Differentiated consistency for worldwide gossips

Davide Frey, Achour Mostefaoui, Matthieu Perrin, Pierre-Louis Roman, Francois Taiani

Differentiated consistency for worldwide gossips

Davide Frey, Achour Mostefaoui, Matthieu Perrin, Pierre-Louis Roman, and François Taïani

Abstract—Eventual consistency is a consistency model that favors liveness over safety. It is often used in large-scale distributed systems where models ensuring a stronger safety incur performance that are too low to be deemed practical. Eventual consistency tends to be uniformly applied within a system, but we argue a demand exists for differentiated eventual consistency, e.g. in blockchain systems. We propose UPS to address this demand. UPS is a novel consistency mechanism that works in pair with our novel two-phase epidemic broadcast protocol GPS to offer differentiated eventual consistency and delivery speed. We propose two complementary analyses of the broadcast protocol: a continuous analysis and a discrete analysis based on compartmental models used in epidemiology. Additionally, we propose the formal definition of a scalable consistency metric to measure the consistency trade-off at runtime. We evaluate UPS in two simulated worldwide settings: a one-million-node network and a network emulating that of the Ethereum blockchain. In both settings, UPS reduces inconsistencies experienced by a majority of the nodes and reduces the average message latency for the remaining nodes.

Index Terms—Distributed system, Epidemic protocol, Eventual consistency, Blockchain

1 Introduction

Today's distributed computer systems have reached unprecedented sizes. Modern data centers routinely comprise tens of thousands of machines [2], [3]; social networks and IoT services combine cloud and edge resources into world-spanning applications [4], [5]; blockchains [6]–[8] execute on tens of thousands of machines organized in peer-to-peer networks [9], [10]. Such enormous scales come with a host of challenges that distributed-system researchers have focused on over the last decades. One key challenge arises from the inherent tension between performance, consistency, and fault tolerance, as proven by the CAP impossibility theorem [11].

To overcome this fundamental barrier, large-scale distributed systems often adopt an eventual consistency model [5], [12] for their data [13], [14]. Intuitively, eventual consistency allows the replicas of a distributed shared object to temporarily diverge, as long as they eventually converge back to a unique global state. Formally, this global consistent state should be reached once updates on the object stop (with additional constraints usually linking the object's final value to its sequential specification) [15]. In a practical system, a consistent state should be reached every time updates stop for *long enough* [5]. How long is long enough depends on the properties of the underlying communication service, notably on its *latency* and *ordering guarantees*. These two fundamental properties stand in a natural trade-off, where latency can be traded for better (probabilistic) ordering properties [16]–[18]. This inherent tension builds a picture in which an eventually consistent object must find a compromise between speed (how fast changes are visible to other nodes) and consistency (to which extent different nodes agree on the system's state).

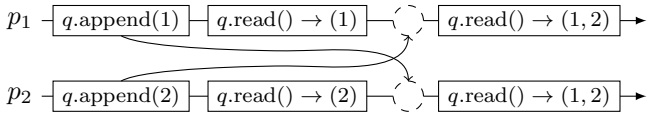


Fig. 1: An eventually consistent append-only queue.

This tension is exemplified in Fig. 1 that shows processes p_1 and p_2 using a distributed append-only queue q, i.e. an abstract representation of a blockchain [19, §16.7]. A queue q supports two operations: $\mathrm{append}(x)$ adds the integer x to the queue, and $\mathrm{read}()$ returns the content of q. In Fig. 1, p_1 and p_2 eventually converge to the same consistent global state $(1, 2)$ that includes all modifications: $q.\mathrm{append}(1)$ by p_1 and $q.\mathrm{append}(2)$ by p_2, in this order. However, p_2 experiences a temporary inconsistent state when it reads (2): this read "misses" p_1's append operation which has been ordered before that of p_2 and is thus inconsistent with the final state $(1, 2)$. Process p_2 could delay its first read operation to increase its chances of receiving p_1's append operation on time (dashed circle), ultimately avoiding this inconsistency. However, such delays naturally impede change propagation across processes and are hard to set correctly.

This tension between speed and consistency in systems relying on eventual consistency is generally resolved by uniformly selecting one trade-off point that all nodes in a system adhere to [5], [16]. However, large-scale systems routinely need to cater to diverging requirements, e.g. due to hardware heterogeneity or applications offering heterogeneous services. For instance, blockchain systems are composed of miners and clients with different priorities. Miners are incentivized to receive data as fast as possible [20]–[22] while clients mainly seek security, i.e. consistency [23], [24].

We argue in this paper for *differentiated levels of consistency* to satisfy the heterogeneity inherent to large-scale systems. Specifically, we focus on the implementation of eventually consistent replicated data structures in large message-

- *D. Frey and F. Taïani are with Univ Rennes, CNRS, Inria, IRISA, Rennes F-35000, France. E-mails: davide.frey@inria.fr, francois.taiani@irisa.fr.*
- *A. Mostefaoui and M. Perrin are with LS2N, Université de Nantes, Nantes F-44300, France. E-mails: achour.mostefaoui@univ-nantes.fr, matthieu.perrin@univ-nantes.fr.*
- *P.-L. Roman is with École Polytechnique Fédérale de Lausanne (EPFL), Lausanne CH-1015, Switzerland. E-mail: pierre-louis.roman@epfl.ch.*

A preliminary version of this paper appears in the proceedings of SRDS 2016 [1]. Please refer to the Appendix in page 15 for a summary of the changes.

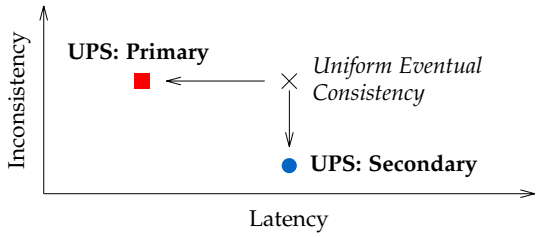


Fig. 2: Aimed consistency/latency trade-off in UPS.

passing distributed systems. We investigate how different levels of consistency can be provided in this context using a novel, judiciously crafted, epidemic broadcast protocol. Unlike past hybrid consistency conditions [25]–[29], our solution differentiates eventual consistency in and of itself to favor speed for specific nodes (e.g. blockchain miners) and consistency for others (e.g. blockchain clients). It leverages the internal dynamics of probabilistic gossip/epidemic protocols [30], [31] whose scalability has been put to use in a broad range of recent systems [8], [32]–[34].

Evaluating such a protocol in practice raises, however, an important methodological point: how to measure consistency. Consistency conditions are typically defined as predicates on execution histories; a system execution is either consistent or it is not, leaving no room for quantification. Practitioners who deploy eventual consistency are interested in the current *level* of consistency of a running system, i.e. how far the system currently is from a consistent situation. Quantitatively measuring the inconsistencies in a system is not straightforward. We can measure the level of *agreement* between nodes by counting how many nodes see the same state [5], but this approach can thus lead to paradoxes. For instance, the system in Fig. 1 appears close to agreement if most nodes read the same state (2) as p_2, even though (2) is inconsistent with the final converged state $(1, 2)$.

This paper makes the following contributions:

- *Update-query consistency with Primaries and Secondaries (UPS)*, a novel consistency mechanism offering two levels of eventual consistency in a system (§3). UPS extends the update-query consistency protocol [15] by exploiting a novel two-phase epidemic broadcast—*Gossip Primary-Secondary (GPS)*—that involves two classes of nodes. Primary nodes (Primaries in short) seek to receive object updates as fast as possible while Secondary nodes (Secondaries in short) strive to minimize the inconsistencies they perceive. Fig. 2 summarizes these goals.
- A formal analysis of GPS' latency and network overhead via a continuous analysis as well as of its consistency via a compartment-based discrete analysis (§4).
- A scalable consistency metric to assess the inconsistency observed by Primaries and Secondaries using UPS (§5).
- Experimental evaluation of the consistency and latency properties of UPS both in a large-scale simulated one-million-node network and in a network simulating the Ethereum blockchain [7] (§6). We show in both settings that the cost paid by each node class is small compared to an undifferentiated system: Primaries experience a lower latency with levels of inconsistency close to those of undifferentiated nodes, while Secondaries observe fewer inconsistencies at a minimal latency cost. Besides, GPS only incurs a small overhead in message complexity, proportional to the ratio of Primaries in the network.

2 Background & Problem Statement

2.1 Linearizability

Linearizability [35] ensures that a replicated data structure behaves in a way that is indistinguishable from that of a single shared copy. It typically imposes a total order on the operations applied to the data structure and is therefore considered one of the strongest available consistency criteria.

Linearizability provides a convenient abstraction to developers but comes with great costs in large-scale systems. Failures are common in these systems and communication delays are typically unbounded, i.e. the network is said to be asynchronous. These two constraints make it impossible to emulate any linearizable data structure without abandoning fundamental properties such as availability or partition tolerance, a trilemma elegantly captured by the CAP theorem [11]. Worse, Attiya and Welch have proved [36] that even in failure-free synchronous systems, the implementation of strongly consistent data structures (e.g. stack, queue, set) implies execution times for local operations that are linear with respect to network latency, resulting in poor performance.

2.2 Weak Consistency Conditions

Weaker consistency conditions overcome the limits and costs of linearizability by striking a balance between agreement, speed, and dynamicity within a system. Such conditions include PRAM consistency, causal consistency [37]–[39], and eventual consistency [12]. PRAM and causal consistency expect the local histories observed by each process to be plausible, regardless of the other processes, but they do not impose state convergence. On the other hand, eventual consistency tolerates temporary inconsistent states but requires that all replicas eventually converge to the same state.

Unlike linearizability, these weaker consistency conditions can be implemented in any distributed system where communication is possible, even when facing unbounded delays (asynchrony) and node failures. In addition, their local time complexity (e.g. the time taken by one given node to execute an operation locally) does not depend on communication delays, allowing for efficient implementations.

2.3 Implementing Consistency: the Role of Broadcast

In message-passing distributed systems, a replicated data structure is typically implemented by (1) hosting replicas of the shared data on each process, and by (2) leveraging a broadcast primitive to disseminate the operations that processes execute on their copy of the shared data structure. The consistency level exhibited by the resulting data structure is tightly related to the guarantees of the underlying broadcast primitive they use. Linearizability typically requires a total-order broadcast, which is computationally equivalent to consensus [40], [41]. By contrast, weaker consistency conditions can be implemented using weaker broadcast abstractions. Causal broadcast can be used to implement a causal memory and reliable broadcast is the communication primitive indicated to implement eventual consistency [42].

2.4 Update Consistency

As we hinted at in §1, eventual consistency requires replicated objects to converge to a globally consistent state when update operations stop for *long enough*. On its own, this condition turns out to be too weak for most practical purposes as the convergence state does not depend on the operations performed on the object. For this reason, actual implementations of eventually consistent objects usually refine eventual consistency by linking the convergence state to its sequential specification.

In this paper, we focus on one such refinement: *Update consistency* [15]. Consider the append-only queue object of Fig. 1; its sequential specification consists of two operations:

- append(x), appends the value x at the end of the queue;
- read(), returns the sequence of all the elements ever appended, in their append order.

Update consistency requires the final converged state to result from a total ordering of all the append operations of all agents. This ordering must also respect the order in which each agent issues its append operations (a.k.a. the *process order*). For example, the scenario in Fig. 1 satisfies update consistency since the final convergence state results from the ordering $\langle q.\text{append}(1), q.\text{append}(2)\rangle$. An equivalent definition [15] states that an execution history respects update consistency if it contains an infinite number of updates or if a finite number of reads can be removed from it such that the resulting pruned history is sequentially consistent. In Fig. 1, this is achieved by removing the read() that returns 2.

Alg. 1 shows an algorithm from [15] that implements the update-consistent append-only queue of Fig. 1. Alg. 1 pairs a broadcast primitive (cf. Line 8) with Lamport clocks [43] to reconstruct a total order of operations after the fact [15]. Relying on this after-the-fact total order allows update consistency to support non-commutative operations such as append without requiring type-constrained CRDTs [13].

