LaBRI

université de BORDEAUX

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEURE

DE L'UNIVERSITÉ DE BORDEAUX

ECOLE DOCTORALE MATHÉMATIQUES ET INFORMATIQUE

SPÉCIALITÉ INFORMATIQUE

Par **Augusta MUKAM**

Échantillonnage de pairs tolérant aux fautes byzantines pour les systèmes distribués à grande échelle

Sous la direction de : **Laurent RÉVEILLÈRE**

Membres du jury :

M. Laurent RÉVEILLÈRE	Professeur des universités, Université de Bordeaux	Directeur de thèse
Mme Sonia BEN MOKHTAR .	Directrice de recherche, CNRS (Villeurbanne)	Rapporteure
M. Yérom-David BROMBERG	Professeur, Université de Rennes (Rennes)	Rapporteur
Mme Patricia THÉBAULT ...	Professeur, Université de Bordeaux (Talence)	Présidente du jury
M. Léo MENDIBOURE	Maître de Conférences, Université de Pau et des Pays de l'Adour (Pau)	Examinateur

Membres invités :

M. Joachim BRUNEAU-QUEYREIX	Maître de Conférences, Bordeaux INP	Co-encadrant

Échantillonnage de pairs tolérant aux fautes byzantines pour les systèmes distribués à grande échelle

Résumé : Les services d'échantillonnage de pairs constituent des primitives fondamentales des systèmes distribués à grande échelle. Ils fournissent à chaque nœud une vue continuellement actualisée et quasi uniforme de la population du système. Les implémentations fondées sur des protocoles gossip sont particulièrement adaptées à cette tâche, grâce à leur passage à l'échelle, leurs propriétés d'auto-guérison et leur robustesse face à la dynamique des nœuds. Cependant, dans des environnements ouverts et adversariaux tels que les réseaux blockchain publics, des nœuds byzantins peuvent biaiser systématiquement les échanges de vues, corrompant la distribution d'échantillonnage et permettant des attaques telles que les attaques d'éclipse, les dénis de service ou la manipulation de protocoles de consensus.

Cette thèse aborde le problème de l'échantillonnage de pairs tolérant aux fautes byzantines à travers trois contributions complémentaires. Premièrement, nous présentons Aupe, un protocole d'échantillonnage de pairs combinant un Set Cleaner de suivi des fréquences avec un mécanisme de débiaisage collaboratif reposant sur des environnements d'exécution de confiance. Des évaluations expérimentales montrent que le protocole Aupe maintient une tolérance aux fautes optimale en présence de jusqu'à 26 % de nœuds adversariaux. Deuxièmement, nous évaluons de manière systématique des estimateurs de fréquences fondés sur des structures de données compactes appelées sketches, notamment Count-Min Sketch, BitMatcher et plusieurs variantes, dans le contexte de l'échantillonnage de pairs adversarial. Nous proposons de nouvelles métriques d'évaluation adaptées aux objectifs de l'échantillonnage de pairs, au-delà des mesures agrégées classiques. Troisièmement, nous intégrons BitMatcher dans Aupe, remplaçant le comptage exact des fréquences par une structure adaptative économe en mémoire. Nous étudions la problématique du vieillissement, des sketches lorsqu'ils sont amenés à vivre indéfiniment, comme dans le cadre du peer sampling, ainsi que la problématique de l'agrégation de sketches dans le cadre du débiasage collaboratif. Nous évaluons le système obtenu dans des scénarios d'injection adversariale.

Ces contributions font progresser l'état de l'art en matière d'échantillonnage de pairs résilient, scalable et économe en ressources, en montrant que l'échantillonnage uniforme et l'estimation statistique compacte d'occurrences peuvent être efficacement combinés dans des systèmes distribués pratiques.

Mots-clés : Systèmes distribués, échantillonnage de pairs, gossip, tolérance aux fautes byzantines, estimation de fréquences, structures de données probabilistes, résilience aux attaques.

Byzantine-Resilient Peer Sampling for Large-Scale Distributed Systems

Abstract: Peer sampling services are foundational primitives in large-scale distributed systems, providing each node with a continuously refreshed, near-uniform view of the system membership. Gossip-based implementations are particularly well-suited to this task owing to their scalability, self-healing properties, and robustness to churn. However, in open, adversarial environments such as public blockchain networks, Byzantine nodes can systematically bias view exchanges, corrupting the sampling distribution and enabling attacks such as eclipse attacks, denial-of-service attacks, and manipulation of consensus protocols.

This thesis addresses the problem of Byzantine-resilient gossip-based peer sampling through three complementary contributions. First, we present AUPE, a Byzantine fault-tolerant peer sampling protocol that combines a frequency-tracking Set Cleaner with a collaborative debiasing mechanism leveraging trusted execution environments. Experimental evaluations show that AUPE maintains optimal fault tolerance in the presence of up to 26% adversarial nodes. Second, we conduct a systematic evaluation of sketch-based frequency estimators, including Count-Min Sketch, BitMatcher, and several variants, in the context of adversarial peer sampling, and propose evaluation metrics tailored to peer sampling objectives that go beyond standard aggregate measures. Third, we integrate BitMatcher into AUPE, replacing exact frequency counting with a memory-efficient adaptive sketch. We study the issue of sketch decay or aging when sketches are required to live indefinitely, as in peer sampling services, as well as sketch aggregation in the context of collaborative debiasing. We evaluate the resulting system under adversarial conditions.

Together, these contributions advance the state of the art in resilient, scalable, and resource-efficient peer sampling, demonstrating that uniform sampling and compact statistical frequency estimation can be effectively combined in practical distributed systems.
Keywords: distributed systems, peer sampling, gossip protocols, Byzantine fault tolerance, frequency estimation, sketches, adversarial resilience

Unité de recherche
UMR 5800 Université de Bordeaux, 33000 Bordeaux, France.

Remerciements Au terme de cette thèse, je souhaite exprimer ma profonde gratitude à toutes les personnes qui ont, de près ou de loin, contribué à son aboutissement.

Je tiens tout d'abord à remercier très sincèrement mes encadrants de thèse Laurent Réveillère et Joachim Bruneau-Queyreix pour m'avoir offert cette opportunité de thèse. Merci pour votre confiance, votre encadrement et votre exigence scientifique tout au long de ces années. Votre expertise, votre rigueur et votre disponibilité ont été déterminantes dans la conduite de mes travaux. Comme je le souhaitais en démarrant cette aventure, et grâce à vous, j'ai pu approfondir mes connaissances et développer des compétences valorisantes pour la suite de ma carrière professionnelle.

J'adresse mes remerciements aux membres du jury pour l'honneur qu'ils m'ont fait en acceptant d'évaluer ce travail. Je remercie Sonia Ben Mokhtar et Yérom-David Bromberg, qui ont accepté la charge de rapporteur de cette thèse. Je remercie également Patricia Thébault et Léo Mendiboure d'avoir accepté d'examiner mon travail. Merci pour l'intérêt que vous avez porté à mon travail.

Je suis reconnaissante envers l'ensemble de mes collègues du laboratoire Bordelais de Recherche en Informatique (LaBRI), les doctorants de l'Afodib, les membres de l'équipe Progress pour l'environnement stimulant qu'ils ont contribué à créer, les discussions enrichissantes et l'entraide au quotidien. Je remercie infiniment mes camarades du bureau 219 : Lamyae pour ses câlins réconfortants, Nicolas H. pour ses astuces et tutoriels, Oumaima pour ses délicates attentions, Tom pour ses histoires fascinantes, Lucas pour ses blagues toujours inattendues ; ainsi que mes camarades du bureau 125 : Asma, Nolan, Farah ; et tous les autres stagiaires et doctorants, actuels et passés, de l'équipe. Je pense également à Thomas, Romain, Jean-Rémy, Christophe, Xavier, Stéphane et tous les permanents de l'équipe pour leurs précieux conseils.

Un grand merci à mes aînés qui ont soutenu avant moi, notamment Sarah pour sa bienveillance, ainsi que ceux avec qui j'ai soutenu dans la même période : Gustave pour ses coups de main, Meghna pour ses recommandations et Yanis pour sa gaieté. Je remercie mes compatriotes du labo, Françoise pour nos discussions réconfortantes, Sthyve et Voltaire. Merci de cette proximité qui m'a permis d'avoir un bout de chez moi au travail. Je remercie les membres de mes équipes d'enseignement, à l'Enseirb et à l'UBx, notamment les permanents et les Ater, pour leur aide et remplacements d'urgence. Les discussions que nous avons eues, tant scientifiques que personnelles, ont rendu ces années de thèse bien plus agréables. Je remercie également Cathy, Claire, Safia, Auriane, et tout le personnel administratif et technique du laboratoire, dont l'efficacité et la disponibilité facilitent considérablement le travail de recherche. Merci de m'avoir soutenu et motivé depuis la phase de candidature à cette thèse. Je remercie également les

membres de mon comité de suivi annuel, et tous les collègues extérieurs pour les échanges scientifiques fructueux et les discussions passionnantes que nous avons menés ensemble.

Un grand merci à mes ainés de Polytechnique Yaoundé qui ont toujours su me motiver et me conseiller, notamment Véronne, Kévin et Josyanne, ainsi qu'à mes anciens camarades de classe et d'école pour leur encouragement, en particulier Brice, Simon, Caleb, Clinton et Dieudonné. Vos encouragements m'ont été précieux tout au long de cette aventure.

Merci à mes amis, proches et compagnons de route pour leur soutien moral, leur patience et leurs encouragements inestimables au fil de ces années. Je remercie notamment Daniela, Liana et Emmanuelle des "doctoresse fit girls"; Olivia, Joseph, Lionel, Stevy, Jessica et tous mes amis de Bordeaux. Merci pour votre soutien et votre implication dans le bon déroulement de ma thèse et de ma soutenance. A ceux d'entre vous qui sont encore sur ce chemin de doctorat, je souhaite de voir vos efforts récompensés dans les meilleurs délais.

Je remercie profondément ma maman Georgette qui a pu assister à ce grand jour et m'a soutenu chaque jour par ses prières et ses conseils. Je remercie Arielle, Junior et tous les membres de ma famille pour votre amour, votre soutien indéfectible et votre confiance. Votre présence m'a été précieuse à chaque étape de ce parcours.

Enfin, je remercie toutes les personnes qui ont contribué, de près ou de loin, à l'aboutissement de ce travail. Que ce soit par un simple mot d'encouragement ou par une aide concrète, chaque geste a compté pour moi. Merci à ceux qui ont pu assister à ma soutenance, en présentiel ou en ligne, et partager ce moment heureux avec moi. Merci notamment à mes amis pour leur assistance technique et leurs cadeaux.

À toutes et à tous, merci.

Je dédie cette thèse également à mon défunt père Albert, parti trop tôt, de son vivant professeur en physique du Bois et enseignant à l'École Normale Supérieure de Yaoundé.

Je ne saurai terminer sans remercier le Seigneur qui par la prière m'a tenu la main afin de traverser ces trois années et demie de challenges, instructifs et à la fois difficiles, notamment les mois de maladie à la fin de ma thèse. A l'exemple de mon travail qui porte sur la résilience aux fautes, cette thèse a été un véritable parcours de résilience. Que le Seigneur soit au contrôle de la grande aventure qui débute à présent.

Pour finir, le travail de recherche à l'origine de ces résultats a bénéficié d'un financement de l'Agence nationale de la recherche (ANR). Je leur suis reconnaissante d'avoir rendu cette recherche possible.

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

Nodes/Participants in a distributed system

N	Number of nodes
f	Fraction of Byzantine (malicious) nodes in the system ($f < 1$)
t	Fraction of Trusted nodes in the system ($t < 1$)
F	Number of Byzantine nodes ($F = f \times N$)
T	Number of Trusted nodes ($T = t \times N$)

Chapter 1

Introduction

1.1 Motivation

Over the past two decades, large-scale distributed systems have become a cornerstone of the digital economy and modern infrastructure. From the early days of file-sharing networks such as Napster and BitTorrent, through the deployment of global content delivery networks like Akamai, to the emergence of blockchain systems and decentralized finance, the scale, openness, and societal impact of these systems have grown dramatically. Bitcoin alone connects over 22,500 nodes across the globe [Bit25c], processes hundreds of thousands of transactions daily, and secures assets worth hundreds of billions of dollars. Beyond cryptocurrencies, decentralized paradigms now underpin storage systems such as IPFS and Filecoin, machine learning frameworks through federated learning, and an ever-expanding ecosystem of Internet-of-Things devices. These systems operate at a massive scale, span heterogeneous and geographically distributed infrastructure, experience continuous churn as nodes join and leave, and are open to participants whose behavior cannot be assumed to be cooperative.

At the heart of every large-scale distributed system lies a deceptively simple question: *how does a node discover and maintain contact with other nodes in the network?* This question, known as the neighbor discovery or membership problem, is foundational because virtually every higher-level service, from overlay construction and information dissemination to aggregation and consensus, depends on each node having access to a diverse, representative, and up-to-date subset of peers. The peer sampling service [JGKvS04] was introduced precisely to address this need. It provides each node with samples drawn uniformly at random from the global system or population of nodes, and refreshed continuously to reflect the current state of the network.

Among the various techniques proposed for peer sampling, gossip-based ap-

proaches [JVG+07b] have emerged as the method of choice in both research and practice. In gossip-based peer sampling, nodes periodically exchange portions of their local neighbor lists, called *views*, with randomly selected peers. This simple mechanism produces views that converge rapidly toward a well-mixed, statistically representative picture of the network, without any centralized coordination or global knowledge. Gossip-based protocols such as Cyclon [VGvS05b] and Newscast [TJ09] have been shown to yield overlays with properties close to those of random graphs, including high connectivity, small diameter, and robustness to node departures.

However, the openness that makes large-scale distributed systems powerful also makes them vulnerable. In permissionless environments such as public blockchains, any entity can join the network and participate in the gossip protocol without prior authorization or identity verification. This openness creates vulnerability in the peer sampling service. In fact, the exchange of peer information, which ensures the diversity and update of node samples, becomes the attack surface through which adversaries can subvert the system.

1.2 Problem Statement

In this thesis, we aim to address several problems faced by peer sampling protocols in an adversarial context.

Adversarial attacks on gossip-based peer sampling. Contrary to a correct node that respects the prescribed protocol, a Byzantine (or malicious) node may deviate arbitrarily. A Byzantine node can send fabricated messages, withhold information, forge identifiers, or collude with other malicious nodes to mount coordinated attacks. In the context of gossip-based peer sampling, an adversarial strategy is identifier flooding, in which Byzantine nodes inject their own identifiers, or those of colluding peers, into the gossip exchanges at a rate far exceeding that of correct participants. Because gossip protocols rely on the stream of received identifiers to construct and update local views, this flooding progressively poisons the views of correct nodes. Over time, the views of correct nodes become dominated by adversarial identifiers, a phenomenon that has cascading consequences.

First, with polluted views, correct nodes can be eclipsed. They lose connectivity with the rest of the correct network and can only communicate with adversarial nodes. Eclipse attacks have been demonstrated in practice against Bitcoin by Heilman et al. [HKZG15], who showed that an attacker controlling a modest number of IP addresses could successfully isolate targeted Bitcoin nodes from the other correct nodes. Second, adversarial nodes gain disproportionate visibility, enabling them to manipulate higher-level protocols built on top of peer sampling, includ-

ing consensus mechanisms, data dissemination, and decentralized learning. Third, the statistical properties of the views are destroyed. Instead of providing uniform random samples of the network, the peer sampling service delivers biased samples that overrepresent the adversary, undermining the assumptions on which the correctness and performance of upper-layer protocols rely.

Two broad families of defenses have been proposed in the literature. Detection-based approaches, such as Secure Peer Sampling (SPS) [JMv10b] and Secure Cyclon [AV23a], aim to identify and exclude misbehaving nodes by using cryptographic techniques to verify the legitimacy of exchanged views. However, these approaches face two key limitations. First, detection introduces additional overhead with nodes performing cryptographic computations, managing auxiliary data structures, and exchanging proofs of misbehavior. In the case of SPS, correct nodes remain vulnerable to rapid flooding attacks, in which adversaries can overwhelm them before they are blacklisted. Secure Cyclon mitigates this by detecting violations within a few rounds, but at a higher computational and memory cost that may be prohibitive in resource-constrained environments. Second, both approaches permanently exclude detected nodes from the overlay. While this removes misbehaving participants, it also reduces the overall population, thereby progressively degrading the overlay's connectivity and the quality of the peer sampling service.

Tolerance-based approaches, such as Brahms [BGK+08], and its extensions Raptee [PBQY+22] and Basalt [ABF+23], take a different route. Rather than attempting to identify which nodes are malicious, they design the protocol to limit the influence that any set of Byzantine nodes can exert on the views of correct nodes. Brahms, for instance, restricts the number of received membership messages a node can process per round and uses min-wise independent samplers to extract uniform samples from a potentially biased stream. Basalt introduces a stubborn chaotic search to counter the overrepresentation of adversarial identifiers. These tolerance-based protocols face important limitations that motivate the present thesis.

Uniform node sampling in adversarial environments. The key insight behind the tolerance-based approach is to produce a uniform and up-to-date sample of system nodes, drawn from the adversarially biased stream of received identifiers. To achieve this, Brahms [BGK+08] proposed a sampling component, implemented by each node, and which relies on a randomly chosen min-wise permutations that output a list of node identifiers whose image values are the smallest ever encountered by the permutations. This list is used to update a specific portion of the node views, which we call the sample part. However, Brahms is highly vulnerable to a Byzantine attack, with a small fraction of Byzantine nodes that can pollute

almost all correct nodes' views [PBQY+22]. The authors of RAPTEE [PBQY+22] introduced trusted nodes that operate within trusted execution environments to improve BRAHMS' resilience. These trusted nodes act as a source of trust and accelerate the spread of identifiers among them while filtering information received from untrusted nodes. However, RAPTEE 's effectiveness is limited because bias mitigation is restricted to collaboration among trusted nodes. To improve the BRAHMS results, BASALT [ABF+23] proposes changing the seeds used by the sampling component more frequently and using this component to update the entire view. However, BASALT faces limitations when the fraction of Byzantine nodes in the system is high (greater than 20%).

Anceaume et al. [ABG13] demonstrated analytically that, in order to generate a uniform random sample of nodes, it is necessary to limit the number of messages disseminated by malicious nodes, and to ensure that the correct nodes possess the full system membership. The latter requirement can be explained by the fact that if a node knows that a particular identifier appears ten times more often than expected under a uniform distribution, it can sample it less often, thereby counteracting the adversarial bias. In this way, Anceaume et al. [ABS13] proposed a sampling algorithm capable of approximating on-the-fly a uniform stream from a biased input, provided that the node has access to the frequency of the received identifiers. However, due to high churn and a massive identifier space in large-scale systems, maintaining an exact frequency counter for every distinct identifier that a node has ever encountered requires memory that grows linearly with the number of identifiers in the system. This leads the authors to propose an alternative that relies on a probabilistic, fixed-size data structure known as a Count-Min sketch [CM05], which maps each identifier to a set of counters via hash functions and provides frequency estimates with bounded error.

However, a series of non-exhaustive studies [DR07, YJZ+19b, SJL+24, LX21] have shown that Count-Min sketches exhibit suboptimal performance in frequency estimation for unbalanced data streams. Moreover, these studies proposed novel sketching strategies to address these limitations, but they are not designed or evaluated for a peer-sampling context and, on top of that, are adversarial. In addition, a peer sampling protocol runs indefinitely, so the stream of identifiers received by nodes is unbounded, and a fixed-size sketch will eventually saturate. Finally, in a collaborative debiasing setting where trusted nodes could exchange and combine their frequency information, the sketch must support a merge operation, a property that is straightforward for simple linear sketches but becomes problematic for the more accurate non-linear designs that modern streaming algorithms employ.

These intertwined challenges, namely adversarial bias, stream unboundedness, and mergeability, define the problem space that this thesis addresses.

1.3 Research Goals

The overarching goal of this thesis is to design, implement, and evaluate resilient peer sampling mechanisms that maintain near-uniform sampling quality in the presence of Byzantine adversaries while operating within the memory and computational constraints of large-scale distributed systems. This goal is decomposed into four specific research objectives.

Research Goal 1 Protocol-level Byzantine resilience through collaborative debiasing.
The first objective is to design a peer sampling protocol that leverages frequency tracking and trusted node collaboration to mitigate adversarial bias. The key question is whether a small number of trusted nodes, each with only a partial view of the network, can collectively produce frequency estimates accurate enough to debias the identifier stream for the entire correct node population. This goal addresses the resilience dimension of our problem.

Research Goal 2 Systematic evaluation of sketch-based frequency estimation for adversarial peer sampling.
The second objective is to identify which compact data structures are best suited for frequency estimation under adversarial conditions. Classical aggregate metrics used to evaluate sketches do not capture the specific requirements of peer sampling under balanced attacks, where the critical question is not the average estimation error but the ability to preserve the relative overrepresentation of adversarial identifiers. This goal requires developing new evaluation metrics and conducting a rigorous comparative study across multiple sketch designs.

Research Goal 3 Enabling unbounded operation through decay and merge strategies.
The third objective is to overcome the limitations of fixed-size sketches for unbounded peer-sampling streams. A long-running protocol will eventually saturate any finite sketch, destroying its debiasing capability. We aim to design decay mechanisms that allow sketches to adapt to evolving stream distributions, and merge procedures that enable trusted nodes to combine non-linear sketch instances, a problem with no existing solution for fingerprint-coupled sketches such as BitMatcher [SJL+24].

Research Goal 4 Integration and end-to-end validation.
The fourth objective is to integrate the preceding contributions into a unified system and validate it through extensive simulation under realistic adversarial

scenarios. The key question is whether protocol-level debiasing and sketch-based frequency estimation can be combined without sacrificing either resilience or efficiency.

1.4 Contributions

This thesis advances the state of the art in resilient gossip-based peer sampling through three main contributions, each addressing one or more of the research goals defined above.

AUPE: collaborative Byzantine fault-tolerant peer sampling. Our first contribution is AUPE, a new peer sampling protocol that extends BRAHMS with a frequency-based debiasing mechanism. Each node in AUPE maintains a *Set Cleaner* consisting of two components: a tracking component that records how often each identifier appears in the received gossip stream, and a debiasing component that transforms the biased stream into one that more closely approximates a uniform distribution. The debiasing principle, inspired by the uniform sampling algorithm of Anceaume et al. [ABS13], is to downweight high-frequency identifiers when producing local samples.

The distinctive feature of AUPE is its collaborative debiasing mechanism. A subset of nodes running inside trusted execution environments (TEEs) such as Intel SGX periodically exchange and merge their frequency tracking information. Because each trusted node observes a different slice of the gossip traffic, the merged information provides a more complete and accurate picture of the global identifier distribution than any individual node could obtain. Trusted nodes then use this enriched information to produce less biased views, which benefit the entire system through the gossip process.

Experimental evaluation with 10,000 simulated nodes demonstrates that AUPE achieves near-perfect resilience when up to 26% of nodes are adversarial, with the average fraction of Byzantine identifiers in correct node views converging to the actual fraction of malicious nodes in the system, meaning that the adversary gains no advantage from its flooding strategy. This substantially outperforms BRAHMS and BASALT under equivalent attack scenarios.

Robust frequency estimation for adversarial peer sampling. While AUPE demonstrates the effectiveness of frequency-based debiasing, its reliance on exact per-identifier counters is a scalability limitation as the memory required grows linearly with the number of distinct identifiers. Our second contribution addresses this limitation by conducting a systematic study of sketch-based frequency estimation techniques for adversarial peer sampling.

We evaluate six sketching strategies: Count-Min Sketch [CM05], Count-Mean-Min Sketch[DR07], Lossy Conservative Update[GI11], Cold Filter [YJZ+19b], BitMatcher [SJL+24], and Probabilistic Sketch [LX23]. These are evaluated on synthetic streams that model adversarial injection scenarios with varying attack intensities. A key methodological contribution is the introduction of the *bias factor*, a new metric that evaluates the overrepresentation of high-frequency identifiers, often adversarial, relative to low-frequency ones, usually correct. Unlike classical aggregate metrics such as Average Absolute Error, the bias factor error directly captures the information that matters for debiasing *i.e.*, whether the sketch enables the protocol to distinguish adversarial from correct identifiers with sufficient accuracy. Our evaluation identifies BitMatcher as the most promising candidate for integration into peer sampling protocols, owing to its adaptive bit-allocation mechanism and strong performance on the unbalanced distributions characteristic of Byzantine attacks.

Decay and merge strategies for unbounded stream processing. Our third contribution bridges the gap between the sketch evaluation and the protocol integration by addressing two challenges that arise when replacing exact counters with BitMatcher in AUPE.

The first challenge is stream unboundedness. A peer sampling protocol runs indefinitely, processing an ever-growing stream of identifiers. A fixed-size sketch will eventually saturate, its counters reach their maximum values, and it can no longer distinguish frequently occurring identifiers from rare ones. To address this, we design BMDecay, a controlled decay procedure for BitMatcher. BMDecay periodically reduces counter values via a halving procedure triggered by bucket-type transitions, which are themselves constrained to maintain sufficient fingerprint capacity. This mechanism implements a form of controlled aging that gives more weight to recent observations while preserving the relative frequency ordering.

The second challenge is mergeability. BitMatcher is a non-linear, fingerprint-coupled sketch *i.e.*, each counter position stores both a frequency count and a compact fingerprint of the associated identifier. Two independently constructed BitMatcher instances will generally contain different fingerprints at corresponding positions, making naive element-wise merging operations semantically invalid. We design a merge procedure that combines two BMDecay instances by matching fingerprints and taking the maximum of corresponding counters, exploiting the fact that trusted nodes using identical hash functions maintain structurally compatible layouts.

Experimental evaluation demonstrates that AUPE, equipped with BMDecay, mitigates Byzantine attacks with nearly optimal resilience for Byzantine fractions up to 20%, while using only 12% of the memory required by exact counting. With-

out decay, BitMatcher's F1 score degrades by up to 40% after processing streams exceeding 1 million elements. Moreover, with BMDecay, frequency estimation remains viable over streams of 10 million elements.

1.5 Thesis Organization

The remainder of this dissertation is organized as follows.

In Chapter 2, we establish the foundational concepts necessary to understand the rest of the thesis. We review the main families of distributed systems and peer-to-peer architectures, introduce the peer sampling service and its desired properties, and present the fault models commonly considered in distributed systems, from crash faults to adversarial Byzantine behavior. We illustrate these concepts with concrete examples, including Bitcoin's peer discovery mechanism.

In Chapter 3, we survey the evolution of gossip-based peer sampling from foundational designs under crash-fault assumptions (Cyclon, Newcast) through detection-based approaches (Secure Peer Sampling, Secure Cyclon) to tolerance-based protocols (Brahms, Raptee, Basalt). Our analysis highlights how each generation addresses the challenges of randomness and robustness while revealing the remaining gaps, particularly in scalable frequency-based debiasing, that motivate our contributions.

In Chapter 4, we present the design, implementation, and evaluation of Aupe. We formalize the system and adversarial threat models, describe the Set Cleaner mechanism and the collaborative debiasing protocol, and present extensive simulation results demonstrating Aupe's near-perfect resilience even with up to 26% of nodes being adversarial.

In Chapter 5, we present our systematic evaluation of sketch-based frequency estimation techniques. We formalize the problem of bias estimation in peer sampling, introduce the bias factor metric, compare six sketching strategies under adversarial conditions, and identify the design trade-offs between memory efficiency, estimation accuracy, and bias preservation.

In Chapter 6, we present BMDecay, our extension of BitMatcher for unbounded stream processing, and the associated merge procedure for distributed sketch combination. We integrate both mechanisms into Aupe and evaluate the complete system across a range of adversarial scenarios, demonstrating that protocol-level debiasing and sketch-based estimation can be combined into a practical, scalable solution.

In Chapter 7, we summarize the contributions of this thesis, discuss their broader implications and limitations, and outline directions for future research, including formal analysis and deployment on real-world testbeds.

Chapter 2

Background

This chapter establishes the foundational concepts necessary for understanding the design, analysis, and security of peer-sampling protocols in large-scale distributed systems.

We begin by introducing distributed systems and peer-to-peer architectures, with a particular emphasis on unstructured overlays that rely on gossip (epidemic) communication.

We then present the generic peer-sampling abstraction introduced by Jelasity *et al.* [JVG+07a], detailing its four core components, which are view initialization, peer selection, view propagation, and view update. We analyze the topological properties (randomness, connectivity, robustness) that emerge from these protocols and explain why uniform random sampling is essential for scalability and fault tolerance. Using Bitcoin's peer-to-peer layer as a concrete example, we illustrate the critical role of peer sampling in real-world systems.

Finally, we examine the limitations of classic peer-sampling protocols in adversarial environments. We describe Byzantine attack vectors, such as targeted, eclipse, hub, Sybil, and balanced attacks, which exploit the unawareness of standard sampling to bias overlay construction and isolate honest nodes. We also review countermeasures to these attacks.

The concepts and challenges presented in this chapter provide the necessary background for the state-of-the-art survey in Chapter 3 and motivate the Byzantine-resilient peer-sampling protocols developed in the remainder of this thesis.

2.1 Overview of Distributed Systems

Distributed systems have become a cornerstone of today's digital society. They underpin online social networks (such as Instagram and Facebook), video streaming platforms (such as YouTube and Netflix), and file-sharing networks (such as

BitTorrent). Cryptocurrencies (such as Bitcoin and Ethereum), cloud computing services (such as Amazon Web Services and Google Cloud), and the Internet of Things also rely on distributed systems. A *distributed system* is a collection of *nodes*, running on separate physical machines. These latter can include servers, personal computers, mobile devices, or sensors. Nodes communicate over a network to achieve a common goal. Multiple nodes operate simultaneously and collaboratively to provide services and share resources (*e.g.*, file transfer). Users, either humans or external processes (called clients), request these services through specific entry points or servers.

Despite these benefits, distributed systems face several key challenges. The system must present itself as a unified entity despite its distributed nature, supporting seamless user and application interaction. Hiding distribution complexities, such as resource location and communication failures, is essential for a smooth user experience. However, with multiple entry points comes a larger attack surface for adversaries. Despite these vulnerabilities, the system must be designed to remain robust and continue functioning.

Distributed systems can have various architectures, defined by how components and resources are divided, shared, and assigned roles. Examples include *client-server*, *three-tier*, *microservices*, *peer-to-peer*, and *event-driven architectures*. In this thesis, we focus on the *peer-to-peer* architectures.

2.2 Peer-to-Peer Architectures

Peer-to-peer (P2P) networks are networks in which nodes, also known as *peers*, do not rely on a centralized authority for services. All nodes (peers) are equal, acting both as clients and servers simultaneously. They hold the same responsibilities. P2P networks support various applications, such as selecting nearby peers, robust wide-area routing, efficient data item lookup, and redundant storage. Classic examples of P2P applications include file sharing (BitTorrent, Gnutella), content delivery (Skype, Spotify), blockchain networks (Bitcoin, Ethereum), VPNs, and data storage services.

P2P systems offer several key advantages. Resilience increases thanks to their decentralized structure. As peers join with ease, the network's resource capacity grows accordingly. The system also automatically manages node departures. The continuous joining and leaving of nodes is known as *churn*[1].

