \text{ propagates batch derivatives} \\ 0{:}2 \quad \hat{r}_y < 0{:}6 \quad &=) \quad n_y \text{ verifies batch derivative)} \\ 0{:}6 \quad \hat{r}_y \quad 0{:}8 \quad &=) \quad n_y \text{ serves as a bystander} \\ 0{:}8 \quad \hat{r}_y < 1 \quad &=) \quad n_y \text{ runs verification audits} \end{aligned}$$

TABLE III
SUMMARY FROM BATCH DERIVATIVE VERIFIER

Result Header	
Signature	Signed hash of [Header j Result Body]
Batch ID	Hash(B): Hash of the original message
From address	DID URI of Node N_j
Result Body	
Role	Batch Verifier
Proof $_x$	$< h(B); PvK_x >$
Assigner	DID URI of node N_i that assigned the batch derivative
Batch derivative	B' : Batch derivative that was verified
Segments reviewed	1, 3, 4
Verification result	$h(M'_1)$, $h(M'_3)$, $h(M'_4)$
Indices of failed messages	Indices where message verification failed

2) Batch derivative verification with Compute Sharding: The role of batch derivative verification is assumed by the node that generated a random number $\hat{r}_y$ in the range (0.2, 0.6). Dividing the batch derivative B' into 5 equal segments, node n_y re-purposes r_y to identify the number of sub-segments and the corresponding segment indices over which it shall run the verification function (F_b (Algorithm 2)). Each segment index 1,2,3,4,5 corresponds to that one-fifth sub-segment of the batch (or batch derivative). The node could run batch verification on more than one sub-segment. This process of randomized selection of sub-segments of the batch that will be processed introduces *compute sharding* in PoRT protocol. Compute sharding further disincentivizes collusion. Node n_y logs the indices of messages that failed verification, computes the hash of verified messages within each sub-segment (keeping the relative order of verified messages unaltered) and sends its results directly to the leader n_z as illustrated in Table III and broadcasts the same to the rest of the network.

Algorithm 2 Batch Verification Function F_b

```
1:  function VERIFY_BATCH(batch)
2:      batch_size  len(batch:messages)
3:      valid_messages  make( [ ]Message )
4:      invalid_indices  make( [ ]int )
5:      for i = 1 to batch_size do
6:          is_msg_valid
        VERIFY_MESSAGE(batch:messages[i])
7:          if is_msg_valid then
8:              valid_messages
        append batch:messages[i]
9:          else
10:             invalid_indices  append i
11:
        return hash(valid_messages); invalid_indices
```

3) Network sharding in Batch Verification: A proportion of the network, by design, is siphoned away from expending computational resources of verifying a batch derivative. These nodes generated a random number $\hat{r}_y$ that fell in the range (0.6, 0.8). These nodes submit the random number and the proof that it was generated without bias for B to the rest of the network (in fashion similar to Table **??**.

4) Behavioral audits: If the random number $\hat{r}_y$ that node n_y generated from B' fell in the range (0.8, 1), then the node shall check for role conformity. A role non-conformity event could pertain to protocol violations such as (a) a node serving a role on batch B that they were not designated to serve (b) a node using doctored random numbers and biasing the progress of batch verification. Node n_y submits a summary of its work to the rest of the network in the format illustrated in Table IV.

TABLE IV
SUMMARY FROM BATCH DERIVATIVE VERIFIER

Result Header	
Signature	Signed hash of [Header j Result Body]
Batch ID	Hash(B): Hash of the original message
From address	DID URI of Node N_j
Result Body	
Role	Audit
Proof $_x$	$< h(B); PvK_x >$
Conformant nodes	44, 21, 3, 103, 8, 9, 10
Non-conformant nodes	11, 12
Non-conformancy summary	Test logs indicating non conformance

5) Leader based Consensus: The leader n_z gathers verification results submitted and on a per batch-segment basis checks if the hash of the verified messages match the hash of the batch-segment with just the verified messages retained. N_z determines the minimum number of votes needed per batch-segment upon generating random numbers in the range (5 to 10) using the random number r_z. Once there is consensus on the list of verified messages in a batch, the leader calls off any further verification on batch B, adds a block comprising the verified messages to the chain that the block belongs to, to that chain it is serving as the leader of.