2.5 Problem Statement

The critical feature of update consistency lies in its ability to precisely define the nature of the converged state the system should reach once all updates have been issued. In practice, however, the intermediate behavior of a system before it converges (i.e. before it reaches "consensus") is equally critical for its usability [5]. Focusing on this intermediate behavior raises two key challenges. First, existing eventually-consistent systems tend to treat all nodes uniformly, leading to a non-converged behavior that does not cater for the specific needs of individual participants [14], [44]. Second, studying a system's behavior before convergence requires the ability to quantify how well- or ill-converged this system is. Unfortunately, measuring the inconsistency level of non-converged states remains an ill-defined exercise. Despite being easy to compute in a broad range of situations, existing system-oriented metrics do not consider the ordering of update operations (append in our case) [5], [45]–[47]. On the other hand, theoretical metrics require global system knowledge [48], which makes them impractical at large scale.

This paper addresses both challenges. First, we propose a novel scalable broadcast protocol that, when used in Line 8 of Alg. 1, satisfies update consistency and supports different consistency levels for read operations that occur before convergence.[1] Specifically, we exploit the trade-off between delivery speed and consistency to offer different levels of inconsistency within the same system: We distinguish a small fraction of Primary nodes that should receive fast, albeit possibly inconsistent, information, from a significant fraction of Secondary nodes that should only receive stable consistent information, albeit more slowly. Second, we propose a novel metric to measure the level of inconsistency of an append-only queue, and use it to evaluate our protocol.

Algorithm 1: Update consistency for an append-only queue.

```
1:  initialization
2:      int id                                          ▷ Node identifier
3:      set <int, int, V> U ← ∅                         ▷ Set of updates to the queue
4:      int clock_id ← 0                                ▷ Node's logical clock
5:  procedure APPEND(v)                                 ▷ Append a value v to the queue
6:      clock_id ← clock_id + 1
7:      U ← U ∪ {< clock_id, id, v >}
8:      BROADCAST (< clock_id, id, v >)
9:  upon receive (< clock_msg, id_msg, v_msg >) do
10:     clock_id ← MAX(clock_id, clock_msg)
11:     U ← U ∪ {< clock_msg, id_msg, v_msg >}
12: procedure READ( )                                   ▷ Read the current state of the queue
13:     q ← ()                                          ▷ Empty queue
14:     for all < clock, id, v > ∈ U sorted by (clock, id) do
15:         q ← q · v
16:     return q
```

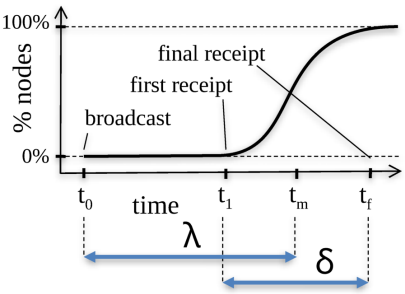


Fig. 3: Two sorts of speeds: latency (λ) and jitter (δ).

3 The GPS Broadcast Protocol

3.1 System Model

We consider a large set of nodes $p_1, ..., p_N$ that communicate using point-to-point messages. Any node can communicate with any other node, given its identifier. We use probabilistic protocols that are naturally robust to crashes and message losses, but do not consider these aspects in the rest of the paper for simplicity. Nodes are categorized in two classes: a small number of Primary nodes (Primaries) and a large number of Secondary nodes (Secondaries). The class of a node is a parameter set by the application that captures the requirements in terms of update consistency: Primaries should perceive updates as fast as possible, while Secondaries should experience as few inconsistencies as possible.

1. Importantly, "consistency level" here does not refer to the *time* required to achieve (eventual) agreement, but on the transient *level of disagreement* before agreement is achieved. This transient *level of disagreement* has a direct impact of the quality of service and trust users can put in a system, and is therefore of prime practical importance [5].

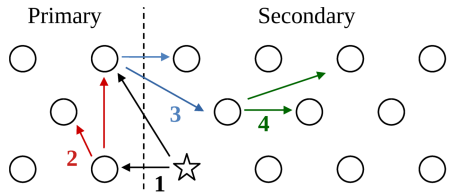


Fig. 4: Model of GPS and path of an update in the system.

3.2 Intuition & Overview

We have repeatedly referred to the inherent trade-off between speed and consistency in eventually consistent systems. This trade-off may appear contradictory: if Primaries receive updates faster, why should not they also experience higher levels of consistency? This apparent paradox arises because we have so far silently confused *speed* and *latency*. The situation within a broadcast is in fact more subtle and involves two sorts of speeds (Fig. 3): *latency* (λ, shown in the figure as the mean delay $t_m - t_0$ over all nodes, where t_m is the mean reception time) is the time a message m takes to reach individual nodes, from the point in time of m's sending (t_0). *Jitter* (δ), by contrast, is the delay between the first (t_1) and the last receipt (t_f) of a broadcast.[2] Inconsistencies typically arise in Alg. 1 when some updates have only partially propagated within a system, and are thus predominantly governed by the jitter δ rather than the average *latency* λ.

The gossip-based broadcast protocol we propose, Gossip Primary-Secondary (GPS), exploits this distinction and proposes different δ/λ trade-offs. Primaries have a reduced λ, thus accelerating update receptions, but also a slightly increased δ, while Secondaries have a reduced δ, thus increasing consistency by improving the order of update receptions, at the cost of a slightly higher λ.

Intuitively, GPS uses the set of Primaries as a sort of message "concentrator" that accumulates copies of an update u before collectively forwarding it to Secondaries in a better order. Fig. 4 depicts the main phases of GPS:

(**1**) A new update u is first sent to Primaries;
(**2**) Primaries disseminate u among themselves;
(**3**) Once most Primaries have received u, they forward it to Secondaries;
(**4**) Finally, Secondaries disseminate u among themselves.

A key difficulty in this sequence consists in deciding when to switch from Phase **2** to **3**. A coordinated transition would require a global synchronization mechanism, which is costly and generally impracticable in very large systems. Instead, GPS relies on a less accurate but scalable local procedure based on broadcast counts: each Primary node decides locally when to start forwarding to Secondary nodes.

For completeness' sake, let us note that since Phase **1** exclusively relies on Primaries, GPS stops working if all Primaries fail. In cases when such a systemic catastrophic failure is plausible, GPS requires additional fault-tolerance mechanisms. These mechanisms are however deemed out of scope of the current paper.

2. In most large-scale epidemic broadcast scenarios, $t_0 - t_1$ corresponds to a single communication exchange and is therefore small in comparison to λ, which is driven by the average path length in the random gossiping graph. As a result, λ and δ tend to capture the same underlying dynamics, and δ is ignored.

Algorithm 2: Gossip Primary-Secondary (GPS) for a node.

```
1:  parameters
2:      int fanout                          ▷ Number of nodes to send to
3:      int rpsViewSize                     ▷ Out-degree per class of each node
4:      boolean isPrimary                   ▷ Class of the node
5:  initialization
6:      set{node} Γ_P ← ∅                   ▷ Set of Primary neighbors
7:      set{node} Γ_S ← ∅                   ▷ Set of Secondary neighbors
8:      map{message, int} R ← ∅             ▷ Counters of message
                                            ▷ duplicates received
9:  periodically
10:     Γ_P ← rpsViewSize nodes from Primary-RPS
11:     Γ_S ← rpsViewSize nodes from Secondary-RPS
12: procedure BROADCAST(msg)                ▷ Called by the application,
13:     GOSSIP(msg, Γ_P)                    ▷ e.g. in Line 8 of Alg. 1
14: procedure GOSSIP(msg, targets)
15:     for all j ∈ {fanout random nodes in targets} do
16:         send(msg) to j
17: upon receive (msg) do
18:     if msg ∉ R then R[msg] ← 0
19:     R[msg] ← R[msg] + 1
20:     if R[msg] = 1 then DELIVER(msg)     ▷ Deliver 1st receipt
21:     if isPrimary then
22:         if R[msg] = 1 then GOSSIP(msg, Γ_P)
23:         if R[msg] = 2 then GOSSIP(msg, Γ_S)
24:     else
25:         if R[msg] = 1 then GOSSIP(msg, Γ_S)
```

Algorithm 3: Uniform gossip for a node (baseline).

(only showing main differences to Alg. 2)

```
9′:  periodically Γ ← rpsViewSize nodes from RPS
14′: procedure GOSSIP(msg)
15′:     for all j ∈ {fanout random nodes in Γ} do
16′:         send(msg) to j
17′: upon receive (msg) do
18′:     if msg ∈ R then return             ▷ "Infect and die"
19′:     R[msg] ← 1
20′:     DELIVER(msg)                       ▷ Deliver 1st receipt
21′:     GOSSIP(msg)
```

3.3 The GPS Algorithm

Alg. 2 shows the pseudo-code of GPS. GPS follows the standard models of reactive epidemic broadcast protocols [49], [50]. Each node keeps a history of the messages received so far (in the R variable, Line 8), and decides whether to retransmit a received broadcast to fanout other nodes based on its history. However, contrary to a standard epidemic broadcast, GPS handles Primaries and Secondaries differently:

- First, GPS uses two instances of random peer sampling (RPS) protocols [51], [52] (Lines 10 and 11) to track each node class. Both Primaries and Secondaries use the RPS view of their class (i.e. a small random sample of nodes from this class) to retransmit a message they receive for the first time to fanout other nodes in their own class (Lines 22 and 23), thus implementing Phases **2** and **4**.
- Second, nodes in GPS handle retransmissions differently depending on their class. Primaries use the inherent

presence of message duplicates in gossip protocols to decide locally when to switch from Phase **2** to **3**. Each node keeps count of the received copies of individual messages (Lines 8, 18 and 19). Primaries use this count to detect duplicates, and forward a message to fanout Secondaries (Line 23) when a duplicate is received for the first time, thus triggering Phase **3**.

In short, we can classify Primaries as *infect twice and die* and Secondaries as *infect and die*. For comparison, a standard *infect and die* gossip without classes is shown in Alg. 3; we refer to it as *Uniform gossip*. Uniform gossip serves as our baseline for our analysis (§4) and our experiments (§6).

4 Formal Analysis of GPS

In the following we analyze the expected performance of GPS and compare it to Uniform gossip in terms of message complexity and latency in §4.1, then analyze the behavior of nodes in GPS using a compartmental model in §4.2. Our analysis uses the following notations:

- $N = \{p_1, ..., p_{|N|}\}$ is the set of nodes in the system;
- P is the set of Primaries;
- S is the set of Secondaries such that $S = N \setminus P$;
- $d = |P|/|N|$ is the density of Primaries;
- $f \in \mathbb{N}$ is the fanout (assumed $\geq 1/d$).

4.1 Asymptotic Continuous Analysis

We proposed an asymptotic analysis of GPS in a preliminary version of this paper [1, §III.D]. This analysis shows that Primaries receive messages rounds earlier of nodes in Uniform gossip and that, assuming a small density d, GPS induces small overheads both in message latency for Secondaries and in message complexity. We summarize here the main findings to compare them with our evaluation results further.

The latency of Primaries is $log_f(d)$ rounds smaller compared to nodes in Uniform gossip:

$$\Delta\lambda_P \sim -\log_f(d). \tag{1}$$

Assuming a small fraction of Primaries (i.e. $d \ll 1$), the latency of Secondaries can be approximated as

$$\lambda_S \sim \log_f(|S|) + 1 + 2C. \tag{2}$$

with C a constant independent of N.

Since Primary nodes gossip twice (once towards other Primary nodes at Line 22 of Alg. 2, and once towards Secondary nodes at Line 23), and Secondary nodes gossip once (Line 25), the message complexity of GPS can be approximated as

$$|msg|_{GPS} \approx (1 + d) \times |msg|_{uniform}. \tag{3}$$

Eq. (3) shows that GPS induces a fraction d of additional messages compared to Uniform gossip, e.g. a network with 1% Primaries induces only 1% more messages.