In P2P systems, nodes construct an *overlay network*. This overlay is a logical layer formed by connections established between nodes for application-specific communication. It operates on top of the physical Internet infrastructure, meaning nodes use the existing physical network but maintain their own logical links to

[1] In most peer-to-peer systems, 30–60% of peers leave within the first hour [SGG03].

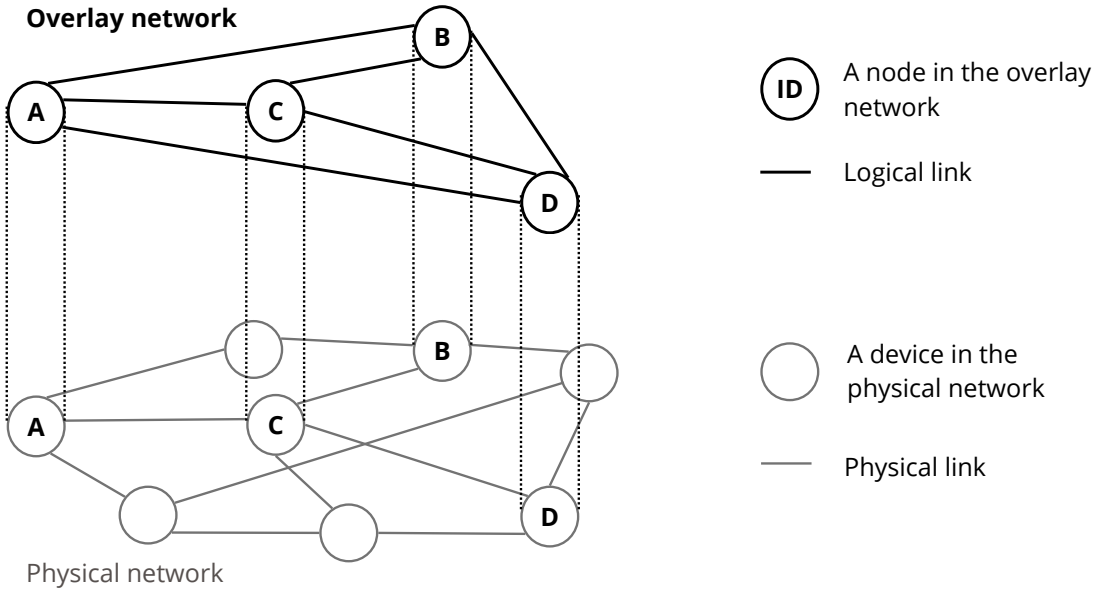


Figure 2.1: Overlay network with 4 nodes, on top of a physical network of 8 nodes.

communicate with one another. These logical links may differ from the underlying physical connections, allowing nodes to interact according to the requirements of the P2P system, with custom routing policies and communication paths. These logical links are communication channels set by protocol specifications, and use transport mechanisms such as TCP or UDP. As shown in Figure 2.1, the overlay network connects participating nodes, each identified by an identifier, and linked to individual processes running on the underlying physical infrastructure.

Nodes of a P2P network create and maintain the overlay topology by updating their routing table. A node can communicate only with its neighbors listed in its routing table. A routing table contains a fixed-sized list of peer identifiers with their IP addresses, drawn from the overall system population. Depending on how the nodes organize themselves (membership management), overlay topologies can be structured or unstructured. This organization determines the topology type.

In *structured* overlays, the network topology and the data placement follow deterministic rules. These overlays rely on a uniform identifier space, typically derived from hashing (*e.g.*, SHA-1). Both nodes and data are assigned identifiers within this space. Distributed Hash Tables [BKK+03] map data to specific nodes based on unique identifiers. For example, in the Chord system [SMLN+03], peers are logically organized into a ring-like topology. Data is requested by progressively looking among peers with higher identifier similarity, as defined by the membership protocol. Such deterministic system behavior brings about efficient resource retrieval.

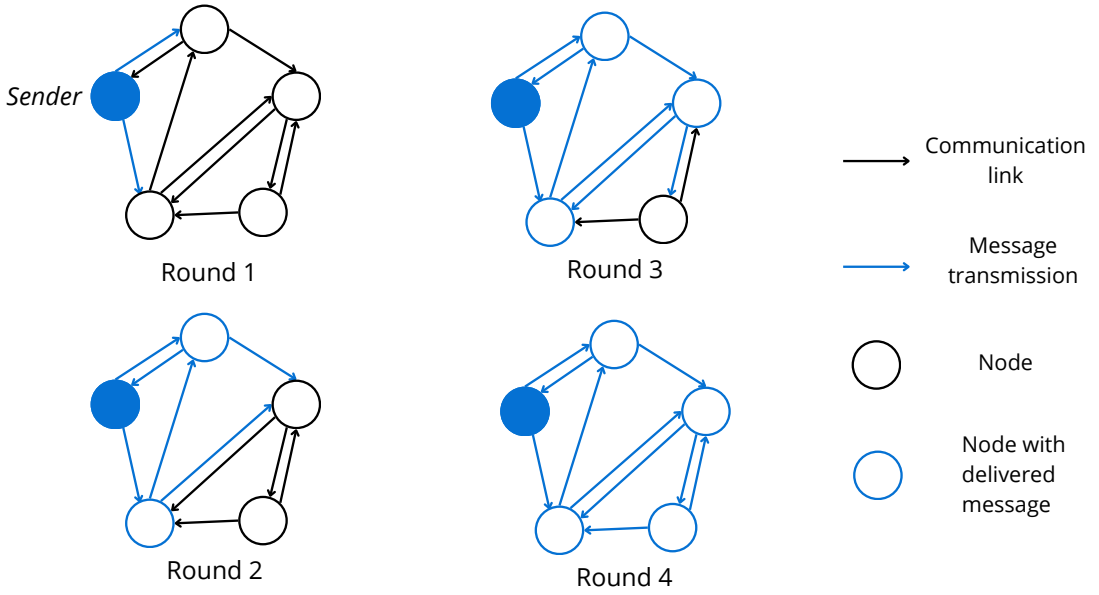


Figure 2.2: Gossip-based broadcast.

By comparison, in *unstructured* overlays, the topology emerges stochastically. Nodes connect to other peers based on probabilistic computations, which favors flexibility in overlay construction and thus resilience to node failures. Each node requests data by flooding or gossiping with its neighbors. For example, in the Gnutella P2P network [MIF02], a node searches for resources by broadcasting messages only to its neighbors. This approach distributes the cost of flooding the network. Figure 2.2 illustrates a gossip-based broadcast performed with five nodes, each associated with two neighbors. The sender forwards a message to its neighbors, which in turn relay it to theirs. In four rounds, all the nodes received the message. However, maintaining the overlay in a large-scale dynamic environment subject to churn remains challenging.

To scale reliably, large-scale dynamic unstructured P2P systems cannot have complete global knowledge of system-wide membership from which to select their neighbors. In fact, maintaining such knowledge would entail significant synchronization costs and an unrealistic workload for the system. Instead, each node relies on a membership protocol to continuously produce and update its neighbors.

2.3 Overview of the peer sampling service

To obtain neighbor contacts, a node queries a *peer sampling service*, which returns a random subset of peers (a *sample*). Figure 2.3 illustrates the peer sampling

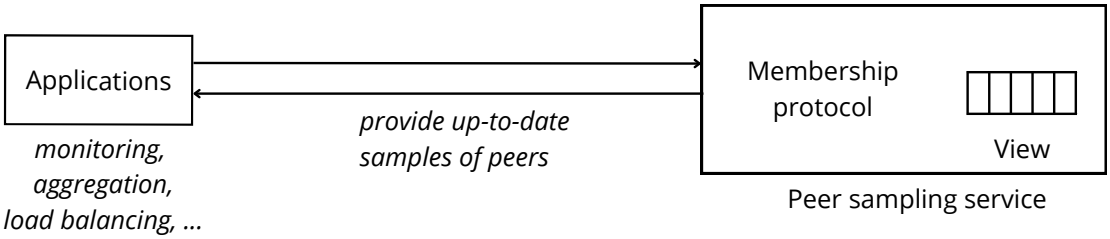


Figure 2.3: Overview of the peer sampling service.

service. Gossip-based protocols and P2P applications regularly invoke the service to obtain samples. Ideally, these samples are uniform, independent, and drawn from the full set of live (non-failed) nodes. But in practice, they are taken from the node's local routing table, or *view*. The service runs a membership protocol that updates this view. This protocol runs continuously, exchanging information with instances of the service running on other nodes to reflect system dynamics.

An efficient way to implement the membership protocol is to use the P2P gossip communication pattern. Nodes *periodically* exchange membership messages to advertise themselves and discover new peers. This approach has been widely used in several applications to support information dissemination [Pit87], overlay management [VGvS05b], load balancing [JMB04], and aggregation [KDG03]. Gossip-based peer sampling is thus a cornerstone of modern large-scale dynamic systems and the primary focus of this thesis. This section describes the peer sampling service, its objectives, the properties of the resulting topology, its core functionalities, and the sampling challenges, as initially described by Jelasity *et al.* [JVG+07a].

The objective of peer sampling membership protocols is to maintain a connected overlay network among nodes and ensure it remains connected even during churn and failures, so that all nodes can eventually reach each other. The effectiveness of this approach depends on the uniformity and randomness of the sampling process. The protocol must ensure that every live node has an equal, non-null chance of being selected as a neighbor of other nodes. In fact, these properties have been shown to lead to an overlay topology that is resilient to node failures and balances well the communication and computational load across the system [JGKvS04]. Depending on the purpose of the application that relies on the peer sampling service (load balancing, aggregation, *etc.*), the uniformity or randomness of samples can be approximated.

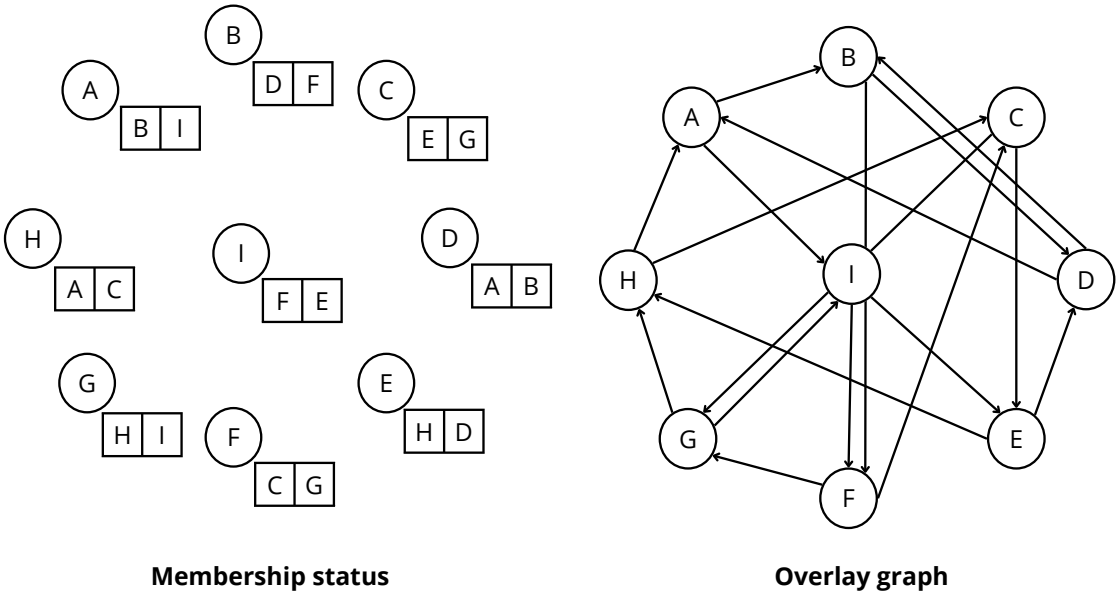


Figure 2.4: Overlay graph of nine nodes.

2.4 Properties of the overlay graph

The set of nodes and their logical links together form an overlay graph. A well-designed membership protocol maintains an overlay graph that closely approximates a random graph [KMG03]. This latter has properties such as a low network diameter, low clustering, and a balanced in- and out-degree distribution, which favor scalability in system membership size, network connectivity, low partitioning, and robustness to network failures.

To obtain an overlay graph of the network, overlay nodes are represented as vertices, and the logical links between them define the graph's edges. Figure 2.4 proposes an illustration of a simple network's membership status and its associated overlay graph. Each node has a table representing its views (here with two descriptors) that reference its actual neighbors. Each neighbor reference in this view is associated with a directed edge from the node to the referenced neighbor in the overlay graph. The following paragraphs provide a closer look at each property of the overlay graph and its impact on the overlay's behavior.

Degree distribution. In an oriented graph, the degree of a node u is the sum of its *in-degree* and its *out-degree*, that is, respectively, the number of directed edges going to node u and the number of directed edges going from node u to other neighbors. The degree of an overlay graph is the average degree of its nodes. A degree distribution with low standard deviation is preferable to ensure network

robustness and load balancing. In an overlay graph, a node's out-degree is the number of nodes it is connected to, or the size of its view. Its in-degree is the number of neighbors that are connected to the node. For example, in Figure 2.4, nodes have an equal in-degree and an out-degree (of two), meaning that the workload of protocols relying on this membership management will be equally shared between the nodes.

In an overlay, nodes with low in-degree are less well known than those with high in-degree, and both pose distinct challenges. The former risks being partitioned from the system if its in-degree value drops to zero and it loses its own connections. The latter can become a highly connected hub and face a high volume of incoming requests.

Clustering coefficient. The clustering coefficient of a node indicates how close its neighbors are to forming a complete cluster. It is calculated by dividing the number of undirected edges that actually exist between its neighbors by the total number of possible undirected edges between them. The result is a number between 0 and 1. For example, in Figure 2.4, the clustering coefficient of node A is zero because there are no edges among its neighboring nodes B and I.

The clustering coefficient of a graph defines how its nodes cluster together. It is computed as the average clustering coefficient of its nodes. The smaller this coefficient is, the less the neighbors cluster together, meaning that nodes possess more diverse knowledge even with few neighbors. For instance, in a complete graph, the clustering coefficient is 1, whereas in a tree, it is 0. In an overlay network, the clustering coefficient measures the diversity of nodes' views. Nodes in a cluster are more likely to be isolated from the rest of the system and also receive a large number of redundant messages.

Path lengths. The shortest path length between two nodes is the minimum number of edges that are encountered to go from one node to another. For example, in Figure 2.4, the shortest path to reach node D from node A is $A \rightarrow B \rightarrow D$ with a shortest path of 2. It represents the minimum number of hops a message must cross to reach another node in the overlay network. The maximum path length between two nodes in a graph is its *diameter*. The average path length is the mean of all shortest paths between all node pairs. Low diameter and average path length ensure fast information dissemination in the overlay network, resulting in lower communication costs and high scalability.

The experiments in the paper [JVG+07a] showed that, for specific peer sampling designs (presented in Section 2.5), the dissemination of gossiping membership messages and the continuous updating of node views induce the convergence of the overlay graph to stable properties analogous to those of random graphs. This con-

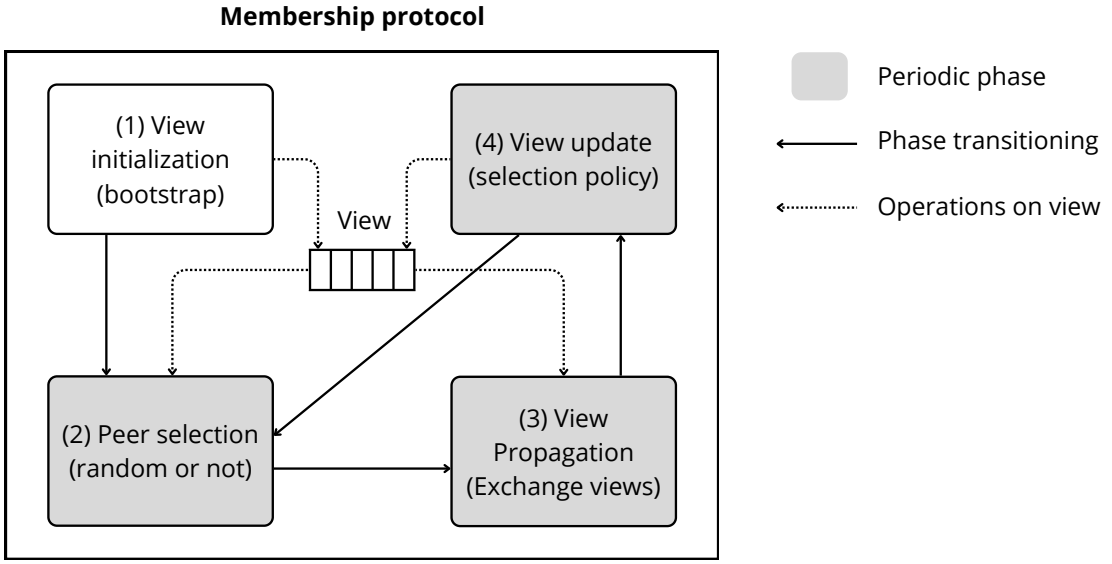


Figure 2.5: Membership protocol.

vergence is independent of the initial graph configuration, provided churn ceases.

2.5 Description of the membership protocol

Each overlay node requests two main functionalities from the service. First, when the node joins the system, it instantiates and runs the service. Then, the service registers itself with the gossip membership protocol instance that maintains the overlay network. Second, the node continuously requests the service to obtain samples of peers derived from its view, to start communicating with.

Each entry in the view describes a unique neighbor, providing its identifier (a number or key) and its address (IP:port) for communicating with it on the network. Some peer sampling designs use an additional field, the peer's age or timestamp, to detect stale descriptors *i.e.*, references to failed or departed nodes, as well as some misbehaviors (see Chapter 3 for protocol examples). This field indicates how long the reference to this peer has existed.

Each node in the overlay executes the same membership protocol. The four main phases of the protocol are view initialization, during which the node obtains an initial view after joining the network; peer selection, where the node selects one or more peers to communicate with; view propagation during which nodes exchange membership information; and view update during which each node updates its view accordingly. Figure 2.5 shows the operations flow of the protocol. Except for the view initialization, all these phases are executed periodically to adapt to

the network's dynamics.

Although the protocol is fully asynchronous and requires no global clock, it executes to *rounds* of *cycles*, during which a node executes the same sequence of operations. The protocol initiates two threads per node: an active thread that selects peers and sends membership messages (or requests), and a passive thread that listens to and responds to neighbor requests. The active thread is activated every round of τ seconds. The four phases of a peer sampling protocol are presented below.

View initialization. This phase is executed when a node first requests the peer sampling service. There are two main strategies for providing a node with its first view. Either the service provides a list of peers with a high likelihood of being available to bootstrap the node view, as hard-coded in the peer sampling code or published by a public server, or the node executes a short random walk [KS04] starting from a proximity contact. Then, the node can start gossiping with these new contacts. To anticipate future temporary disconnection, a node can retain (or hard-code) its previous view. When it reconnects, it resumes gossiping immediately using this old knowledge, avoiding the need for a full bootstrap.

Peer selection. This involves selecting a live peer from the view for future membership exchanges. This can be done multiple times during a given round. The aliveness of a peer can be checked with ping messages; if unsuccessful, the selection can be restarted. Two peer selection policies can be considered. Either the selection is uniform at random, or the oldest peer in the view is chosen. The idea behind the latter strategy is that peers who have not been contacted for the longest time tend to have the least correlated views, thereby increasing the likelihood of obtaining new peers. Selecting the most recent peer, *i.e.*, the one we communicated with in the previous round, is not relevant because the views are similar.

View propagation. After selecting a peer for communication, the node exchanges membership management messages with that peer to share information. There are two basic types of messages: push and pull messages. With *Push* messages, the node unsolicitedly sends a reference of itself and some or all of its neighbors to the contacted peer. And with *Pull* messages, the node requests its neighbor's peer references. Figure 2.6 illustrates a bunch of 3 system nodes, with their views representing their outgoing connections with other peers. During this round, node I receives a push message from node A and sends a pull message to its neighbor H, and subsequently updates its view.

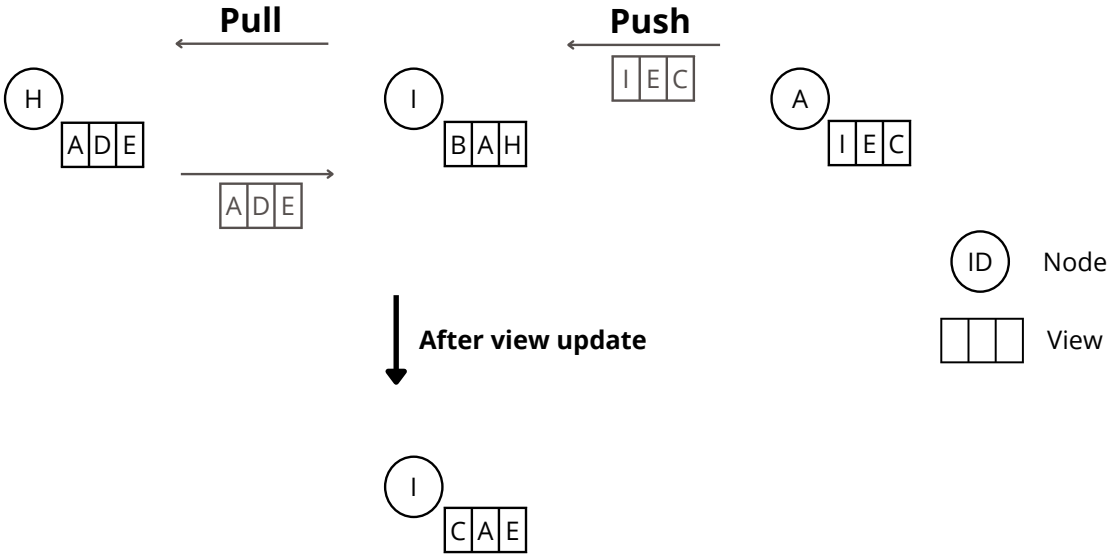


Figure 2.6: View propagation and view update of node I in a given round.

A pure-pull strategy is ineffective because a node that loses all of its outgoing connections becomes isolated and can no longer initiate communication. Similarly, experiments in [JVG+07a] demonstrate that a pure-push strategy fails to balance in-degrees and significantly increases the risk of network partitions. The *push-pull* strategy, in which both communicating nodes actively exchange their views, resolves these issues. With push-pull, the in-degree distribution remains tightly concentrated around the view size and has low variance, which is ideal for load balancing.

View update. After the view propagation phase, a node temporarily stores all peer references received during the current round in a buffer. It then constructs a candidate set by taking the union of its previous view and the buffered entries. In case of duplicate descriptors for the same node, it takes the freshest one and discards the others. Finally, it trims this set back to the fixed view size v according to one of the various update policies. The first strategy is to form the new view by selecting v entries uniformly at random from the candidate set, without any preference (*blind* policy). The second strategy is to retain the v most recent entries from the candidate set, according to descriptor timestamps or ages, thereby favoring freshness and quickly removing stale references (*healer* policy). And the last strategy is that the node replaces exactly the entries it sent to its peer with the entries received from that peer; the remaining (unsent) entries from its old view are kept unchanged (*swapper* policy). Furthermore, the view update strategy can be a mix of these strategies, depending on the system's needs.

The choice of view update policy has the most decisive influence on the statistical properties of the resulting overlay, as demonstrated by Jelasity *et al.* [JVG+07a]. First, they showed that regardless of the policy, the service always guarantees *local randomness*. In fact, from any given node's perspective, the returned samples appear uniformly random. However, when considering *global randomness* and the independence of samples across the network (which is crucial for overlay connectivity and load balancing), the policies differ markedly. Using the *swapper* strategy yields greater similarity between the overlay graph and a random graph, with views that are highly independent. *Healer* is the second-best option: it introduces moderate correlation between the generated views, yet converges quickly to a well-connected overlay. And *Blind* performs worst on global randomness metrics, as it tends to preserve existing correlations longer. Second, with respect to fault-tolerance properties, although all strategies are robust to failures, churn, and partitions, the *healer* strategy is the most effective at managing them, even in catastrophic scenarios, because it accelerates the elimination of dead links.

Consequently, the swapper strategy is the preferred choice for most practical deployments requiring strong randomness guarantees. In contrast, the healer strategy is preferred for maintaining high-robustness overlays in the presence of churn and failures. Note that these results are obtained with a push-pull view propagation policy, as it performs better.

In conclusion, Gossip-based peer sampling protocols exhibit strong resilience to failures thanks to their randomness, which provides multiple independent dissemination paths between nodes. Information about newly joining nodes spreads progressively, while information about departed or failed nodes progressively disappears from the network. Experimental results in [JVG+07a] confirm that various peer-sampling service designs tolerate severe failure scenarios, including catastrophic failures (sudden crash of a large fraction of nodes), massive churn (simultaneous departure of many nodes followed by the arrival of new ones), and network partitions. A distinct and more challenging class of failures involves arbitrary (Byzantine) behavior, which is addressed in Section 2.6.

Example: Peer sampling in Bitcoin

Blockchain systems such as Bitcoin rely on a peer-to-peer (P2P) overlay to disseminate transactions and blocks without centralized coordination. Each node maintains a limited number of active connections to other peers, which are selected from a much larger pool of reachable nodes. Measurements of the Bitcoin network report an average of approximately 22,500 reachable nodes in July 2025 [Bit25c]. Peer sampling in Bitcoin is implemented through the address manager component of Bitcoin Core [bit25a], which maintains and updates a local view of known peers.

View initialization. When a node is started for the first time, it must bootstrap its initial view of the network. To do so, it queries a set of hardcoded DNS seed hostnames or fallback seed addresses to obtain IP addresses of active peers [MLP+15]. In addition, Bitcoin Core stores peer addresses in a local database, allowing nodes to reuse previously known peers across restarts. Each address is associated with a timestamp indicating the last successful interaction with that peer.

Peer selection. Bitcoin Core maintains a fixed upper bound on the number of outgoing connections, typically eight simultaneous peers. New connections are established by sampling from the local address table according to a probabilistic policy that favors recently reachable or successfully contacted peers, while still allowing exploration of new addresses. Peer selection and connection management are performed periodically to ensure sustained connectivity.

View propagation. Nodes exchange peer information through `GETADDR` and `ADDR` messages. Upon establishing a new connection, a node sends a `GETADDR` message, to which the peer replies with up to three `ADDR` messages, each containing up to 1,000 addresses randomly selected from its tables, for a total of up to 2,500 addresses. These replies exclude outdated peers and duplicates already sent on the same day, ensuring a fresh sample of addresses is provided. Nodes also disseminate unsolicited `ADDR` messages containing up to 10 recent addresses to a fixed subset of their peers. Additionally, each node advertises itself once daily to all connected peers.

View update. The local address table is continuously updated based on communication outcomes. Addresses that are recently used or successfully contacted are retained, while stale or unreachable peers are gradually removed to keep the table size bounded. This ongoing maintenance ensures that the node's view remains populated with active and responsive peers.

Although this peer sampling mechanism provides robust connectivity and supports network scalability, it does not yield a perfectly uniform sample of the network. Earlier versions of Bitcoin Core were shown to be vulnerable to Sybil [Dou02] and Eclipse attacks [HKZG15], in which adversaries inject a large number of malicious addresses to bias a node's peer view. Subsequent design choices mitigate these risks, but they remain an inherent challenge in open, address-based P2P networks.

2.6 Byzantine Faults in peer sampling

Byzantine (or malicious) faults [LSP82] represent the most general and adversarial failure model. In this model, faulty nodes may behave arbitrarily, sending inconsistent, misleading, or colluding messages. In the context of peer sampling, Byzantine nodes can actively bias the sampling process by disproportionately promoting their own identifiers and suppressing those of correct nodes. This increases their representation in the views of correct nodes and enables a range of attacks aimed at isolating nodes, partitioning the network, or undermining higher-level protocols (e.g., consensus in blockchains). Such malicious behavior directly threatens the topological properties obtained through gossip-based peer sampling, including robustness to partitioning, fast recovery from failures, and uniform random sampling (see the previous section about the peer sampling service).

A Byzantine node behaves arbitrarily, attempting to intentionally damage the sampling process. An adversary controlling even a small fraction of colluding Byzantine nodes can dictate their behavior and launch powerful attacks. Byzantine nodes can deviate from the peer sampling specification at different phases.

During the peer selection phase, Byzantine nodes can contact more peers than the protocol allows (e.g., by sending additional push messages) or selectively target correct nodes to deliver biased information. When answering pull requests from correct nodes, Byzantine nodes can fill their answered view exclusively with descriptors of nodes from their Byzantine group, thereby accelerating the spread of their group across the network. Moreover, they can manipulate descriptor timestamps or ages to favor their selection. In addition, depending on their resource capacity, they can retain complete knowledge of the correct nodes by expanding their view beyond protocol limits, thereby enabling them to orchestrate attacks more effectively.

These strategies exploit the randomness and locality of standard view-update policies, leading to Byzantine identifiers being sampled and propagated more frequently. Many attack strategies can be implemented, depending on the attacker's goal in the overlay. The following paragraphs provide a closer look at these strategies.

Targeted Attack. The simplest attack consists of flooding a single correct node with Byzantine node descriptors via various push requests and false views sharing. This gradually fills the victim's view with Byzantine identifiers, increasing the likelihood that it selects more Byzantine peers for future exchanges and fewer correct nodes. Over time, the victim becomes heavily influenced, with some of its connections controlled. Targeting multiple nodes in the overlay can enable powerful attacks. For example, an adversary can launch Denial-of-Service attacks that overload and disable parts of the network, or disrupt higher-layer protocols

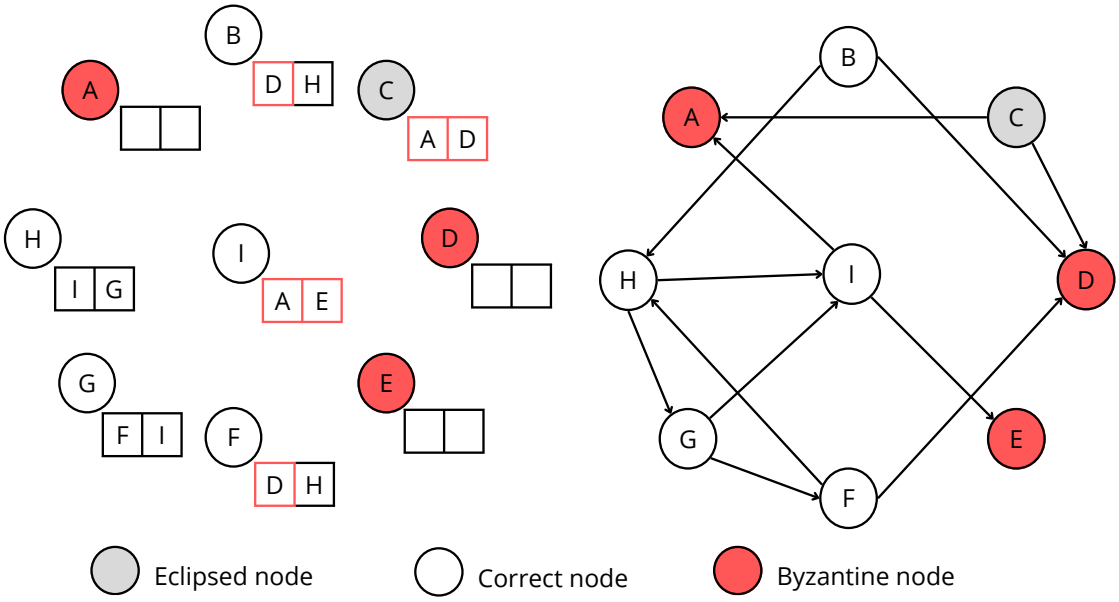


Figure 2.7: Eclipse attack on the overlay network.

that depend on the overlay membership (e.g., a blockchain consensus protocol).

