Watch Flow reach its 1 billionth transaction in real-time here.

Technical Vision

Core Protocol Vision

Data Availability Vision

Share this page

Table of contents

Motivations + principles

Motivations for Flow

Sharing those benefits with the masses

Pipelining as Key to Scalability

Scaling Transaction Linearization via Hierarchical Consensus

Parallel execution for horizontal scaling

Distributed Verification

Summary and Conclusions

Motivations + principles

Global distribution +

Byzantine Fault Tolerant +

Efficiently-scalable platform for general-purpose computation +

CRYP2NOVA

Key concepts and terminology for a globally-distributed, fault-tolerant, scalable operating system

Nodes

Nodes are the building blocks that collectively form our globally-distributed, BFT, open operating system. Informally, a node can be described as an individual actor. It may be a single server or a group of many microservices depending on the node's responsibilities. The important aspect is that a node is a single entity that either plays by the rules or is considered malicious (formally 'byzantine'). Nodes are run by 'node operators',

Archived copy · cryp2nova.com

typically companies, non-profit organizations, governments, or individuals with sufficient domain expertise.

Protocol

The **protocol** specifies how nodes are supposed to interact. An 'honest' node *always* follows the rules, which includes being available to accept work.

Byzantine node

A malicious or **byzantine node** can deviate from the protocol in any way it desires or be offline [1]. A byzantine node may behave honestly for a very long time and only reveal its malicious intent at some point later (or never) potentially in a coordinated attack together with other byzantine nodes. A system is **Byzantine Fault Tolerant**, if it continues to produce correct outputs even if byzantine nodes are participating.

CRYP2NOVA

Motivations for Flow

Archived copy · cryp2nova.com

Flow should scale to global adoption by 1 billion users and be capable of running frequently-interacting, potentially safety-critical applications with moderate computational and data storage requirements. Algorithmically, Flow belongs to the family of blockchain protocols. Though, we feel there is much value in putting a strong focus on our final goal: engineering a globally-distributed, efficiently scalable, BFT operating system. Throughout Flow's design and architecture, we consequently applied the following guiding principles that are all motivated by this end goal.

Usability: absorbing complexity into the platform rather than outsourcing it to developers

From our perspective, the following best practices broadly strengthen usability, which Flow implements today except for the last item, which is on our roadmap.

- We believe that consistency brings a huge benefit for a software developer. Working with eventually-consistent databases or implementing concurrent algorithms substantially increases software complexity, engineering time and maintenance cost [3]. In technical terms, we desire ACID-compliant transactions.
- An intuitive concept of a user account is very important, where an account can have multiple access keys with potentially different permissions, keys can be cycled, etc.
- We desire a programming language, where objects have a native notion of access permissions. Thereby, we can represent concepts like "You give me an object, where you as the creator determine what I can do with the object (including whether or not I can copy it)." Cadence is Flow's programming language with native language features to express ownership and access permissions.
- Lastly, having a (cryptographically secure) pseudo random number generator [PRNG] natively within the system is immensely useful. Therefore, Flow implements a DFinity-style random beacon [4] (including a distributed key generation). Among other low-level protocol functions, the Flow Virtual Machine utilizes the random beacon to seed PRNGs for smart contracts. The resulting random numbers are unpredictable, verifiable and tamper-proof – all in the presence of a colluding group of adversarial nodes within the network [5].

CRYP2NOVA

Archived copy · cryp2nova.com

By implementing broadly-used functionality on a platform level, such random number generation, we reduce development overhead. Developers can focus on their core goals, instead having to work around shortcomings of the platform.

Efficiency as a fundamental requirement

In 2020, researchers estimated that information and communications technology causes approximately 2% - 4% of global greenhouse gas emissions [6]. In addition, there is a substantial ecological impact from extracting and refining the required resources, such as cobalt, copper, lithium and rare-earth elements. Therefore, efficient resource consumption is a central architectural metric for Flow.

In 2021 Flow consumed a total of 0.18 GWh in energy [7], which equals the power consumption of about 30 people [8]. Already today, Flow is among the most efficient BFT compute platforms in existence [9]. As Flow matures and transaction load increases, we expect Flow's scaling efficiency to generate a more pronounced advantage.