4.2 Compartment-based Discrete Analysis

We next analyze the dissemination of messages in GPS using a discrete-time compartmental model from epidemiology. Compartmental models split the population into *compartments* (or states) and analyze population flows between compartments based on transition rates. The simple three-compartment model Susceptible-Infectious-Recovered (SIR) [53] can be used to analyze Uniform gossip: at first every node is *susceptible*, then becomes *infectious* upon the first reception of a message m, gossips m as a result, and *recovers* (or dies) by ignoring m further. The SIR model defines the infection rate β and recovery rate γ such that the population follows the sequence $\mathcal{S} \rightarrow_\beta \mathcal{I} \rightarrow_\gamma \mathcal{R}$.

GPS distinguishes two populations—Primaries and Secondaries—and thus cannot be analyzed using a straightforward SIR model. Since Primaries first gossip to Primaries then to Secondaries, intuitively, a fitting model should contain two infectious compartments, e.g. SISIR, while Secondaries follow the classic SIR sequence.

Model. We assume a synchronous execution, in which messages sent in round r are received in round $r + 1$. Each round r unfolds in two phases: first, messages from round $r - 1$ are received, then messages from round r are sent.

We reuse some of the notation from SIR, e.g. β and γ, but introduce our own notation for compartments for readability. We identify the compartments based on node class and the number of messages sent or received by nodes. We will note

$$X_{i,j}$$

the compartment of nodes of class X (with $X \in \{P, S\}$) that have received i copies of a message and sent j messages. For Primaries $i \in \{0, 1, 2+\}$, while for Secondaries $i \in \{0, 1+\}$, where $y+$ means *y or more*; and $j \in \{0, 1, 2\}$, where $j = 2$ is only used for Primaries.

Fig. 5 represents all compartments as boxes, and events, i.e. the reception or sending of one or multiple messages, as arrows between the compartments. For instance, the dashed arrow from $P_{0,0}$ to $P_{2+,0}$ corresponds to the situation in which a susceptible Primary receives 2 or more messages in the same round. Similarly, the solid arrow from $P_{2+,0}$ to $P_{2+,2}$ corresponds to the sending of two messages (to f other Primaries and f Secondaries). The number of messages sent or received by an arrow is derived from the compartments connected by this arrow, e.g. 2 sendings in the latter example. Nodes in blue compartments send messages in the second phase of a round hence change compartment.

4.2.1 Primary nodes

By construction, a Primary node never sends more than 2 messages, and never more than the number of messages it has seen so far, i.e. $i \geq j$ in the above notation. As a result, the compartments of interest for Primaries are $P_{0,0}$, $P_{1,0}$, $P_{2+,0}$, $P_{1,1}$, $P_{2+,1}$, and $P_{2+,2}$.

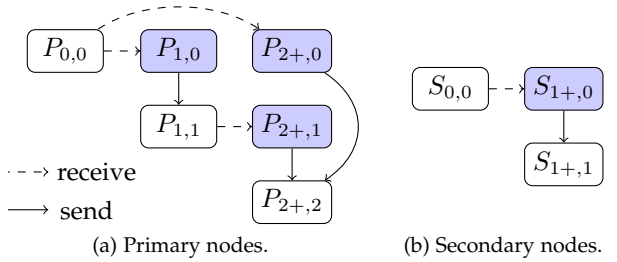
Originally all Primaries are in the $P_{0,0}$ compartment since they have not received any message, which mirrors the *susceptible* compartment of SIR. Symmetrically, nodes are in $P_{2+,2}$ when they have received 2 or more messages, and sent 2 (the first to other Primaries, and the second to Secondaries). These nodes will not send any more messages (will not "infect" any other node), and $P_{2+,2}$ can therefore be interpreted as the compartment of *recovered* nodes of SIR.

For the compartments $P_{0,0}$, $P_{1,1}$ and $P_{2+,2}$ (filled in white in Fig. 5a), we will note

$$P^r_{i,j} \qquad \text{for } j = \min(2, i),$$

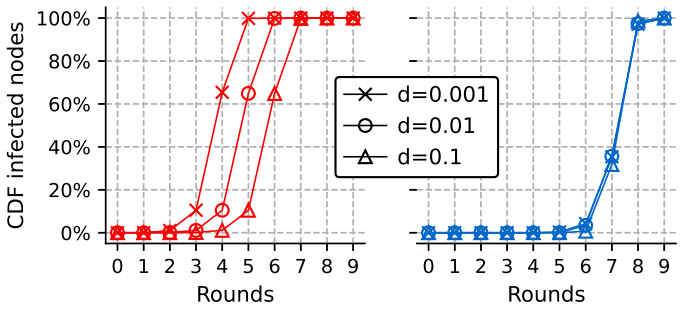
the number of nodes in $P_{i,j}$ at the *start* of round r.

The remaining compartments require a somewhat different treatment. This is because we have assumed each round



(a) Primary nodes. (b) Secondary nodes.

Fig. 5: The compartments used in our discrete analysis.



(a) Dissemination to Primaries. (b) Dissemination to Secondaries.

Fig. 6: Numerical application of Thms. 1 and 2 on message dissemination to Primaries and Secondaries for a network of $1,000,000$ nodes and a density $d \in \{0.001, 0.01, 0.1\}$.

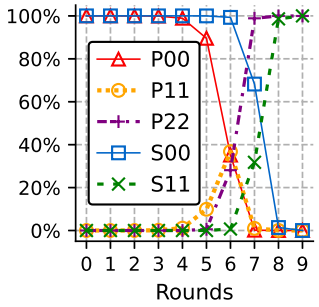


Fig. 7: Evolution of compartment sizes (cf. Fig. 6) following Thms. 1 and 2 for a network of $1,000,000$ nodes and a density $d = 0.1$.

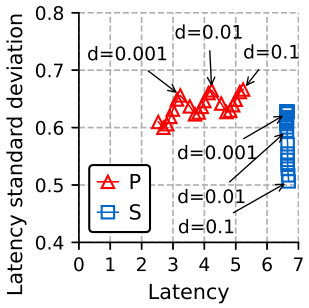


Fig. 8: Effect of density d on message latency and jitter (as the latency std. dev.) for Primaries and Secondaries using Thms. 1 and 2.

unfolds in two phases: first, messages from round $r-1$ are received, then messages from round r are sent. Because the sending of messages is deterministic (e.g. all nodes in compartment $P_{1,0}$ send a message in the second phase of a round), the nodes present in the three compartments $P_{1,0}$, $P_{2+,0}$ and $P_{2+,1}$ (filled with light blue in Fig. 5a) always send messages in the second phase of a round, and therefore change compartments. As a result, $P_{1,0}$, $P_{2+,0}$ and $P_{2+,1}$ are systematically empty between two rounds. By convention, for these transitory compartments, we will note

$$P^r_{i,j} \qquad \text{for } i > j,$$

the number of nodes in compartment $P_{i,j}$ just *after* the first phase of round r, and *before* the sending of any message.

We are interested in the evolution of $P^r_{0,0}$, $P^r_{1,1}$ and $P^r_{2+,2}$ in each round. Because the number of Primaries, $|P|$, remain constant we can derive $P^r_{2+,2}$ from $P^r_{0,0}$ and $P^r_{1,1}$ using

$$P^r_{2+,2} = |P| - P^r_{0,0} - P^r_{1,1}. \tag{4}$$

In the following we will therefore focus on $P^r_{0,0}$ and $P^r_{1,1}$.

Theorem 1. *The evolution of Primary nodes follows the formulae*

$$P^{r+1}_{0,0} = a(P^r_{0,0}, P^{r-1}_{0,0}), \tag{5}$$
$$P^{r+1}_{1,1} = b(P^r_{0,0}, P^{r-1}_{0,0}, P^r_{1,1}), \tag{6}$$

where

$$a(x,y) = x\left(1 - \frac{f}{|P|}\right)^{y-x}, \text{ and}$$
$$b(x,y,z) = \left(z - \frac{f \times x\,(x-y)}{|P| - f}\right)\left(1 - \frac{f}{|P|}\right)^{y-x}.$$

Proof. Please refer to the Appendix for the proof. □

4.2.2 Secondary nodes

The compartments of Secondaries (Fig. 5b) are simpler, as Secondaries can only be in two states at the end of a round with respect to a message m: either they have not received m and not forwarded it ($S_{0,0}$), or they have received m once or more, and forwarded it once ($S_{1+,1}$). As for Primaries, $S_{1+,0}$ (Secondaries that have received m, but not yet forwarded it) is a temporary compartment, used for clarity, that is filled and emptied in the course of a round.

Theorem 2. *The evolution of Secondary nodes follows the formula*

$$S^{r+1}_{0,0} = c\Big(P^r_{0,0}, P^{r-1}_{0,0}, P^r_{1,1}, P^{r-1}_{1,1}, S^r_{0,0}, S^{r-1}_{0,0}\Big) \tag{7}$$

where

$$c(x,y,z,u,v,w) = v \times \left(1 - \frac{f}{|S|}\right)^{y+u+w-x-z-v} \tag{8}$$

Proof. Please refer to the Appendix for the proof. □

4.2.3 Numerical application

Figs. 6 to 8 use Thms. 1 and 2 to chart the behavior of GPS under various parameters. In particular, Figs. 6 and 7 confirm the asymptotic results of §4.1: a low density causes Primaries to receive a broadcast earlier, but has little impact on the dissemination to Secondaries, with the gain by Primaries of the form $-log_f(d)$ (Eq. (1)). Fig. 8 confirms our intuition regarding the impact of density on the jitter/latency trade-off between Primaries and Secondaries: although a high Primary density leads to reduced latency gains for Primaries (x-axis, 'P' curve, $d = 0.1$ annotation), it also yields a reduced jitter for Secondaries (y-axis, 'S' curve, $d = 0.1$). In §6, we will see that this reduced jitter delivers an improved consistency for Secondaries, matching the intuition presented in Fig. 2.

5 Consistency Metric

To assess more precisely how the UPS/GPS algorithm we are proposing (§3.3) can improve the intermediate behavior of an update consistency data structure, we need to quantify how consistent this data structure is. As such, we propose in this section a novel consistency metric that overcomes the weaknesses of existing solutions by (i) considering the ordering of operations, and (ii) being locally computable.

5.1 A General Consistency Metric

We first observe that the algorithm for an update-consistent append-only queue (cf. Alg. 1) guarantees that all its execution histories respect update consistency. To measure the consistency level of temporary states, we therefore evaluate how the history deviates from a stronger consistency model, *sequential consistency* [54]. An execution respects sequential consistency if it is equivalent to some sequential (i.e. totally ordered) execution that contains the same operations, and respects the sequential (process) order of each node.

Since update consistency relies itself on a total order, the gist of our metric consists in counting the number of read operations that do not conform with a total order of updates that leads to the final convergence state. Given one such total order, we may transform the execution into one that conforms with it by removing some read operations. In general, a data object may reach a given final convergence state through different possible total orders, and for each such total order we may have different sets of read operations whose removal makes the execution sequentially consistent. We thus count the level of inconsistency by taking the minimum over these two degrees of freedom: choice of the total order, and choice of the set.

More formally, we define a *set of temporary inconsistencies* (or temporary inconsistency set for short) w.r.t. an execution Ex as a *finite* set of read operations E that, when removed from Ex, makes it sequentially consistent.[3] We denote the set of all the temporary inconsistency sets of an execution Ex over all total orders by $TI(Ex)$. We then define the *relative inconsistency* RI of an execution Ex as the minimal number of read operations that must be removed from Ex to make it sequentially consistent:

$$RI(Ex) = \begin{cases} \min\limits_{E \in TI(Ex)} |E| & \text{if } TI(Ex) \neq \emptyset \\ +\infty & \text{otherwise.} \end{cases} \quad (9)$$

For example, in Fig. 1, several total orders (of all operations) can lead to a sequentially consistent execution, once pruned of problematic read operations. If we consider the total order $\langle\, q.\mathrm{append}(1), m.\mathrm{append}(2), q.\mathrm{read}(1), q.\mathrm{read}(2), q.\mathrm{read}(1,2), q.\mathrm{read}(1,2)\,\rangle$ then the execution becomes sequentially consistent by removing $q.\mathrm{read}(1)$ and $q.\mathrm{read}(2)$. By contrast, with $\langle\, q.\mathrm{append}(1), q.\mathrm{read}(1), q.\mathrm{append}(2), q.\mathrm{read}(2), q.\mathrm{read}(1,2), q.\mathrm{read}(1,2)\,\rangle$, removing just $q.\mathrm{read}(2)$ suffices to make the execution sequentially consistent. As the execution itself is not sequentially consistent, the level of inconsistency in Fig. 1 is therefore 1.