Eclipse Attack. An eclipse attack [SNDW06] extends the targeted attack to control all the connections to and from the victim node, effectively isolating it from the rest of the network. This vulnerability was famously demonstrated against Bitcoin's peer-sampling mechanism [HKZG15]. In fact, as the victim loses connectivity to other correct nodes, those nodes will progressively remove the victim's descriptors from their views. Figure 2.7 illustrates an eclipse attack on the correct node C. Node C is isolated and can communicate only with the Byzantine nodes A and D. Although the targeted node I knows only Byzantine nodes (A and E), it can still be contacted by other correct nodes (G and H). So it is not isolated yet.

Since Byzantine nodes behave arbitrarily and independently of their views, their view content and outgoing connections have been omitted from the figure for clarity. This will be the case for the following attack scenarios.

Hub Attack. In a hub attack [JMv10a], Byzantine nodes coordinate multiple eclipse or targeted attacks to maximize their visibility in the overlay. The objective is to elevate these adversarial nodes into high-degree hubs under the attacker's control, allowing them to intercept and steer a large portion of the network's traffic. As a result, the overlay fragments into clusters of correct nodes, which are effectively captured by Byzantine hubs. Once this structure is in place, a coordinated departure of these hubs can trigger widespread disconnections, amounting

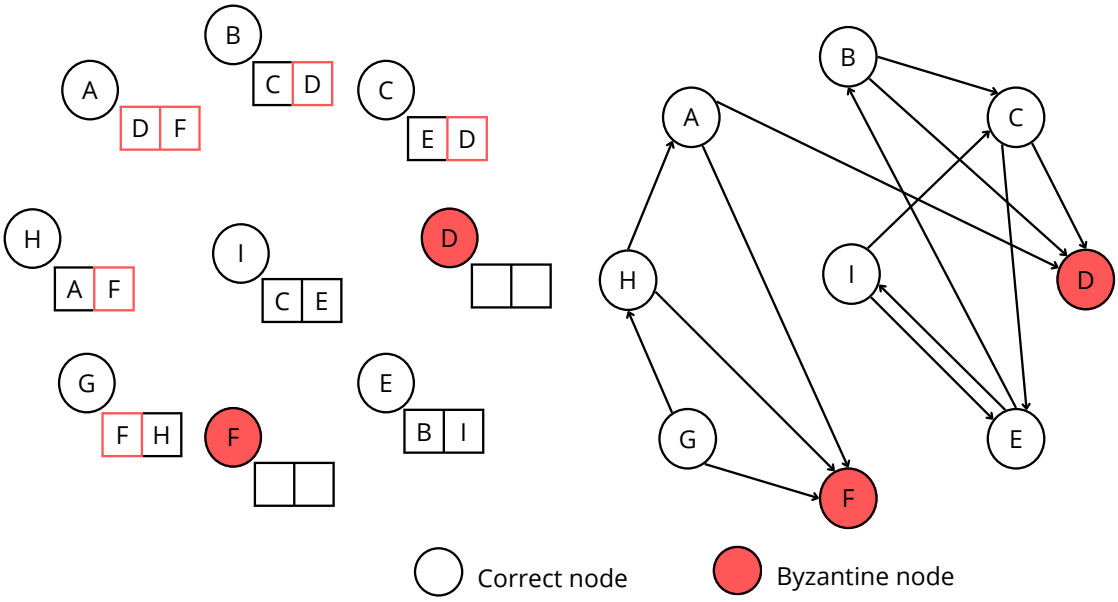


Figure 2.8: Hub attack on the overlay network.

to a Denial-of-Service attack on the entire system. Figure 2.8 illustrates a hub attack on the overlay network. Byzantine nodes D and F became hubs, controlling traffic between the two generated clusters $A - H - G$ and $B - C - E - I$. They can thus partition the network into these 2 clusters. Recovering from a hub attack requires substantial rebuilding of the overlay topology.

Balanced Attack. The most insidious attack is the balanced attack [BGK+08]. In this attack, Byzantine nodes collaboratively distribute their identifiers evenly across all correct nodes. This requires the Byzantine nodes to build a full knowledge of the correct nodes to coordinate the attacks. These attacks emerged to counteract the protective measures put in place against previous attacks. By respecting per-node push limits and avoiding the creation of an obvious hub, they achieve high representation while remaining stealthy. This attack is especially perilous because it violates the uniformity assumption of peer sampling while evading simple anomaly detectors. Figure 2.9 presents a scenario of a balanced attack in the overlay network. Byzantine nodes G, I, and D spread their descriptors evenly to increase the chance that each correct node has at least one Byzantine neighbor. The views of the legitimate nodes are then gradually corrupted by requests sent to these Byzantine nodes.

Sybil Attack. In a Sybil attack [Dou02], a single node creates numerous fake identities (different IP addresses and keys) and injects them into the system. By

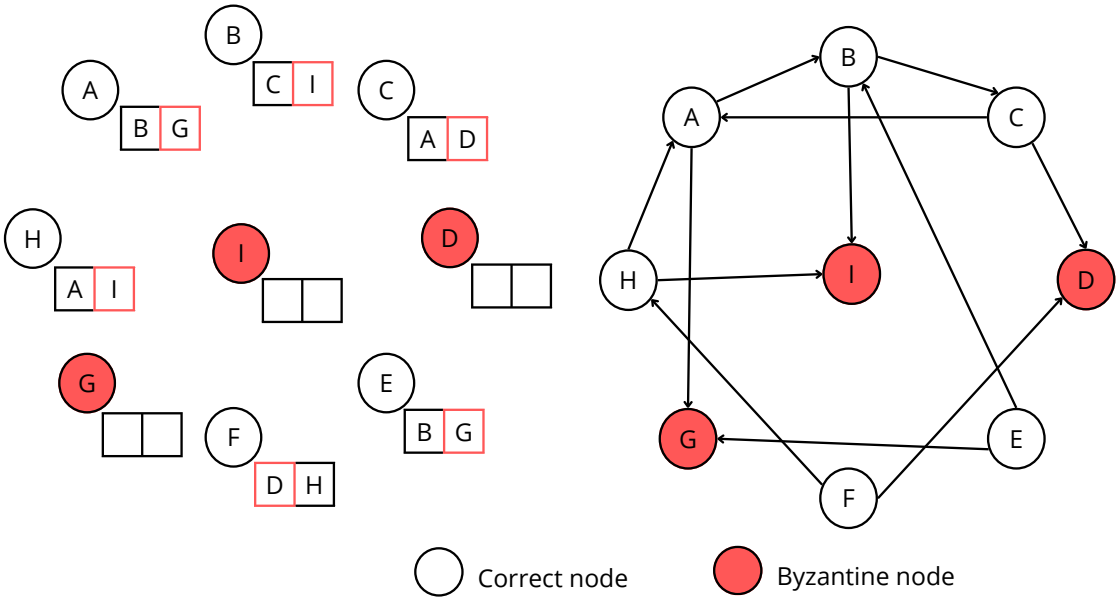


Figure 2.9: Balanced attack on the overlay network.

creating fake identities, a Byzantine node increases its chances of being sampled through them. These identities can then be used to mount eclipse or hub attacks more effectively. Sybil attacks can be mitigated by making identity creation costly (*e.g.*, proof-of-work, proof-of-stake, or trusted certification), though such mechanisms are computationally expensive in large, open systems.

We do not consider identity impersonation in this list of attacks, as basic cryptographic protections, such as message authentication and descriptor signatures, prevent impersonation and ensure integrity.

2.7 Uniform sampling in adversarial settings

Byzantine attacks compromise the overlay topology, rendering it unable to achieve its desirable properties, such as a low network diameter, low clustering, and a balanced in- and out-degree distribution (see Section 2.4). Hence, the samples provided by the peer sampling service are more likely to include Byzantine nodes. In this section, we present strategies to mitigate bias introduced by Byzantine attacks on the membership information sent by nodes (in push and pull reply messages) and to make the samples provided by the service more uniform.

Because Byzantine nodes have abnormally high degrees, an effective way to mitigate their overrepresentation is to monitor the global topology and re-equilibrate node degrees when updating their views. But the computational costs this strat-

egy incurs render it infeasible in peer-to-peer settings. Consequently, the peer sampling service needs to add policies to reduce bias that can be generated and received on each node. A solution is to limit the number of membership requests (push and pull) a node can process via computational puzzles (*e.g.*, Merkle's puzzles [Mer78]), and the size of pull responses by checking the size of the returned view. Although this solution does not entirely resolve the attack, it slows it down and pushes it back over several rounds.

Malicious nodes can still build and share false views; consequently, correct nodes will receive a high proportion of Byzantine descriptors during the view propagation. Because of the service's core randomness, the correct node's view will be updated at random without prior knowledge of which group each identifier belongs to. The node could monitor the peer occurrences received during the round. However, in a balanced attack, this would be of no use, as the Byzantine node descriptors are evenly distributed among the nodes. Mitigating this bias requires that, from a node's point of view, every peer encountered during protocol execution should have the same probability of being selected to update the view, regardless of how often their descriptors have been seen since the beginning of the sampling process. This involves restoring uniform sampling under adversarial conditions.

In their study on the uniform sampling problem [ABG13], the authors demonstrated that two requirements must be met to generate a uniform, dynamic, and random sample of nodes under adversarial conditions. First, the dissemination rate of identifiers through push messages must be bounded to limit the bias that Byzantine nodes can introduce to the stream of identifiers received by a node (as explained previously). Second, correct nodes must have access to a sufficiently complete membership view proportional to the system's size to enable unbiased selection. However, these requirements are too demanding to be fulfilled in large-scale dynamic systems. Moreover, a common strategy to randomize the selection of view descriptors is to use hash-based permutations. In Min-Hash permutations approaches [BCFM98], each received identifier is mapped to the output of a uniform independent hash function, and the identifier with the smallest hash value is selected as the sample. Since hash outputs are independent of the number of occurrences, the selection probability remains unbiased even when the frequency is biased adversarially. However, once the identifier with the minimum hash value is selected, the sample output cannot change unless the hash function is modified. Finally, Anceaume *et al.* [ABS13] propose a probabilistic sampling algorithm that outputs a uniform sample stream when computing an input stream, using the frequency distribution of node identifiers. In conclusion, these threat models motivate the design of Byzantine-resilient peer sampling protocols (see Chapter 3), which explicitly aim to restore uniformity under adversarial pressure.

2.8 Conclusion

Peer sampling is a fundamental building block of large-scale distributed systems. It enables efficient overlay construction, robust gossip dissemination, and reliable peer discovery in blockchain networks by continuously providing each node with a small, dynamic partial view that approximates a uniform random sample of the membership. However, the randomness and uniformity guarantees of classic gossip-based peer-sampling protocols are fragile in adversarial settings. Even a small fraction of Byzantine nodes can exploit the blindness of standard view-update policies to bias sampling outcomes, leading to targeted and eclipse attacks, hub formation, or subtle, system-wide manipulation (balanced attacks). Defending against such threats, especially balanced attacks, requires moving beyond naive gossip. Effective countermeasures include bounded dissemination (to limit flooding), frequency-aware peer sampling, and hash-permutation-based sampling mechanisms that restore the uniformity of the stream of identifiers processed by nodes. Ultimately, Byzantine-resilient peer sampling transforms a once-trusted primitive into a provably robust service that operates correctly in fully open and hostile environments.

Chapter 3

State-of-the-art on Gossip-based peer sampling protocols

This chapter surveys the evolution of gossip-based peer sampling protocols, from crash-fault designs that assume cooperative participants susceptible to crash failures to advanced techniques resilient to Byzantine attacks. We begin with protocols that construct overlays under crash-fault assumptions, including classical designs such as Cyclon and Newscast that target uniform random topologies, as well as the more recent Elevator protocol, which introduces hub sampling to produce overlays with designated hub nodes. We then examine detection-based protocols that leverage cryptography to identify and exclude malicious nodes. Finally, we review fault-tolerant protocols that mitigate adversarial bias without relying on attacker detection, including protocols targeting uniform random sampling and LIFT, which addresses the specific problem of securing hub election in hub-based topologies. Our analysis highlights how each category addresses key challenges such as randomness and robustness, while revealing remaining gaps that motivate our contributions in subsequent chapters.

3.1 Crash-fault peer sampling

In Crash-fault designs where all nodes are cooperative and follow the protocol, gossip-based peer sampling achieves excellent approximations of uniform randomness through simple, periodic view exchanges. For instance, protocols like Cyclon [VGVS05a] and Newscast [TJ09] rapidly mix membership information, yielding overlays with balanced degrees, low clustering, and short paths. More recently, the Elevator protocol [LPBTF24] introduced the concept of hub sampling, where the gossip mechanism is designed to produce overlays with designated hub nodes rather than uniform random graphs. This section outlines how these protocols

work and the topological properties they achieve.

3.1.1 Cyclon

Cyclon [VGVS05a] is a foundational gossip-based membership protocol. Each node maintains a fixed-size partial view of v descriptors, each containing a peer's address (IP:port) and an age counter. The basic cycle of Cyclon is as follows.

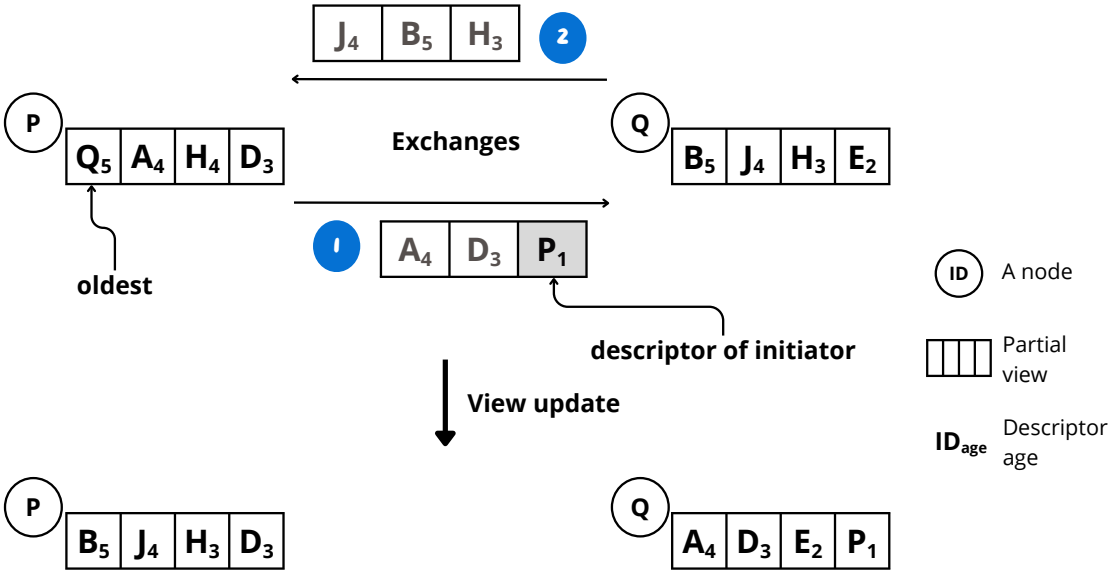


Figure 3.1: Cyclon shuffling process (round 5).

View initialization. A new node must start with at least one contact (an introducer) in the overlay. The node obtains an initial view by copying the introducer's view or using a bootstrap server. This gives the node a view of v descriptors, which may include duplicates or placeholders. Over time, the view evolves to continually include active peers.

Peer selection. Each node P periodically initiates a gossip round (e.g., every 10 seconds). First, it increments the age of every descriptor in its view. Then it selects the oldest descriptor to initiate gossip communication. Intuitively, selecting the oldest ensures that stale information is quickly updated. Figure 3.1 illustrates this procedure. At round 5, node P selects its oldest neighbor, Q, to initiate gossip exchanges. In this example, view descriptors are ordered by age, from oldest (left) to youngest (right).

View propagation. Node P prepares a message containing a set of $k-1$ randomly chosen descriptors (excluding Q) and a new descriptor of itself (P) with an age of 0. Node P then sends this updated set to Q. Upon receiving the message, node Q randomly selects k neighbors from its view, generates a set, and sends it back. Thus, each exchange carries up to k descriptors each way. The shuffling length is $k = 3$ in the example in Figure 3.1, but it is typically 6–8 in real configurations.

View update. After the exchange, each party (P and Q) incorporates the descriptors it received into its own view. They discard any descriptors that point to themselves or are duplicates. They replace the descriptors they just sent with the received descriptors while ensuring the view keeps its fixed size. P then becomes Q's neighbor, and Q is no longer a neighbor of P. This completes the shuffle. As a result, both views are partially refreshed with new random contacts.

Cyclon's strength lies in its simplicity and efficiency. Each view exchange involves only $O(k)$ messages and local updates. Yet, over repeated rounds, the network converges to a well-mixed random overlay. According to experimental evaluations [VGVS05a], Cyclon guarantees low average path lengths and clustering coefficients that converge to those of a random graph. It handles churn well, as stale descriptors are naturally evicted.

However, Cyclon is highly vulnerable to malicious attacks. For instance, a small group of colluding malicious nodes comprising less than 1% of the system can present fake views consisting solely of identifiers from their group to the correct nodes during gossip exchanges, as discussed in the Secure Cyclon [AV23b]. Within a few cycles, this group can seize control of all connections to and from the correct nodes, resulting in a hub attack (see section 2.6 about attacks).

3.1.2 Newscast

Newscast [TJ09] is another popular peer sampling service. Like Cyclon, it maintains a fixed-size view of v, but it uses timestamped descriptors rather than age and is based on full-view exchanges. The protocol works as follows.

View initialization. Similar to Cyclon, each node begins with an initial set of v contacts, which may include bootstrap nodes with empty entries. Each descriptor stores the node's address and a timestamp indicating the last update time.

Peer selection. Unlike Cyclon, in Newscast, a node P randomly selects one neighbor Q from its view to initiate communication with.

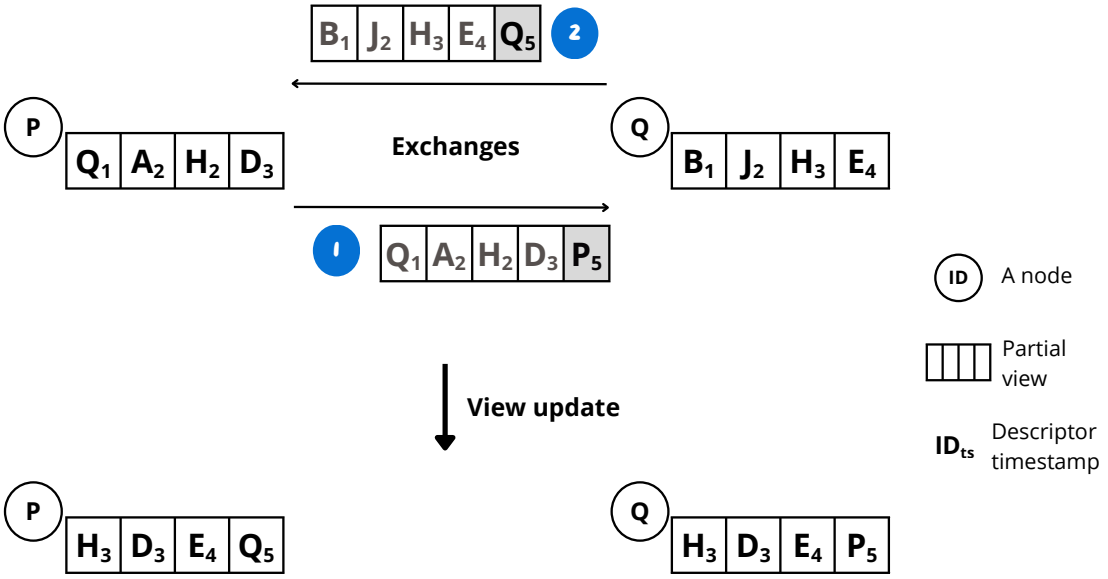


Figure 3.2: Newscast gossip exchanges (round 5).

View propagation. Node P sends Node Q its entire view, along with a fresh descriptor of itself. Specifically, P creates a buffer by merging its view with a self-descriptor (timestamped with the current time) and sends the resulting buffer to Q. Upon receiving an update from P, node Q creates a buffer with its own view (plus its own fresh descriptor) and sends the merged buffer back to P. Figure 3.2 illustrates this procedure with two peers, P and Q. At round 5, node P randomly selects node Q and exchanges descriptors with it. In this example, view descriptors are ordered by timestamp, from oldest (left) to youngest (right).

View update. Both nodes then merge the received buffer with their view and reduce the buffer size to v by retaining the v most recent descriptors with the highest timestamps, while avoiding duplicates. After the exchange, both P and Q have views containing the freshest contacts known to both nodes. The use of timestamps enables Newscast to heal itself: new nodes quickly propagate through the network, and stale descriptors naturally drop out. Empirical studies show that Newscast yields a low diameter and good randomness in the overlay topology.

3.1.3 Elevator

The protocols presented so far, Cyclon and Newscast, construct overlay topologies that approximate random graphs with roughly uniform node connectivity. While random graph topologies offer well-understood resilience and mixing properties, they are not necessarily optimal for all applications. In particular, applica-

tions such as decentralized federated learning benefit from highly connected nodes (hubs), which can accelerate information dissemination across the network. Conventional approaches to hub-based topologies rely on statically designated super-peers [Mon04], which introduce single points of failure and are vulnerable to targeted attacks.

Elevator [LPBTF24] addresses this gap by introducing a new type of peer sampling service called *hub sampling*. Rather than producing uniform random overlays, hub sampling constructs topologies that lie between a random graph and a star network. A small number of nodes organically emerge as hubs through the protocol dynamics, while the remaining connections are maintained randomly. The key idea is to hybridize two attachment strategies within a single gossip-based protocol.

The first strategy is *preferential attachment*, inspired by the Barabási-Albert model [BA99]. In each gossip cycle, a node examines the identifiers in its neighbors' views and identifies the most frequently referenced nodes. It then establishes connections to these popular nodes, up to a predefined number h. When a node receives a connection request, it records the requester in a separate list and responds with a random address from that list, along with its own view. This mechanism causes certain nodes to accumulate incoming connections over successive cycles, progressively elevating them to hub status without any explicit designation or election.

The second strategy is *random attachment*, which operates identically to classical gossip-based peer sampling. Each node maintains $v-h$ connections to randomly selected peers, where v is the node's view size. These random connections ensure that the overlay retains the diversity and resilience properties of a random graph, preventing excessive clustering around hubs and providing alternative paths when hubs fail or depart.

The target topology produced by Elevator has three properties: (i) exactly h nodes act as hubs, where h is a parameter common to all nodes and fixed before the network starts, (ii) the remaining connections (without hubs) follow a random distribution, and (iii) each node maintains v outgoing connections, of which h point to hubs and $v - h$ point to random peers.

Simulation results show that Elevator converges rapidly, typically within 3 to 4 gossip cycles, to the target topology with the desired number of hubs. The protocol demonstrates self-healing properties as lost hubs are replaced without external intervention. Evaluations under churn, crash failures (up to 50% of nodes), and targeted hub removal confirm that the overlay remains connected and recovers its structural properties within a small number of additional cycles. However, the authors of LIFT [LRPBT25] show that even a small fraction of Byzantine nodes, as low as 2% of the total network, is sufficient to subvert the hub selection process.

3.2 Byzantine fault-detection peer sampling

Byzantine detection approaches aim to identify malicious participants within gossip-based peer sampling protocols, mitigating their influence by filtering or penalizing suspicious nodes.

3.2.1 Secure Peer sampling

Secure Peer Sampling (SPS) [JMv10b] extends Newscast (Section 3.1.2) with a prestige-based mechanism to detect hub attacks, where malicious nodes inflate their in-degree to dominate views. Each node maintains two auxiliary descriptor tables in addition to its view.

- A standard view of descriptors (ID, timestamp). Furthermore, to prevent node impersonation, each node signs its descriptor using its private key.
- A Prestige Table (PTABLE): tracks observed peers with the associated tuple $(ts, hits, ttl)$, where ts is the most recent timestamp of the peer, $hits$ counts appearances in received views (prestige measure), and ttl is the time-to-live for this peer in the PTABLE.
- A Whitelist (WLIST): stores low-prestige (likely correct) peers from expired PTABLE entries. It is a list of descriptors (ID, timestamp).

In the example in Figure 3.3, view descriptors are ordered by timestamp, from oldest (left) to youngest (right).

View initialization. SPS inherits the join mechanism from Newscast. First, a new node obtains contacts (e.g., from a bootstrap registry) to establish its view. Then, SPS operates in rounds.

Peer selection. Similar to Newscast, a node selects one peer in its view for view exchange. In addition, in SPS, the node selects one or more additional peers to collect statistics on the prestige (degree) of other nodes. Each round also ages PTABLE entries by decrementing the ttl. When the ttl of a peer reaches 0, it is interpreted as the peer not having been seen recently. For example, in Figure 3.3, the ttl of the descriptor of node F, which has been inserted in node P's PTABLE at round 1, drops to 0 in the current round 5. So its descriptor is moved from PTABLE to WLIST.

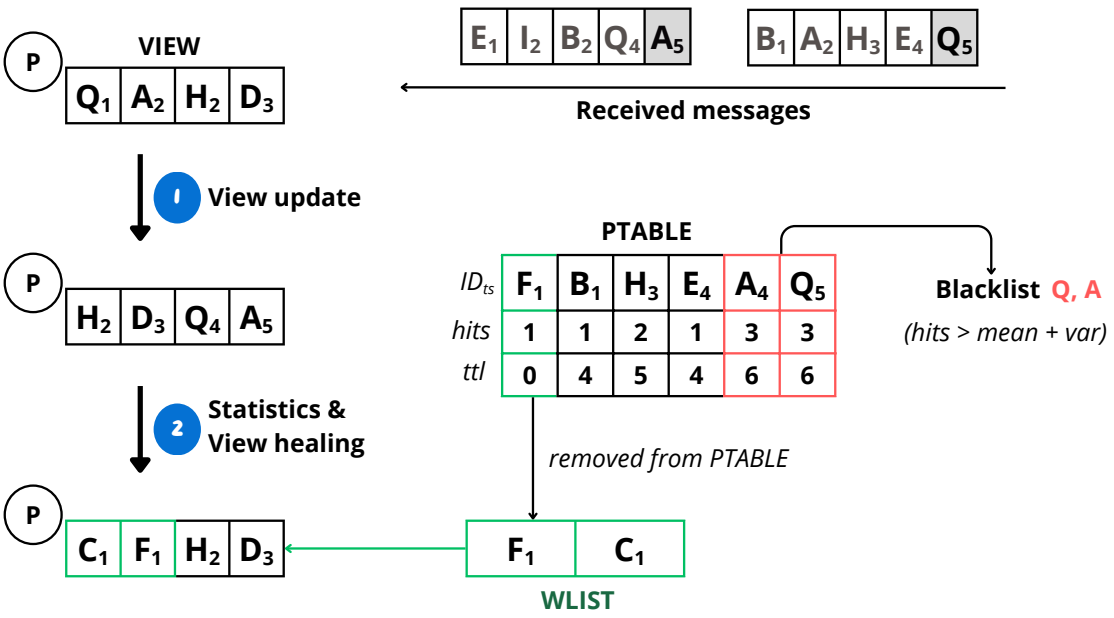


Figure 3.3: Prestige-based view update of SPS (round 5).

View propagation. During each gossip, the same push-pull view exchange occurs as in Newscast. The node sends its view and its own fresh descriptor to the selected neighbor. The neighbor replies similarly. In Figure 3.3, node P receives two views from its exchanges, one from node A, followed by one from node Q. The first reply initiates a Newscast merge operation. So, the node P combines its old view with A's descriptors to form a candidate set for the view update. It is essential to note that, in SPS, the node ignores any incoming message if the creator is blacklisted in the current PTABLE.

View update. Similar to Newscast, the node selects the v newest descriptors from the candidate set to form its view. The other messages received by the node update the local statistics it computes about its neighbors. The node either looks up or creates a tuple in the PTABLE for these message identifiers, updating the timestamp fields to the current time, incrementing the hits counter, and updating the *ttl*. In the example in Figure 3.3, Q's descriptors are used to update P's PTABLE. Descriptors of nodes B and E from Q's view are added (with a *ttl* of 4), while descriptors of nodes Q, A, and E are updated.

After that, a blacklist policy is applied by the protocol. In our example, nodes Q and A are blacklisted because their hit values dominate over the other entries (hit greater than or equal to the mean plus the standard deviation). Whenever an identifier in the view is blacklisted, it is removed and replaced with an identifier

from the WLIST (here, F and C) to maintain a full-sized view.

To handle mistakes, SPS includes a false-positive check. Once per cycle, a node tests a blacklisted peer by exchanging messages with it. If the peer's message contains any other suspects, the suspicion is confirmed. Otherwise, the peer is exonerated and returned to the WLIST. This mechanism ensures that correct nodes are not penalized indefinitely.

Experimental evaluations demonstrate that SPS significantly reduces view pollution. Without this defense mechanism (standard Newscast), 20 malicious nodes in a 1000-node network took over the overlay in approximately 20 cycles, resulting in complete view pollution of correct nodes. With SPS (with two gossip exchanges per cycle), the average pollution fell to approximately 1%. Thus, SPS maintains a nearly random network. However, as shown in [ABF+23], SPS remains vulnerable to a rapid flooding attack managed by a large group of colluding nodes (e.g., 30% of the system). Correct nodes cannot quickly identify and blacklist attackers before they are overwhelmed and isolated.

3.2.2 Secure Cyclon

Secure Cyclon [AV23b] hardens Cyclon (Section 3.1.1) by treating descriptors as cryptographic certificates with ownership chains, preventing forgery and depletion attacks. In Cyclon, each node maintains a fixed-size view of randomly selected peers and periodically performs push-pull gossip exchanges with a neighbor, using a swapping strategy. The view contains descriptors of neighbors that the node has learned about, and each descriptor references a neighbor's identifier, address, and timestamp. Secure Cyclon strengthens this strategy by adding a signature chain to each descriptor, which records its creation and transfer history.

Each node is assigned a pair of public/private keys and creates a descriptor containing its address, timestamp, and public key. When node P advertises itself to node Q, it appends node Q's public key to the descriptor and signs it with its private key to ensure its integrity. Then, node P sends the descriptor to node Q. Each time the descriptor is handed off, the new holder appends a signature. If node A wants to initiate a gossip exchange with node P, it must first return the descriptor to its creator, P, and prove that it is the actual owner. Node P can then redeem (delete) the descriptor, and Node A is prohibited from retaining a copy of the descriptor for reuse.

This chain of signatures makes the descriptor unforgeable. Figure 3.4 presents the different phases of a descriptor's lifespan, from creation to deletion and traversing. If an attacker illegally clones or generates descriptors, the mismatch in signature histories will be evident. Nodes share evidence of any such violation via gossip, enabling the network to collectively detect and blacklist cheaters.