Thesis I: mass adoption requires high throughput and large state space. +

Thesis II: high value of software composability – from open source to open execution +

Thesis III: Scalability is the long-term challenge to usability and efficiency. +

Sharing those benefits with the masses

The following points are the summary of Flow's architectural goals as motivated in the previous sections:

- Throughput of at least 1 Million transactions per second (TPS);

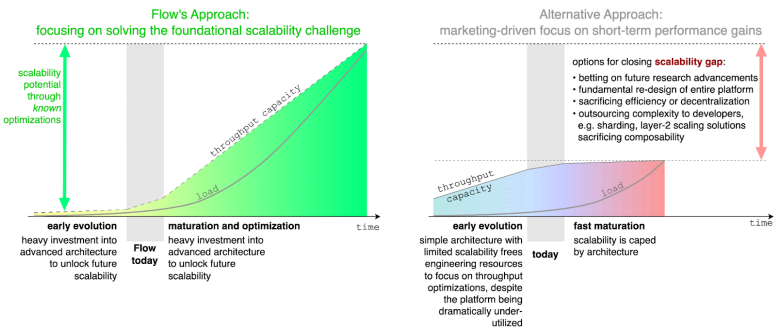
Archived copy · cryp2nova.com

- Ingest at least ½ gigabyte of transactions and data per second (realistically 1 GB and upwards);
- Very large state of one Petabyte (size of state snapshot at a single block) and beyond;
- Supporting large-scale software composability, i.e. running frequently-interacting, turing-complete applications
- Absorb complexity into the platform by solving common challenges on a platform-level once, so all developers benefit. This frees the developer to focus on higher-level work, catalyzes creativity, and reduces cost, which in turn benefits users and fosters innovation.

Flow is solving the scalability challenge now

From a platform-developer's perspective, it is certainly convenient to delegate scaling challenges into the future. However, scalability is the most fundamental and persistent challenge for all distributed, BFT operating systems, including Flow and all other smart-contract-focused blockchain platforms. Therefore, we strongly believe that high-scalability requirements should be de-risked very early in the development process, being considered at every step in the architecture. Otherwise, platform developers outsource the risks of limited scalability to the developers and users. We encourage developers to ask what changes will be necessary on their part should the blockchain platform they are considering achieve mainstream adoption and run into scaling limitations (the overwhelmingly likely scenario).

CRYP2NOVA



For Flow, we are solving the scaling challenge today by building an operating system that transparently scales to high computational loads. We invest significant engineering resources into a more complex but highly scalable architecture, postponing well-understood throughput and latency optimizations into the future. Even at today's capacity levels and housing some major mainstream

Archived copy · cryp2nova.com

experiences such as NBA TopShot, Flow can accommodate a load increase by a factor of 10×. In comparison, many other platforms narrowly focus on throughput optimizations for marketing purposes despite their platforms being under-utilized by orders of magnitude. We believe that continued scalability will be the largest win for developers in the long run as opposed to investing into short-term gains by special-purpose optimizations.

Applications for mainstream-audiences such as Youtube, NBA, Ticketmaster, American Airlines, Doodle 2, LaLiga and many more chose Flow specifically for its long-term scalability. While established brands tend to require scalability early on, grass-root developers and indie projects are impacted by scaling limitations on longer time scales. Nevertheless, as the cost of platform shift is high, selecting a computational platform with long-term scalability is an important strategic decision also for small projects at early stages.

Professionally-operated nodes

Topics of frequent controversy are the hardware and operational requirements for nodes. For resilience and decentralization, we generally desire a large number of redundant independently-operated nodes. Therefore, decentralization maximalists demand that hobbyists with common commodity hardware should be able to operate nodes. However, from a scientific perspective, there is no substantial evidence that arbitrary computation can be efficiently distributed among many small hardware devices, even if we assume correct working of all hardware. In fact, the trend in data centers is towards increasing concentration of compute power, where CPUs deliver increasing throughput per core, despite hardware cost increase and growing challenges with cooling [15]. The reason is that vertical scaling (using the same number of progressively stronger computers) is generally substantially more efficient with respect to energy and hardware consumption as well as software engineering complexity than horizontal scaling (progressively increasing the number of computers, without increasing the individual machine's computer power).