The metric RI is particularly adapted to compare the consistency level of implementations of update consistency: the lower, the more consistent. In the best-case scenario where Ex is sequentially consistent, $TI(Ex)$ contains the empty set, resulting in $RI(Ex) = 0$. In the worst case scenario where the execution never converges (i.e. some nodes indefinitely read incompatible local states), every set of reads that needs to be removed to obtain a sequentially consistent execution is infinite. Since TI only contains finite sets of reads, $TI(Ex) = \emptyset$ and $RI(Ex) = +\infty$.

3. Note that when Ex is finite, such a set E always exists, as removing all read operations from Ex will make it consistent. By contrast, if Ex is infinite, it may be that no finite set of reads can make it consistent. In this latter case, we will say that Ex exhibits an infinite relative inconsistency.

5.2 Update-consistent Append-only Queue

In general, $RI(Ex)$ is complex to compute, in contrast to more practice-oriented proposals [5], [45]–[47]: one must consider all possible total orders of events that can fit sequential consistency, and all possible finite sets of reads to identify temporary inconsistency sets. But for an append-only queue implemented with Alg. 1, we can easily show there exists exactly one minimal set of temporary inconsistencies for executions in which a convergence state is reached.

To understand why, we first observe that the append operation is non-commutative. This implies that there exists a single total order of append operations that yields a given final convergence state. Second, Alg. 1 guarantees that the size of the successive sequences read by a node can only increase and that read operations always reflect the writes made on the same node. Consequently, in order to have a sequentially consistent execution, it is necessary and sufficient to remove all the read operations that return a sequence that is not a prefix of the sequence read after convergence. These read operations constitute the minimal set of temporary inconsistencies TI_{min}.

6 Evaluation

In this section, we evaluate GPS and UPS in order to answer the following three research questions (RQs):

- RQ1: How does UPS compare against a uniform update consistency in terms of consistency (§ 6.4), message latency (§ 6.5), and message complexity (§ 6.6) when experimented on a one-million-node network?
- RQ2: How do the experimental results for RQ1 fare against the expected results from the continuous and compartment-based analyses? (§6.4–§6.6)
- RQ3: How does GPS perform when applied to a practical usecase, namely a blockchain network? (§6.7)

This evaluation explores two complementary large-scale setups. We opt for an extreme system size (1,000,000 nodes) in RQ1 to analyze the behavior of UPS and GPS well beyond the size of most existing P2P systems. In doing so, we aim to inform the design of P2P systems at scales that do not exist yet, but might appear in the future. By comparison, in RQ3, we study the impact of GPS in a concrete present-day P2P system (of ≈23,000 nodes) using recent public datasets.

We first detail the evaluation methodology in §6.1–§6.2 and summarize the results for RQ1 in §6.3.

6.1 Methodology Common to RQ1 & RQ3

We evaluate UPS and GPS using PeerSim [55], a well-known simulator for large-scale peer-to-peer networks. Both code and results are available online for reproducibility [56].

RPS implementation. Both GPS and Uniform gossip require all nodes to execute RPS protocols to fill their respective network views (e.g. Line 10 and Line 11 of Alg. 2), which serve as peer pools to select gossip targets (Line 14 of Alg. 2).

We implement the two RPSs for GPS (i.e. one per node class) and the one for Uniform gossip as global oracles that fill the views of every node with other nodes sampled randomly from the entire network. Typical RPS implementations [57], [58] differ in that they fill one node's view by only

sampling its nearest nodes in the topology (e.g. neighbors of neighbors) for practical reasons, thus taking longer to produce a randomness comparable to that of an oracle.

Scenario. We consider a scenario where all nodes share an instance of an update-consistent append-only queue, as defined in §2.4. Following the definition of update consistency, nodes eventually converge into a strongly consistent state once they stop modifying the queue.

We opt for a scenario where 10 $\mathrm{append}(x)$ update operations, with $x \in \mathbb{N}$, are performed on the queue by 10 random nodes with one update per round during the first 10 rounds. Local clocks start at 0 and are incremented every round. Each append is timestamped with the round number of its emission (cf. Line 7 of Alg. 1). All nodes also execute a read operation every round on their local copy of the queue.

We expect the system to experience two periods: (1) a temporary phase where updates are issued and disseminated, emulating a system continuously performing updates, then (2) a stabilized state once updates finished propagating and most nodes have converged to a strongly consistent state.

Consistency metric. The 10 append operations of our scenario, that we note $(\mathrm{append}_i)_{i\in[0..9]}$ for simplicity, are eventually ordered according to the round in which they were sent (round i for operation append_i). As a result a read operation by a node q at round r is inconsistent in case the set of all append operations received by q so far do not form a prefix of the full $(\mathrm{append}_i)_{i\in[0..9]}$ sequence.

We refine the generic consistency metric proposed in §5 to suit our experimental setting. As such, since the number of updates is finite and each node reads the state of the queue at each round, we define a per-round metric to compare the evolution of the inconsistency of Primaries and Secondaries through time. For each round r, we define $R_P(r)$ and $R_S(r)$ as the sets of all the read operations performed at round r by Primaries and Secondaries, respectively. Using $R_P(r)$ and $R_S(r)$, we define the per-round inconsistency metrics $Incons_P(r)$ and $Incons_S(r)$ as the ratio of Primaries and Secondaries that observe an inconsistent read at round r (i.e. the corresponding reads are in TI_{min}):

$$Incons_P(r) = \frac{|TI_{min} \cap R_P(r)|}{d \times |N|}, \tag{10}$$

$$Incons_S(r) = \frac{|TI_{min} \cap R_S(r)|}{(1-d) \times |N|}. \tag{11}$$

The metric for all nodes $Incons_{P+S}(r)$ naturally follows:

$$Incons_{P+S}(r) = d \times Incons_P(r) + (1-d) \times Incons_S(r). \tag{12}$$

In the following, we focus on these *instantaneous* per-round metrics which provide an equivalent yet more accurate view than the relative inconsistency of an experimental run, $RI(Ex)$. In fact, $RI(Ex)$ can be computed as follows:

$$RI(Ex) = |N| \times \sum_{r=0}^{\infty} (Incons_{P+S}(r)). \tag{13}$$

Statistical significance and plots. We evaluate each configuration 25 times and record the resulting distribution. Figs. 10 and 13 depict mean values as symbols and minimum and maximum values as error bars. Low variability may render error bars unnoticeable. Figs. 14 and 15 depict mean values as symbols and 95% confidence intervals, instead of extrema, as error bars since they rely on realistic datasets.

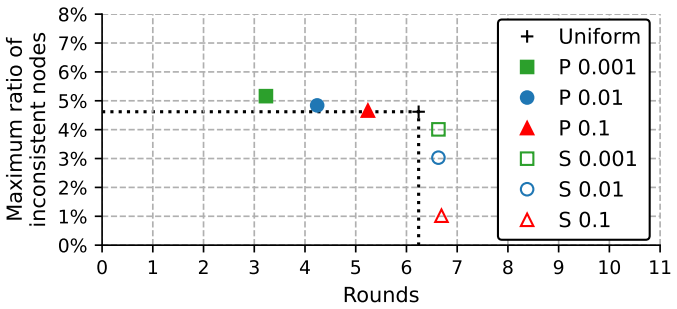


Fig. 9: Summary of the experimental consistency/latency trade-offs in UPS reflecting the goals expressed in Fig. 2 (closer to the bottom left is better). Primaries are faster but *a bit* less consistent than nodes in Uniform gossip, while Secondaries are more consistent but *a bit* slower.

6.2 Methodology Specific to RQ1

We target a world-scale network to evaluate RQ1.

Network parameters. We use a network of 1,000,000 nodes, a fanout of 10 and an RPS view size of 100. These parameters yield a high broadcast reliability (i.e. the probability that a node receives a message) (cf. §6.6). Reliability may be improved by increasing the fanout [49], however we choose powers of 10 for these parameters for readability.

Configurations. We use three configurations for GPS, one per density d of Primaries: 000.1, 00.1, 0.1. In addition, we also evaluate Uniform gossip (cf. Alg. 3) as a baseline.

6.3 RQ1 – Summary of World-scale Experiments

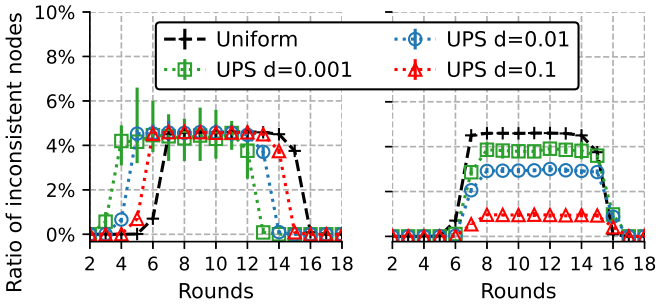
Fig. 9 mirrors Fig. 2 presented in §1. Fig. 9 overviews the experimental results depicting the trade-offs between consistency and latency for the different sets of nodes involved in our scenario. UPS configurations are each shown with two points: Primaries are shown using solid symbols, Secondaries using hollow ones. Uniform gossip is represented by a single black cross. The position on the x-axis shows the average update latency experienced by each set of nodes, and the y-axis their worst perceived level of inconsistency, i.e. the maximum $Incons_X(r)$ value over all runs.

Fig. 9 shows UPS delivers the differentiated consistency/latency trade-offs we set out to achieve in §1: Secondaries enjoy higher consistency levels than they would in a uniform update-query consistency protocol, while paying only a small cost in terms of latency. The consistency boost strongly depends on the density of Primaries in the network, as we further show in §6.4, while the cost in latency does not, reflecting our analysis in §4.1. Primaries present the opposite behavior, i.e. Primaries' latency gains evolve in the reverse direction of Secondaries' consistency gains.

6.4 RQ1 & RQ2 – Consistency Level

Fig. 10a and Fig. 10b show the evolution over time of $Incons_P(r)$ (cf. Eq. (10)) and $Incons_S(r)$ (cf. Eq. (11)), respectively. Since most nodes are Secondaries, the results of $Incons_{P+S}(r)$ (cf. Eq. (12)) are close to those of $Incons_S(r)$.

In short, we note a clear improvement of the maximum inconsistency level of Secondaries over the baseline and an



(a) Primary nodes. (b) Secondary nodes.

Fig. 10: Ratio of inconsistent nodes among Primaries (Fig. 10a) and Secondaries (Fig. 10b) using UPS for densities $d \in \{000.1, 00.1, 0.1\}$ and a baseline using Uniform gossip (lower is better). Primaries reach a strongly consistent state faster than nodes in the baseline, with the same fraction of inconsistent nodes. The set of Secondaries is more consistent than the baseline; the higher the density, the more consistent.

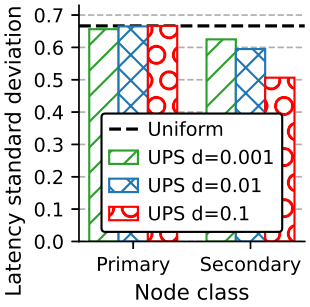


Fig. 11: Jitter as the latency standard deviation (lower is better). A higher density d reduces Secondaries' jitter.

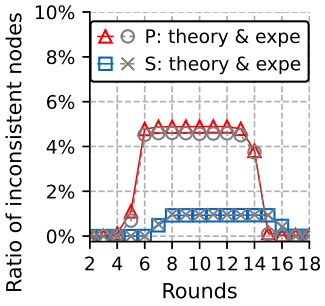


Fig. 12: Theoretical (Thms. 1 and 2) and experimental (Fig. 10) inconsistencies for $d = 0.1$ are almost identical.

equivalent but more volatile inconsistency level for Primaries over the baseline. In addition, Fig. 10b clearly shows the impact of the density on the consistency of Secondaries.