TABLE V
PoRT PROTOCOL'S DEFENSE AGAINST ATTACK VECTORS

Attack vector	PoRT Defense
Replay attack: Producer sends duplicate messages to the network, undermining message entropy and stressing the network	Nodes run hash-collision test on incoming messages by comparing their hash with that of mempool messages
Sybil attacks: A malicious node operates under multiple identities simultaneously	In a stake based network, with role fluidity, Sybil attacks are not rewarding.
Censorship attack: The leader for the chain withholds messages from certain producers, not including them in a batch	Network Discovery Service API's let any node run queries off of leader's mempool and detect instances of withheld messages.
Batch incongruence: The leader or batch derivative creators and propagators may add or remove or shuffle messages thereby affecting consensus and state machine	Audit nodes check for the hashes of batch derivatives with control messages removed (as determined by batch verifiers) and trace origins of batch incongruence if any
Freeloadership: Fellow Byzantine nodes may establish out-of-band communication and share answer keys saving on compute	Dynamic subnets introduce role unpredictability rendering such collusion stratagies less viable and less effective
Grinding attacks: Leaders could attempt adding blocks to chain that will favor them in reelection. Leaders could attempt making batches that favor fellow Byzantines	Both leader election and batch verification depend on a verifiable random number that is generated with one's own private key, making griding attacks ineffective.
Adaptive attacks: Malicious nodes may stage coordinated attacks (DDoS) selectively on certain nodes (when they serve as leader) or on certain batches to delay consensus	Dynamically evolving subnets diminish success of coordinated attacks through the lack of predictability in the roles that other nodes will serve
Non-conformity in roles: Malicious nodes may perform operations their scope	Periodic validation of node's actions outside through audits dissuade such attacks.

K. How PoRT defends against attacks

Defense against attacks is one of the primary focusses behind the design of PoRT protocol. Attacks on the platform can be staged by actors inside and outside of the network at various stages of message handling, for instance at message arrival, at batch formation and dispatch, at batch verification and at block creation. PoRT's defense to attacks stem from the adoption of decentralized public key infrastructure, from its message design which comprises payload encryption and carefully designed message headers, from mempool state replication and a dynamically evolving message verification protocol where roles served by the network are non-deterministic apriori. Crash faults are handled with backup leaders preidentified and with redundancy in roles baked into the design for nodes serving non-leader roles. In addition, PoRT makes Network Discovery Service (NDS) APIs available, which bring in transparence in message handling. The NDS APIs are available to producers and network nodes to query the status of messages, the status of mempools, the operational status of nodes, chain-based leader designations and their backups. Finally, PoRT protocol disincentivizes attacks by tying economic models for rewards for honest work whereby the leader and non-leader nodes are incentivized for speed and accuracy in data handling. On the other hand, strict penalties are applied for malicious work, ones that can diminish a node's stake in effect its returns from serving in the network. Table V details the different attack vectors and the defense provided by PoRT protocol.

IV. BENCHMARKING

This section provides benchmarks we derived on network, compute and memory utilization of the proposed consensus protocol. We performed tests on AWS instances that were spread across the United States, Europe and Asia to evaluate network throughput and compute and memory utilization on today's commodity platforms. The consensus' primary objective is to ensure high scalability using commodity hardware and lowered costs for our customers and reducing entry barriers for participating nodes.

A. Network throughput

Messages sent to the network shall adopt JSON format. Figure 4 illustrates the size of message with breakups provided for the header fields. While the header size shall remain fixed at 298 bytes (215 bytes for the header fields and 83 bytes for JSON overhead), the size of the encrypted message body can vary from one domain to another. Assuming that the message body measures to a few hundred bytes in size, the message as a whole shall measure 0.5 KB and a batch comprising of 1000 such messages shall be around 0.5 MB. Upon compressing each batch of 1000 messages using gzip (*version 353.100.22*), we identified the median size of compressed batches to be 132 KB. Transporting such batches over a dedicated 1 Gbps line sets the theoretical throughput limit at *976K messages per second*, assuming a 4% network loss. Below is a summary of our observations from the AWS experiments on network throughput.

AWS experiment #1 (US East to EU West): Between two AWS Instances (t2.micro) one in EU-West and another in US-East, on different VPCs, using concurrent streams on a single threaded single-CPU, we observed the data transfer rate to be 774 Mbps.

AWS experiment #2 (US East to US West): Between two AWS Instances (t2.micro) one in US-East and another in US-West, under the exact same setting as the above experiment, we observed the data transfer rate to be 843 Mbps.