In summary, we believe that for a highly-scalable distributed operating system, professionally-operated nodes are essential for efficiency. A network with nodes operated by a diverse mix of companies, non-profit organizations, governments, and individuals with domain expertise is sufficiently decentralized. In fact, we expect a network's resilience against malware attacks or acts of cyberwarfare to be higher when it is composed of professionally-operated nodes compared to hobbyists' nodes.

Archived copy · cryp2nova.com

Pipelining as Key to Scalability

We discussed the benefits and potential large-scale applications of a globally-distributed, byzantine fault-tolerant operating system from the perspective of IT security and infrastructure resilience. We introduced Flow as an archetype of such an operating system and derived the following ground-breaking scalability goals:

- Throughput of at least 1 Million transactions per second (TPS). We anticipate that this transaction volume will require the platform to ingest at least ½ gigabyte of data per second; more realistically 1 GB and upwards.
- Very large state of one Petabyte (size of state snapshot at a single block) and beyond;
- Support for large-scale software composability, i.e. running frequently-interacting, turing-complete applications

Now, we analyze the tradeoff between redundancy, resilience and scalability. We will conclude that the cost and benefits of high redundancy vary drastically throughout the lifecycle of transactions. Thus, Flow implements a modular pipeline for processing the transactions, which allows to tune the level of redundancy for each step independently.

We emphasize that scalability is one of the most fundamental and persistent challenges for distributed, BFT operating systems, including Flow and all other smart-contract-focused blockchain platforms. Therefore, we strongly believe that high-scalability requirements must be de-risked very early in the development process, being considered at every step in the architecture. Otherwise, one would outsource the risks of limited platform scalability to developers and users.

CRYP2NOVA

The importance of redundancy for resilience and its cost to scalability +

Transaction linearization and transaction execution have different tradeoffs between efficiency and redundancy +

Separating Consensus from Transaction Execution +

Archived copy · cryp2nova.com

Scaling Transaction Linearization via Hierarchical Consensus

In the previous section, we have seen that pipelining is key for platform efficiency. This chapter focuses on the first pipeline step: transaction linearization. We explain how Flow's consensus system scales beyond the networking capacity of individual servers, which is unmatched in the blockchain space to the best of our knowledge.

Consistently ingesting at least half a gigabyte of transaction data per second on a single node is quite challenging and requires very powerful hardware. We don't want to rely on a "few strong nodes" for transaction linearization, because this would compromise BFT and decentralization as explained in the last section. Conceptually, the answer to this dilemma is intuitive: individual nodes only ingest a subset of all incoming transactions.

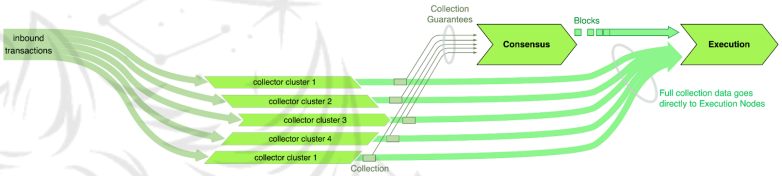


Figure 3: **horizontally scaling transaction linearization**. Collector clusters ingest a portion of the inbound transactions and batch them into collections. Collector nodes within the cluster sign the collection hash to signal their commitment to temporarily storing the full collection until the transactions are processed. Only the signed hash ('collection guarantee') goes to the consensus nodes, while the full collection data is sent directly to the execution nodes. Therefore, the few execution nodes with their strong hardware are the only nodes that have to ingest the full transaction data.