Detailed results. As expected, we observe an increase of inconsistent nodes during the temporary phase for all configurations and a return to a network-wide consistent state once every node has received every update. During the temporary phase, the inconsistency level of Primaries is equivalent to that of nodes in Uniform gossip, i.e. $\approx 4.6\%$ of Primaries are inconsistent with a higher variability for lower densities. In that phase, the inconsistency level of Secondaries is much lower than that of Primaries; we observe that the higher the density the better, i.e. the number of inconsistent nodes remains under 1.0% with a density $d = 0.1$ but reaches $\approx 4.0\%$ with a density $d = 0.001$.

The jitter (cf. §3.2) can be used to compare the consistency of different sets of nodes; a lower jitter results in a lower probability for a node to receive updates in the wrong order, i.e. a higher probability for a node to be consistent. Fig. 11 depicts the latency standard deviation, as a measure of jitter, experienced by Primaries and Secondaries using UPS and all nodes using Uniform gossip. While Primaries experience similar jitter regardless of their density (0.656 for $d = 0.001$, 0.665 for $d = 0.01$, 0.666 for $d = 0.1$)—which is on par with nodes using Uniform gossip (0.667)—Secondaries' jitter decreases visibly as the density increases. These results confirm the causality between jitter and consistency.

Once enough Primaries are infected, the dissemination rapidly reaches all Secondaries. For instance, most Secondaries can receive an update at the same time from Primaries (via Line 23 from Alg. 2) if the density is high enough. Since Secondaries receive all messages in fewer rounds than Primaries, Secondaries are more consistent as a result.

Comparison with analysis. We use Thms. 1 and 2 (cf. §4.2), we derive closed-form formulas for the predicted inconsistency levels of both classes of nodes in our evaluation.

Following the definition of our consistency metric in §6.1, we note Received_q^r the set of append operations received by q at the start of round r, and p_i^r the probability that operation append_i has been received by q at the start of r. We have

$$p_i^r = \Pr\left[\text{append}_i \in \text{Received}_q^r\right]. \tag{14}$$

($p_i^r = 0$ when $r \leq i$, since append_i is broadcast during round r.) Then, the probability that q has exactly received a prefix of length ℓ when round r starts is

$$\Pr\left[\text{Received}_q^r = (\text{append}_i)_{i\in[0..\ell-1]}\right] = \prod_{i=0}^{\ell-1} p_i^r \times \prod_{i=l}^{10}(1-p_i^r). \tag{15}$$

As a result, the probability of a node performing an inconsistent read at round r is given by

$$\begin{aligned}&\Pr\left[\text{read}_q^r \in TI_{min} \cap R_X(r)\right]\\ &\quad= 1 - \sum_{l=0}^{10} \Pr\left[\text{Received}_q^r = (\text{append}_i)_{i\in[0..\ell-1]}\right]\\ &\quad= 1 - \sum_{l=0}^{10}\left(\prod_{i=0}^{\ell-1} p_i^r \times \prod_{i=l}^{10}(1-p_i^r)\right),\end{aligned} \tag{16}$$

where read_q^r is the read operation by process q at round r, $X \in \{P, S\}$ denotes the class of node q, and TI_{min} and $R_X(r)$ are the quantities defined in §6.1.

In Eq. (16), p_i^r can be derived from Thms. 1 and 2 using

$$p_i^r = \begin{cases} 1 - X_{0,0}^{r-i}/|X| & \text{if } r \geq i \\ 0 & \text{otherwise} \end{cases}, \tag{17}$$

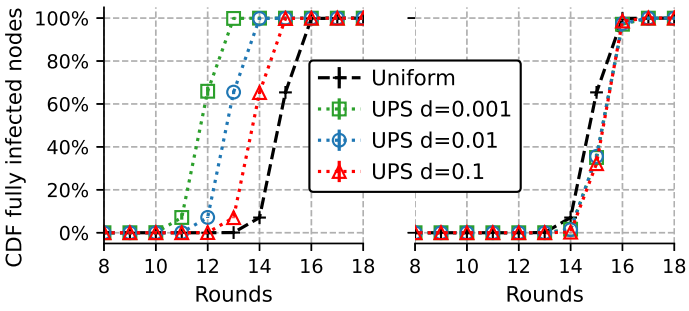
where $X \in \{P, S\}$ stands for the class of node q.

Fig. 12 plots Eq. (16) in which Eq. (17) has been injected for Primaries and Secondaries (solid colored lines) for a density $d = 0.1$, and overlays these theoretical predictions with the experimental results from Fig. 10 (dashed grayed lines). It shows theoretical results follow closely that of experiments up to a small experimental error (with an absolute discrepancy capped at 0.41% for Primaries in round 5).

As for the jitter, Fig. 11 accurately mirrors with experimental data a subset of the theoretical findings of Fig. 8. Since Fig. 11 depicts only three densities, the oscillating behavior of Primaries' jitter is not apparent.

6.5 RQ1 & RQ2 – Message Latency

Fig. 13 compares the dissemination latency of updates using Uniform gossip and UPS with different densities. This figure shows Primaries are infected faster with UPS than with Uniform gossip. Primaries receive all updates 1, 2 and 3 rounds



(a) Primary nodes. (b) Secondary nodes.

Fig. 13: Dissemination latency of all 10 updates to Primaries and Secondaries using UPS for densities $d \in \{000.1, 00.1, 0.1\}$ and a baseline using Uniform gossip (closer to the top left is better). Compared to the baseline, Primaries receive all updates 1, 2 and 3 rounds earlier for densities of $10^{-1}, 10^{-2}$ and 10^{-3}, respectively, while Secondaries receive all updates only half a round later on average.

TABLE 1: Mean number of messages per run of Uniform gossip and GPS alongside reliability and overhead vs Uniform gossip for 10 updates sent to 1 M nodes with a fanout of 10.

Configuration	#messages	Reliability	Overhead
Uniform gossip	99,995,453	0.99995	1.0000000
GPS $d = 0.001$	100,095,431	0.99995	1.0009998
GPS $d = 0.01$	100,995,395	0.99996	1.0099999
GPS $d = 0.1$	109,993,193	0.99998	1.0999819

earlier with densities of 10^{-1}, 10^{-2} and 10^{-3} respectively. Secondaries receive all updates up to 0.5 round later with GPS than with Uniform gossip, independently of the density.

As a result, Primaries and Secondaries experience similar latency gains and losses, respectively, regarding their convergence to a consistent state, as shown in Fig. 10.

Comparison with analysis. The experimental results for message latencies confirm our analysis. Fig. 13, that depicts experimental results, closely resembles Fig. 6 obtained from the compartmental analysis in § 4.2. The latency gain for Primaries of 1, 2, and 3 rounds for densities of $10^{-1}, 10^{-2}$ and 10^{-3}, respectively, corresponds to the expected gains from Eq. (1). The small latency penalty of 0.5 round for Secondaries is also in line with Eq. (2). Both behaviors confirm the validity of the asymptotic analysis in §4.1.

6.6 RQ1 & RQ2 – Network Overhead

Tab. 1 presents three metrics to assess network overheads: *(i)* the number of messages sent over the network; *(ii)* the empirical broadcast reliability; and *(iii)* the normalized network overhead, i.e. the number of messages sent by GPS compared to Uniform gossip. All measures are averaged over 25 experimental runs. The metrics show close to no variability between runs, e.g. the standard deviation ranges between 0.0001% and 0.02% of the mean. The reliability remains high, above four nines, despite the low fanout but barely differs between GPS and Uniform gossip. Considering the experienced reliability, GPS increases network overhead approximately $1.001\times$, $1.01\times$ and $1.1\times$ compared to Uniform gossip for densities of 0.001, 0.01 and 0.1, respectively.

Comparison with analysis. The overhead in terms of number of messages sent by GPS confirms Eq. (3) in §4.1.

6.7 RQ3 – Applicability to Blockchain Networks

We now replace the gossip protocol of a blockchain system with GPS to evaluate its impact on a practical application.

Motivation. Most existing permissionless blockchains rely on epidemic dissemination mechanisms—to broadcast blocks, transactions, and smart contracts—and on probabilistic consensus mechanisms such as the Nakamoto consensus [6], GHOST [59], or their many derivatives—to determine which blocks constitute the blockchain. These mechanisms fundamentally rely on eventual consistency to converge on the same view of the blockchain [60] and only provide strong consistency with high probability for old blocks in the blockchain (i.e. "blockchain's common prefix" [61]). There are no consistency guarantees for recent blocks as they may be replaced in the chain by other blocks—creating so-called blockchain forks—by attackers [62], [63] or even by well-behaving miners. Clients suffer from these inconsistencies as it directly affects their security [24]. Miners also suffer from these concurrent block creations whose frequency is exacerbated by block propagation delays [23]. These delays induce lost mining time for them, leading to a reduction of transaction throughput and ultimately profitability [20].

We thus evaluate the use of GPS in the context of a blockchain network where miners (writers) are Primaries and clients (readers) are Secondaries. We focus on the following blockchain-related research questions (BRQs):

BRQ1: Can GPS improve the consistency, thus the security, of recent blocks for clients?

BRQ2: Can GPS accelerate block propagation between miners, thus reducing lost mining time and ultimately enabling faster block creation?

GPS can also be used to disseminate transactions and smart contracts. This could for instance help improve the ordering of transactions for clients, and raise the bar for double-spend attacks in fast payments [64]. Research questions tackling this usage of GPS are deemed future work.

Gossiping in blockchains. We note that the gossip protocols used in networks such as Bitcoin and Ethereum offer little guarantees of (1) connectivity—the network graph may be partitioned, for instance by an attacker—or (2) reliability—a message may not be fully disseminated thus increasing the likelihood of inconsistencies. This lack of security-related guarantees mostly results from the lack of a proper RPS. Both networks use a protocol similar to Uniform gossip, which is built-in for Bitcoin [65] and libP2P gossipsub for Ethereum [66]. Replacing these protocols by Uniform gossip or GPS, therefore, should not degrade the security of these systems. In fact, since both Uniform gossip and GPS assume the existence of an RPS, they should actually improve the connectivity and reliability of the dissemination.

We discuss in § 7 secure RPSs, required in adversarial networks such as blockchains, and avenues to secure GPS.

Target system. We chose to simulate the Ethereum system as it is a large-scale and well-studied system with a

diverse set of available datasets documenting its behaviors. Since we cannot always find accurate approximations of all the parameters required to properly simulate the Ethereum network, we sometimes use those of Bitcoin.

As described in § 6.1, we use a scenario where 10 updates (blocks) are created (by miners). We evaluate two configurations, Uniform gossip and GPS, using the following described datasets to obtain realistic results.

Topology. We use 22,982 nodes in total, including 328 miners, as the most recent study on Ethereum shows [10, Tab. 2]. While the number of miners may seem low, note that miners may form mining pools. Since miners are Primaries when we evaluate GPS, this setup corresponds to a Primary density of 0.014. We use the associated distribution of miner computation power [10, Fig. 1], i.e. the probability for each miner to create a block, showing 90% of blocks are mined by only 14 entities and 99% by 45 entities.

Nodes are randomly interconnected following the out-degree distribution of the Bitcoin test network [9, Fig. 6a] with a mode of 8, a median of 13, and a maximum of 59.

Note that when we evaluate the impact of using the described realistic topology against an RPS-based topology, we only observe that the latter fully disseminates blocks slightly faster at the price of higher inconsistency variability.

Latencies. Inter-node block propagation latencies follow the heavy-tailed distribution of the Ethereum network [10] with a mode of 95 ms, a median of 123 ms, and a maximum of 4,938 ms. Each pair of nodes is attributed a latency from that distribution and it remains static throughout a simulation.