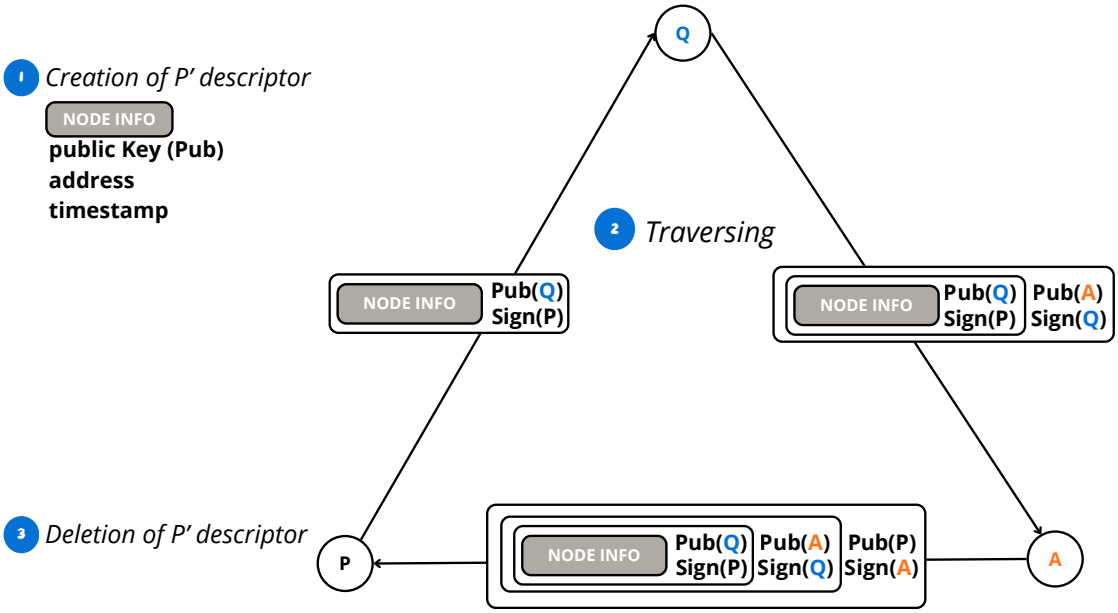


Figure 3.4: Descriptor lifecycle in SECURE CYCLON: creation, transfer, and redemption/deletion.

View initialization. A joining node starts with an empty view. SECURE CYCLON introduces *non-swappable* descriptors to bootstrap such nodes. Initially, the node obtains a small set of valid descriptors (e.g., from a trusted bootstrap peer) and marks them as non-swappable. This means that the node cannot swap them with another node. In other words, it cannot transfer the ownership of these descriptors. These descriptors can only be redeemed and serve to initiate the first gossip exchanges with bootstrap nodes.

Peer selection. During each cycle, a SECURE CYCLON node selects one peer from its view to communicate with. As in basic Cyclon, this is usually the peer corresponding to the oldest descriptor, i.e., the peer with the oldest timestamp. The node must prove ownership of the selected peer's descriptor by presenting its signed certificate.

View propagation. One key enhancement is *the tit-for-tat* exchange. Instead of sending all the requested descriptors at once, nodes transfer them one at a time in lockstep. The initiator sends the first descriptor. If the partner replies, the initiator sends the next descriptor, and so on. If any party quits mid-exchange, the other stops sending. This prevents an attacker from causing the correct peer to lose more descriptors than it gains, thus thwarting *link-depletion attacks*.

During a gossip, two nodes swap descriptors. The initiator sends a batch containing a fresh, signed descriptor of itself and $k-1$ randomly chosen descriptors from its view. The partner replies with k descriptors from its view. SECURE CYCLON adds two security layers here: Each descriptor carries the full signature chain (its "ownership history"), which allows recipients to verify its integrity (Figure 3.4). Each node maintains a small *redemption* cache of recently redeemed descriptors. During gossip, a node includes its cache descriptors as extra "samples" (without transferring ownership). This cache prevents attackers from reissuing an old descriptor undetected. When a descriptor is redeemed, meaning it is returned to its creator, the creator retains a copy for a few cycles. Any reuse of that ID can be cross-checked against the cached version to detect cloning.

View update. After a swap, each node merges the received descriptors into its view. Specifically, the node keeps the v freshest descriptors from all the descriptors it has (the old view, minus the ones it sent, plus those it received). The oldest ones are discarded to maintain a fixed view size.

SECURE CYCLON then runs security checks. If a node notices two descriptors with the same creator ID but inconsistent signature chains, it has proof of misbehavior. The detecting node broadcasts this proof in subsequent gossip, so that eventually all correct nodes see the evidence and blacklist the offender. Once blacklisted, a node's descriptors are ignored by correct peers, effectively removing it from the overlay. Thus, SECURE CYCLON not only repairs the view by dropping malicious links, but also actively purges malefactors from future gossip.

Experimental evaluations [AV23b] show that when 2% of the nodes are attackers, the number of links pointing to malicious nodes temporarily exceeds 2% of all the system's links. However, this number quickly drops back in less than 10 cycles. Moreover, the system recovers even under aggressive attacks. In simulations with up to 40% malicious nodes, SECURE CYCLON quickly detects and removes the malicious links despite an initial spike of up to 90%. Over time, the overlay returns to near uniformity.

One drawback of SECURE CYCLON is that it adds cryptographic computations and tasks such as managing non-swappable flags, caching, and sending redeemed descriptors. Nodes must also track timestamps, keys, and chains for each view descriptor. For very lightweight environments, this can result in significant overhead. Furthermore, once malicious nodes are banished, they cannot participate in further exchanges, resulting in a significant loss of population in the overlay and degrading its connectivity. One alternative would be to tolerate their misbehavior without punishing them. We explore this aspect in the next section.

In conclusion, Byzantine fault-detection protocols identify and exclude malicious nodes in gossip-based peer sampling by embedding verifiable evidence (e.g.,

signatures, counters) into exchanges. This maintains overlay randomness while purging attackers, but requires cryptographic overhead.

3.3 Byzantine fault-tolerant peer sampling

Byzantine fault-tolerant protocols maintain uniform randomness in peer sampling without detecting or excluding malicious nodes. Instead, they use probabilistic techniques (e.g., min-wise permutations) to bound the adversarial influence, even when a significant fraction of nodes are Byzantine. The protocols presented in this section target two distinct objectives. BRAHMS [BGK+08], RAPTEE [PBQY+22], and BASALT [ABF+23] aim to maintain uniform random sampling despite adversarial bias in the identifier stream. LIFT [LRPBT25], by contrast, targets the specific problem of securing hub election in hub-based topologies built by Elevator, using deterministic hub redistribution rather than stream-level debiasing.

3.3.1 BRAHMS

BRAHMS [BGK+08] is the first protocol designed for large, dynamic systems prone to Byzantine failures and Sybil attacks. With high probability, BRAHMS prevents an attacker from creating a partition between correct nodes and ensures that each node's view converges to a uniform random sample over all participating nodes over time. The core ideas of BRAHMS lie in its two components used by all nodes: a push-pull gossip-based communication pattern, and a uniform sampler based on min-wise independent permutations [BCFM00]. Through the gossip component, each node spreads locally known IDs across the system and maintains a dynamic view V of v entries. In the sampling component, each node maintains a *sample list* S of l entries, each of which is uniformly sampled from the received IDs.

Initially, each node maintains a list of node IDs and addresses obtained from a bootstrap node. Periodically, through the gossip component, each node selects $\alpha \times v$ nodes from its dynamic view V to send push messages containing its own ID. It also selects $\beta \times v$ nodes to send pull request messages to, in order to retrieve their views. At the end of each communication round, the sample list S is updated via the sampling component. At the same time, the dynamic view V is renewed by selecting uniformly at random: (i) $\alpha \times v$ IDs received from push messages, (ii) $\beta \times v$ IDs from pull answers, and (iii) $\gamma \times v$ IDs from the sample list (referred to as the *history sample*), with $\alpha + \beta + \gamma = 1$, as shown in Figure 3.5.

The sampling component outputs a list of l IDs from a stream of identifiers obtained via push and pull messages. The component consists of a set of l samplers,

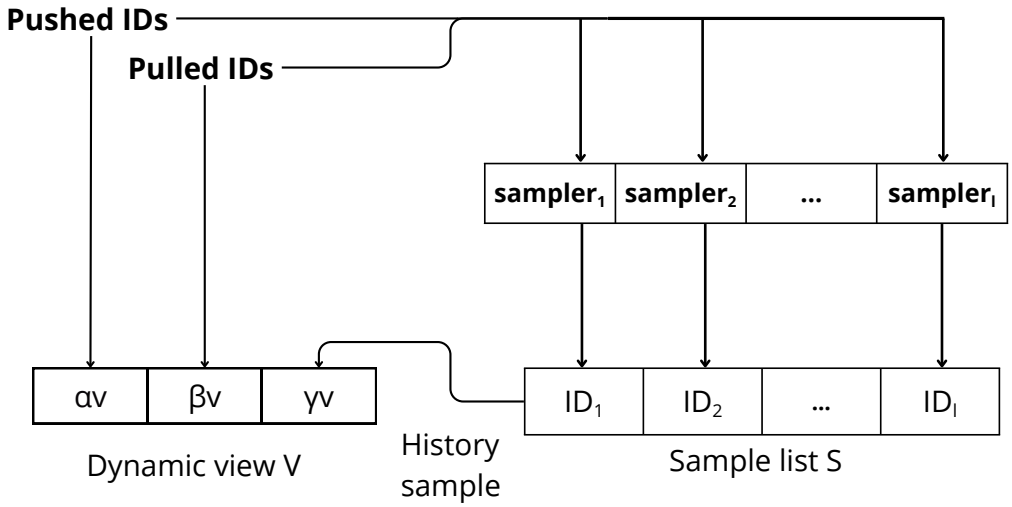


Figure 3.5: Brahms view computation.

each comprising a hash function with a seed selected pseudo-randomly at bootstrap and a locally stored ID. For each input ID, a sampler stores it and outputs the ID that yields the smallest hash value between the current ID and the stored one.

The four defense mechanisms that prevent partitioning and allow convergence to uniform sampling are the following: (i) **limited pushes**, (ii) **attack detection and blocking**, (iii) **controlling the contribution of pulls versus pushes**, and (iv) **history sampling**. We detail these mechanisms below.

(i) An adversary could forge identities or flood the system with push requests, leaving correct IDs propagated mainly through pulls and diminishing their representation exponentially. Brahms assumes a mechanism that limits the message sending rate of nodes, for example, via computational challenges like Merkle's puzzles, virtual currency, *etc.*

(ii) Limiting push messages prevents a simultaneous attack on all correct nodes but does not protect against flooding a targeted node. To do so, Brahms blocks dynamic view updates if more than the expected $\alpha \times v$ pushes are received. This policy slows down progress, but its expected impact in the absence of attacks is bounded, and thanks to limited pushes, some nodes make progress even under attack.

(iii) Node views are threatened by pulls from neighbors more than by adversarial pushes. Pushes from correct nodes are correct, but pull answers from correct nodes may contain some identifiers of Byzantine nodes. Hence, the contribution of pushes and pulls to V must be balanced. Brahms updates V with randomly chosen $\alpha \times v$ pushed IDs to protect targeted nodes, and $\beta \times v$ pulled IDs to protect the rest.

(iv) The attack detection and blocking technique slows down a targeted attack but cannot prevent it altogether. A node victimized by targeted pushes will pull more IDs from Byzantine nodes, send fewer pushes to correct nodes, and see its system-wide representation decrease, leading to fewer correct pushes received. BRAHMS overcomes such an attack using a self-healing mechanism by incorporating in the view an unbiased historical sample of $\gamma \times v$ IDs from the sample list S. Once some correct ID becomes the permanent sample of the node under attack (or if the node's ID becomes a permanent sample of another correct node), the threat of isolation is eliminated.

In short, BRAHMS provides a theoretical guarantee that the output of each correct node will converge to a uniform sample of all nodes, regardless of adversarial pushes into its view. This results in excellent self-healing, as no global partition can form with high probability. However, its resilience breaks down if the adversary controls too many nodes. In fact, later evaluations in [PBQY+22] observed that, when 18% of the nodes were malicious, BRAHMS left 81% of the sampled IDs Byzantine in the correct nodes' views. This shows the need for a more robust, fault-tolerant peer sampling protocol.

3.3.2 RAPTEE

RAPTEE [PBQY+22] is a peer sampling protocol built on top of BRAHMS. In fact, in RAPTEE, nodes are equipped with a push-pull gossip component and a sampling component, with defense mechanisms similar to those in BRAHMS. To improve BRAHMS's resilience, the authors introduced trusted nodes operating on trusted execution environments. These environments have desirable properties, such as authentication, integrity, and confidentiality. These environments ensure that nodes that rely on them execute the protocol using code that cannot be tampered with. RAPTEE assumes that a small percentage of nodes (e.g., 1–5%) have secure enclaves and secretly cooperate to enhance security. These nodes are called trusted nodes.

The type of communication between two nodes is determined by executing a secure mutual authentication protocol. This protocol is executed before issuing a pull request to verify that both parties have the same symmetric key. The communication can be one of three types: *trusted-to-untrusted*, *trusted-to-trusted*, or *untrusted-to-any*. Figure 3.6 illustrates these communication types. Trusted nodes use the same symmetric key to authenticate each other. Failed authentication between two nodes implies insecure communication and the use of the classical BRAHMS algorithm.

The idea behind RAPTEE secure communication is to have trusted nodes perform additional operations alongside the basic BRAHMS pull-gossip steps. First, after successful authentication, two trusted nodes exchange their full views to ac-

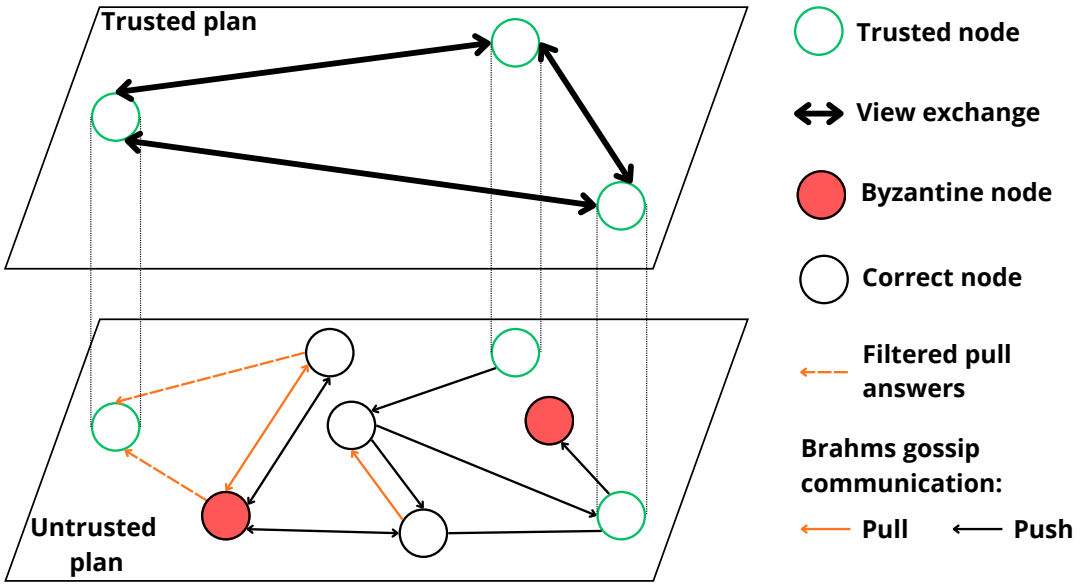


Figure 3.6: Different type of communication in RAPTEE.

celerate the spread of identifiers among them. Second, they filter information from untrusted nodes, which may be biased. These nodes may be Byzantine, or their pull responses may contain Byzantine identifiers. The aim is to slow down the propagation of identifiers received from correct nodes. The protocol defines an eviction ratio, the proportion of pull responses a trusted node uses to update its view during the round. This ratio depends on the proportion of trusted communication that the trusted nodes have made during the round.

As a result, RAPTEE improves BRAHMS' resilience by up to 21% when 10% of nodes are trusted, and 10% are Byzantine. Experimental analyses demonstrate that RAPTEE can withstand attacks on trusted nodes, including the identification of these nodes and the injection of view-poisoned trusted nodes by an attacker. However, RAPTEE's effectiveness is limited because only trusted nodes actively participate in debiasing identifier streams.

3.3.3 BASALT

Similarly to BRAHMS, BASALT [ABF+23] exploits a stubborn chaotic search based on min-wise permutations. Instead of using random samples of received gossip identifiers to update its local view, each node maintains a list of random ranking functions and retains the identifiers that best match those functions. BASALT extends BRAHMS's min-wise independent permutation techniques to build the entire view instead of only a subpart. Figure 3.7 illustrates the BASALT view update and

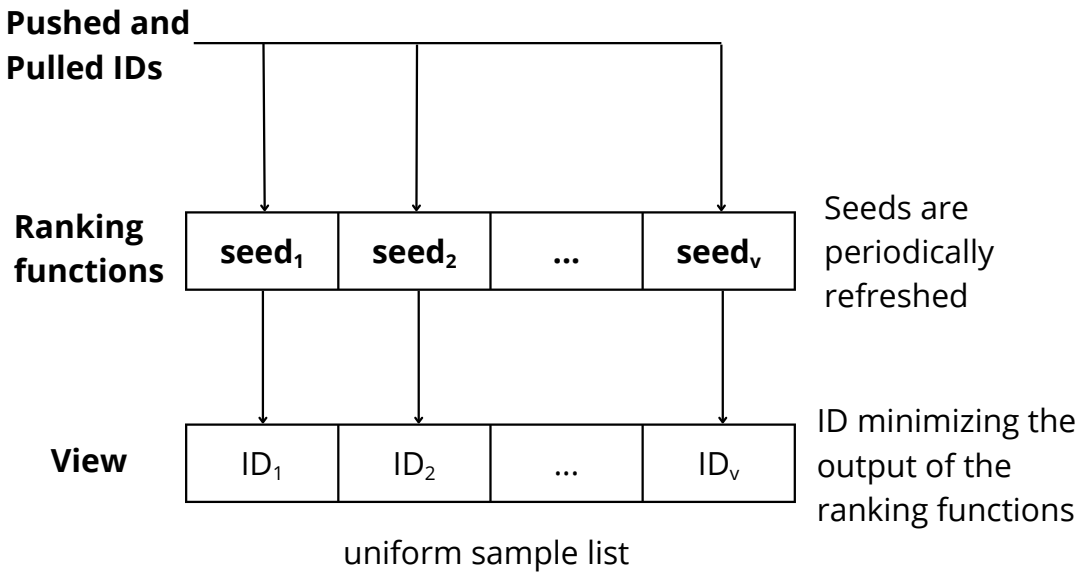


Figure 3.7: Basalt sampling mechanism.

sampling mechanism.

Each node maintains a view of size v and periodically initiates some gossip exchanges with its neighbors. Every τ time units, the protocol randomly selects one neighbor from its view from whom to request a view (pull request) and one neighbor to whom to send its own view (push request). Then the node receives peer descriptors and updates its view as follows. Each entry in the view is associated with a ranking function to which is assigned a seed $seed_i$, with $0 \leq i < v$. Each ranking function is fed by the identifiers of all received descriptors and selects one to store in a view entry. From the initial bootstrap to all gossip exchanges, whenever a ranking function encounters an identifier, it computes its ranking with that function, which is a hash computation. If the ranking of the processed identifier is lower than that of the stored identifier, the processed identifier replaces the stored identifier in the view. The choice of the ranking seed is entirely local to each node, and no other node can interfere with it.

A ranking function that encounters the identifier with the lowest rank across the entire identifier space retains this value for the duration of the protocol. To prevent the sample from becoming static, every ρ time units (ρ is the sampling rate), a node randomly chooses k of its v ranking functions. It renews the seeds of those functions, and then resamples its view by passing its stored identifiers to the new ranking functions. On average, a node's seed is refreshed every v/ρ time units. Refreshing the seeds provides fresh identifiers from the sampling service and avoids the static behavior of the sampling component in Brahms.

Evaluations in the BASALT paper confirm that it outperforms state-of-the-art solutions such as BRAHMS with refreshed sampler seeds and SPS (described in Section 3.2.1).

3.3.4 Lift

As discussed in Section 3.1.3, Elevator constructs hub-based overlay topologies through a preferential attachment mechanism where hub selection is driven by observed connectivity patterns. LIFT [LRPBT25] showed that Byzantine nodes can exploit the preferential attachment mechanism by aggressively disseminating their own identifiers during gossip exchanges, leading to a successful hub attack. Once a Byzantine node occupies a hub position, it is connected to the entire network and can intercept, manipulate, or drop messages at scale.

To address this vulnerability, LIFT introduces a one-shot deterministic hub redistribution mechanism. After an initial phase of hub formation using the Elevator protocol, LIFT deterministically derives the set of hub identifiers from the view and uses it to initialize a pseudo-random number generator (PRNG). Then, it generates h new hub identifiers by drawing uniformly from the node identifier space, discarding duplicates and unreachable nodes, and updates the node view with the newly selected hubs. This approach ensures that although Byzantine nodes may successfully infiltrate hub positions in the first phase by flooding the network with their identifiers, they cannot increase their probability of being selected hubs afterward, because the selection does not depend on observed popularity. The redistribution is performed exactly once, after which the network resumes running the standard Elevator protocol.

Simulation results show that LIFT reduces adversarial influence and remains effective even with up to 5% of nodes in the network being Byzantine. Under these conditions, the protocol maintains its target topology and prevents Byzantine nodes from capturing hub positions. However, as highlighted by experiments involving more malicious nodes, the one-shot nature of the redistribution is a structural weakness, as the attack persists and the number of Byzantine hubs increases.

Finally, it is important to note that LIFT, like Elevator, targets a topological objective (hub-based overlays) that is fundamentally different from the uniform random sampling objective pursued by the protocols discussed earlier in this chapter. This distinction positions LIFT as a complementary line of research rather than a direct competitor to the tolerance-based protocols for uniform peer sampling.

3.4 Conclusion

As distributed systems have grown in adversarial exposure (e.g., open P2P networks, public blockchains), research has progressively expanded beyond crash-fault designs to include more resilient peer-sampling services. Crash-fault peer sampling protocols, such as Cyclon and Newscast, assume unbiased collaboration among nodes and focus on network properties such as scalability, fast mixing, and low overhead. Byzantine fault-detection peer sampling protocols, such as Secure Peer Sampling and SECURE CYCLON, embed cryptographic evidence into view exchanges to detect and exclude malicious nodes from the overlay. Byzantine fault-tolerant peer sampling protocols, such as BRAHMS, RAPTEE, and BASALT, do not rely on detecting attackers but just tolerating them.

Byzantine-tolerant peer sampling protocols preserve sampling quality by design, even under significant adversarial pressure, through probabilistic filtering of received streams. In Chapter 4, we introduce a novel protocol that enhances tolerance to severe Byzantine attacks, built on top of the Brahms strategy (Section 3.3.1).

Chapter 4

AUPE: Collaborative Byzantine fault-tolerant peer-sampling

Peer sampling is a crucial primitive in distributed systems, used to manage overlays and disseminate information in large-scale scenarios such as permissionless blockchain systems. Malicious actors often target these protocols to disrupt higher-level protocols. In this chapter, we introduce AUPE, another extension of BRAHMS (presented in Section 3.3.1) designed to withstand Byzantine attacks even when a significant number of nodes are compromised. The AUPE protocol enhances Byzantine resistance by equipping nodes with a *Set Cleaner* that consists of two key components: a tracking component that records the frequency of identifiers (IDs) received during view propagation, and a debiasing component that transforms a given set of IDs into one that more closely matches the expected uniform distribution of the IDs observed thus far. This reduces the likelihood that frequently encountered nodes will be resampled.

Similarly to RAPTEE, AUPE also benefits from the presence of some nodes running on a trusted execution environment. In this work, we consider Intel SGX, but any TEE technology providing similar code integrity and remote-attestation properties would be a fit for our approach, *e.g.*, ARM's TrustZone [PS19]. In AUPE, trusted nodes track, in a distributed and collaborative fashion, the dissemination of identifiers in the system by merging their respective tracking components. This merge strategy allows them to quickly gain a better understanding of the frequency of node ID advertisements across the overall system. It provides a broader information base for operating the debiasing mechanism. Hence, trusted nodes collectively serve as a source of less biased views for the rest of the system. Our simulation results, including 10,000 nodes, show that AUPE achieves near-perfect resilience with 26% of malicious nodes, where the average percentage of Byzantine samples in correct nodes' views converges to the proportion of malicious nodes in the system.

In section 4.1, we describe our system model and objectives as well as the attack model of our adversary. In Section 4.3, we describe AUPE and detail how it complements BRAHMS with its debiasing approach and Section 4.4 presents the experimental methodology used to evaluate the performance and resilience of AUPE under different configurations and discusses the results. We review the state of the art in Section 4.5 before concluding in Section 4.6.

4.1 System model.

We consider a system of N active nodes, each identified by a unique ID, that communicate via a routed network. This system runs a gossip-based peer sampling protocol for peer discovery, operating in rounds. We assume that the system is at time T_0, when no nodes join or leave. Each node has a local view of l_1 entries that represent its neighbors.

The system is composed of a fraction $f < 1$ of malicious nodes managed by an attacker trying to over-represent itself, a fraction t of nodes operating on a trusted execution environment, referred to as *trusted nodes*, and a fraction $h = 1 - f - t$ of correct nodes following the instructions of the peer sampling protocol AUPE.

Correct nodes execute AUPE, the extension of the iconic BRAHMS peer-sampling protocol. AUPE relies on both the push-pull gossip and sampling components of BRAHMS as well as on its defense mechanisms. Push messages include only the sender's ID, while responses to pull requests contain the full view of the requested node. *AUPE nodes* integrate a local component, coined *AUPE's set cleaner*, responsible for correcting the representation bias in favor of Byzantine nodes that exist in the sets of received IDs at the end of each round. Similarly, trusted nodes adopt the same behavior as correct nodes. In addition, interactions among trusted nodes are designed to boost their set cleaner performance and help them more effectively debias their sets of received IDs, thereby providing slightly less biased membership information to correct nodes.

4.2 Attack model and design goal

We consider an adversary controlling all Byzantine nodes, aiming to undermine the system by manipulating the correct nodes' perception of the proportion of Byzantine nodes. The adversary has access to the system's global membership, including Byzantine and correct nodes, but is unaware of the location or number of trusted nodes.

In the original BRAHMS paper [BGK+08], the authors prove that a balanced attack, which spreads faulty pushes evenly among correct nodes, maximizes the

expected system-wide fraction of faulty IDs. Additionally, thanks to its sampling mechanism, BRAHMS sustains targeted attacks in which the adversary attempts to partition the network by targeting a subset of nodes and sending more pushes than in balanced attacks. As AUPE is built on top of BRAHMS and inherits its properties, our adversary advertises only Byzantine IDs to correct nodes via pull request responses and balanced push messages.

We rule out Sybil attacks [Dou02] by relying on the Sybil resilience of BRAHMS that limits the message sending rate of nodes via computational challenges like Merkle's puzzles, virtual currency, *etc.* We assume that trusted nodes can only crash fault. They cannot act maliciously, even though Byzantine IDs can bias their view. Trusted nodes run on Intel's SGX framework, and we trust Intel to certify genuine SGX-enabled CPUs. We also assume that the code running inside enclaves is properly attested before being given access to secrets. We assume Byzantine nodes cannot break cryptographic primitives or read data in the trusted environment of trusted nodes. We also assume that it is impractical for an adversary to extract secrets from enclaves via side-channel attacks, even without physical access or the ability to colocate a process on the target machine. Communications between any two nodes, including trusted ones, are encrypted with symmetric encryption to protect against eavesdropping. Finally, we assume that the adversary does not have global access to communication links that do not involve it and, therefore, cannot infer information from communication patterns.

4.3 The AUPE protocol

This section describes the two main blocks of the AUPE design. First, AUPE provides a local debiasing solution for each node, and second, AUPE enables collaborative debiasing between trusted nodes.

4.3.1 AUPE set cleaner

In BRAHMS, the view of correct nodes is composed of $\gamma \times v$ identifiers from BRAHMS's history sample. This provides Byzantine resilience once enough node identifiers have passed through the sampling component, but only for these $\gamma \times v$ entries. The remaining $(\alpha + \beta)v$ entries are selected directly from the sets of received identifiers, which are biased towards the representation of malicious node IDs due to the Byzantine node attack. To mitigate this problem, AUPE integrates a *Set Cleaner* that is responsible for mitigating this bias during the computation of each node's view, as depicted in Figure 4.1,

At the end of each round, the sets of received identifiers from push and pull requests are injected into the Set Cleaner of AUPE. The Set Cleaner consists of

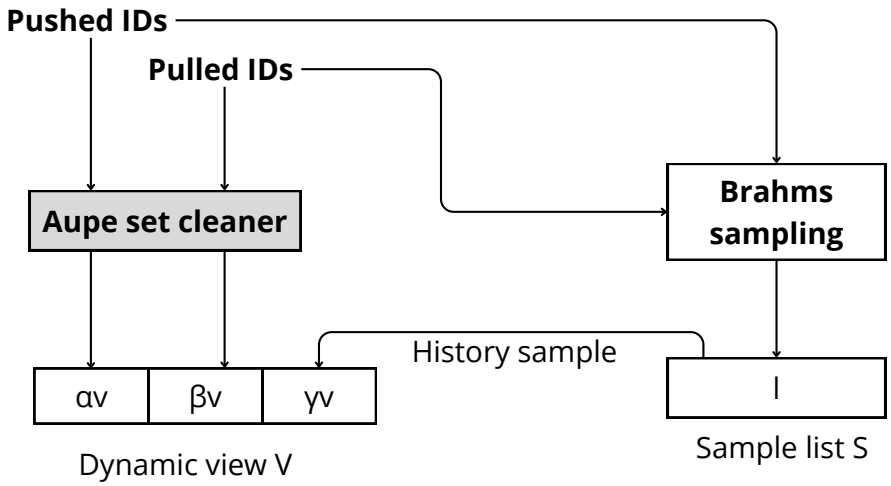


Figure 4.1: Overview of AUPE's view computation.

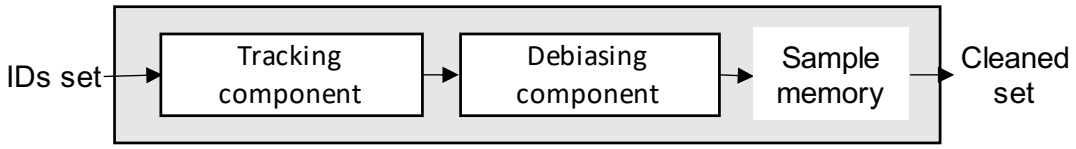


Figure 4.2: Overview of AUPE's set cleaner.

two components: a tracking component that records the occurrences of received IDs from the start of the protocol's execution, and a debiasing component that transforms a given ID set into one that more closely matches the expected uniform distribution of IDs seen up to that point. The output set is used to select the required number of identifiers for the view construction of AUPE. Figure 4.2 depicts the general operations of AUPE's Set Cleaner. In AUPE's case, the two sets resulting from the pulled and pushed IDs are passed through the Set Cleaner, and two sets of $\alpha \times v$ and $\beta \times v$ are created to fill the *push* and *pull* parts of AUPE's dynamic view.