Flow realizes horizontally-scaling transaction ingestion through a hierarchical process illustrated in Figure 3. We briefly summarize the process here and refer interested readers to our paper [20] for details. In a nutshell, there is one class of nodes called 'collectors', whose job is to create batches of linearized transactions. The large set of collectors (potentially a few thousand in the mature, maximally scaled protocol) are partitioned into smaller groups (50 - 100 nodes), called clusters, on a weekly basis. Each transaction goes to one particular cluster depending on its hash. If we have k clusters in the protocol, each cluster only needs to ingest $1/k$ of the global transaction data. A single batch, called collection, of linearized transactions might contain as many as 1000 transactions. The full collection goes directly to the Execution Nodes once enough collectors from the respective cluster have signed it and thereby

Archived copy · cryp2nova.com

promised to store the collection. However, only a reference to the collection, called collection guarantee, is sent to the consensus committee. The consensus nodes linearize the collection guarantees and put them into blocks. As an individual collection guarantee is only about 128 bytes, consensus participants can ingest thousands of them per second with light hardware requirements. The resulting processing pipeline takes the form depicted in Figure 4.

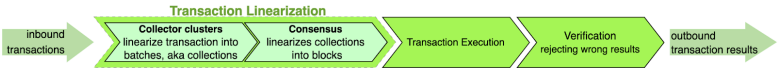


Figure 4: **Flow's transaction processing pipeline** (simplified) with hierarchical transaction linearization.

Early 2020, we published a paper describing the approach of hierarchical, horizontally scalable transaction linearization [20], as we were finalizing the corresponding collector implementation in Flow. Since mid 2021, Ethereum is thinking in a conceptually very similar direction, though without tangible release dates, using the keywords 'data availability shards' [21, 22] or more recently 'danksharding' [23, 24].

Parallel execution for horizontal scaling

After describing how to scale linearization to 1 Million transactions per second (previous chapter), we now shift our focus to transaction execution. This chapter explains how Flow achieves massive efficiency and scalability benefits by concentrating execution within a limited number of dedicated strong nodes.

We believe that an operating system tailored towards high software composability provides tremendous benefits to developers and catalyzes creativity, which in turn benefits users and fosters platform adoption. Developers heavily utilizing composability require the computing platform to efficiently run a network of frequently-interacting, turing-complete applications. With computational load approaching the anticipated levels (petabyte state and gigabyte(s) per second inbound transactions), no single server would be capable of storing the state entirely or computing all transactions at acceptable speed. Hence, transaction execution needs to scale horizontally, i.e. applications must be distributed across servers under the hood.

Archived copy · cryp2nova.com

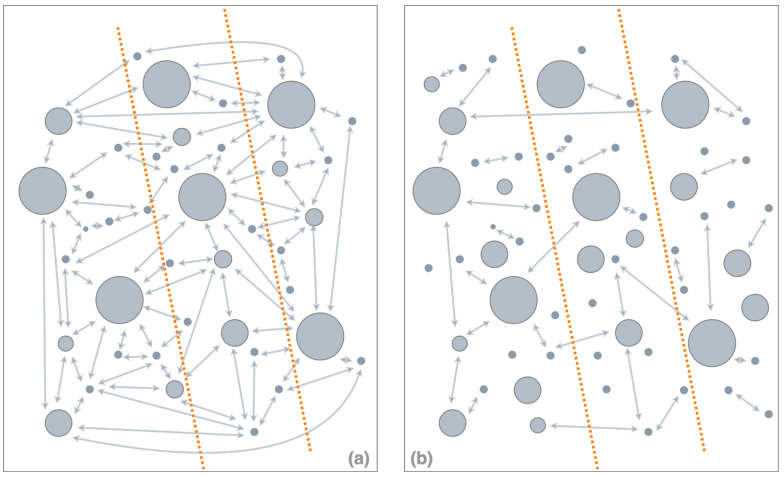


Figure 5: **(a) a network of frequently-interacting applications** and **(b) sparsely-interacting applications**. Circles depict applications and arrows their interactions. The orange lines illustrate a distribution of the applications across 3 servers. An arrow crossing an orange line implies that two different servers have to exchange data due to applications interacting with each other. Applications heavily utilizing composability (figure a) result in significant communication volume. In contrast, distributing rarely-interacting applications (figure b) is easier, even if inter-server communication is resource intensive, because the costly operations are relatively rare.

Applications heavily utilizing composability (figure 5a) result in significant communication volume. In this scenario, data exchange between the servers needs to be highly efficient. Otherwise, the computational platform cannot be scaled without resource consumption becoming impractical.