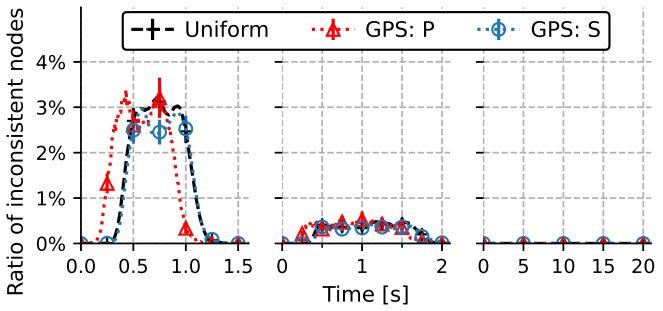
Block sizes. According to Etherscan's daily averages [67], the Ethereum block size has steadily increased to reach 43 kB at the end of 2020. Therefore, we assume miners can emit such small blocks instantaneously. Bandwidth has thus no impact on transmission delays in our simulations.

Block creation periods. We use Etherscan's daily averages from July 2015 to the end of 2020 [68] to derive the block creation periods in our experiments. The distribution exhibits low variability: 10^{th} percentile of 13.08 s, median of 14.15 s, 90^{th} percentile of 17.17 s, and maximum of 30.31 s.

Such a low period nullifies the probability of concurrent block creations, and hence forks, in Ethereum. In such a setting, GPS cannot offer a differentiated service. GPS however makes it possible to mitigate the impact of concurrent block creations, hence to use higher block frequencies. To explore this potential benefit, we artificially increase block creation frequencies in the following by either 10, 100 or 200.

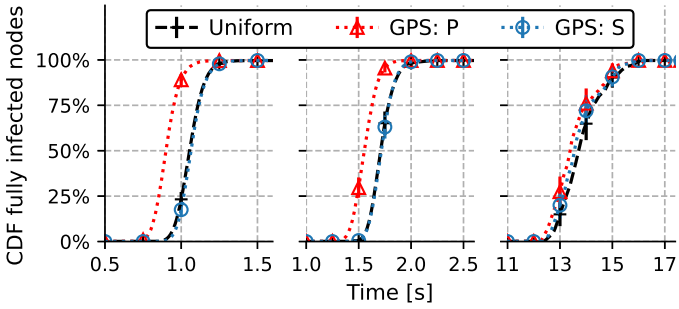
Impact of block creation periods. Fig. 14 depicts the ratio of nodes that receive updates (blocks) in the wrong order (akin to Fig. 10), while Fig. 15 depicts the latency needed for nodes to receive all blocks (akin to Fig. 13), for the three acceleration factors, 200, 100, and 10.

BRQ1 results. We first observe on Fig. 14 that increasing the frequency of block creation induces an increase in the number of inconsistent nodes, as expected. The ×10 scenario does not show any noticeable difference in inconsistency or latency between GPS and the uniform baseline. In the ×100 and ×200 experiments, Primaries (miners) experience a greater variability of the inconsistency metric compared to



(a) Frequency×200. (b) Frequency×100. (c) Frequency×10.

Fig. 14: Ratio of inconsistent nodes during dissemination for Uniform gossip and GPS when block creation is accelerated 200, 100 and 10 times. Creating blocks faster leads to more inconsistencies (Fig. 14a) while creating them slowly removes the expected benefit for Secondaries (clients) (Fig. 14c).



(a) Frequency×200. (b) Frequency×100. (c) Frequency×10.

Fig. 15: Dissemination latency of 10 blocks for Uniform gossip and GPS when block creation is accelerated 200, 100 and 10 times. Primaries (miners) receive blocks ≈160 ms faster than Secondaries (clients). This gain is hardly noticeable when updates are created slowly (Fig. 15c).

Secondaries (clients) and nodes in Uniform gossip, similarly to Fig. 10. In these last two scenarios, clients are slightly more consistent, hence secured, with GPS than miners and nodes in Uniform gossip. With GPS we see an average of 2.70% inconsistent miners and 2.60% inconsistent clients in Fig. 14a, and 0.35% inconsistent miners and 0.32% inconsistent clients in Fig. 14b. We believe the small difference is due to the very small density of miners of 0.014.

BRQ2 results. We see on Fig. 15 visible latency gains for miners compared to clients and nodes in Uniform gossip in the ×100 and ×200 scenarios. These gains allow miners to spend less time mining on a stale blockchain. Miners actually receive all blocks ≈160 ms faster on average compared to clients and nodes in Uniform gossip in all scenarios since latency gains are tied to the density of miners (cf. Fig. 13).

Results summary. We summarize this evaluation w.r.t. the two BRQs presented initially. For BRQ1, we note clients' security is slightly improved by GPS. As for BRQ2, GPS greatly benefits miners, when blocks are created at a high frequency, by enabling them to converge to a consistent state up to 13% faster (cf. Fig. 15a) than with Uniform gossip. We conclude GPS improves blockchain systems, and its impact greatly depends on the density of miners, 0.014 here.

7 Security Discussion

As is, GPS and UPS are not resilient to attackers, or Byzantine nodes [69] in general, but can be extended towards this goal.

Secure RPSs. As hinted at in §6.7, securing GPS requires the use of *secure* RPSs [52], [70] to establish *robust* random network topologies. Such topologies maintain strong network connectivity with high probability under adversarial conditions, which is instrumental in ensuring the reliability of a secure gossip algorithm. By contrast, a weak network connectivity can be exploited by attackers to launch eclipse attacks [71] against targeted nodes, which, in the case of blockchains [63], [72]–[75], can lead to miners wasting resources and clients accepting double-spent coins.

More concretely, a secure RPS limits the number of Byzantine nodes in the views it returns. This makes it possible to select a fanout that guarantees that gossip algorithms using this secure RPS are eventually reliable, i.e. they eventually deliver their messages to every node with high probability. This fanout value, logarithmic in the number of correct nodes, can be determined precisely using a known analysis [76] based on the probability of connectivity of random graphs [77].

Attackers could also sabotage the performance of GPS by lying on which nodes are Primary or Secondary. The secure RPSs used by GPS should protect against these attacks. For instance, they can require nodes to sign their membership and ensure that a node cannot belong to both classes.

BRB GPS. Securing GPS beyond the use of secure RPSs could be achieved by exploiting techniques from Byzantine reliable broadcast (BRB) [19, §9.3]. A BRB algorithm guarantees reliability and prevents *equivocation* [78], i.e. conflicting messages from a node, which can be used in blockchains to double-spend coins. BRB protocols are far more complex and costly than best-effort protocols akin to GPS. A BRB version of GPS would likely (1) use signatures to authenticate message senders, (2) form probabilistic Byzantine quorums using said signatures and samplings from the RPSs, akin to Contagion [34], and (3) be composed of additional communication phases which multiply the message complexity, e.g. Bracha's BRB [79] includes Echo and Ready phases.

However, the Byzantine model of BRB limits its uses. For instance, this model does not fit permissionless blockchains, while GPS' does (cf. § 6.7). Designing a secure, reliable broadcast for permissionless networks is an open question.

Secure UPS. Once GPS is secured, securing UPS would likely require updates to (1) match application-specific security criteria for validity and/or (2) be signed by its source. For instance, transactions in Bitcoin must be signed while blocks must have a valid proof of work but are not signed.

8 Related Work

Differentiated consistency. Hybrid consistency conditions have been extensively studied for distributed shared memory [25], [80], [81] and geo-distributed systems [26]–[29], [82]. RedBlue [26] and Fisheye [28] both propose hybrid conditions for geo-replicated systems. RedBlue proposes a consistency/latency trade-off on operations, rather than on nodes like UPS does. While with Fisheye, a strong consistency condition is achieved by topologically-close nodes and a weaker one is achieved by remote nodes. Fisheye does not consider eventual consistency, nor convergence speed.

Measuring inconsistency. Several approaches exist to evaluate the consistency of a system. Zellag and Kemme have proposed [48] to uses a dependency graph of transactions (nodes) to highlight the conflicts between them (edges) in cloud services; cycles in the graph represent inconsistencies. This metric requires a global knowledge of the system which is impractical in large-scale systems. Golab et al. proposed two metrics to measure data staleness against Lamport-atomicity [83] in key-value store traces: Δ-atomicity [45] and Γ [46]. However, we cannot use them as they do not consider the ordering of update operations.

Other approaches evaluate consistency practically by using system-based metrics such as the read-after-write latency [47] or the similarity between different cache levels [5].

Finally, measuring inconsistencies becomes unnecessary when using CRDTs [13] since inconsistencies due to operation ordering are impossible with these data types. CRDTs naturally lead to eventual consistency with no extra effort.

Biased gossip protocols. Many approaches have explored the use of bias in gossip protocols to accommodate the inherent heterogeneity of systems. Yet, none has tackled heterogeneous consistency requirements before GPS.

For example, Directional gossip [84] improves overall reliability by favoring weakly connected nodes. The work in [85] reduces broadcasts' message complexity by differentiating its quality of services between good nodes and bad nodes, as defined by the user. Messages are first rapidly broadcast to good nodes using a reactive gossip protocol, while a slower but cheaper periodic push gossip is used to reach bad nodes. The periodic gossip reduces the number of messages but increases the delivery latency for bad nodes. Gravitational gossip [86] offers differential reliability balancing communication workload between them according to their capacities. Nodes receive a fraction r, a user-defined quality rating, of the messages before they time out. Gravitational gossip hence offers a cost/reliability trade-off. Hierarchical gossip [87] greatly reduces message complexity by leveraging the physical network topology. Nodes favor gossip targets that are close in the network hierarchy, resulting in a slight increase in delivery latency since message flooding is avoided. Perigee [88] accelerates dissemination by learning and adapting each node's neighborhood based on their round-trip latencies with other nodes. HEAP [89] reduces video streaming delivery latency by adapting each node's fanout to account for heterogeneous bandwidth capabilities.

In the context of blockchain networks, Marçal et al. [90] reduce the transaction message complexity of Bitcoin miners, by reducing the size of their neighborhood, and their transaction latencies, by orienting new transactions towards miners first. This approach exclusively favors miners without improving clients' experience, unlike GPS as deployed in §6.7. Similarly, gossiping tailored for Hyperledger Fabric [91] also reduces message complexity and overall transaction latencies for miners, but does not consider clients either.

Finally, ecBroadcast [17] and EpTO [16] achieve probabilistic total order and can thus be used to implement probabilistic strong consistency conditions at the cost of higher latency, and a higher message complexity for EpTO.

9 Conclusion

Update-query consistency with Primaries and Secondaries (UPS), with its underlying gossip protocol Gossip Primary-Secondary (GPS), provides a novel eventual consistency mechanism offering differentiated data consistency and delivery latency properties. Primary nodes deliver updates faster at the cost of a small consistency penalty, Secondary nodes experience stronger consistency with a slightly higher latency. Both node classes reach a consistent state with high probability once all updates' dissemination is completed.

Our formal analyses and evaluations on a one-million-node network both highlighted the impact of the density (fraction) of Primary nodes on the trade-off between consistency and latency experienced by all nodes. A low density favors a fast dissemination to Primary nodes, while a high density favors higher consistency for Secondary nodes. We further evaluated GPS in a simulated blockchain network showing that it improves miners' efficiency and clients' security, slightly, when blocks are created at a high frequency.

Acknowledgments

The authors wish to thank the reviewers of IEEE TPDS for their valuable feedback and Lucianna Kiffer for generously sharing her datasets [10]. This work was partially funded by the French ANR grants O'Browser (ANR-16-CE25-0005-03) and ByBloS (ANR-20-CE25-0002-01).