By reducing the bias introduced by the Byzantine nodes in its sets, the AUPE node ends up with less biased views. This, in turn, helps spread less biased information about the system's membership, benefiting other correct nodes.

Tracking component. Each node has a *tracking datastructure* providing the occurrences of received IDs. This is simply a key-value store where the keys are the received identifiers and the values are their occurrences (i.e., the number of times they were received). Each pushed or pulled ID feeds this datastructure throughout the execution of the protocol.

Debiasing component. Debiasing is performed at the end of each protocol round. The two sets of pulled and pushed IDs received are transformed by this component. Its algorithm is detailed in Algorithm 1. We consider an AUPE node that receives a set σ of IDs at the end of round r. For each ID in σ, the debiasing component produces an ID to build the output set σ'. The node maintains a local *sample memory* Γ of size $sm << N$. The purpose of this sample memory is to hold nodes' IDs that will be selected to create the output set σ' for current and future rounds. When processing the input set σ, each encountered node ID j is used to update the node's occurrence table Φ with its current occurrence count Φ_j (line 2). If the sample memory Γ is not already full and does not contain node j's ID, it is added to the sample memory (line 4). If full, the node computes the probability of inserting node j into the sample memory, denoted as p_j (line 6), as the minimum known occurrence learned so far min divided by the actual occurrence of node j, Φ_j. The values min and Φ_j are known from the occurrence table maintained in the tracking component. If selected for insertion in the sample memory Γ, the ID of node j replaces another existing entry chosen uniformly at random.

Finally, an ID is selected uniformly at random from Γ to fill the output set σ' (lines 10 and 11).

Algorithm 1: Local debiasing component for a node receiving the set of identifiers σ.

Input: Input stream σ in round r
Output: Output stream σ'
Data: Set of sm identifiers Γ

1 **for** $j \in \sigma$ **do**
2 $\Phi_j \leftarrow \Phi_j + 1$;
3 **if** $|\Gamma| < sm$ **then**
4 $\Gamma \leftarrow \Gamma \cup \{j\}$;
5 **else**
6 $min \leftarrow \min \Phi_j$; $p_j \leftarrow \frac{min}{\Phi_j}$;
7 **if** $rand() \leq p_j$ **then**
8 choose uniformly at random k_1 from Γ;
9 $\Gamma \leftarrow (\Gamma \setminus \{k_1\}) \cup \{j\}$;
10 choose uniformly at random k_2 from Γ;
11 $\sigma' \leftarrow \sigma' \cup \{k_2\}$;

4.3.2 Secret collaborative debiasing

We provide AUPE's trusted nodes the capacity to recognize their trusted peers hidden in the mass. Further, we improve their behavior relative to correct nodes by allowing them to exchange and aggregate information they have collected about the dissemination of identifiers. In turn, the debiasing performed by trusted nodes is based on a larger amount of information, enabling more accurate cleansing of the received sets of identifiers. Consequently, trusted nodes build less-biased views and serve as sources of more representative samples of the system's membership, thereby benefiting the correct nodes.

Mutual authentication. To allow trusted nodes to verify beforehand whether they are interacting with other trusted nodes, we rely on a secure mutual authentication protocol, executed by all nodes before sending push or pull messages to a chosen neighbor. We assume that each node possesses a symmetric secret key. In AUPE, untrusted nodes generate a random secret key during the initialization phase. In contrast, trusted nodes share a common secret key that is provisioned during the remote attestation phase.

The mutual authentication protocol between two nodes, A and B, operates as follows. First, A generates a pseudo-random number r_A and sends it to B via a cryptographic challenge. In turn, B generates another pseudo-random number r_B and computes the hash of the concatenation of r_A and r_B, $H(r_A \cdot r_B)$, and encrypts it with its own secret key obtaining $[H(r_A \cdot r_B)]_{K_B}$. Then, B sends r_B and $[H(r_A \cdot r_B)]_{K_B}$ to A. Upon reception, A computes $H(r_A \cdot r_B)$ and deciphers $[H(r_A \cdot r_B)]_{K_B}$ using its own secret key K_A. If the two values are identical (*i.e.*, A and B share the same secret key), A can identify B as trusted. Then, A sends $[H(r_B \cdot r_A)]_{K_A}$ to B. Like A, B deciphers this encrypted hash using its own secret key K_B and compares it with $H(r_B \cdot r_A)$. If the two are equal, B can also identify A as trusted, and the protocol succeeds. Note that in this protocol, if one of the two considered nodes is untrusted, no information about the trustworthiness of the trusted node is revealed, keeping the identity of trusted nodes secret.

Remembering trusted peers and aggregating tracking components. We aim to have trusted nodes function as a group of actors who track the dissemination of identifiers they observe, collectively sharing this information to accelerate the detection of nodes attempting to over-represent themselves. Ideally, one could devise a collaborative tracking oracle in which all trusted nodes know one another and share and use a single global tracking component. Although not practically feasible, this oracle would enable instant tracking of identifier propagation within the system. Essentially, the oracle represents a configuration in which all trusted nodes merge their tracking components during each round.

With AUPE, we aim to approximate this oracle by employing well-known gossip-based aggregation techniques [JMB05a]. The key is to merge tracking components in a commutative and associative manner between subsets of trusted nodes in every round. Based on the work of Jelasity *et al.* [JMB05a], we select the average operation as the candidate for merging tracking components. Eventually, with sufficient merge operations between trusted nodes, they will possess tracking components that expose the same information. The merge operation averages the corresponding entries for each key in the key-value store.

Since trusted nodes play a crucial role in accelerating AUPE's debiasing operations, each trusted node maintains a *trusted peer list* of the last M trusted node identifiers it has contacted. This list is updated as the trusted node contacts other trusted peers by replacing the oldest entry with each newly acquired trusted node identifier. In each round of AUPE, each trusted node contacts the M peers on its trusted peer list, sending its tracking component before receiving theirs. The trusted node then merges each received tracking component with its own. To ensure that the identities of trusted nodes remain undetectable, correct nodes perform similar operations, *e.g.*, maintaining a random list of M identifiers and contacting them in every round. However, they do not merge as they cannot successfully execute the mutual authentication protocol.

4.4 Evaluation

We compare the Byzantine-tolerance of AUPE against its competitor BRAHMS, as well as BASALT (presented in 3.3.3). As described in the initial BASALT paper [ABF+23], we consider the value of 1 reset seed per round as the refresh rate, implying that each seed is reset every v rounds on average.

We used a simulation infrastructure implemented in Rust to conduct a simulation campaign and assess the extent to which AUPE, BRAHMS and BASALT provide resilience against Byzantine nodes attempting to manipulate the view of correct nodes by over-representing themselves. Our simulation infrastructure is also based on the official BASALT implementation, available on GitHub [Auv20]. To assess each protocol's resilience, we measure the average proportion of malicious identifiers in the local views of non-Byzantine nodes at the end of each simulation run. We run simulations on a system with $N = 10,000$ nodes. Views are composed of $v = 160$ entries. We set the parameters α, β and γ to $1/3$, and $l = 160$. Byzantine nodes run the attack described in section 4.1. Each simulated configuration executed over 200 rounds represents a system with $f \times N$ malicious nodes, $t \times N$ trusted nodes, and $(1-f-t) \times N$ correct nodes. Once over, we collect each node's logs and tear down the simulator.

Our evaluation of AUPE considers a key-value approach for the tracking com-

ponent and a sample memory of size 100 for the debiasing component. To evaluate the effectiveness of AUPE's debiasing strategy alone, we first consider a system with no trusted nodes. We refer to this AUPE variant as *Aupe-simple*. We vary the proportion f of Byzantine nodes from 8% to 50% in increments of 2% and compare *Aupe-simple* against its two competitors. Figure 4.3 depicts the evolution of the average proportion of Byzantine identifiers in the views of non-Byzantine nodes for each protocol as f increases.

As expected, *Aupe-simple* always provides views composed of fewer Byzantine IDs than BRAHMS. While BRAHMS generates views polluted, on average, by 77% Byzantine IDs when $f = 26\%$, *Aupe-simple* only reaches 46%. This is because *Aupe-simple* identifies frequent IDs in the sets of pushed and pulled IDs and selects them less often than infrequent IDs for the construction of its push (*i.e.*, $\alpha \times l_1$) and pull (*i.e.*, $\beta \times l_1$) view subparts. *Aupe-simple* also surpasses BASALT in all configurations and goes beyond the limit reached by BASALT: when $f > 24\%$, BASALT output views composed of more than 90% of malicious IDs, while *Aupe-simple* keeps the proportion of Byzantine IDs below, and only reaches about 80% of malicious IDs in its views when $f > 32\%$.

To better understand the impact of AUPE's debiasing strategy, Figure 4.4 shows the evolution over time of the proportion of Byzantine IDs in the views of non-Byzantine nodes for $f = 26\%$. In addition, Figure 4.5 shows the evolution of the proportion of Byzantine IDs in the push, pull, and history sample view subparts for $f = 26\%$. At the end of this configuration's simulation, *Aupe-simple* provides views containing 46% malicious IDs, whereas BRAHMS reaches 77%. Detailed in Figure 4.5, the push and pull view subparts of *Aupe-simple* are significantly less polluted than those of BRAHMS, 31% and 30% of Byzantine IDs, respectively, against 92% and 96% for BRAHMS. In turn, this also positively affects the history sample view subpart, as *Aupe-simple* nodes can discover new correct nodes faster than in BRAHMS.

To study the impact of introducing trusted nodes that cooperate to track the spread of IDs in the system, we first capture the potential of this collaborative approach by evaluating the oracle described in section 4.3.2, referred to as *Aupe-oracle* in the following figures. Additionally, we examine three variants of AUPE by introducing $t = 10\%$, $t = 20\%$, and $t = 30\%$ of trusted nodes in the system, respectively. The number of remembered trusted nodes, denoted by M, is set to 10 to limit the number of merges of tracking components per round. Figure 4.6 illustrates the evolution of the average proportion of Byzantine IDs in the views of non-Byzantine nodes for each protocol as f increases.

Furthermore, Figure 4.7 reveals the impact of AUPE's collaborative debiasing on each view subpart, *e.g.*, push, pull, and history sample. Comparatively, *Aupe-oracle* outperforms all other protocols, providing views with a proportion of

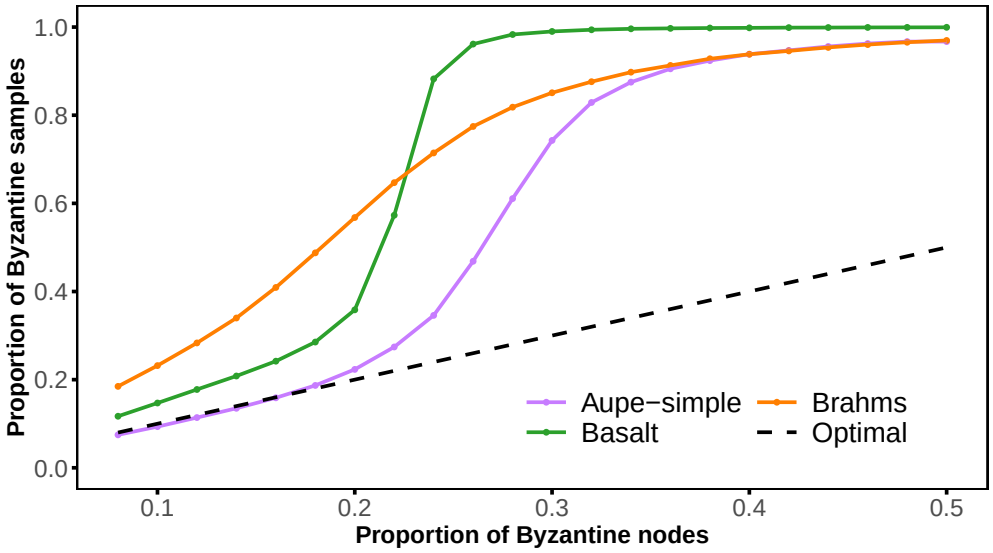


Figure 4.3: AUPE's Byzantine resilience.

Byzantine samples close to the optimal until f exceeds 40%. However, while its push and pull view subparts are already optimal, the history sample view becomes polluted with Byzantine IDs that propagate faster than correct and trusted IDs.

As the proportion of trusted nodes increases, AUPE's variants become more

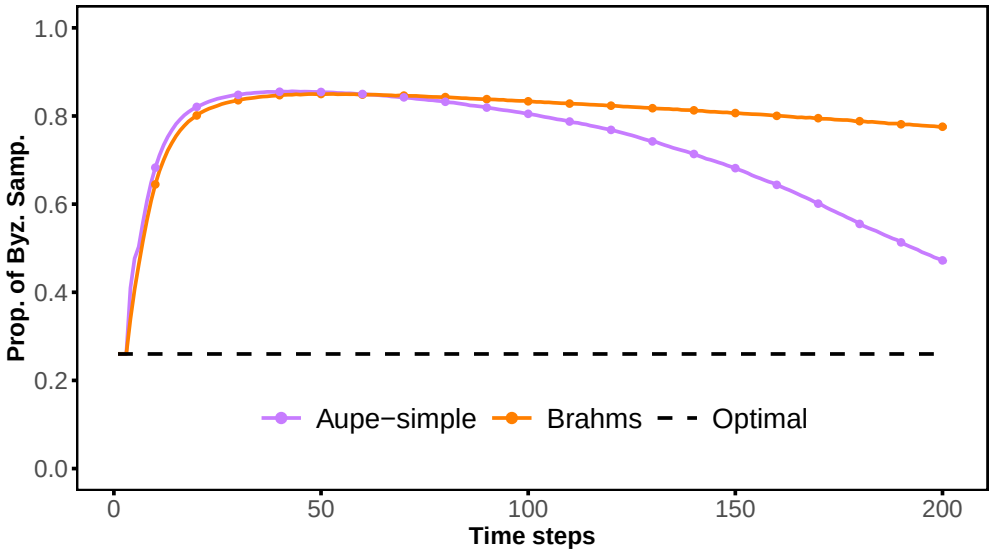


Figure 4.4: Evolution of proportion of Byzantine samples in the system, for $f = 26\%$.

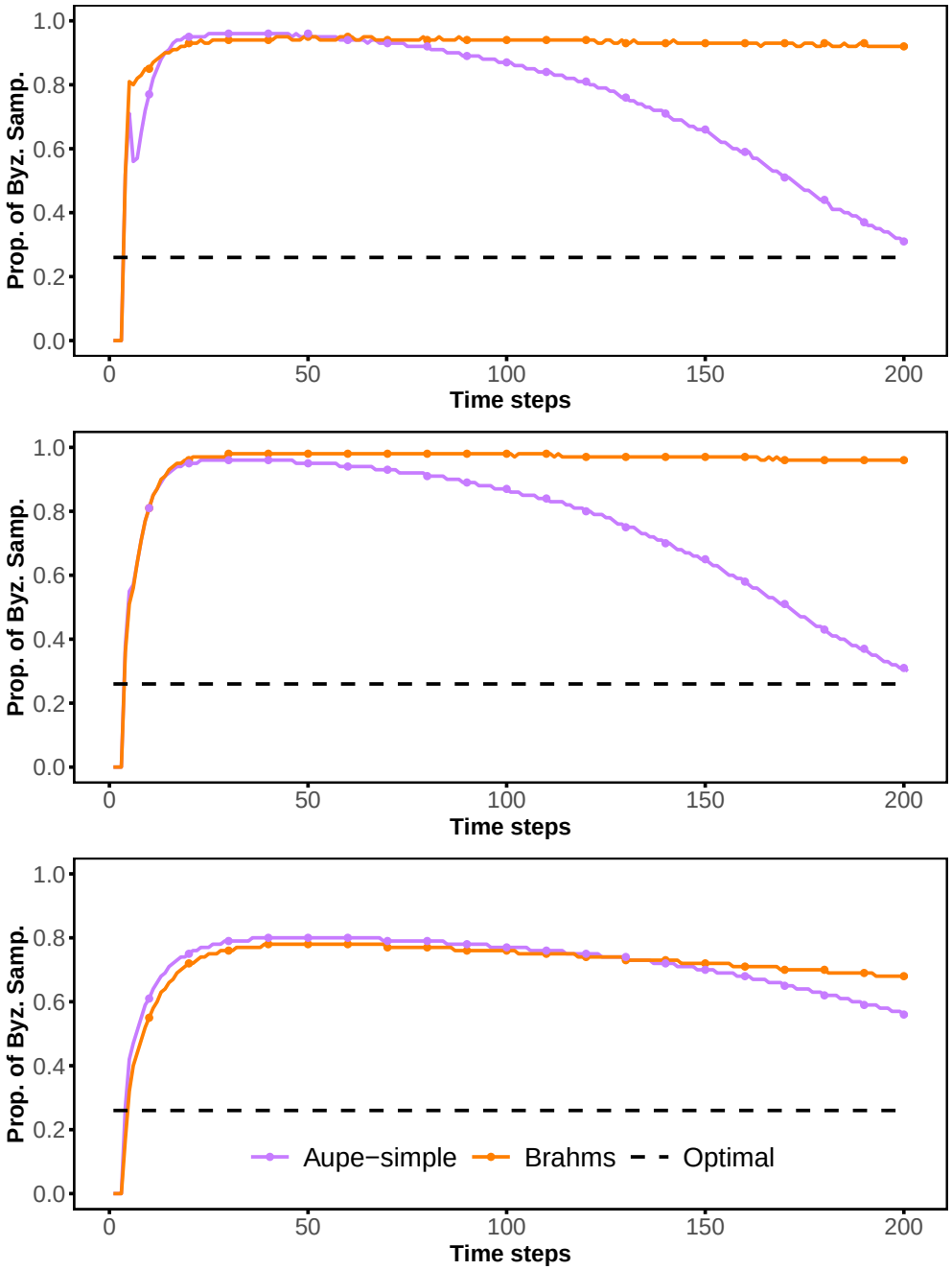


Figure 4.5: Evolution in time of Byzantine sample proportion inside the push view subpart (top figure), pull view subpart (middle figure), and history sample view subpart (bottom figure) for $f = 26\%$.

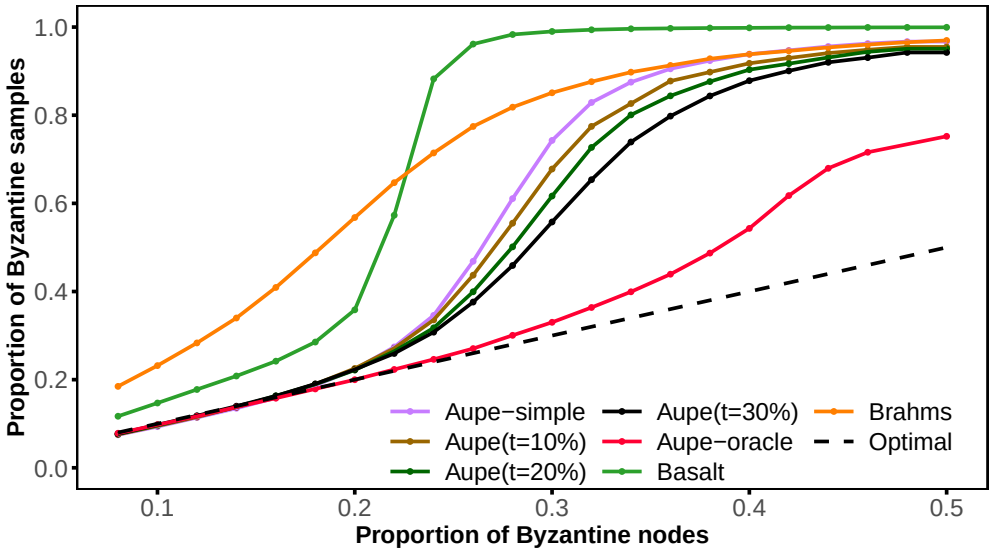


Figure 4.6: Impact of collaborative debiasing on AUPE.

resilient than *Aupe-simple*. This is due to the high effectiveness of the collaborative tracking strategy, as depicted in Figure 4.7. The push-and-pull view subparts of each of AUPE's variants become significantly less polluted when trusted nodes are added to the system.

In Figure 4.8, the graph illustrates the resilience gains of AUPE compared to BRAHMS. This refers to the percentage reduction in Byzantine samples as seen by non-Byzantine nodes. Overall, for values of $f < 24\%$, AUPE provides up to 60% resilience gains over BRAHMS. As f increases from 24% to 40%, the adversary's attack becomes more effective, causing the resilience gains to gradually drop to 0% at $f = 40\%$. In these configurations, the specific role played by trusted nodes becomes evident. The presence of more trusted nodes helps mitigate the reduction in resilience gains. With $f = 30\%$, incorporating 30%, 20%, and 10% of trusted nodes reduces the proportion of malicious IDs in AUPE's views by 34%, 27%, and 20%, respectively, in comparison to BRAHMS.

Figure 4.9 shows the evolution of the proportion of Byzantine IDs in the view of non-Byzantine nodes over time for $f = 30\%$. While each of AUPE's variants initially exhibits similar behavior, increasing the proportion of trusted nodes helps them gather more information about the spread of Byzantine IDs over time. This, in turn, enables them to reduce the proportion of Byzantine samples in their views and, consequently, in the views of correct nodes.

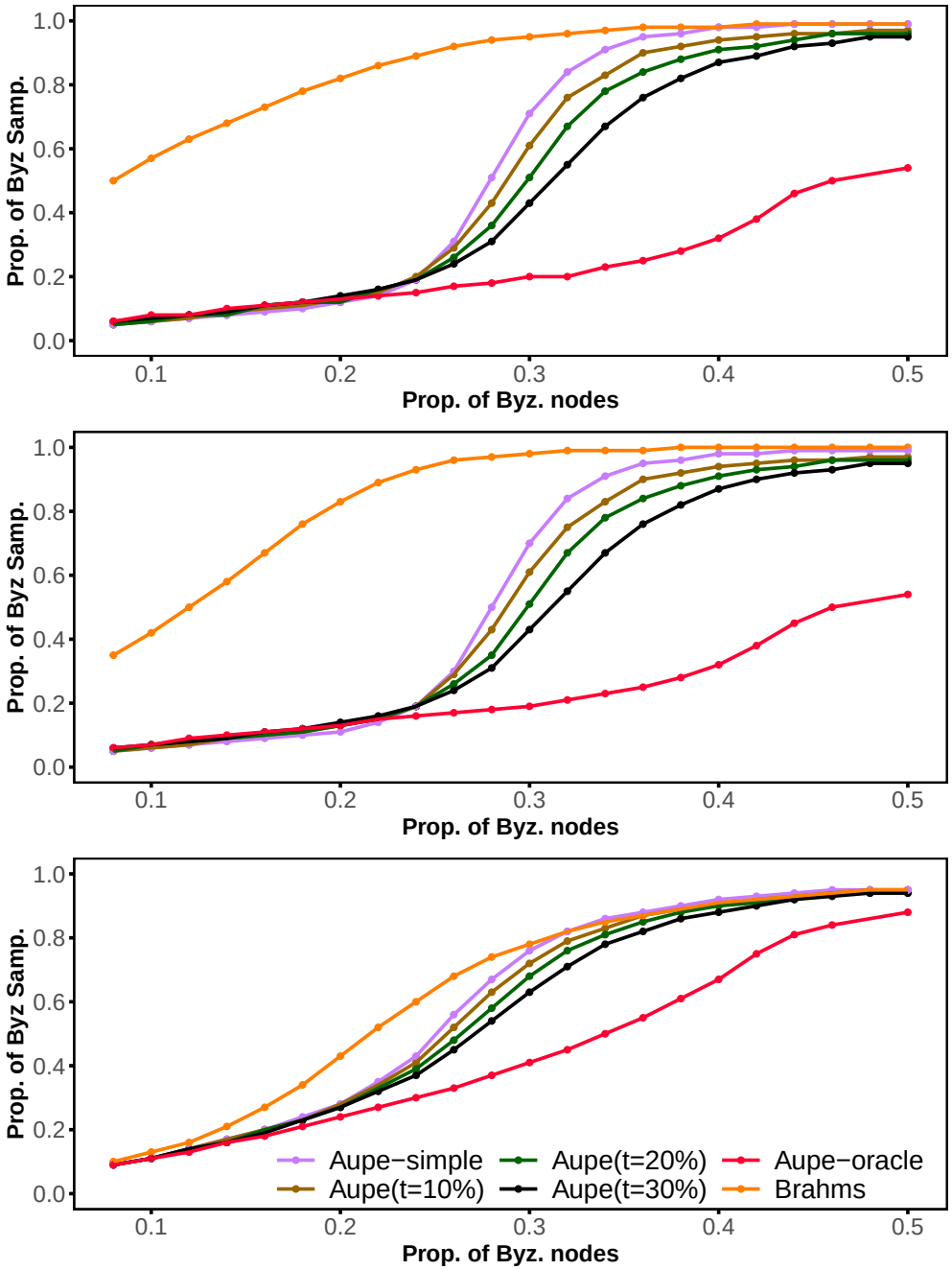


Figure 4.7: Byzantine sample proportion inside the push view subpart (top figure), pull view subpart (middle figure), and history sample view subpart (bottom figure) when varying f.

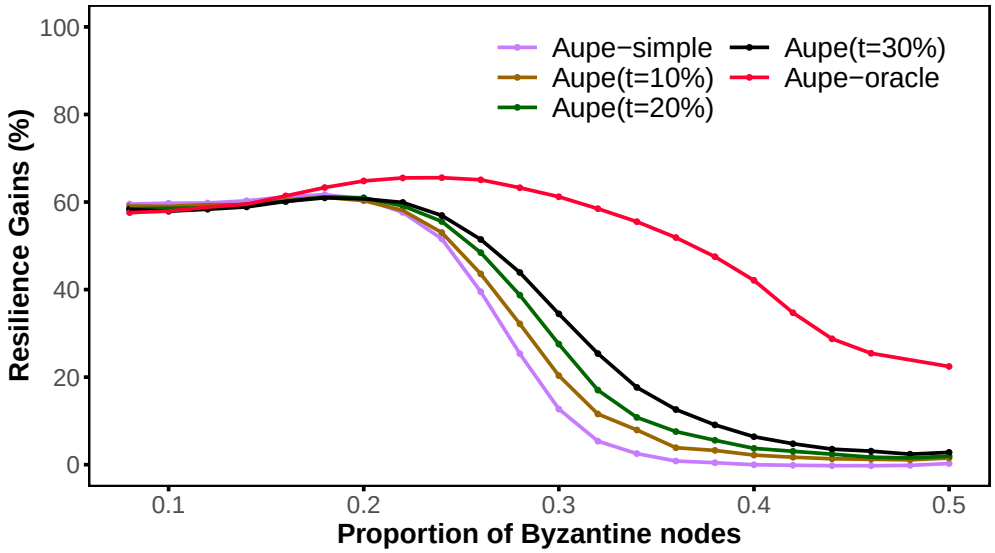


Figure 4.8: AUPE's resilience gains compared to BRAHMS (the higher the better).

4.5 Discussion

The seminal work BRAHMS [BGK+08] employs a min-wise permutation technique [BCFM98] to ensure that a subpart of the view is composed of an actual uniform sample of the set of identifiers each node has seen. Additional coun-

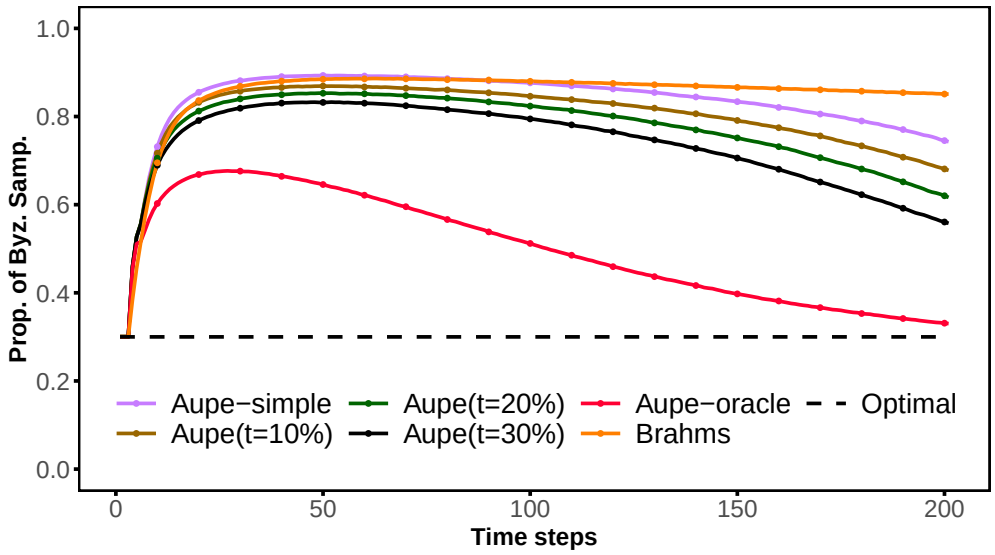


Figure 4.9: Evolution of merge for 200 rounds. $f = 30\%$.

termeasures to specific Byzantine attacks complement the approach to prevent node isolation and eviction. Similarly to BRAHMS, BASALT [ABF+23] exploits a greedy epidemic procedure towards random nodes that are implicitly defined using min-wise independent permutations. In BASALT, each node composes its local view from the current state of this greedy epidemic procedure, extending the use of BRAHMS's min-wise independent permutation techniques to build the entirety of the view instead of only a subpart. The authors of RAPTEE [PBQY+22] introduced trusted nodes operating on trusted execution environments to improve the resilience of BRAHMS. Communications between the different types of nodes (honest-to-honest, honest-to-trusted, and trusted-to-trusted) are designed to accelerate the spread of identifiers among trusted nodes and to slow their propagation from honest nodes (which possess slightly more polluted views) to trusted nodes. As a result, RAPTEE improves BRAHMS' resilience by up to 21% when 10% of nodes are trusted, and 10% are Byzantine. Similar to our work, trusted nodes in RAPTEE can serve as sources of slightly less biased views for honest nodes. In our work, we go one step further by enabling collaboration among trusted nodes, allowing them to share and collectively use the knowledge they have gained about the actual dissemination of identifiers. Doing so, AUPE outperforms RAPTEE by providing up to 60% of resilience gains for $f \leq 24\%$ while mitigating the impact of the adversary's attack, even when having control over 40% of the nodes.

4.6 Conclusion

We introduced AUPE, the first Byzantine-tolerant random peer sampling protocol that utilizes collaborative trusted debiasing. The inclusion of trusted nodes in the system model enabled collaborative tracking of the spread of identifiers and local debiasing of Byzantine nodes within the set of received descriptors. Through simulations involving 10,000 nodes, AUPE demonstrated superior resilience compared to state-of-the-art solutions. It exhibited near-perfect resilience, even in the presence of an adversary controlling 26% of the nodes. By incorporating as few as 10% trusted nodes, AUPE increased BRAHMS' resilience by up to 60% and mitigated the impact of the adversary's attack, even when the adversary controlled 40% of the nodes. However, maintaining an array for tracking requires memory that grows linearly with the number of identifiers in the system. This raises the question of memory-efficient alternatives, such as sketches.