In comparison, rarely interacting applications (figure 5b) can be distributed across multiple servers even if data exchange between servers is resource intensive. Limiting a computational platform to sparsely-interacting applications is a convenient shortcut for the platform developer. However, thereby they outsource significant complexity to developers, because many developers have to now work around shortcomings of the platform. For a platform supporting only sparsely-interacting applications, scalability challenges have to be addressed over and over by many developer teams separately, instead of solving them once on the platform level.

The Flow team is a strong believer of absorbing complexity into the platform instead of outsourcing it to developers. We architected Flow to support large-scale composability and describe in the following how Flow efficiently scales transaction execution horizontally.

Flow employs a very limited number of dedicated execution nodes (8 to 10 max in the mature protocol). We envision that each execution node would be a small data center. Execution nodes are

Archived copy · cryp2nova.com

fully redundant to each other, meaning every execution node *holds the entire state* locally and *executes all transactions*. Our approach has the following ground-breaking efficiency benefits (also illustrated in Figure 6):

- Due to **high data locality**, i.e. short communication distances, limited hardware and energy resources are needed for data exchange at scale. Moreover, within the data center, specialized network infrastructure can be used that would be cost-prohibitive over longer distances, thereby further minimizing resource consumption.
- **Spatially dense computation** implies low-latency communication. Thereby, Flow can run heavily-interacting applications. The same computation would bottleneck on latency if executed on a widely-distributed network where servers communicate over the internet.
- As an execution node is operated by a single operator, the **servers within the execution node trust each other**. Thereby, the servers can retrieve state and exchange application data without relying on CPU- and energy-intensive signatures and cryptographic correctness proofs.
- The Flow protocol is oblivious to the specifics of how execution nodes *internally* distribute the workload across their numerous servers. The only requirement is that the execution result must be identical to sequential execution of the linearized transactions. Hence, **execution nodes may internally re-order transactions and execute them in parallel, as long as the result remains invariant**. This is a game-changing advantage of Flow over most other protocols for the following reasons:

  - Different execution nodes can employ different heuristics for parallelizing the transactions. Most likely, there is no single algorithm for parallelizing transactions that is optimal in all scenarios. With a diverse mix of heuristics, different execution nodes will be temporarily faster than others depending on the currently prevalent load patterns. The speed of the overall platform is determined by the fastest execution node(s). Hence, diverse execution heuristics make the Flow protocol faster.

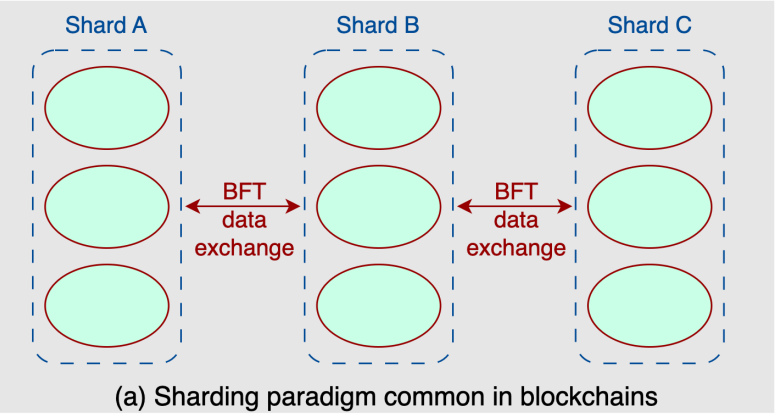
  - Execution nodes can locally evolve their heuristic without needing to coordinate time-intensive protocol upgrades. Execution nodes are rewarded for speed (a topic, which we will cover in a future article) and they can treat their heuristic for parallelizing execution as intellectual property. Therefore, we expect continuous, fast-paced optimizations of speed and efficiency.