References

[1] D. Frey, A. Mostefaoui, M. Perrin, P.-L. Roman, and F. Taïani, "Speed for the elite, consistency for the masses: differentiating eventual consistency in large-scale distributed systems," in *SRDS*, 2016.
[2] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *EuroSys*, 2015.
[3] Y. Feng, Z. Liu, Y. Zhao, T. Jin, Y. Wu, Y. Zhang, J. Cheng, C. Li, and T. Guan, "Scaling Large Production Clusters with Partitioned Synchronization," in *USENIX ATC*, 2021.
[4] "Google Data Centers Location," https://www.google.com/about/datacenters/locations/, accessed 2021-10-01.
[5] H. Lu, K. Veeraraghavan, P. Ajoux, J. Hunt, Y. J. Song, W. Tobagus, S. Kumar, and W. Lloyd, "Existential Consistency: Measuring and Understanding Consistency at Facebook," in *SOSP*, 2015.
[6] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.
[7] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," *Ethereum Project Yellow Paper*, 2014.
[8] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and Probabilistic Leaderless BFT Consensus through Metastability," arXiv:1906.08936v2, 2020.
[9] S. Delgado-Segura, S. Bakshi, C. Pérez-Solà, J. Litton, A. Pachulski, A. Miller, and B. Bhattacharjee, "TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions," in *FC*, 2019.
[10] L. Kiffer, A. Salman, D. Levin, A. Mislove, and C. Nita-Rotaru, "Under the Hood of the Ethereum Gossip Protocol," in *FC*, 2021.
[11] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, 2002.
[12] W. Vogels, "Eventually Consistent," *CACM*, 2009.
[13] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-Free Replicated Data Types," in *SSS*, 2011.
[14] V. Enes, P. S. Almeida, C. Baquero, and J. Leitão, "Efficient Synchronization of State-Based CRDTs," in *ICDE*, 2019.
[15] M. Perrin, A. Mostefaoui, and C. Jard, "Update Consistency for Wait-free Concurrent Objects," in *IPDPS*, 2015.
[16] M. Matos, H. Mercier, P. Felber, R. Oliveira, and J. Pereira, "EpTO: An Epidemic Total Order Algorithm for Large-Scale Distributed Systems," in *Middleware*, 2015.
[17] R. Baldoni, R. Guerraoui, R. R. Levy, V. Quéma, and S. Tucci Piergiovanni, "Unconscious Eventual Consistency with Gossips," in *SSS*, 2006.
[18] A. Sousa, J. Pereira, F. Moura, and R. Oliveira, "Optimistic Total Order in Wide Area Networks," in *SRDS*, 2002.
[19] M. Raynal, *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer, 2018.
[20] T. Cao, J. Decouchant, J. Yu, and P. Esteves-Verissimo, "Characterizing the Impact of Network Delay on Bitcoin Mining," in *SRDS*, 2021.
[21] "BloXroute," https://bloxroute.com/, accessed 2021-10-01.
[22] "FIBRE Fast Internet Bitcoin Relay Engine," https://bitcoinfibre.org/, accessed 2021-10-01.
[23] C. Decker and R. Wattenhofer, "Information Propagation in the Bitcoin Network," in *P2P*, 2013.
[24] L. Kiffer, R. Rajaraman, and a. shelat, "A Better Method to Analyze Blockchain Consistency," in *CCS*, 2018.
[25] R. Friedman, "Implementing Hybrid Consistency with High-Level Synchronization Operations," *Dist. Comp.*, 1995.
[26] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues, "Making Geo-Replicated Systems Fast as Possible, Consistent when Necessary," in *OSDI*, 2012.
[27] C. Xie, C. Su, M. Kapritsos, Y. Wang, N. Yaghmazadeh, L. Alvisi, and P. Mahajan, "Salt: Combining ACID and BASE in a Distributed Database," in *OSDI*, 2014.
[28] R. Friedman, M. Raynal, and F. Taïani, "Fisheye Consistency: Keeping Data in Synch in a Georeplicated World," in *NETYS*, 2015.
[29] R. Guerraoui, M. Pavlovic, and D.-A. Seredinschi, "Incremental Consistency Guarantees for Replicated Objects," in *OSDI*, 2016.
[30] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic Algorithms for Replicated Database Maintenance," in *PODC*, 1987.
[31] P. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, "Lightweight Probabilistic Broadcast," *ACM TOCS*, 2003.
[32] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on BFT consensus," arXiv:1807.04938v3, 2019.
[33] P. Li, G. Wang, X. Chen, F. Long, and W. Xu, "Gosig: A Scalable and High-Performance Byzantine Consensus for Consortium Blockchains," in *SoCC*, 2020.
[34] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovic, and D.-A. Seredinschi, "Scalable Byzantine Reliable Broadcast," in *DISC*, 2019.
[35] M. P. Herlihy and J. M. Wing, "Linearizability: A Correctness Condition for Concurrent Objects," *ACM TOPLAS*, 1990.
[36] H. Attiya and J. L. Welch, "Sequential Consistency versus Linearizability," *ACM TOCS*, 1994.
[37] P. Bailis, A. Ghodsi, J. M. Hellerstein, and I. Stoica, "Bolt-on Causal Consistency," in *SIGMOD*, 2013.
[38] S. Almeida, J. Leitão, and L. Rodrigues, "ChainReaction: a Causal+ Consistent Datastore based on Chain Replication," in *EuroSys*, 2013.
[39] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS," in *SOSP*, 2011.
[40] T. D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *JACM*, 1996.
[41] D. Dolev, C. Dwork, and L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consensus," *JACM*, 1987.
[42] S. Dubois, R. Guerraoui, P. Kuznetsov, F. Petit, and P. Sens, "The Weakest Failure Detector for Eventual Consistency," *DC*, 2019.
[43] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *CACM*, 1978.
[44] S. Burckhardt, A. Gotsman, H. Yang, and M. Zawirski, "Replicated Data Types: Specification, Verification, Optimality," in *POPL*, 2014.
[45] W. Golab, X. Li, and M. A. Shah, "Analyzing Consistency Properties for Fun and Profit," in *PODC*, 2011.
[46] W. Golab, M. R. Rahman, A. AuYoung, K. Keeton, and I. Gupta, "Client-Centric Benchmarking of Eventual Consistency for Cloud Storage Systems," in *ICDCS*, 2014.
[47] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi, "YCSB++: Benchmarking and Performance Debugging Advanced Features in Scalable Table Stores," in *SoCC*, 2011.
[48] K. Zellag and B. Kemme, "How Consistent is Your Cloud Application?" in *SoCC*, 2012.
[49] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh, "Probabilistic Reliable Dissemination in Large-Scale Systems," *IEEE TPDS*, 2003.

[50] F. Taïani, S. Lin, and G. S. Blair, "GossipKit: A Unified Component-Framework for Gossip," *IEEE TSE*, 2014.
[51] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. Van Steen, "Gossip-Based Peer Sampling," *ACM TOCS*, 2007.
[52] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer, "Brahms: Byzantine resilient random membership sampling," *Compututer Networks*, 2009.
[53] V. Capasso, *Mathematical Structures of Epidemic Systems*, ser. Lecture Notes in Biomathematics. Springer, 2008.
[54] L. Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," *IEEE TOC*, 1979.
[55] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *P2P*, 2009.
[56] "GPS/UPS code repository," https://gitlab.inria.fr/WIDE/gps.
[57] S. Voulgaris, D. Gavidia, and M. van Steen, "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays," *JNSM*, 2005.
[58] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, "Peer-to-Peer Membership Management for Gossip-Based Protocols," *IEEE TOC*, 2003.
[59] Y. Sompolinsky and A. Zohar, "Secure High-Rate Transaction Processing in Bitcoin," in *FC*, 2015.
[60] E. Anceaume, A. Del Pozzo, R. Ludinard, M. Potop-Butucaru, and S. Tucci-Piergiovanni, "Blockchain Abstract Data Type," in *SPAA*, 2019.
[61] J. A. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin Backbone Protocol: Analysis and Applications," in *EUROCRYPT*, 2015.
[62] I. Eyal and E. G. Sirer, "Majority is not Enough: Bitcoin Mining is Vulnerable," in *FC*, 2014.
[63] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack," in *EuroSP*, 2016.
[64] G. O. Karame, E. Androulaki, and S. Capkun, "Double-Spending Fast Payments in Bitcoin," in *CCS*, 2012.
[65] "Bitcoin documentation – P2P network," https://developer.bitcoin.org/devguide/p2p_network.html, accessed 2022-08-22.
[66] "Ethereum documentation – Gossip," https://ethereum.org/en/developers/docs/networking-layer/#gossip, accessed 2022-08-22.
[67] "Etherscan - Ethereum Average Block Size Chart," https://etherscan.io/chart/blocksize, accessed 2021-10-01.
[68] "Etherscan - Ethereum Average Block Time Chart," https://etherscan.io/chart/blocktime, accessed 2021-10-01.
[69] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM TOPLAS*, 1982.
[70] A. Auvolat, Y.-D. Bromberg, D. Frey, and F. Taïani, "BASALT: A Rock-Solid Foundation for Epidemic Consensus Algorithms in Very Large, Very Open Networks," arXiv:2102.04063, 2021.
[71] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach, "Eclipse Attacks on Overlay Networks: Threats and Defenses," in *INFOCOM*, 2006.
[72] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse Attacks on Bitcoin's Peer-to-Peer Network," in *USENIX Security*, 2015.
[73] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies," in *SP*, 2017.
[74] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network," in *SP*, 2020.
[75] M. Saad, S. Chen, and D. Mohaisen, "SyncAttack: Double-Spending in Bitcoin Without Mining Power," in *CCS*, 2021.
[76] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié, "Epidemic Information Dissemination in Distributed Systems," *IEEE Computer*, 2004.
[77] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, 1960.
[78] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested Append-Only Memory: Making Adversaries Stick to Their Word," in *SOSP*, 2007.
[79] G. Bracha, "Asynchronous Byzantine agreement protocols," *Information and Computation*, 1987.
[80] H. Attiya and R. Friedman, "Limitations of fast consistency conditions for distributed shared memories," *Inf. Proc. Letters*, 1996.
[81] P. Keleher, A. L. Cox, and W. Zwaenepoel, "Lazy Release Consistency for Software Distributed Shared Memory," in *ISCA*, 1992.
[82] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh, "Consistency-Based Service Level Agreements for Cloud Storage," in *SOSP*, 2013.
[83] L. Lamport, "On interprocess communication," *DC*, 1986.
[84] M.-J. Lin and K. Marzullo, "Directional Gossip: Gossip in a Wide Area Network," in *EDCC*, 1999.
[85] N. Carvalho, J. Pereira, R. Oliveira, and L. Rodrigues, "Emergent Structure in Unstructured Epidemic Multicast," in *DSN*, 2007.
[86] K. Hopkinson, K. Jenkins, K. Birman, J. Thorp, G. Toussaint, and M. Parashar, "Adaptive Gravitational Gossip: A Gossip-Based Communication Protocol with User-Selectable Rates," *IEEE TPDS*, 2009.
[87] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh, "Efficient and Adaptive Epidemic-Style Protocols for Reliable and Scalable Multicast," *IEEE TPDS*, 2006.
[88] Y. Mao, S. Deb, S. B. Venkatakrishnan, S. Kannan, and K. Srinivasan, "Perigee: Efficient Peer-to-Peer Network Design for Blockchains," in *PODC*, 2020.
[89] D. Frey, R. Guerraoui, A.-M. Kermarrec, B. Koldehofe, M. Mogensen, M. Monod, and V. Quéma, "Heterogeneous Gossip," in *Middleware*, 2009.
[90] J. Marçal, L. Rodrigues, and M. Matos, "Adaptive Information Dissemination in the Bitcoin Network," in *SAC*, 2019.
[91] N. Berendea, H. Mercier, E. Onica, and E. Rivière, "Fair and Efficient Gossip in Hyperledger Fabric," in *ICDCS*, 2020.

Davide Frey has been a researcher at Inria Rennes Bretagne-Atlantique since 2010. He received his PhD from Politecnico di Milano in Italy in 2006; he then worked as a post-doctoral researcher both at Washington University in St. Louis (MO), and at Inria Rennes before being recruited as a permanent researcher in 2010. His research interests focus on the systemic aspects of large-scale distributed systems.

Achour Mostefaoui received the MSc degree is computer science in 1991, and the PhD degree from the University of Rennes in 1994. He is a professor of computer science at the University of Nantes, France. He is the head of a master's diploma in computer science, University of Nantes, and is a co-head of the GDD research team within the LINA Lab.

Matthieu Perrin is an associate professor at the University of Nantes. His scientific interests focus on the wide area of distributed computing, and in particular algorithms in shared memory and message-passing distributed systems, including modeling of weakly consistent shared objects and message broadcast primitives.