Chapter 5

Robust Frequency Estimation for Peer sampling in Adversarial context

In large-scale, dynamic systems, peer sampling services provide each node with samples of other active peers in the network. Ideally, these samples are drawn uniformly at random from the entire system. These samples are continuously and locally obtained at each node from the sequence of neighbor identifiers (IDs) it receives, also known as a *stream*. However, as previous chapters have shown, these protocols are vulnerable to Byzantine attacks by malicious nodes that overrepresent themselves in the correct node streams. This causes the streams to become biased toward Byzantine IDs, favoring the selection of malicious nodes as samples. In this malicious context, to render a uniform random sample of nodes, the authors of [ABG13] demonstrated the necessity of ensuring that correct nodes possess full system membership.

In the previous chapter, we presented AUPE, in which each node maintains an array of per-ID counters to track the frequency of seen neighbors over time. AUPE queries this array to transform the received stream into an output stream that more closely matches the expected uniform distribution of the IDs observed thus far. In fact, AUPE computes the probability of selecting each ID as a sample of the output stream, based on its frequency. Consequently, frequent IDs are selected less frequently than rare IDs.

As a node encounters new neighbors, its array grows, eventually reaching an unlimited size. However, nodes must operate with constrained memory and computational resources, particularly in resource-constrained environments [KSW21]. Consequently, exact tracking is infeasible, motivating the need for compact approximate data structures, known as *sketches*, that can effectively estimate ID frequencies at low cost. In their seminal work [ABS13], authors proposed a space-

efficient sampling algorithm to approximate a uniform stream processed *on-the-fly* from a biased input stream, with bounded error. This approach uses the probabilistic fixed-size data structure known as the Count-Min sketch [CM05], which is fed by the input stream and used to estimate ID frequencies.

However, a series of non-exhaustive studies [DR07, YJZ+19b, SJL+24, LX21] have shown that Count-Min sketches exhibit suboptimal performance in frequency estimation for unbalanced data streams. These studies have also proposed novel sketching strategies to address these limitations. In an adversarial context, an effective sketch must accurately estimate the frequencies of IDs belonging to two classes: high-frequency (often referring to adversarial nodes) and low-frequency (usually belonging to correct or underrepresented nodes), while preserving the relative bias between the two classes present in the stream.

In addition, many existing evaluations of sketches [RD07, DR07, YJZ+19b, SJL+24] use aggregate metrics, such as *average absolute* or *relative error*, to compare the accuracy of different sketching techniques. While applicable in general contexts, these metrics do not directly indicate how well the frequencies of IDs for each class (low-frequency or high-frequency) are estimated, nor whether one class is estimated better than the other. In fact, these indicators would influence how the IDs are sampled and, in turn, the extent to which the adversarial bias is mitigated.

To find effective sketches for frequency estimation in a malicious context, we make the following contributions:

- In section 5.2, we formalize the problem of bias estimation in peer sampling by incorporating the statistical properties of the observed stream by correct nodes during a balanced attack in which malicious nodes try to equally overrepresent themselves. This has led to the development of a new metric, the *bias factor*, that describes the extent to which the class of high-frequency IDs dominates over the class of low-frequency IDs in a frequency distribution.

- In section 5.3, we compare a range of sketches, from established designs such as Count-Min Sketch [CM05] and Lossy Conservative Update [GI11] to recent proposals such as Probabilistic Sketch [LX23] and BitMatcher [SJL+24], in various scenarios and on multiple metrics, in particular on how accurately they preserve the bias factor of an initial frequency distribution while producing frequency estimates.

- In section 5.4, we present experimental results from parametrized synthetic streams, highlighting the trade-offs between memory usage, frequency estimation accuracy, and bias preservation.

Finally, Section 5.5 discusses the results, and Section 5.6 concludes the paper and outlines directions for future work. Our results provide practical insights into

which sketches best support frequency estimation in an adversarial setting, paving the way for more efficient and Byzantine-tolerant uniform sampling algorithms.

5.1 System model

This section formalizes the assumptions and threat model used throughout the paper. We consider a system of active nodes, each uniquely identified, communicating over a routed network, and executing a gossip-based peer sampling protocol, such as those presented in previous chapters. We assume the node identifiers (IDs) are randomly assigned. Each node maintains a list of neighbors of logarithmic size, and periodically exchanges this list with its neighbors. At the end of each round, the node uses the received IDs to update its list. This infinite sequence of received IDs is called the node stream.

Ideally, a node stream approximates a uniform distribution of the IDs seen so far [JVG+07a]. However, a fraction $f < 1$ of the system nodes are malicious and controlled by an adversary. This adversary may actively tamper with the stream of any correct node by sending disproportionate messages and false information to promote the selection of IDs from their malicious group of nodes for updating the correct node's view. Designing Byzantine fault-tolerant sampling algorithms requires mitigating bias in the streams from correct nodes.

Specifically, we consider balanced attacks (see Section 2.6) in which Byzantine nodes distribute the IDs of their group members evenly across all correct nodes in a round-robin manner. This ensures that each correct node receives as many Byzantine IDs as possible while avoiding detectable local concentrations. To counter this type of attack, prior work [ABS13] and AUPE propose tracking the frequencies of received IDs from the beginning of the sampling protocol. Then, they adjust the sampling strategy to downweight overrepresented IDs (likely Byzantine) and restore uniformity to the sampling process.

5.2 Stream modeling

In this section, we characterize the stream received by a correct node under a balanced attack. We also define a metric, the *bias factor*, to quantify the extent to which this stream is biased by Byzantine IDs. To make the notion of uniform sampling meaningful, we assume that the stream is received starting at time t_0, after which no new peers are discovered and no known nodes are removed from the system (*i.e.*, churn ceases). The stream is received and processed in one pass by a sampling algorithm.

Let $E = \{x_0, x_1, \ldots, x_{N-1}\}$ be the set of IDs for all nodes in our system.

This set is composed of Byzantine IDs from the subset $E_B = \{x_0, \ldots, x_{F-1}\}$, with $F = fN$, and correct IDs from the subset $E_C = \{x_F, \ldots, x_{N-1}\}$, without overlapping nor gap between these subsets.

We model the *stream* S as a sequence of M identifiers drawn from E, denoted by $S = (S(k))$, where $0 \leq k \leq M-1$ and $S(k) \in E$. Under a balanced attack, each ID $S(k)$ in the stream S appears with probability w if $S(k) \in E_B$ and with probability $1-w$ if $S(k) \in E_C$, where $0 \leq w \leq 1$.

The stream S can be represented as a mix of two independent and identically distributed substreams S_B and S_C with their IDs belonging respectively to the subset E_B and to the subset E_C. These substreams follow a uniform distribution over their respective subsets.

In a fault-free setting, *i.e.*, without overrepresentation of malicious nodes, w is approximately equal to $w_{unif} = |E_B|/N$ and each ID is encountered with probability $1/N$ in the stream S. Under attack, w is much higher than w_{unif}, which leads to Byzantine identifiers having higher per-identifier frequencies than correct ones.

To quantify this imbalance, we define the *bias factor* γ as the ratio of the probability of sampling an ID of the class of Byzantine nodes over the probability of sampling an ID of the class of correct nodes. It is computed as:

$$\gamma = \frac{w/|E_B|}{(1-w)/|E_C|}$$

Given the empirical frequency distribution ϕ over S, this ratio can be approximated as the ratio of the average frequency of a Byzantine ID to that of a correct ID:

$$\gamma = \frac{\sum_{k=0}^{F-1} \phi(x_k)/|E_B|}{\sum_{k=F}^{N-1} \phi(x_k)/|E_C|}$$

where $x_k \in E$ and $\phi(x_k)$ is its observed frequency.

A value of $\gamma \approx 1$ indicates no bias (uniformity). Otherwise, Byzantine identifiers are overrepresented. This metric provides a precise way to evaluate the severity of bias and to identify suitable sketches for estimating ID frequencies while preserving the bias.

5.3 Sketching strategies for peer sampling

Maintaining an exact frequency array is not scalable in large systems. Instead, probabilistic data structures known as *sketches* provide a memory-efficient alternative, offering approximate frequency estimates within rigorous error bounds using only a fixed amount of memory. Each node maintains its own sketch and feeds

every peer ID it receives into it during gossip rounds. Over time, IDs belonging to Byzantine nodes that flood the network will accumulate abnormally high estimated frequencies.

Each sketch supports two core operations: (1) *update*, which modifies the sketch to reflect the arrival of a new occurrence of an ID, and (2) *query*, which returns an estimate of an ID's frequency. Classical sketches, such as Count-Min sketch [CM05], provide lightweight baselines for frequency estimation with strong guarantees. However, their basic structure does not fit well with distributions that have a small proportion of dominant IDs (unbalanced distributions), leading to wasted memory or reduced accuracy, especially for low-frequency IDs [GDI11].

More recent approaches have evolved to account for the distribution of ID frequencies when designing the sketch. These approaches can be divided into two categories: (i) *hierarchical-based* sketches [RKA16, YZJ+17, YJL+18a, YJZ+19b], which structure the sketch into many components or layers to track ID frequencies depending on their deduced class (low- or high-frequency). Cold Filter [YJZ+19b] is one of the most accurate approaches in this category. (ii) *self-adjusting based* sketches [BEMV21a, ZLT+21, SJL+24], which adapt the sketch counter sizes on the fly to fit the distribution. As shown in their paper, BitMatcher [SJL+24] is the most prominent in this category.

The Probabilistic sketch [LX23] offers a novel approach to this challenge by computing the frequency of an ID based on its probability of appearing.

A thorough investigation of these sketches offers a dual perspective, encompassing both historical insights and contemporary advancements. This section describes and compares several sketching techniques suitable for estimating frequency distributions for unbalanced streams. The design principles of these methods will be outlined, including their update and query mechanisms and their error guarantees.

5.3.1 Count-Min Sketch

The Count-Min Sketch (CMS) [CM05] is a hash-based structure that estimates frequencies with guaranteed upper bounds. It consists of a $d \times w$ matrix of counters, denoted Y, where d is the number of hash functions (depth) and w is the number of counters per row (width). Each row is paired with an independent 2-universal hash function $h_1, \ldots, h_d : \mathcal{U} \to [w]$.

Update. When a node receives ID x during a gossip exchange, it updates the sketch as follows:

$$Y[k, h_k(x)] = Y[k, h_k(x)] + 1, \forall 1 \leq k \leq d$$

This increments exactly one counter in each of the d rows (see Figure 5.1).

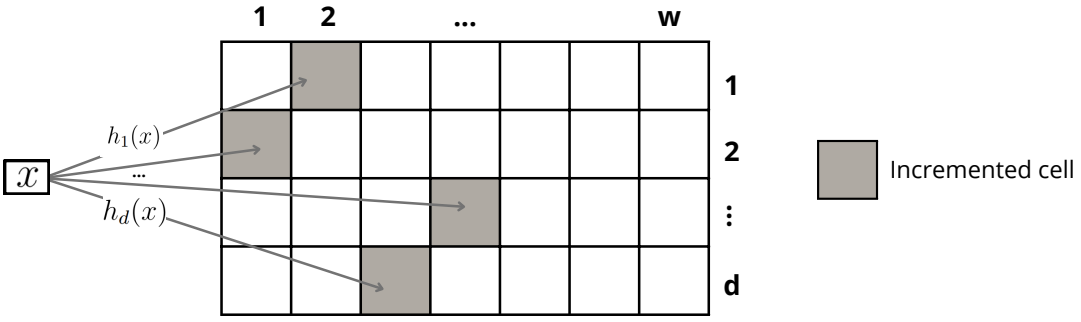


Figure 5.1: Update operation in the Count-Min Sketch.

Query. The estimated frequency $\hat{\phi}(x)$ of ID x is obtained by taking the minimum value among the d associated counters:

$$\hat{\phi}(x) = \min_{1 \leq k \leq d} Y[k, h_k(x)]$$

Error guarantee. The CMS guarantees that the estimated frequency is greater than the true frequency by an error of size ε, within $\mathcal{O}(wd)$ space:

$$\hat{\phi}(x) \leq \phi(x) + \varepsilon \cdot M, \text{ with probability at least } 1 - \delta$$

where $\varepsilon = \frac{e}{w}$ with $e = exp(1)$, $\delta = e^{-d}$, and M is the total number of IDs processed so far. ε can be made arbitrarily small by increasing the width w. The standard CMS has been observed to produce substantially inflated counts, particularly for low-frequency IDs, due to collision issues [GDI11].

An additional advantage of the Count-Min Sketch is its *mergeability*: the pointwise sum of two sketches is equivalent to the sketch of the concatenated streams processed by each sketch. This property enables nodes to aggregate frequency information (e.g., during push-pull exchanges) without revealing individual observations.

Count-Min Sketch with Conservative Updates (CMSCU).

The *conservative update* technique [EV03] aims to mitigate CMS collisions by incrementing only the counters with the minimum current value. The query procedure remains the same.

Update.

$$Y[k, h_k(x)] = max\{Y[k, h_k(x)], c_{min}(x) + 1\}, \forall 1 \leq k \leq d$$

, where

$$c_{min}(x) = \min_{1 \leq k \leq d} Y[k, h_k(x)]$$

The conservative update variant maintains the same theoretical error bound as the standard CMS. However, it typically achieves lower empirical overestimation by avoiding redundant counter increments, especially for low-frequency IDs in a biased distribution. In fact, [GDI11] showed that conservative update reduces the average relative error by a factor of at least 1.5.

5.3.2 Count-Mean-Min Sketch

The Count-Mean-Min (CMM) sketch [DR07] addresses the overestimation error in CMS by compensating for the expected noise introduced by hash collisions. As shown in [GDC12], conservative Updates further improve the average error of the Count-Mean-Min sketch. Here, we consider the conservative update version (CMMCU).

Update. As with CMSCU, updates are applied solely to the counters of the minimum value.

Query. The estimated frequency is corrected by subtracting estimated noise, defined as the average of other counter values in the same row:

$$Noise_k(x) = \frac{S_k - Y[k, h_k(x)]}{w - 1}, \forall 1 \leq k \leq d$$

, where S_k is the sum of counter values in row k.

The estimated frequency is the median of adjusted counters:

$$\hat{\phi}(x) = median_k(Y[k, h_k(x)] - Noise_k(x)), \forall 1 \leq k \leq d$$

Error guarantee. The CMMCU sketch provides equivalent theoretical error bounds to those of the standard CMS. However, unlike the CMS, the CMMCU sketch does not guarantee a one-sided error bound. The noise-subtraction process can lead to an underestimation of true frequencies. In particular, CMMCU demonstrates a lower error rate in estimating low-frequency IDs, as evidenced by the findings reported in [GDC12]. Conversely, CMMCU tends to produce high levels of underestimation error when estimating mid- and high-frequency IDs.

5.3.3 Lossy Conservative Update

The Lossy Conservative Update (LCU) sketch [GI11], inspired by Lossy Counting [MM02], extends CMSCU by periodically decreasing certain counts. The LCU window-split variant splits the stream into windows of fixed size $d \times w$. LCU periodically prunes small counts by decrementing all counter values in the sketch by 1, reducing overestimation of low-frequency IDs. This variant is the most accurate among those proposed in the paper.

Update. Similar to CMSCU, each ID insertion increments its associated counters using the conservative approach.

At each window boundary t,

If $(Y[k,l] > 0$ and $Y[k,l] \leq t)$, then $Y[k,l] = Y[k,l] - 1$,
$\forall 1 \leq k \leq d, 1 \leq l \leq w$

Query. The query process is similar to CMS.

Error guarantee. The LCU guarantees both underestimation and overestimation errors:

$$\phi(x) - \frac{1}{d \times w} \cdot M \leq \hat{\phi}(x) \leq \phi(x) + \frac{e}{w} \cdot M,$$

with probability at least $1 - \delta$.

Experiments in [GDC12] showed that LCU provides lower underestimation errors than CMMCU but higher overestimation errors. Compared to CMSCU, LCU provides a better estimation of low-frequency IDs and similar performance on high-frequency IDs.

5.3.4 Cold Filter

Cold Filter (CF) [YJZ+19b] is a two-layer sketching technique designed to efficiently differentiate between low- and high-frequency IDs using small counters. It is inspired by the Bloom filter [Blo70]. CF defines low-frequency IDs as those with a frequency below a predefined threshold, T, and which must be retained in the filter. High-frequency IDs are placed in a second-stage structure, which can be any sketch with a large counter size (e.g., CMSCU). This helps to estimate both categories of IDs accurately.

To achieve this filtering, the first stage must determine whether an ID is low-frequency. To improve the accuracy of this task, the first stage is divided into two layers, L_1 and L_2. Each layer is composed of w_i counters, each of b_i bits, and uses d_i hash functions. The global threshold T is split into two local thresholds:

$T = T_1 + T_2$, where $T_1 \leq 2^{b_1} - 1$ and $T_2 \leq 2^{b_2} - 2$. These thresholds control the overflow conditions of each layer.

Update. Given an incoming ID x, the CF algorithm proceeds as follows. It computes d_1 hash values and retrieves the corresponding counters from L_1, obtaining the minimum V_1. If $V_1 < T_1$, CF increments only the counters in L_1 whose values are equal to V_1 (conservative updates), and ends the insert operation. When all the d_1 counter values of x in L_1 reach T_1 (concurrent overflow), CF promotes the ID to the high layer L_2. In the layer L_2, CF computes the minimum V_2 among the d_2 hashed counters. If $V_2 < T_2$, it increments those counters whose values are equal to V_2. Once all d_2 counters of x reach T_2, CF considers the ID to have surpassed the global threshold T. Then the algorithm uses the second-stage structure ϕ to record the remaining frequency of x.

Query. To estimate the frequency of an ID x, CF applies the following rules. It queries the layer L_1 to obtain the estimated frequency of x in this layer, again denoted V_1. If $V_1 < T_1$, then $\hat{\phi}(x) = V_1$. If $V_1 = T_1$, it queries the estimated frequency of x in layer L_2, similarly denoted V_2. If $V_2 < T_2$, then $\hat{\phi}(x) = V_1 + V_2$. Otherwise $\hat{\phi}(x) = V_1 + V_2 + V_\phi$, with V_ϕ the query result of sketch ϕ.

Error guarantee. To assess the efficacy of the Cold Filter, its authors have proposed a metric called the "misreport rate." This metric quantifies the proportion of low-frequency IDs entered into the second-stage structure and that have subsequently been identified as errors. Through experimental inquiry with predetermined thresholds, it was demonstrated that the specified rate is less than 0.1% when CF memory occupies one-third of the space that an array would otherwise occupy. Moreover, experimental evaluations demonstrate that, in terms of average absolute error, CF can outperform CMSCU by a factor of 3.

5.3.5 BitMatcher

BitMatcher (BM) [SJL+24] is a recent sketching technique that dynamically adapts bit counter sizes based on the imbalance of the input stream. This fine-grained self-adjustment enables accurate estimation of ID frequencies while keeping the memory overhead for low-frequency IDs low. This makes it particularly well-suited to unbalanced distributions, such as those arising from Byzantine attacks in peer sampling.

BitMatcher maintains two parallel arrays of w buckets. Each bucket contains B entries, where each entry stores a fixed-size *fingerprint* obtained by hashing the processed ID, and a *counter* with variable bit width. The entries are ordered

by increasing maximum bit width, and the bucket configuration adapts dynamically to local frequency variations. A metadata flag encodes the current counter configuration, enabling efficient state transitions during counter resizing.

Update. To insert an element x, BitMatcher computes two candidate buckets for the two arrays, using Cuckoo hashing [FAKM14]:

$$h_1(x) = hash(x) \text{ and } h_2(x) = h_1(x) \text{ XOR } hash(fp(x))$$

where $fp(x)$ is the fixed-size fingerprint of ID x, $h_1 \mod w$ and $h_2 \mod w$ the two candidate buckets for storing $fp(x)$.

Then, BitMatcher searches for an entry to update the ID frequency. The sketch begins by scanning all entries of both buckets, looking for a fingerprint that matches $fp(x)$. If found, the associated counter is incremented. If not found and space is available, x is inserted into an empty entry, and the counter is set to 1. If both buckets are full, the counter value of one of their first entries is decreased by one (replacement strategy).

After an increment, if a counter reaches its maximum capacity, BitMatcher performs a corrective operation within the affected buckets to prevent future overflows. It first searches the two corresponding buckets for an empty entry with a larger capacity and, if found, moves the counter to that entry. This strategy increases the likelihood that high-frequency IDs will be assigned larger counters. If no such entry exists, BitMatcher triggers a state transition for the bucket containing the overflowing entry. This transition may involve shrinking other counters within the bucket or sacrificing the smallest entry to free additional bit space. These transitions follow a predefined table of valid bucket configurations (see Figure 5.2), with state transition rules detailed in the author's paper [SJL+24], ensuring that memory usage remains tightly bounded.

Query. To estimate the frequency of an ID, BitMatcher probes both corresponding buckets for an entry matching its fingerprint: If the fingerprint is found, the associated counter value is returned. If the fingerprint is not found and there is free space available, the estimated frequency is set to 0, indicating that the element is not present in the sketch. If both buckets are full, BitMatcher returns the minimum counter value among the two buckets.

Error guarantee. BitMatcher produces two types of errors: overestimation due to possible fingerprint collisions and underestimation due to the decremental measure (replacement strategy) applied when an ID cannot be placed in the sketch.

The average absolute error is $M/(w \cdot 2^{\mathcal{F}})$, where $\mathcal{F}$ is the fingerprint size. Typically, $\mathcal{F} = 8$ bits in practice [SJL+24]. This error is smaller than the CMSCU,

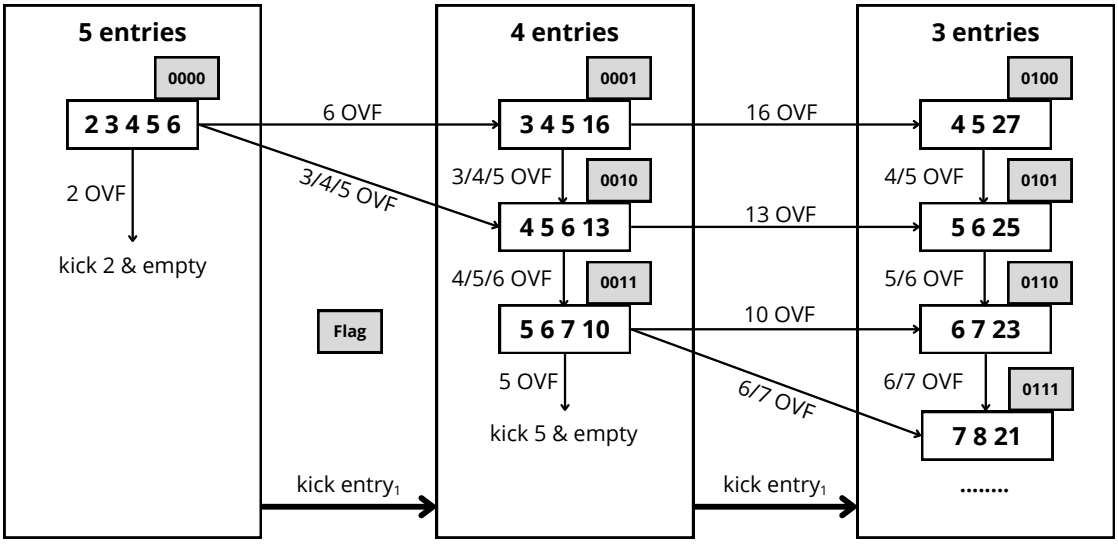


Figure 5.2: BitMatcher bucket state transition table [SJL+24].

CMMCU, and LCU errors by a factor of $2^{\mathcal{F}}$. This improvement has been validated experimentally, as shown in [SJL+24]. BM outperforms CMS in accuracy across both small and large datasets, regardless of the imbalance.

5.3.6 Probabilistic Sketch

The Probabilistic Sketch [LX23] (also known as the XY Sketch) is a recent sketching technique for identifying low-frequency IDs in large-scale data streams. Unlike conventional hash-based sketches, such as the Count-Min Sketch, which are susceptible to overestimation due to collisions, the XY Sketch uses a decomposition-and-recomposition strategy to encode stream IDs utilizing a set of fundamental elements probabilistically. The sketch's design is intended to enhance space efficiency and scalability.

XY Sketch is structured as a two-dimensional matrix $Y \in \mathbb{N}^{d \times w}$, where $d > 1$ denotes the number of rows, each of which corresponds to an *element* used in ID decomposition. $w = 2^b$ is the number of columns, which is determined by the bit-width b of each element. Each cell $Y[i, j]$ is a counter.

Update. Given an ID x, a bijective function decomposes it into d elements: $(x_1, x_2, \ldots, x_d)$, which are taken from the set of w possible elements. Each element x_i corresponds to a column index, and is linked to the i^{th} row of the matrix. To insert x, the following update is performed:

$$Y[i, x_i] \leftarrow Y[i, x_i] + 1, \quad \forall\, i = 1, \ldots, d$$

Query. To estimate the frequency $\phi(x)$, XY Sketch reconstructs the d elements $(x_1, \ldots, x_d)$, and computes:

$$\hat{\phi}(x) = M \cdot \prod_{i=1}^{d} \frac{Y[i, x_i]}{M}$$

, where M is the total number of elements inserted. This probabilistic recomposition yields estimates that are unbiased for low-frequency IDs and less affected by heavy hitters.

Error guarantee. The sketch guarantees the following probabilistic bound on the estimation error:

$$\left|\hat{\phi}(x) - \phi(x)\right| \leq \frac{2M}{\delta N}, \quad \text{with probability at least } 1 - \delta.$$

Experiments in [LX23] show that XY outperforms CMS, CMSCU, and CF (for any CF filter percentage) in accuracy across datasets, regardless of the imbalance. The extended version of XY Sketch generalizes the basic matrix structure by allowing rows of varying lengths (determined by the number of bits assigned to each element), ensuring that any given memory budget is fully utilized. A greedy space allocation algorithm tunes the number of counters per row while preserving the ability to recompose IDs and maintaining the same error bounds as the basic version. This flexibility reduces wasted space and improves estimation accuracy under tight memory constraints.

5.3.7 Comparison of sketches

Table 5.1 summarizes the characteristics of the sketches introduced in the previous section. w and d are sketch width and depth; ε is the error bound, δ the failure probability, M is the total number of inserted IDs (the stream length), N is the number of distinct elements; c and b are, respectively, the number of bits in an identifier and in an element for XY Sketch; $\mathcal{F}$ is the fingerprint size in BitMatcher. The table emphasizes the sketches' memory requirements, space complexity, and associated error guarantees. The Cold Filter is excluded from the table because its parameters are essentially tuned.

This section aims to evaluate the suitability of these sketches for peer sampling. To this end, the section analyzes them, focusing on the key trade-offs relevant in this context. These trade-offs include memory efficiency, robustness to low- and high-frequency IDs, scalability, and performance under high unbalanced distributions.

Counting strategy	d	w	Space	Underest. Error	Overest. Error
CMSCU [EV03]	$\log \frac{1}{\delta}$	e/ε	$\mathcal{O}(wd)$	–	$\varepsilon \cdot M$
CMMCU [GDC12]	$\log \frac{1}{\delta}$	e/ε	$\mathcal{O}(wd)$	$\varepsilon \cdot M$	$\varepsilon \cdot M$
LCU [GI11]	$\log \frac{1}{\delta}$	e/ε	$\mathcal{O}(wd)$	$\frac{1}{wd} \cdot M$	$\varepsilon \cdot M$
BitMatcher [SJL+24]	2	$1/\delta\varepsilon 2^{\mathcal{F}}$	$\mathcal{O}(w)$	$E(AAE) = \frac{1}{w2^{\mathcal{F}}} \cdot M$	
XY [LX23]	c/b	2^b	$\mathcal{O}(wd)$	$\frac{2}{\delta N} \cdot M$	$\frac{2}{\delta N} \cdot M$

Table 5.1: Comparison of counting strategies. AAE denotes Absolute Average Error. "–" marks value not defined.

Memory efficiency. In peer sampling, nodes operate under a fixed memory budget. It is therefore crucial to examine how efficiently each sketch uses its allocated space. Approaches based on the Count-Min Sketch (CMSCU, CMMCU, LCU) require $\mathcal{O}\left(\frac{e}{\varepsilon}\log\frac{1}{\delta}\right)$ memory. For the same values of ε and δ, BitMatcher occupies a memory space in the order of $\mathcal{O}(\frac{1}{\delta\varepsilon 2^{\mathcal{F}}})$. This bound is smaller than that of the Count-Min Sketch when $\mathcal{F}$ is sufficiently large ($\mathcal{F} \geq 6$).

The XY Sketch has a memory bound of $\frac{c}{b} \cdot 2^b$, where c is the number of bits in the binary decomposition of an ID given the entire system space, and b is the number of bits of an element. This bound grows exponentially with b. The smallest possible configuration, with $b = 1$, requires only $2 \cdot c$ bits, which is very compact but potentially less accurate.

Robustness to low- and high-frequency IDs. An effective sketch for peer sampling must accurately estimate both low- and high-frequency IDs. Count-Min Sketch variants have an overestimation error bound of $\frac{e}{w} \cdot M$. This error decreases with larger w but grows with stream length. The LCU and CMMCU variants reduce collision noise, improving estimates for low-frequency IDs, but introduce underestimation errors.

These theoretical error bounds and prior comparisons of these sketches in word processing tasks [GDC12] suggest: (1) For low-frequency IDs: CMMCU > LCU > CMSCU in estimation accuracy. (2) For high-frequency IDs: LCU and CMSCU perform similarly, and both outperform CMMCU.

Cold Filter explicitly separates low-frequency from high-frequency IDs, reducing collisions between the two groups and improving query accuracy across the board.

Scalability. As the stream size increases, the estimation error for all strategies also increases. However, CF, BM, and XY are less sensitive to changes in stream length than CMSCU, CMMCU, and LCU, as indicated by their error bounds. Notably, the XY Sketch error is inversely proportional to the number of distinct IDs, making it highly scalable with system size.

Performance under high imbalance. High imbalance amplifies the frequency gap between low- and high-frequency IDs. Strategies that explicitly mitigate collisions between these two groups, such as the Cold Filter strategy, maintain better accuracy under unbalanced conditions.

Summary. This qualitative comparison reveals different design priorities. The BitMatcher is memory efficient. The Cold Filter maintains high accuracy under imbalance. Both perform well with low-frequency IDs, along with CMMCU and LCU. The CMSCU is the simplest but less robust for low-frequency IDs.

These differences motivate the experimental evaluation in realistic peer sampling settings presented in the next section.

5.4 Evaluations and results

The goal of this section is to evaluate the effectiveness of the sketches presented earlier, under memory constraints and in the context of peer sampling. To identify suitable sketches for estimating ID frequencies in received streams, we address several research questions. First, given a memory budget, how accurately does each sketch estimate frequencies when the stream follows a uniform random distribution (i.e., no adversarial bias)? Second, given a memory budget, how accurately does each sketch estimate frequencies under a balanced attack? And to what extent can the sketches maintain an unchanged bias factor with the provided estimations? Finally, how much does the performance of these sketches degrade as the proportion of malicious nodes, the number of distinct IDs, and the total number of IDs in the stream size increase?