  - Horizontal scaling of transaction execution being an

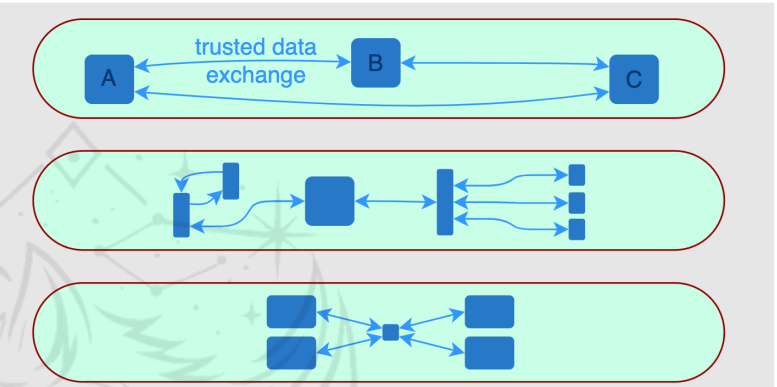
CRYP2NOVA

Archived copy · cryp2nova.com

implementation detail of the execution node also dramatically simplifies the Flow protocol.



(a) Sharding paradigm common in blockchains



(b) Horizontal scaling *within* the Execution Node in Flow

Figure 6 (a) depicts the sharding paradigm common in blockchains. Each node (green oval) maintains a local copy of a single shard or a small subset of shards. For an amalgamation of densely-interacting software (as illustrated in Fig. 5a), the various applications calling each other run frequently on different nodes, which don't share a trust relationship. Hence, most function calls require a BFT data exchange between nodes, which imposes a substantial overhead, such as: additional correctness proofs for the payload data needs to be transmitted; long-distance communication over the internet entails latency; often inter-shard communication needs to be mediated by consensus, resulting in even more latency and severe throughput limitations.

Figure (b) illustrates Flow's horizontal scaling approach within the execution nodes (green ovals). Execution nodes have different heuristics for parallelizing the computation internally. For instance, the top execution node internally partitions the computation in a similar way to the shards from figure (a). However, as the data exchange is entirely internal to the node, there is no overhead. Moreover, different execution nodes implement different heuristics for horizontally scaling their computation. A diverse mix of scaling heuristics increases Flow's resilience and alleviates load-specific bottlenecks of individual transaction parallelization heuristics.

CRYP2NOVA

Archived copy · cryp2nova.com

In conclusion, there are massive efficiency and scalability benefits from concentrating transaction execution within a limited number of dedicated strong nodes. We highlight that Flow achieves this without compromising security or decentralization. This is because for a given starting state and a sequence of computational instructions, there is a *single objectively-correct result*. The verification step described in the following chapter guarantees that only correct results are accepted. Therefore, correctness of the system outputs is guaranteed despite low redundancy.

Distributed Verification

We have argued that for a highly-efficient, scalable compute platform, redundancy must be carefully controlled to keep resource consumption practical. However, with low redundancy, the majority (or all) execution nodes being compromised becomes a plausible scenario. To guarantee that only correct transaction execution results are accepted, we utilize redundancy. Essentially, we are asking a group of nodes, called 'Verification Nodes' or 'Verifiers' in short, to re-run the computation. If a larger fraction of Verifiers agree with the result, Flow will consider the result as correct and otherwise reject it as faulty.

Observant readers will notice that we are reducing redundancy of execution nodes to be more efficient but then adding redundancy at the verification step to prevent faulty results to slip though. Nevertheless, by substantially cutting execution redundancy we can dramatically reduce energy and hardware consumption. But we only need to add a little bit of redundancy, with limited hardware and energy cost, at the verification state. With such an architecture, the entire platform is still substantially more efficient than having high redundancy at the execution state.

To illustrate the underlying idea, consider a hypothetical scenario where evil Eve and byzantine Bob are contractors installing a fire alarm and trying to get away with a bad, incorrect job. The established process is that at the end Eve and Bob will have to show their work to a licensed inspector, who has to sign off. In simplified terms, we have a redundancy of 3, because Eve, Bob and the inspector are all spending time. Eve and Bob are trying to bribe the inspector to sign off on their faulty work. Depending on the inspector, some might be vulnerable, accept the bribe and become byzantine, while other inspectors withstand the compromising offer. Assume we have 1000 licensed inspectors working in the city, ⅓ of which are vulnerable to be compromised. For one inspector checking the work, Eve and Bob have a statistical change of 33.3% to get away with their faulty result. To make it more challenging, we can assign $\alpha = 5$ inspectors and require Eve and Bob to present

Archived copy · cryp2nova.com

signatures from at least ⅔ of them. In this case, they have to get lucky enough so that there are 3 vulnerable inspectors, which reduces the chance of accepting faulty work to less than 5%. Assigning 50 inspectors already decreases the probability to less than 1 in a million!