Pierre-Louis Roman is a postdoctoral researcher in computer science at the École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. He holds a PhD from the University of Rennes 1 obtained in 2018 for his work on decentralized systems. His research interests revolve around distributed systems with a particular focus on secured and scalable systems such as distributed ledgers and cryptocurrencies.

François Taïani is a Professor in Distributed Computer Systems at the University of Rennes 1, and at IRISA/Inria in Rennes, France, where he heads the WIDE Inria research team. His main research interest lies in the scalability and programmability of complex distributed systems (e.g. overlays, on-line social networks, data centers), with a focus on resilience, concurrency, and self-organization.

APPENDIX
CHANGELOG

A preliminary version of this article titled "Speed for the elite, consistency for the masses: differentiating eventual consistency in large-scale distributed system" appears in the proceedings of SRDS 2016 [1]. In addition to general improvements, the current article notably adds:

- A compartment-based discrete analysis of GPS in §4.2 that utilizes models originally developed in epidemiology, to help us study the consistency of GPS, and complements the asymptotic analysis in §4.1 that studies latency and message complexity;
- A comparison between analytical and experimental results on consistency in § 6.4 obtained thanks to the new compartment-based discrete analysis of GPS;
- Precise numbers on the reliability and overhead of GPS in our evaluation in §6.6 and Tab. 1;
- An evaluation of GPS in §6.7 on a simulated blockchain network that closely matches the characteristics of Ethereum's network;
- A security discussion in §7 that presents possible improvements towards securing GPS and UPS.

APPENDIX
PROOFS FOR THE COMPARTMENT-BASED DISCRETE ANALYSIS

We here provide the proofs for Thm. 1 and Thm. 2 presented in the compartment-based discrete analysis described in §4.2. For convenience, we repeat the two theorems and Fig. 5.

Theorem 1. *The evolution of Primary nodes follows the formulae*

$$P_{0,0}^{r+1} = a(P_{0,0}^r, P_{0,0}^{r-1}), \tag{18}$$
$$P_{1,1}^{r+1} = b(P_{0,0}^r, P_{0,0}^{r-1}, P_{1,1}^r), \tag{19}$$

where

$$a(x,y) = x\left(1-\frac{f}{|P|}\right)^{y-x}, \text{ and}$$
$$b(x,y,z) = \left(z - \frac{f\times x\,(x-y)}{|P|-f}\right)\left(1-\frac{f}{|P|}\right)^{y-x}.$$

Proof. Starting with $P_{0,0}$, the Primaries that remain in the compartment $P_{0,0}$ at the start of round $r+1$ are those that did not receive any message during round r:

$$P_{0,0}^{r+1} = P_{0,0}^r \times P_P^r[receive = 0]. \tag{20}$$

The messages received during round r by Primaries are those sent during round $r-1$ by the nodes in $P_{1,0}^{r-1}$ and $P_{2+,0}^{r-1}$. We therefore have

$$P_P^r[receive = 0] = (1-\beta_P)^{(P_{1,0}^{r-1}+P_{2+,0}^{r-1})}, \tag{21}$$

where $\beta_P = \frac{f}{|P|}$ is the infection rate of Primaries.

Let us note $\Delta^r(P_{i,j})$ the population change of nodes in compartment $P_{i,j}$ during round r, i.e.

$$\Delta^r(P_{i,j}) = P_{i,j}^{r+1} - P_{i,j}^r. \tag{22}$$

Because $P_{1,0}$ and $P_{2+,0}$ are systematically emptied between two rounds, $P_{1,0}^{r-1} + P_{2+,0}^{r-1}$ can be also written as

$$P_{1,0}^{r-1} + P_{2+,0}^{r-1} = -\Delta^{r-1}(P_{0,0}) = P_{0,0}^{r-1} - P_{0,0}^r, \tag{23}$$

which once injected in Eq. (21) yields

$$P_P^r[\text{receive} = 0] = (1-\beta_P)^{P_{0,0}^{r-1}-P_{0,0}^r}. \tag{24}$$

Coming back to Eq. (20), Eq. (24) allows us to write

$$P_{0,0}^{r+1} = P_{0,0}^r \times (1-\beta_P)^{P_{0,0}^{r-1}-P_{0,0}^r} \tag{25}$$

which proves Eq. (18) of the theorem.

The case of $P_{1,1}$ is slightly more involved. The compartment $P_{1,1}$ loses nodes that have received some message(s) in round r (and moved to $P_{2+,1}$), but also gains all the nodes originally in $P_{1,0}$ that have sent one message. We thus have

$$\Delta^r(P_{1,1}) = P_{1,0}^r - P_{2+,1}^r. \tag{26}$$

$P_{1,0}^r$ corresponds to the nodes that were originally in $P_{0,0}$ at the start of round r, and exactly received one message during round r. $P_{1,0}^{r-1} + P_{2+,0}^{r-1} = -\Delta^{r-1}(P_{0,0})$ messages were gossiped to Primaries in round $r-1$, with a probability of reception by individual nodes of β_P. Applying a binomial law gives a probability of receiving exactly one message as

$$\begin{aligned} P_P^r[\text{receive} = 1] =& \binom{1}{-\Delta^{r-1}(P_{0,0})}\beta_P(1-\beta_P)^{-\Delta^{r-1}(P_{0,0})-1} \\ =& -\Delta^{r-1}(P_{0,0})\beta_P(1-\beta_P)^{-\Delta^{r-1}(P_{0,0})-1} \\ =& -\Delta^{r-1}(P_{0,0})\frac{\beta_P}{1-\beta_P}\gamma_p(P_{0,0}^r, P_{0,0}^{r-1}), \end{aligned} \tag{27}$$

where $\gamma_P()$ is the helper function

$$\gamma_P(x,y) = (1-\beta_P)^{(y-x)}. \tag{28}$$

This yields

$$\begin{aligned} P_{1,0}^r &= P_{0,0}^r \times P_P^r[\text{receive} = 1] \\ &= P_{0,0}^r \times -\Delta^{r-1}(P_{0,0})\frac{\beta_P}{1-\beta_P}\gamma_p(P_{0,0}^r, P_{0,0}^{r-1}). \end{aligned} \tag{29}$$

$P_{2+,1}^r$ counts the nodes from $P_{1,1}^r$ that have received at least one message during round r:

$$\begin{aligned} P_{2+,1}^r =& P_{1,1}^r \times (1 - P_P^r[\text{receive} = 0]), \\ =& P_{1,1}^r \times \left(1-\gamma_p(P_{0,0}^r, P_{0,0}^{r-1})\right). \end{aligned} \tag{30}$$

Substituting Eqs. (29) and (30) in Eq. (26), we get

$$\begin{aligned} \Delta^r(P_{1,1}) = P_{0,0}^r \times -\Delta^{r-1}(P_{0,0})\frac{\beta_P}{1-\beta_P}\gamma_p(P_{0,0}^r, P_{0,0}^{r-1}) \\ - P_{1,1}^r \times \left(1-\gamma_p(P_{0,0}^r, P_{0,0}^{r-1})\right). \end{aligned} \tag{31}$$

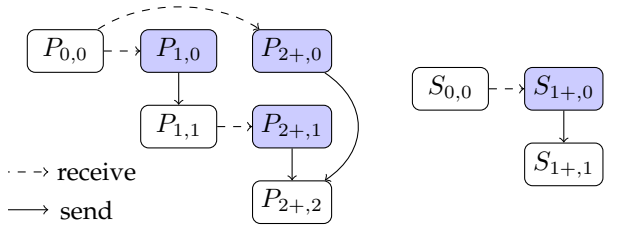
Replacing $\Delta^r(P_{1,1})$ and $\Delta^{r-1}(P_{0,0})$ by their definition from Eq. (22), and solving to obtain $P_{1,1}^{r+1}$ yields Eq. (19) of the theorem and concludes the proof. □

Theorem 2. *The evolution of Secondary nodes follows the formula*

$$S_{0,0}^{r+1} = c\left(P_{0,0}^r, P_{0,0}^{r-1}, P_{1,1}^r, P_{1,1}^{r-1}, S_{0,0}^r, S_{0,0}^{r-1}\right) \tag{32}$$

where

$$c(x,y,z,u,v,w) = v \times \left(1-\frac{f}{|S|}\right)^{y+u+w-x-z-v} \tag{33}$$



(a) Primary nodes. (b) Secondary nodes.

Fig. 5: The compartments used in our discrete analysis

Proof. The reasoning for $S_{0,0}$ follows that of $P_{0,0}$. As for $P_{0,0}$, we will assume for simplicity that a Secondary node might select itself when gossiping a message. As a result, we do not need to distinguish between sender and receiver in terms of probability of receiving a message, or to distinguish between broadcasts from Primaries to Secondaries on one hand, from broadcasts within Secondaries on the other.

In the following we will note $\beta_S = \frac{f}{|S|}$ the probability that a Secondary node receives a given broadcast (whether originating from a Primary or Secondary node).

During a round r, messages sent to Secondaries originate from Primaries that received a second message during round $r-1$ (making up $P_{2+,1}^{r-1} + P_{2+,0}^{r-1}$ messages), and from Secondaries that received their first message during round $r-1$ (i.e. $S_{1+,0}^{r-1}$ messages). These messages are received during round r and determine the size of the Secondary compartments $S_{0,0}$ and $S_{1+,1}$ at the start of round $r+1$. (Recall that $S_{0,0} + S_{1+,1} = |S|$ when a round starts.)

The nodes remaining in the compartment $S_{0,0}$ are those that receive none of the $P_{2+,1}^{r-1} + P_{2+,0}^{r-1} + S_{1+,0}^{r-1}$ messages sent during round $r-1$:

$$S_{0,0}^{r+1} = S_{0,0}^{r} \times (1-\beta_S)^{P_{2+,1}^{r-1}+P_{2+,0}^{r-1}+S_{1+,0}^{r-1}} \tag{34}$$

Nodes in $S_{1+,0}^{r-1}$ are those leaving $S_{0,0}^{r-1}$

$$S_{1+,0}^{r-1} = -\Delta^{r-1}(S_{0,0}) = S_{0,0}^{r-1} - S_{0,0}^{r}. \tag{35}$$

$P_{2+,1}^{r-1} + P_{2+,0}^{r-1}$ corresponds to the nodes entering $P_{2+,2}$

$$P_{2+,1}^{r-1} + P_{2+,0}^{r-1} = \Delta^{r-1}(P_{2+,2}). \tag{36}$$

Since $P_{0,0}^{r} + P_{1,1}^{r} + P_{2+,2}^{r}$ is constant (equal to $|P|$), we have $\Delta^{r-1}(P_{2+,2}) + \Delta^{r-1}(P_{1,1}) + \Delta^{r-1}(P_{0,0}) = 0$ and hence

$$\begin{aligned}\Delta^{r-1}(P_{2+,2}) &= -\Delta^{r-1}(P_{1,1}) - \Delta^{r-1}(P_{0,0}) \\ &= P_{0,0}^{r-1} - P_{0,0}^{r} + P_{1,1}^{r-1} - P_{1,1}^{r}.\end{aligned} \tag{37}$$

Replacing Eqs. (35) to (37) in Eq. (34), we obtain

$$\begin{aligned}S_{0,0}^{r+1} &= S_{0,0}^{r}\,(1-\beta_S)^{P_{0,0}^{r-1}+P_{1,1}^{r-1}+S_{0,0}^{r-1}-P_{0,0}^{r}-S_{0,0}^{r}-P_{1,1}^{r}} \\ &= S_{0,0}^{r}\left(1-\frac{f}{|S|}\right)^{P_{0,0}^{r-1}+P_{1,1}^{r-1}+S_{0,0}^{r-1}-P_{0,0}^{r}-S_{0,0}^{r}-P_{1,1}^{r}} \\ &= c\Big(P_{0,0}^{r}, P_{0,0}^{r-1}, P_{1,1}^{r}, P_{1,1}^{r-1}, S_{0,0}^{r}, S_{0,0}^{r-1}\Big)\end{aligned} \tag{38}$$

with $c(..)$ defined as above. □