The initial phase of this study outlines the experimental protocol, including the critical parameters and scenarios defining our streams, the selection of sketch parameters, and the metrics used to evaluate the performance of the sketches examined. Lastly, we present the results obtained and their interpretation.

5.4.1 Stream parameters

To evaluate the accuracy of our sketches under realistic constraints, we simulate our streams based on the Bitcoin network characteristics (version 0.11) and its peer

sampling protocol, as described in Section 2.5. In Bitcoin, each node can initiate up to 8 outgoing connections to neighbors. Bitcoin Core deliberately sticks to the same randomly chosen peers for a day. Upon establishing a new connection, a node sends a `GETADDR` message, to which the neighbor replies with `ADDR` messages that can hold up to 2,500 peer IDs in total. Consequently, a node typically receives about 2,500 identifiers from each of its eight neighbors in a single day, for a total of approximately 20,000 identifiers at most.

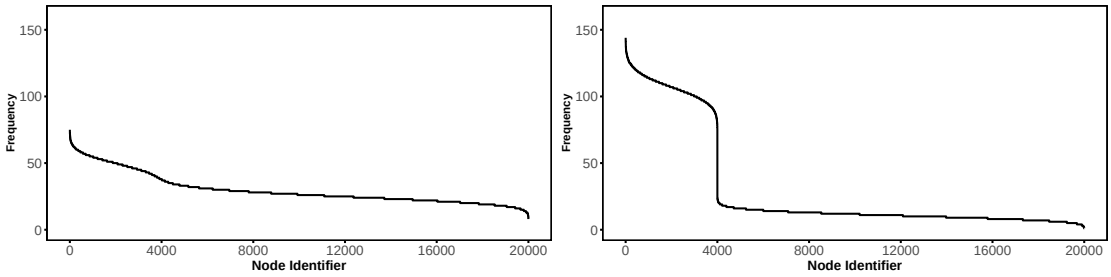
To simulate this network, we use a fixed system size of 20,000 unique IDs, which is consistent with long-term Bitcoin measurements [Bit25c]. These measurements reported an average of 20,283 reachable nodes between July 2024 and July 2025. Each node ID is encoded as a 4-byte integer. Thus, an array-based strategy that stores all ID frequencies would require 80 KB of storage space. For our evaluation, we varied the memory budget allocated to our sketches from 20 KB to 80 KB. The stream size M ranges from 600,000 IDs (representing 30 days of operation) to 7,200,000 IDs (one year). We also varied the number of distinct IDs N from 20,000 to twice the system size, *i.e.*, 40,000.

We study two scenarios representing the settings outlined in Section 5.2: a fault-free scenario with a uniform peer selection, and a malicious scenario with Byzantine nodes attempting to bias the system via a balanced attack. In the fault-free scenario, the stream of identifiers is generated uniformly across the identifier space. In the malicious scenario, a fraction f of the identifier space consists of Byzantine nodes that overrepresent themselves by a bias factor γ. To evaluate the impact of this bias on estimation accuracy, we varied the proportion of Byzantine IDs to 10%, 20%, and 30%, and the bias factor to 2 and 10. Given a fixed proportion of malicious nodes, a bias factor of 2 indicates a subtle attack, whereas a bias factor of 10 represents an aggressive adversarial strategy. Figure 5.3 illustrates two streams from our malicious scenario with $f = 20\%$, by showing the frequency distribution of identifiers sorted in decreasing order. As the value of γ increases (from Figure 5.3a to Figure 5.3b), Byzantine IDs become more and more dominant at the top of the distribution. The combination of f and γ yields six streams in our malicious scenario. Together with the fault-free scenario, these produce a total of seven synthetic streams.

5.4.2 Sketch parametization

We evaluate six strategies: BitMatcher [SJL+24] (BM), Cold Filter [YJZ+19b] (CF), Count-Mean-Min with Conservative Updates [GDC12] (CMMCU), Count-Min Sketch with Conservative Updates [EV03] (CMSCU), Lossy Conservative Updates [GI11] (LCU) and Probabilistic sketch [LX23] (XY).

For CMSCU, LCU, and CMMCU, the number of hash functions is $d = 3$, as recommended in [GDC12]. For CF, the thresholds are the same as in the original



(a) $\gamma = 2$. (b) $\gamma = 10$.

Figure 5.3: Frequency distribution of input stream in malicious setting. $N = 20,000, M = 600,000, f = 20\%$.

paper ($T_1 = 15$ and $T_2 = 241$). Additionally, 90% of the memory budget is allocated to the two-layer cold filter, and the remaining 10% to the side structure, which is a CMSCU sketch. This latter configuration has been shown to yield optimal results in prior experimental work [YJZ+19b, LX23]. The XY encoding process uses $\lfloor \log_2(N) \rfloor + 1$ bits to represent IDs in binary. Moreover, we used the extended version of XY, which optimizes space utilization.

The counters used in the listed sketches occupy 4 bytes, except for Cold Filter, which uses predefined counter sizes, and BitMatcher, which implements counter-size adjustment. For each of these sketches, the available space (the memory budget) is used to its fullest. The code of our sketches was implemented in Rust, totaling approximately 2,000 lines, except for BitMatcher and Cold Filter, for which the authors' implementations were used.

5.4.3 Evaluation metrics

To compare the estimation accuracy of our sketches for a given stream, we store the stream in a buffer large enough to compute the exact frequency distribution of IDs. Then we feed each sketch the stored stream and query it to obtain an output frequency distribution over the stream IDs. After querying all sketches, we compare their output frequency distributions using various metrics. Our goal is to assess not only whether a sketch's output frequency distribution approximates the original distribution well, but also whether it preserves ID classes and the overrepresentation of the high-frequency class over the low-frequency class during frequency estimation in a malicious scenario.

Statistical Accuracy via KL Divergence. We first assess how closely the estimated probability distribution p' matches the true probability distribution p

using Kullback-Leibler divergence (D_{KL}):

$$D_{KL}(p'||p) = -\sum_{k=0}^{N-1} p'(x_k) log \frac{p(x_k)}{p'(x_k)}$$

Here, N is the number of distinct identifiers, and $p(x_k)$, $p'(x_k)$ denote the true and estimated probabilities of finding the ID x_k in the stream, respectively. The distribution p (respectively p') is obtained by dividing each frequency (resp. each estimated frequency) by the sum of all frequencies (resp. all estimated frequencies). A lower KL divergence indicates greater similarity between the two distributions.

Classification Accuracy via KMeans. In a malicious scenario, an additional objective is to differentiate between Byzantine and correct ID classes. Given the frequency distribution output by a sketch, the K-Means clustering algorithm is applied with $K = 2$ clusters to differentiate between these two classes. Based on the clustering results, the following calculations are performed: *precision*, *recall*, and the harmonic mean of precision and recall, known as the F_1-*score*.

Bias Factor Estimation. The bias factor indicates the extent to which Byzantine IDs are overrepresented relative to correct IDs within a given frequency distribution. Given an output frequency distribution, we compute its bias factor, γ', in the hope that it will approach the original factor, γ, of the input frequency distribution. Our objective is to ascertain the discrepancy between the original and the approximate bias factors. To this end, the relative error, γ_{err}, is computed as follows:

$$\gamma_{err} = \frac{\gamma' - \gamma}{\gamma}, \quad \text{with } \gamma, \gamma' \geq 0$$

5.4.4 Results in fault-free scenario

In the fault-free scenario, we examine how accurately our sketches estimate frequencies under varying memory budgets, stream sizes (M-scalability), and identifier space sizes (N-scalability). For each sketch, we evaluate the KL divergence between the input and output frequency distributions.

Memory efficiency. Figure 5.4 reports the KL divergence for the four memory budget configurations. The streams comprise 600,000 IDs drawn from a space of 20,000 IDs. CF and CMSCU consistently achieve the lowest divergence values (close to zero), independently of the available memory. In contrast, other methods, especially CMMCU and LCU, exhibit higher divergence as memory increases. This degradation is due to a reduction in hash collisions while residual values are still

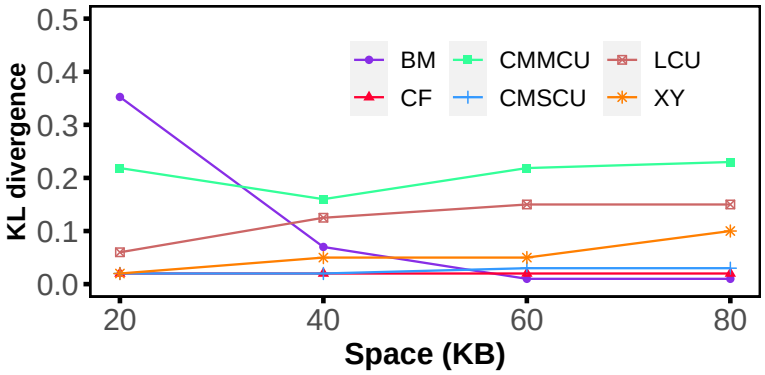


Figure 5.4: KL divergence vs. Budget memory on fault-free scenario. $N = 20,000, M = 600,000$.

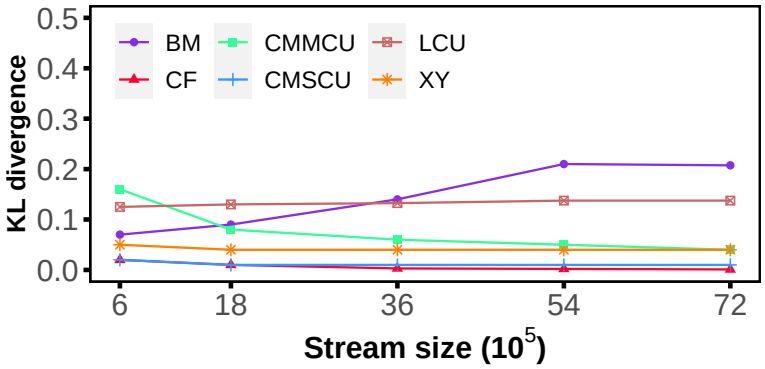


Figure 5.5: KL divergence vs. stream size on fault-free scenario. Budget is 40 KB, $N = 20,000$.

discarded, thereby introducing estimation bias. BM benefits from larger memory budgets and achieves its best performance when the memory exceeds 60 KB.

M-scalability. The size of the ID stream affects the accuracy of our sketch estimations due to the increased number of insertions. To understand this influence, we evaluated the scalability of our estimators as a function of stream size. For this experiment, we consider our system to consist of 20,000 nodes and to have a budget of 40 KB, as this configuration produced good results for all sketches in the previous analysis. Figure 5.5 depicts the evolution of the KL divergence of the output streams for each estimator as the stream size increases. It shows that BM is the most sensitive to increases in stream size. Similarly, CF and CMSCU provide stable results and achieve the best divergence. CMMCU improves as more items are observed due to the increasing noise being subtracted from the estimates.

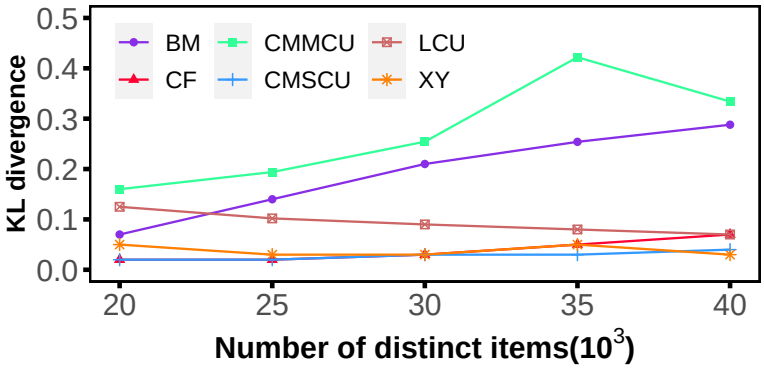


Figure 5.6: KL divergence vs. number of distinct node identifiers on fault-free scenario. Budget is 40 KB, $M = 600,000$.

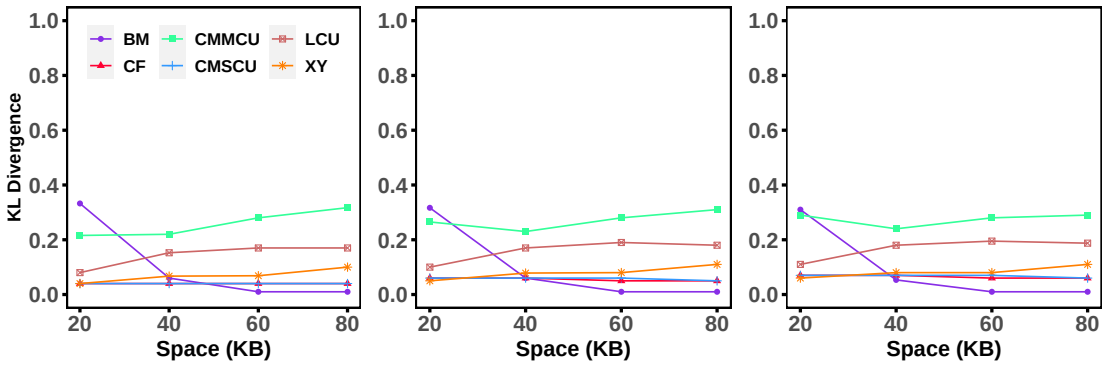
N-scalability. The size of the identifier space is a decisive parameter because it influences collisions in these sketches. Figure 5.6 illustrates the impact of increasing the system size, N, with the stream size fixed at 600,000 IDs and a budget of 40 KB. This shows that CMMCU and BM are the most sensitive to an increase in the number of distinct items. For CF and CMSCU, however, the output distributions are statistically closest to the original distribution.

5.4.5 Results in malicious scenario

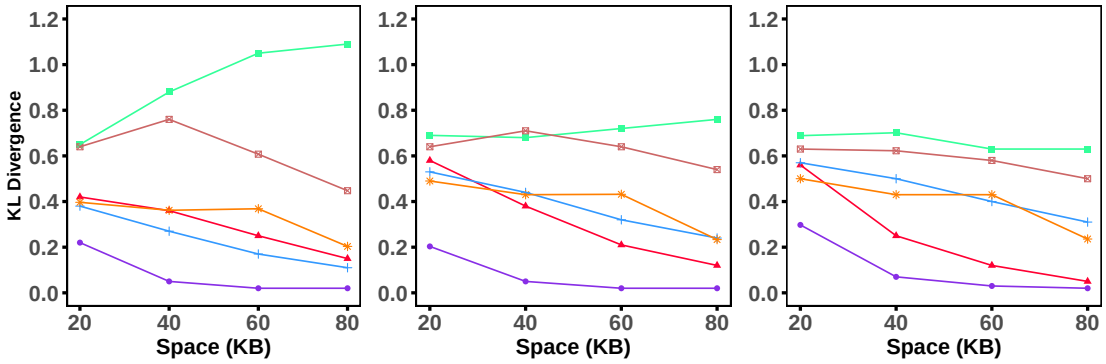
In the malicious scenarios, we examine how accurately our sketches estimate frequencies across varying memory budgets, stream sizes, and identifier-space sizes. We evaluate the KL divergence, the clustering metrics, and the bias factor estimation between the input and output frequency distributions.

Memory efficiency. As in the fault-free scenario, we varied the memory budget for the estimators from 20 KB to 80 KB. Figures 5.7, 5.8, and 5.9 show the impact of the memory budget on the Kullback-Leibler divergence metric, the F1 score, and the bias factor error, respectively. For each metric, we examine two bias factor values: $\gamma = 2$ and $\gamma = 10$. The streams comprise 600,000 IDs drawn from a space of 20,000 IDs.

Kullback-Leibler divergence. For $\gamma = 2$, the results for the KL divergence (Figure 5.7a) are quite similar to those of the fault-free scenario. Regardless of the memory budget, CF and CMSCU produce stable, lower divergence. Other strategies are sensitive to memory constraints. As the memory budget increases, the performance of CMMCU, LCU, and XY may worsen. For memory budgets greater than 40 KB, BM performs best with a divergence close to zero.



(a) $\gamma = 2$.

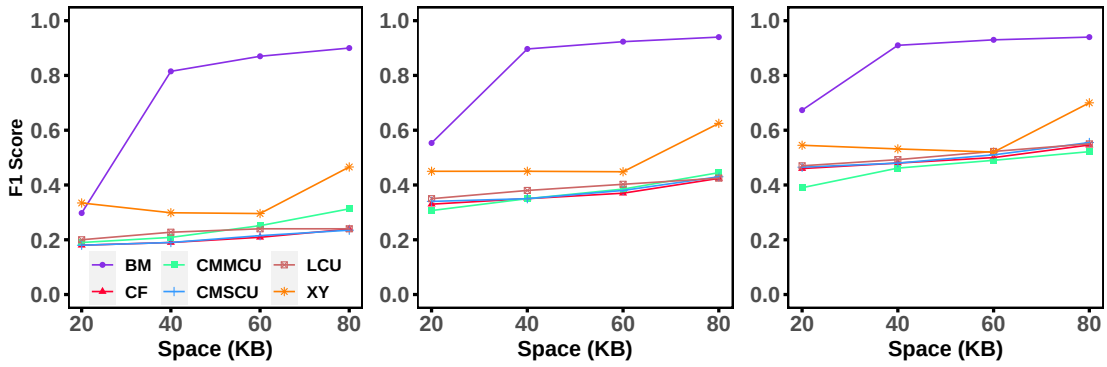


(b) $\gamma = 10$.

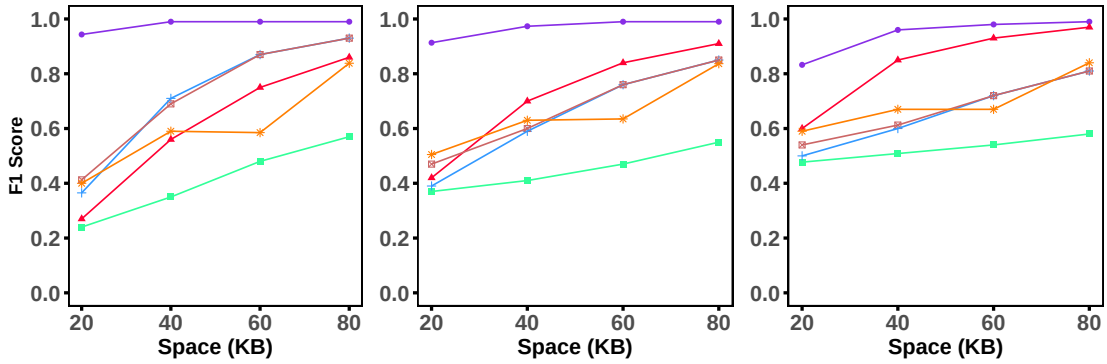
Figure 5.7: KL divergence vs. budget memory on malicious case. $N = 20,000$, $M = 600,000$, $f = 10\%$ (left figures), 20% (middle figures) and 30% (right figures) respectively.

For $\gamma = 10$, BM achieves the lowest divergence (Figure 5.7b). For the other strategies, a KL divergence greater than 0.2 implies a significant difference between the input and output distributions. CMMCU and LCU are the most distant from the truth. Overall, the results are quite similar for a given bias factor, regardless of the Byzantine proportion.

Classification metric. Figure 5.8a shows that for $\gamma = 2$, BM achieves the highest F1-score (over 80%) when there is sufficient memory (40 KB or more). This suggests that BM is more effective than other strategies at distinguishing Byzantine identifiers from correct ones. Table 5.2a describes the precision and recall results of the six sketches. In this and all subsequent tables, the best results are highlighted in bold. The table shows that the memory budget does not significantly affect the precision of the sketches for a given proportion of Byzantine nodes, which remains poor (below 0.5). Except for BM, which shows a substantial improvement, and for



(a) $\gamma = 2$.



(b) $\gamma = 10$.

Figure 5.8: F1score vs. budget memory on malicious case. $N = 20,000, M = 600,000, f = 10\%, 20\%$ and 30%.

XY, which improves when $f = 30\%$. As memory increases, BM misclassifies fewer correct nodes. Furthermore, the more Byzantine nodes there are, the higher the precision across all strategies. Additionally, CMMCU has higher precision than LCU and CMSCU but lower recall, highlighting its robustness to low-frequency items. These results are consistent with our initial expectations outlined in Section 5.3.7. CF and CMSCU identify nearly all Byzantine nodes, as evidenced by their recall. CF achieves the best recall, thanks to its low misreport rate. All strategies perform well at identifying Byzantine identifiers, with a recall greater than 0.5. XY and CMMCU are the least efficient.

When $\gamma = 10$ (Figure 5.8b), BM consistently achieves an F1-score above 80%. When the memory budget exceeds 60 KB, both CMSCU and LCU perform well. However, they degrade as the proportion of Byzantine nodes increases and CF surpasses them. CMMCU performs the worst in all cases. For the precision (Table 5.2b), BM is the most reliable, achieving nearly 100% precision. The impact of

f%	BM	CF	CMMCU	CMSCU	LCU	XY
10	.19/.77	.10/**1**	.10/.85	.10/.99	.11/.87	**.21**/.78
	.71/.96	.10/**.99**	.12/.79	.11/.98	.13/.83	.18/.78
	.79/.97	.12/**.98**	.15/.83	.12/**.98**	.14/.90	.18/.77
	.84/.97	.13/**.99**	.19/.89	.14/**.99**	.14/.96	.33/.79
20	**.45**/.74	.20/**1**	.22/.69	.20/.99	.22/.83	.32/.77
	.85/.94	.21/**.99**	.23/.74	.22/.96	.25/.79	.32/.72
	.89/.95	.23/**.97**	.25/.79	.24/.95	.26/.85	.32/.72
	.92/.96	.27/.97	.30/.83	.28/**.98**	.28/.94	.53/.77
30	**.66**/.69	.30/**1**	.33/.60	.30/.98	.33/.82	.44/.70
	.90/.93	.31/**.99**	.34/.76	.32/.95	.37/.77	.44/.67
	.93/.93	.34/**.97**	.36/.75	.35/.93	.38/.82	.47/.57
	.95/.94	.39/**.95**	.41/.74	.39/**.95**	.40/.90	.65/.76

(a) $\gamma = 2$.

f%	BM	CF	CMMCU	CMSCU	LCU	XY
10	**.93**/.96	.15/**1**	.14/.90	.22/**1**	.27/.90	.25/**1**
	.98/**1**	.39/**1**	.21/**1**	.55/**1**	.53/**1**	.45/.86
	.98/**1**	.60/**1**	.31/**1**	.76/**1**	.76/**1**	.45/.85
	.99/**1**	.75/**1**	.40/**1**	.87/**1**	.87/**1**	.85/.83
20	**.92**/.90	.27/**1**	.24/.81	.24/.99	.35/.78	.45/.58
	.98/.97	.53/**1**	.27/.88	.42/**1**	.43/.96	.49/.88
	.98/.99	.73/**1**	.31/.98	.61/**1**	.61/**1**	.50/.89
	.99/**1**	.84/**1**	.38/**1**	.74/**1**	.74/**1**	.85/.82
30	**.78**/.90	.43/**1**	.34/.80	.34/.97	.44/.70	.49/.73
	.98/.94	.73/**1**	.38/.75	.43/**1**	.48/.85	.54/.90
	.99/.98	.87/**1**	.39/.88	.56/**1**	.57/.98	.53/.90
	.99/.99	.94/**1**	.42/.95	.68/**1**	.68/**1**	.86/.82

(b) $\gamma = 10$.

Table 5.2: Evaluating Precision/Recall depending on memory budget of 20, 40, 60, 80 KB, respectively. $N = 20,000, M = 600,000$.

the memory budget is evident in improved precision and recall of the sketches. Furthermore, all strategies detect nearly all Byzantine nodes when sufficient memory (40 KB or more) is available.

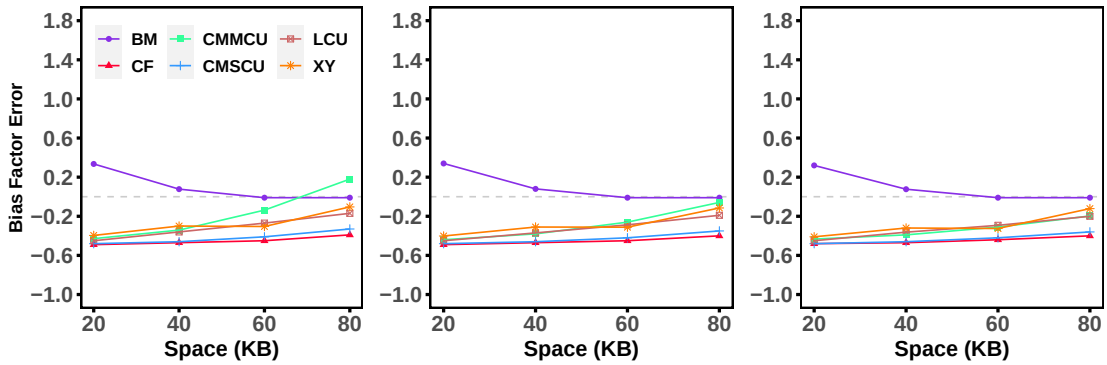
Bias factor estimation. Bias factor errors are lower when $\gamma = 2$ (Figure 5.9a) than when $\gamma = 10$ (Figure 5.9b). Overall, BM provides overrepresentation coefficients close to the optimal value (with an error near 0), and it is the only strategy that occasionally overestimates the bias factor. Therefore, it is the most appropriate strategy for estimating the extent of overrepresentation of Byzantine identifiers.

When $\gamma = 10$ (Figure 5.9b), almost all strategies underestimate the coefficient by more than 50%. Consequently, Byzantine IDs appear underrepresented and would be oversampled by a selection strategy. In contrast, BM provides a nearly optimal estimation of the overestimation factor.

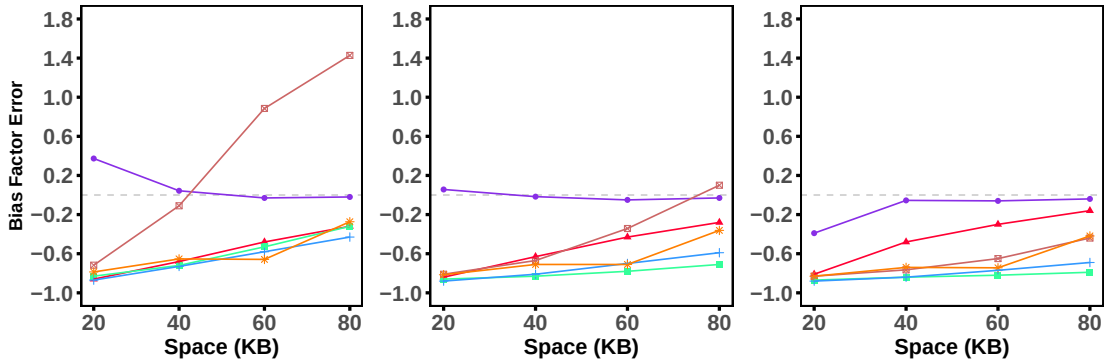
M-scalability. This section examines how stream size affects the estimation accuracy of our estimators. We varied the stream sizes in our synthetic traces from 600,000 to 7,200,000. The memory budget is fixed at 40 KB for all configurations. Figures 5.10, 5.11, and 5.12 show the impact of the memory budget, respectively, on the Kullback-Leibler divergence metric, the F1 score, and the bias factor error. The system comprises 20,000 identifiers.

Kullback-Leibler divergence. For $\gamma = 2$, the results for the KL divergence (Figure 5.10a) show that most strategies maintain stable, low divergence (< 0.2) regardless of stream size. BM's divergence increases slightly with larger streams.

For $\gamma = 10$ (Figure 5.10b), BM again achieves the lowest divergence. The other



(a) $\gamma = 2$.



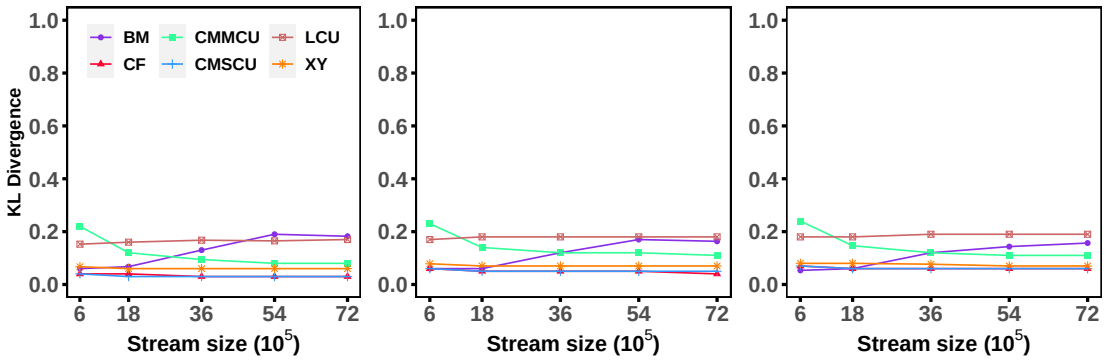
(b) $\gamma = 10$.

Figure 5.9: Error on Bias Factor (γ_{err}) vs. budget memory on malicious case. $N = 20,000, M = 600,000, f = 10\%, 20\%$ and 30%.

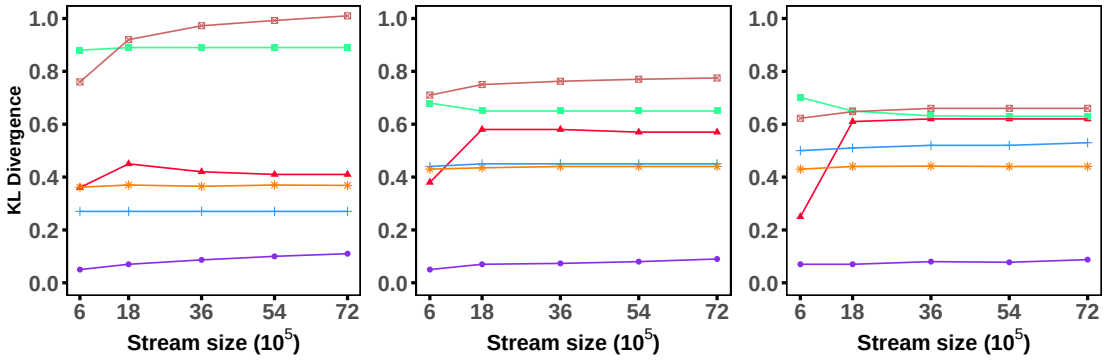
sketches show a significant difference between the input and output distributions. As in the previous case, CMMCU and LCU are still the least accurate.

Classification metric. With $\gamma = 2$, Figure 5.11a shows that BM achieves an F1-score above 70% for stream sizes below 5.4 million with 10% Byzantine node IDs. BM is indeed the most effective at distinguishing Byzantine identifiers from correct ones, but its performance degrades when the stream includes many identifiers and only a small fraction are Byzantine. Other strategies are not sensitive to stream size. As shown in Table 5.3a, the precision of these sketches improves with the proportion of Byzantine IDs in the identifier space. Most strategies identify at least 80% of Byzantine identifiers, with CF and CMSCU performing the best, as shown in the same table.