The crucial insight is that we don't need all 1000 inspectors to check the work. We can still build a *safe and efficient* protocol, by randomly assigning a limited number of inspectors from a large cohort.

Conceptually, we apply the same approach to reject faulty execution result in Flow:

- Each execution node publishes their own result, essentially pledging a large safety deposit (in blockchain terms 'stake') to its result being correct.
- Consensus nodes record the unverified results in blocks and randomly assign α Verification nodes to check the result. Through a sophisticated cryptographic process (DFinity-style random beacon [4]), the verifier assignment is unknowable at the time the Execution Node publishes its result.
- ⅔ of the assigned Verifiers need to approve the execution result. If the verifiers find a fault in the result, the execution node's stake is taken (formally 'slashed') as penalty; equally so, if the execution node doesn't reveal computation details to the verification nodes enabling them to check.

Figure 7 shows the probability of accepting a wrong execution result depending on the number of assigned verifiers α. With 10 execution nodes and assigning α=100 verifiers, we have a redundancy of 110, which is comparatively small. Other blockchain systems have execution redundancy of several hundreds, even approaching 2000 [25]. In comparison, Flow's redundancy for transaction execution is about *one order of magnitude less*. We emphasize that Flow limits redundancy only for execution, which does not undermine decentralization and resilience, as execution nodes solely compute the transactions in the order determined by the decentralized, highly-resilient parts of the network.

The only possible attack vector for Execution Nodes is to cease operations completely. Nonetheless, a single honest execution node is sufficient to keep Flow running.

CRYP2NOVA

Archived copy · cryp2nova.com

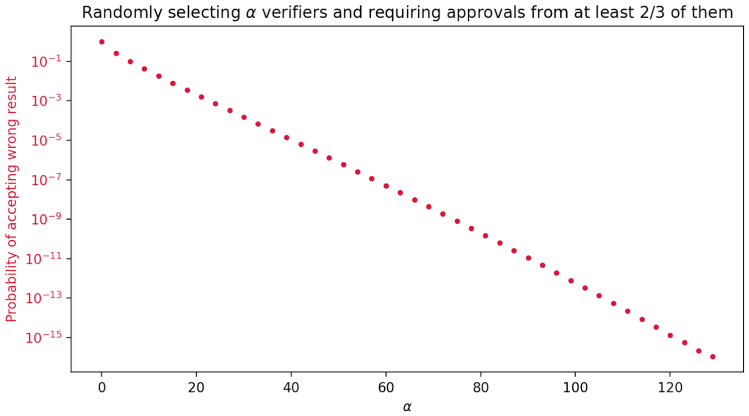


Figure 7: Probability of accepting a wrong result depending on the number of assigned verifiers α. In the mature protocol, we would randomly assign α from a cohort of approximately 1000 verifiers. The probability decreases to about 10^{-12} for α=100. It can be further decreased though increasing the total cohort of verifiers.

The description of Flow's verification stage in this article is strongly condensed and simplified. For example, in the full protocol, we break the execution results into many small chunks. Each verification node only processes about 5 - 8% of all the chunks comprising a block. Thereby, the heavy work of checking a block is distributed and parallelized across the entire cohort of verification nodes.

For further details, we refer readers to our paper [26] on execution and verification. While the approach described in the paper still applies, some of the subtle technical details have been revised and refined. For the interested reader, we will publish an updated paper in the future.