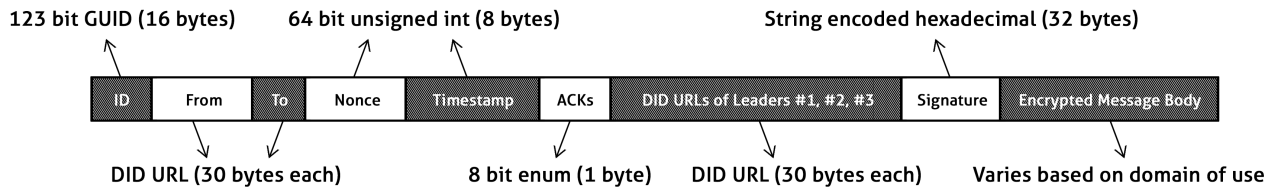

Fig. 4. Size of header fields in a typical message

On an average we observed network throughput to be 0.8 Gbps on the AWS Network, which brought our theoretical throughput limit to 794K messages per sec. on the above network. However, upon leveraging parallel streaming frameworks such as gRPC (HTTP 2.0), we can achieve more economies of scale on network throughput. We have reserved this exercise for the future.

B. Compute and Memory Performance

Our consensus protocol comprises highly parallelizable atomic operations that can remain agnostic of any external I/O dependency. Consequently, verifying a batch of messages can be subdivided into atomic tasks such as batch derivative generation, batch derivative verification and batch derivative propagation using VRF and consensus gathering. Selecting a set of AWS instances the configurations of which measure up to commodity hardware, we ran batch verification tests on each. The Table VI illustrates the configurations of different AWS machines that were tested and the corresponding batch verification throughput on each. Memory consumption was a constant at approximately 2 MB per 1000 transactions

Figure 5 illustrates compute utilization and message throughput of BD verification on AWS EC2 instances with 1 CPU core, 4 CPU cores, 8 CPU cores and 32 CPU cores. In the near future, we will perform a similar analysis leveraging commodity GPUs with thousands of low powered processing cores, which will result in highly improved compute and message throughputs.

TABLE VI
A BENCHMARK ON COMPUTE THROUGPUT

Result Header	Compute Type	Throughput (TPS)
1 (vCPUs)	Virtual cores (AWS)	12,000
4 (vCPUs)	Virtual cores (AWS)	31,000
8 (vCPUs)	Virtual cores (AWS)	98,000
16 (vCPUs)	Virtual cores (AWS)	117,000
32 (vCPUs)	Virtual cores (AWS)	152,000
16 (32 threads)	Physical Cores (baremetal)	213,000
32 (64 threads)	Physical Cores (baremetal)	570,000

The experiments above indicated that memory (RAM) footprint for BD verification was nearly constant at 2 MB per batch (1000 messages). In the event of nodes having to process 1 million messages concurrently, the expected memory footprint for BD verification is well within today's commodity specifications.

V. DISCUSSIONS AND CONCLUSIONS

Proof of Real-time Transfer protocol is designed to power decentralized platforms that serve content creation, social applications, enterprise communications, gaming and metaverse applications. The protocol functions in a gasless setting where no transaction takes higher precedence over another. In content agnostic fashion, it simply provides guarantees towards the delivery of source authenticated, secure, integrity-preserved messages with reasonable message-order guarantees. With deterministic verification, dynamic subnets and compute sharding, PoRT protocol imposes implicit bounds on compute and network resource usage for message verification. Further, by adopting parallel chains, the protocol features instant finality of verified messages. Hence, decentralized platforms that are powered by PoRT protocol can serve applications that demand extremely high throughput and low latency.

Future revisions to PoRT protocol are envisaged along the following lines. PoRT message verification protocol embracing verifiable computations with zero-knowledge proofs [19] generated in trustless settings will further augment scalability of the protocol without compromising security. Cryptographic solutions that enable keeping the number of blocks a leader for a chain is authorized to propose a secret until [20] after the leader has fulfilled his role would further enhance security. Adopting application-specific temporal *rolling* blockchains [21] whereby the genesis block is updated periodically will be critical to better manage storage requirements in the decentralized platform. Token economics and the ensuing incentivization for participating nodes play a vital role in the sustenance of decentralized networks. Future revisions will encompass our findings with regards to optimal designs for the same.

REFERENCES

[1] M. Sporny, D. Longley, M. Sabadello, O. Steele, and C. Allen, "Decentralized identifiers DID v1.0. 2021", https://www.w3.org/TR/2021/PR-did-core-20210803.
[2] Thomas Pornin, "Deterministic usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC, 6979:1–79, 2013.
[3] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", Dec 2008. https: //bitcoin.org/bitcoin.pdf.
[4] Vitalik Buterin. "Ethereum: A next-generation smart contract and decentralized application platform", 2014. https://github.com/ethereum/wiki/wiki/ White- Paper.
[5] Charles Hoskinson,"Why we are building Cardano", 2017. https://whitepaper.io/ document/581/cardano- whitepaper.
[6] Gavin Wood, "Polkadot: Vision for a heterogeneous multi-chain framework", 2020. https: //polkadot.network/PolkaDotPaper.pdf.