Summary and Conclusions

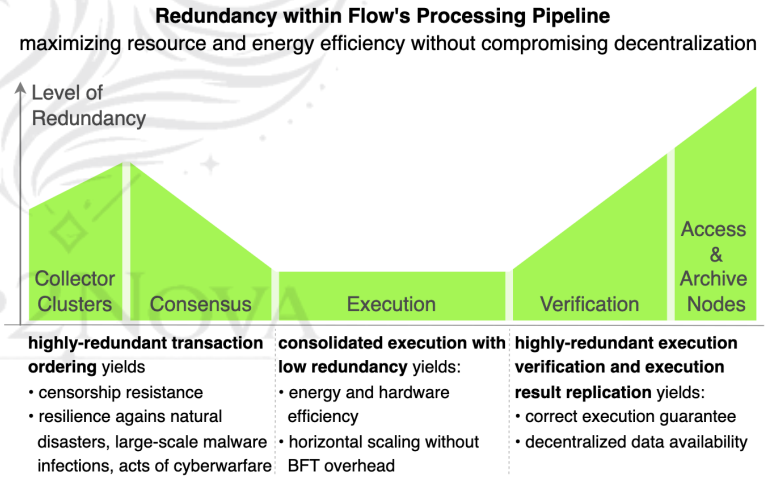
Redundancy is a central parameter in a distributed compute platform, such as Flow. While high redundancy usually benefits resilience and decentralization, it also entails large hardware and energy consumption. For example, replicating 1 petabyte of state across hundreds to thousands of nodes is neither economically nor ecologically justifiable.

In our view, *efficient scalability* and support for highly-interacting, *composable applications* is quintessential for any distributed operating system aiming for mass-adoption and broad utility. Efficient scalability is an ongoing challenge and an area of active research and development. It is well-established that any general-purpose compute platform, from multi-core CPUs to cloud infrastructure, cannot be scaled without sacrificing efficiency or software composability, unless scalability is designed from the start

Archived copy · cryp2nova.com

into the platform. This holds for conventional compute platforms, which can tolerate crash failures. Extending the requirements to also tolerate byzantine failures makes scalability even harder. Therefore, I strongly disagree with the prevailing approach in the blockchain space to design systems with severely limited scalability and betting on future extensions (sharding, rollups, zero-knowledge proofs) to solve it.

For Flow, scalability is an intrinsic part of the architecture, considered at every step of development. On a technical level, we achieve unprecedented scalability by carefully controlling redundancy. Flow's architecture separates processing steps across specialized node roles to individually optimize redundancy. Thereby, we achieve maximal efficiency without sacrificing decentralization or resilience. In particular transaction execution, the dominant driver of hardware and energy consumption, has exceptionally low redundancy in Flow. For example, in 2021 Flow consumed a total of 0.18 GWh in energy [7], which equals the power consumption of about 30 people [8]. Already today, Flow is among the most efficient BFT compute platforms in existence [9]. We stress that transaction throughput for Flow and other more-recent blockchain platforms is still orders of magnitude below their mature levels. As computational load increases, energy consumption will generally increase but we expect Flow to scale much more efficiently than other platforms due to its unique architecture.



As a closing remark, we want to briefly discuss complexity. The Flow protocol might appear complex with its 5 different node roles compared to other popular blockchains. However, we emphasize that the Flow protocol already contains the entire complexity for scaling Flow to global adoption, while other projects delegate the scaling complexity into the future. For example, we concur that Ethereum 1.0 is substantially simpler than the Flow, but it does not scale. To tackle their scaling challenge, Ethereum 2.0 is by orders of magnitude more complex.

We conjecture that in a future comparison of Flow vs other protocols that have solved the scaling challenge (without compromising composability), Flow will be among the simplest and

Archived copy · cryp2nova.com

most intuitive protocols. This is because concentrating execution in dedicated, high-powered execution nodes is an innovation unique to Flow. Other protocols (including Ethereum 2.0, Near, etc) attempt to horizontally scale transaction execution across different nodes without trust relationships. Therefore, they must specify on a protocol level the detailed mechanics to horizontally scale transaction execution.

by Alexander Hentschel, PhD, Chief Protocol Architect of the Flow Blockchain, 2023

Stay up to date with the latest news on Flow.

Stay up to date with the latest news on Flow.

Enter your email address

Submit

© 2026 All Rights Reserved

Archived copy · cryp2nova.com

flow

Archived copy · cryp2nova.com