Figure 5.11b depicts the evolution of the F1-score as the stream size increases with $\gamma = 10$. BM consistently scores near 100%. Table 5.3b details the precision and recall values that led to these F1-score results. CF yields the worst global pre-



(a) $\gamma = 2$.



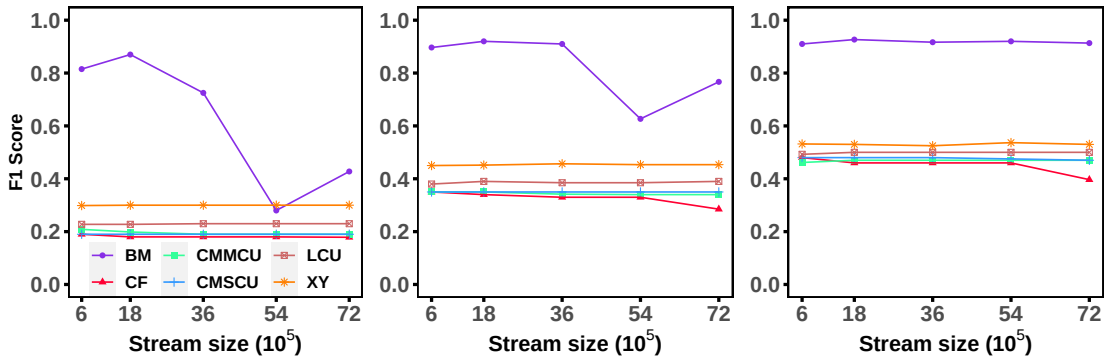
(b) $\gamma = 10$.

Figure 5.10: KL divergence vs. stream size on malicious case. Budget is 40 KB, $N = 20,000$, and $f = 10\%, 20\%, 30\%$.

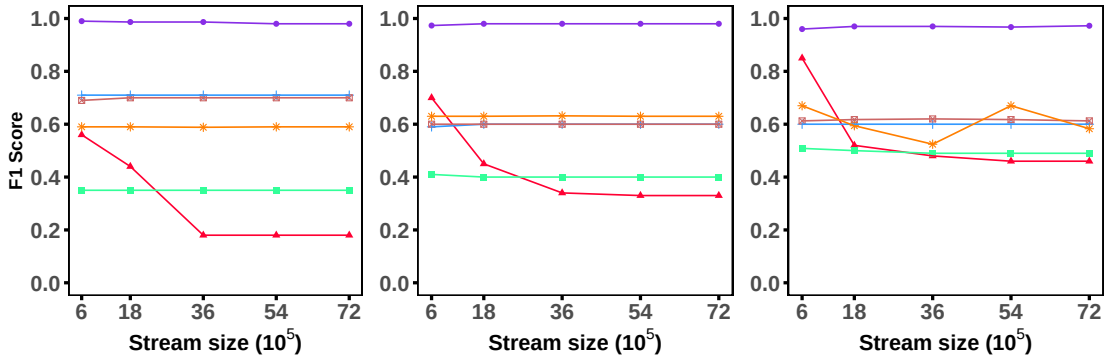
cision. Its sensitivity to stream size stems from the fixed thresholds of its layers, which need to be tuned. More Byzantine IDs lead to better classification performance. Recall values demonstrate that all strategies detect nearly all Byzantine IDs. For a 10% Byzantine fraction, all strategies except XY recover nearly all Byzantine identifiers.

Bias factor estimation. For $\gamma = 2$, as shown in Figure 5.12a, BM provides the lowest bias factor error. However, a larger stream size worsens BM's coefficient estimation. Other strategies are insensitive to increases in stream size.

When $\gamma = 10$ (Figure 5.12b), and f=10%, all strategies, except BM and LCU, underestimate the factor by 60% to 100%. As the proportion of Byzantine IDs increases ($f = 20\%$ and $f = 30\%$), this error increases as well. BM is the only strategy that achieves near-optimal estimation of the factor in all cases.



(a) $\gamma = 2$.



(b) $\gamma = 10$.

Figure 5.11: F1score vs. stream size on malicious case. Budget is 40 KB, $N = 20,000$, and $f = 10\%, 20\%, 30\%$.

N-scalability. Finally, we evaluated the N-scalability of the estimators using our six traces. We varied the number of distinct nodes from 20,000 to 40,000 in increments of 5,000 to simulate the arrival of new items until the system size doubles. Typically, an array strategy cannot scale in these settings due to its fixed number of entries. For all the configurations, the memory budget is fixed at 40 KB. Figures 5.13, 5.14 and 5.15 show the impact of the number of distinct IDs on the Kullback–Leibler divergence metric, clustering metric, F1 score, and bias factor error, respectively. The stream size comprises 600,000 identifiers.

Kullback-Leibler divergence. For $\gamma = 2$, the results for the KL divergence (Figure 5.13a) show that all strategies maintain low divergence (< 0.2) as the number of IDs increases, except for BM and CMMCU, which are more sensitive to an increase in the number of distinct items.

For $\gamma = 10$ (Figure 5.13b), BM again achieves the lowest divergence. The other strategies produce significant divergence between the output and input distribu-

f%	BM	CF	CMMCU	CMSCU	LCU	XY
10	**.71**/.96	.10/**.99**	.12/.79	.11/.98	.13/.83	.18/.78
	.79/.97	.10/**1**	.11/.93	.11/.99	.13/.84	.18/.82
	.62/.97	.10/**1**	.11/.95	.11/.99	.13/.84	.18/.82
	.16/.98	.10/**1**	.11/.97	.11/.99	.13/.85	**.18**/.82
	.32/.98	.10/.93	.10/.97	.11/**.99**	.13/.85	.18/.82
20	**.85**/.94	.21/**.99**	.23/.74	.22/.96	.25/.79	.32/.72
	.89/.95	.20/**1**	.22/.91	.21/.98	.26/.78	.31/.80
	.88/.94	.20/**1**	.21/.94	.21/.99	.26/.79	.31/.83
	.51/.95	.20/**1**	.21/.95	.21/.99	.25/.79	.34/.76
	.69/.94	.20/.64	.21/.95	.21/**.99**	.26/.80	.31/.82
30	**.90**/.93	.31/**.99**	.34/.76	.32/.95	.37/.77	.44/.67
	.93/.91	.30/**.99**	.32/.87	.32/.96	.37/.76	.50/.58
	.93/.91	.30/**1**	.31/.94	.32/.97	.37/.77	.51/.55
	.94/.90	.30/**1**	.31/.95	.31/.98	.37/.76	.45/.71
	.92/.90	.30/.75	.31/.96	.31/**.99**	.37/.77	.50/.61

(a) $\gamma = 2$

f%	BM	CF	CMMCU	CMSCU	LCU	XY
10	**.98**/**1**	.39/**1**	.21/**1**	.55/**1**	.53/**1**	.45/.86
	.97/**1**	.28/**1**	.21/**1**	.55/**1**	.54/**1**	.45/.85
	.97/**1**	.10/**1**	.21/**1**	.55/**1**	.54/**1**	.45/.85
	.96/**1**	.10/**1**	.21/**1**	.55/**1**	.54/**1**	.45/.85
	.97/**1**	.10/**1**	.21/**1**	.55/**1**	.54/**1**	.45/.85
20	**.98**/.97	.53/**1**	.27/.88	.42/**1**	.43/.96	.49/.88
	.97/.98	.29/**1**	.25/.98	.43/**1**	.44/.97	.49/.88
	.97/.98	.21/**1**	.25/.99	.43/**1**	.43/.98	.49/.88
	.98/.99	.20/**1**	.25/**1**	.43/**1**	.43/.98	.49/.88
	.97/.99	.20/**1**	.25/**1**	.43/**1**	.43/.98	.49/.88
30	**.98**/.94	.73/**1**	.38/.75	.43/**1**	.48/.85	.54/.90
	.98/.96	.35/.99	.36/.82	.43/**1**	.49/.85	.62/.68
	.98/.97	.32/**1**	.34/.88	.43/**1**	.48/.85	.73/.49
	.97/.97	.30/**1**	.34/.90	.43/**1**	.49/.84	.53/.91
	.98/.97	.30/**1**	.34/.91	.43/**1**	.48/.85	.65/.66

(b) $\gamma = 10$

Table 5.3: Evaluating Precision/Recall depending on stream size: $600,000$, $1,800,000$, $3,600,000$, $5,400,000$, $7,200,000$ items. Budget is 40 KB, $N = 20,000$.

f%	BM	CF	CMMCU	CMSCU	LCU	XY
10	**.71**/.96	.10/**.99**	.12/.79	.11/.98	.13/.83	.18/.78
	.21/.93	.10/**1**	.11/.8	.11/.97	.12/.82	**.25**/.91
	.2/.88	.11/**1**	.12/.72	.10/.98	.12/.83	.17/.74
	.2/.82	.11/**1**	.11/.73	.10/.98	.11/.83	.15/.84
	.19/.76	.12/**1**	.11/.7	.10/.99	.11/.85	.18/.76
20	**.85**/.94	.21/.55	.23/.74	.22/**.96**	.25/.79	.32/.72
	.69/.91	.21/**1**	.23/.68	.21/.97	.24/.79	.36/.82
	.5/.84	.22/**1**	.22/.71	.21/.97	.23/.78	.32/.68
	.5/.77	.23/**1**	.22/.71	.21/.97	.23/.81	.28/.83
	.47/.71	.24/**1**	.23/.54	.20/.98	.22/.83	.32/.72
30	**.9**/.93	.31/.55	.34/.76	.32/**.95**	.37/.77	.44/.67
	.78/.88	.32/**1**	.34/.65	.31/.96	.35/.75	.48/.79
	.73/.8	.34/**1**	.32/.77	.31/.96	.35/.77	.43/.66
	.65/.73	.36/**1**	.31/.81	.31/.97	.33/.8	.42/.72
	.6/.67	.36/.55	.32/.59	.31/**.97**	.33/.81	.44/.67

(a) $\gamma = 2$

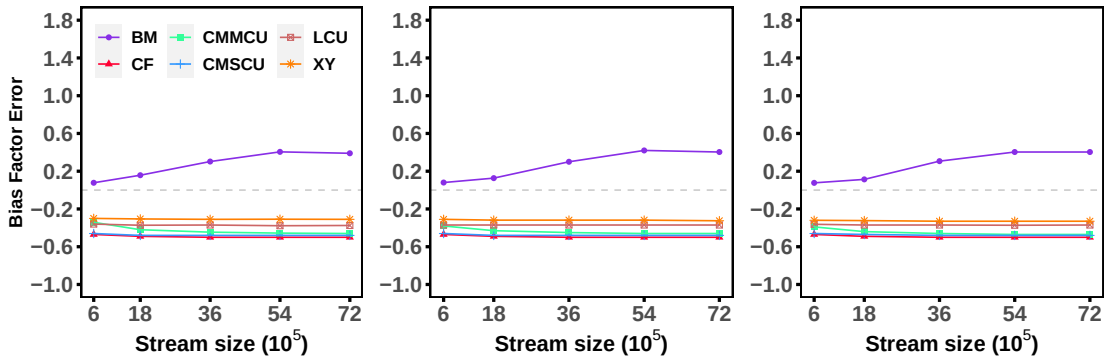
f%	BM	CF	CMMCU	CMSCU	LCU	XY
10	**.98**/**1**	.39/**1**	.21/**1**	.55/**1**	.53/**1**	.45/.86
	.97/.99	.37/**1**	.18/.97	.43/**1**	.40/**1**	.36/.98
	.96/.98	.34/**1**	.16/.89	.34/**1**	.34/.96	.31/.93
	.96/.98	.32/**1**	.15/.84	.26/.99	.29/.94	.28/.94
	.95/.98	.30/**1**	.15/.8	.20/.99	.26/.9	.25/**1**
20	**.98**/.97	.53/**1**	.27/.88	.42/**1**	.43/.96	.49/.88
	.97/.96	.56/**1**	.27/.78	.34/**1**	.38/.88	.40/.98
	.96/.95	.56/**1**	.26/.77	.29/.99	.36/.83	.37/.89
	.96/.94	.55/**1**	.26/.74	.26/.99	.35/.78	.38/.78
	.93/.97	.53/**1**	.25/.74	.24/.99	.34/.75	.39/.73
30	**.98**/.94	.73/**1**	.38/.75	.43/**1**	.48/.85	.54/.9
	.97/.93	.72/**1**	.37/.72	.37/.99	.47/.77	.49/.85
	.96/.98	.70/**.99**	.36/.74	.35/.98	.45/.74	.59/.56
	.94/.96	.67/**.99**	.35/.74	.34/.96	.44/.72	.53/.64
	.94/.92	.64/**1**	.34/.74	.34/.95	.42/.73	.49/.73

(b) $\gamma = 10$

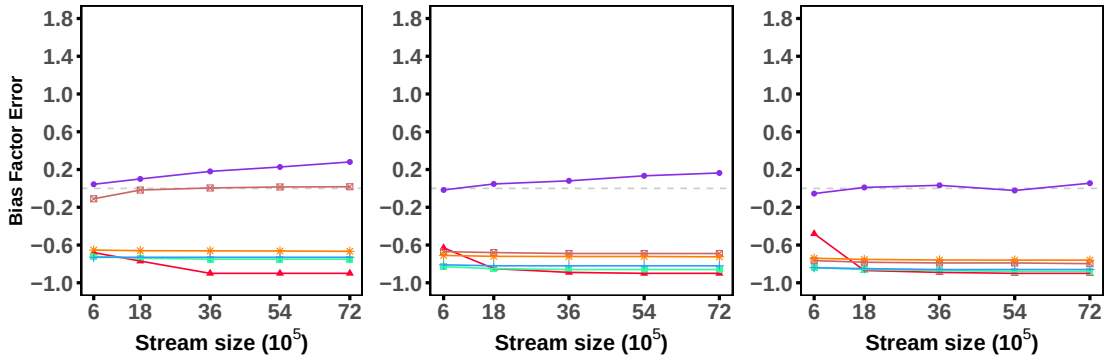
Table 5.4: Evaluating Precision/Recall depending on number of distinct node identifiers: $20,000$, $25,000$, $30,000$, $35,000$, $40,000$, respectively. Budget is 40 KB, $M = 600,000$.

tions, with CMMCU and LCU diverging the most, as before.

Classification metric. For $\gamma = 2$ (Figure 5.14a), when the number of Byzantine



(a) $\gamma = 2$.

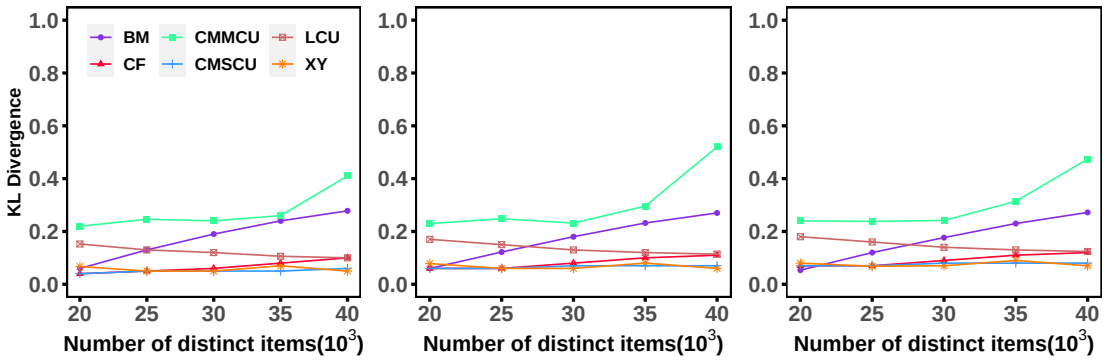


(b) $\gamma = 10$.

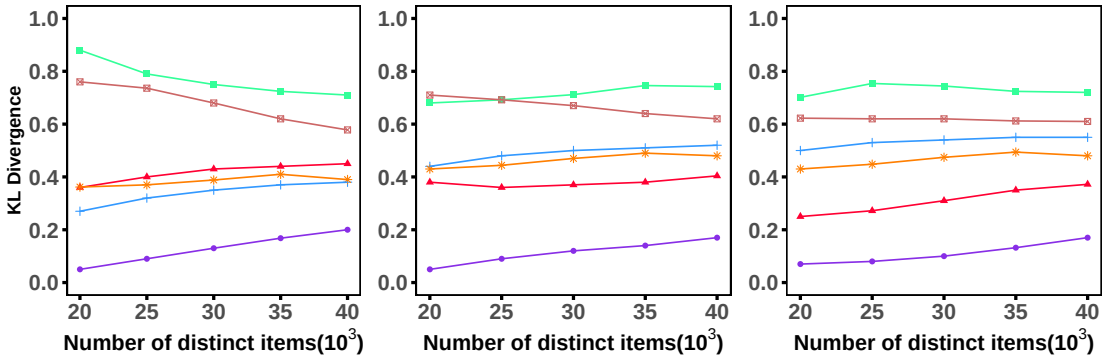
Figure 5.12: Error on Bias Factor vs. stream size on malicious case. Budget is 40 KB, $N = 20,000$, and $f = 10\%, 20\%, 30\%$.

IDs is set to 10%, BM achieves F1-scores of 80% provided that the number of distinct identifiers is less than 25,000. However, at larger system sizes, BM yields suboptimal results, with F1 Scores falling below 0.5. This is due to a drop in precision, as evidenced by the results presented in Table 5.4a, which are explained by an increase in collisions due to a low fingerprint size. For higher proportions of Byzantine identifiers (20% and 30%), BM achieves high F1-scores ranging from 60% to 90%. Regardless of the configuration, other strategies exhibit unsatisfactory F1 scores due to low precision. However, higher Byzantine fractions enhance F1 scores, with XY achieving a maximum score of 60%.

Figure 5.14b depicts the evolution of F1 Score as the stream size increases when $\gamma = 10$. BM scores around 100% precision and recall consistently (see table 5.4b). CMMCU performs the worst in all cases due to low precision and recall. CF demonstrates the poorest precision, indicating that many correct identifiers are reported as Byzantine. Conversely, CF achieves the highest recall rate



(a) $\gamma = 2$.



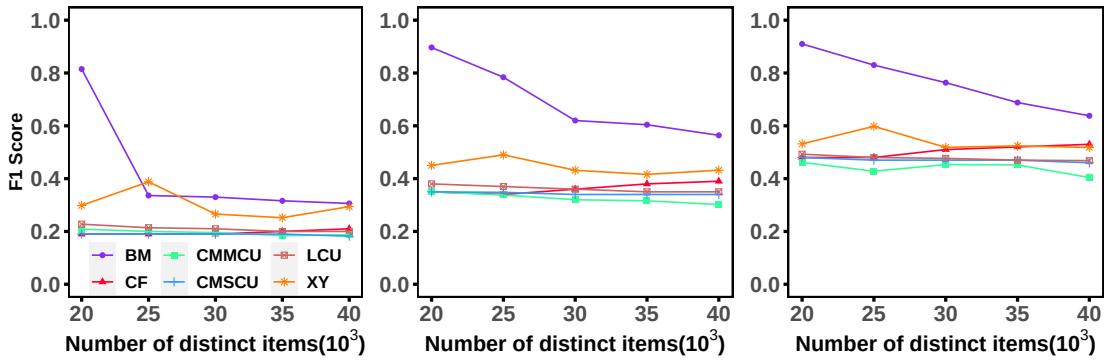
(b) $\gamma = 10$.

Figure 5.13: KL divergence vs. number of distinct node identifiers on malicious case. Budget is 40 KB, $M = 600,000$, $f = 10\%, 20\%$ and 30%.

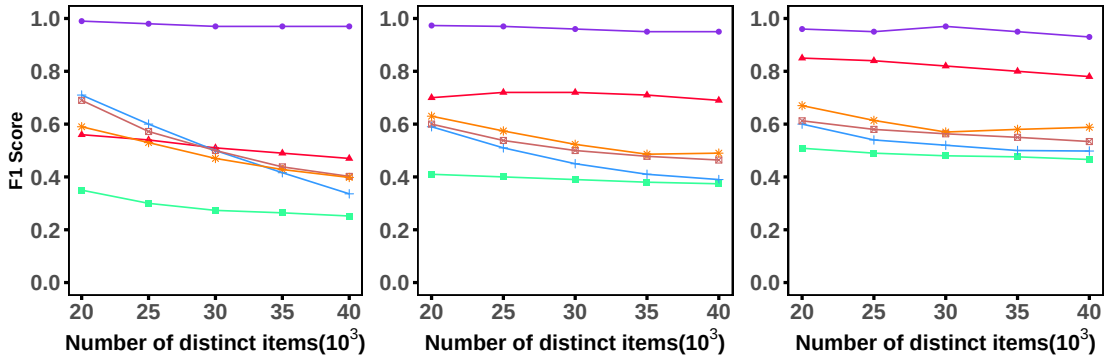
(100%), demonstrating its ability to filter out all Byzantine identifiers. Overall, more Byzantine IDs lead to better classification performance.

Bias factor estimation. For $\gamma = 2$, as shown in Figure 5.15a, BM provides a bias factor that is close to the optimal value. Except for LCU, the bias-factor errors in the other sketches are insensitive to increases in the number of distinct identifiers.

When $\gamma = 10$, underestimation errors remain stable as the system scales across all strategies except for LCU, which worsens in the first configuration ($f = 10\%$). BM achieves the most accurate coefficient estimates with almost no error at $f = 30\%$.



(a) $\gamma = 2$.



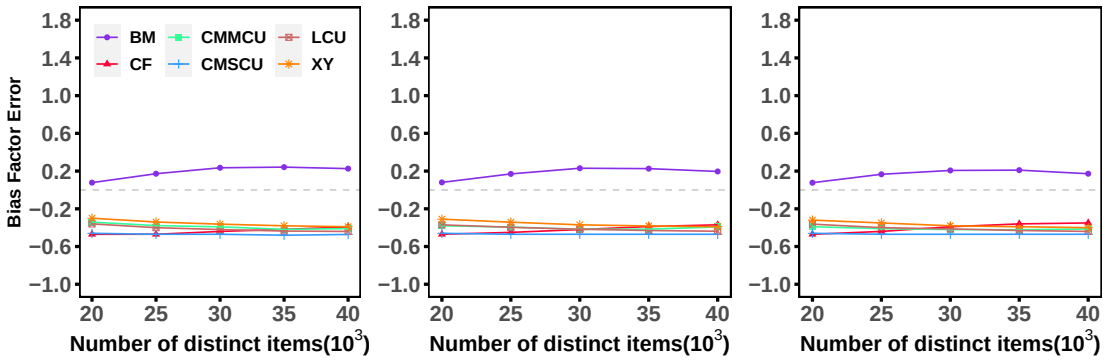
(b) $\gamma = 10$.

Figure 5.14: F1 score vs. number of distinct node identifiers on malicious case. Budget is 40 KB, $M = 600,000$, $f = 10\%, 20\%$ and 30%.

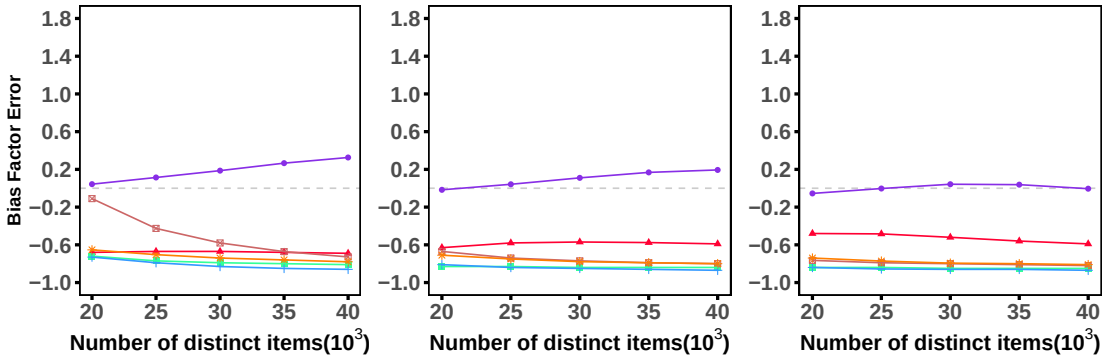
5.5 Discussion

Our experiments highlight several key trade-offs across the six evaluated sketches. In fault-free scenarios, the Count-Min sketch with conservative updates and the Cold Filter consistently achieve the lowest KL divergence, closely preserving the uniform distribution. BM achieves competitive results, but it is more sensitive to stream size. In contrast, strategies that discard residuals after each query (CMMCU, LCU, and XY) exhibit unexpected degradation as memory increases. BM improves with higher memory.

Under adversarial settings, BM offers the most robust performance. It maintains a low KL divergence, high classification metrics (precision, recall, and F1), and accurate estimates of the bias factor γ, even when the Byzantine fraction reaches 30%. Most other estimators, including CMSCU and CF, systematically underestimate γ, which prevents them from distinguishing between low and high frequencies.



(a) $\gamma = 2$.



(b) $\gamma = 10$.

Figure 5.15: Error on Bias Factor vs. number of distinct node identifiers on malicious case. Budget is 40 KB, $M = 600,000$, $f = 10\%, 20\%$ and 30%.

While promising for low-frequency items, XY suffers from low recall and inaccurate bias-factor estimation under adversarial conditions, particularly when a high proportion of nodes are Byzantine. In terms of scalability, CMSCU and CF perform better in both M and N dimensions. BM remains competitive, but it exhibits degradation in precision, classification performance, and factor estimation when the stream or system size is large relative to available memory. Overall, no single estimator dominates, but BitMatcher would be the best choice under malicious pressure, while CMSCU and CF are strong candidates for uniform workloads and bounded-memory environments.

5.6 Conclusion

This work presents a comparative study of six sketches that are suitable for estimating ID frequency in the context of adversarial sampling. Using realistic

datasets inspired by the Bitcoin address propagation mechanism, we evaluated the strengths and limitations of each sketch under fault-free and malicious conditions. BitMatcher stands out for its ability to accurately detect low- and high-frequency identifiers during aggressive attacks. However, scaling remains a challenge for BitMatcher. This motivates the design of an alternative sketch suitable for long-running peer sampling applications.

Chapter 6

Decay and Merge Strategies for Byzantine-Tolerant Peer Sampling

Frequency estimation in data streams is a fundamental problem in distributed systems, with applications ranging from network monitoring to peer sampling protocols. In the context of Byzantine-tolerant peer sampling, accurate frequency tracking enables nodes to identify and counteract adversarial behavior by detecting identifiers that appear with abnormally high frequency. The AUPE protocol, presented in Chapter 4, leverages this principle by equipping nodes with a Set Cleaner component that tracks occurrences of received identifiers and reduces bias in the view construction process accordingly.

In large-scale dynamic environments, the number of distinct identifiers seen by a node can be extremely high, making exact counting impractical due to excessive memory and processing requirements. In Chapter 5, we explored sketches, memory-efficient data structures that estimate identifier frequencies with rigorous error bounds by mapping received identifiers onto a limited set of shared counters. However, this summarization comes at a cost. The fixed-size nature of sketches means that estimation errors grow with the volume of processed data, and counters may eventually approach or exceed their maximum capacity.

To address the challenge of summarizing unbounded streams in fixed-size memory, the literature has proposed a range of strategies that can be broadly categorized into four families. *Temporal decay functions* [CSSX09, CM14] assign decreasing weights to older observations, ensuring that recent data contribute more to frequency estimates, but require per-element bookkeeping (timestamps, floating-point computations) that is unsuitable for resource-constrained settings. *Sliding window mechanisms* [DGIM02, PGD12] restrict analysis to the most recent observations, providing clear semantics but introducing abrupt discontinuities at window boundaries and memory overhead for tracking element expiry. *Hierarchical aggregation* [MSA12, ZYL+21] maintain multiple sketch instances covering differ-

ent time intervals, managing staleness by swapping, merging, or discarding entire sketches at interval boundaries, but with an increased memory footprint. *Counter aging strategies* such as counter halving [EFM17], Lossy Counting [MM02], and probabilistic decrements [DR06], provide implicit aging with the lowest overhead, making them the most natural candidates for integration into lightweight protocols. However, these strategies are designed for linear sketches and for fault-free environments where the input stream, though potentially unbalanced, is not adversarially crafted.

A separate but equally important challenge arises from *sketch merging* in distributed settings. In AUPE peer sampling, trusted nodes operating within secure execution environments can exchange and combine their local sketches to build a more comprehensive picture of system node propagation, based on identifier frequencies. For linear sketches such as Count-Min Sketch, merging is straightforward, with element-wise addition of corresponding counters, and preserves all error bounds [ACH+13]. However, the most accurate modern sketches are non-linear. They employ techniques such as conservative updates, self-adjusting counter widths, or fingerprint-counter coupling that break the linearity property and make distributed merging either ill-defined or semantically unsound. Fingerprint-coupled sketches, those that store a compact hash of the item identity alongside each counter, such as HeavyKeeper [GYZ+18] and BitMatcher [SJL+24], pose the hardest merge challenge. Two fingerprint-coupled sketches constructed independently will usually contain different fingerprints at matching positions. This means that operations based on individual elements are generally invalid without special logic to reconcile them.

Chapter 5 has demonstrated the potential of BitMatcher for frequency estimation, both for uniform and biased identifier distributions. BitMatcher is a fingerprint-coupled sketch that organizes counters into 64-bit buckets, with each bucket containing multiple fingerprint-counter pairs whose bit allocations adapt dynamically to the data stream's frequency distribution. This bit-level counter adjustment mechanism achieves superior memory efficiency compared to traditional sketches such as Count-Min Sketch, while maintaining high accuracy for frequency estimation tasks. However, without a decay operation, counters eventually reach their maximum values, thereby eliminating the ability to distinguish differences in item frequencies. Moreover, it requires a merge procedure to combine two instances of BitMatcher.

To enable AUPE to operate over an unbounded identifier stream, we introduce the BMDecay sketch, which extends the BitMatcher design along two axes. On the *decay* axis, we introduce *controlled transitions*, which restrict bucket state changes to configurations that preserve sufficient fingerprint capacity, ensuring the sketch can continue to track the required number of distinct identifiers. When exhausted,

bucket type transitions trigger a *counter halving* procedure that divides all counters by 2 and rearranges them within the sketch to prevent overflow. Although this decay procedure introduces additional computational overhead, it helps to preserve the sketch accuracy over time. On the *merge* axis, we design a procedure that combines the frequency estimates of two BMDecay instances by taking the maximum of corresponding counters for each stored fingerprint, leveraging the fact that trusted nodes sharing the same hash functions maintain structurally compatible layouts. The resulting merged sketch provides a clearer view of overrepresented identifiers, thereby improving the debiasing quality of the sampling protocol.

The contributions of this chapter are threefold. First, we formalize the decay procedure (BMDecay) and analyze its properties with respect to the accuracy of frequency distributions in long-term stream processing. Second, we introduce the merge procedure, which combines two BMDecay sketches and preserves the fingerprint-counter coupling. These two procedures follow an *extraction-reconstruction* scheme to optimize the storage of fingerprint items. Third, we integrate both mechanisms into the AUPE protocol and conduct experimental evaluation under a range of adversarial scenarios, demonstrating significant improvements in Byzantine resilience compared to configurations without these mechanisms. Our experiments show that without decay, BitMatcher's accuracy degrades by up to 40% in F1 score after processing streams of more than 1 million elements, rendering it unsuitable for long-running peer sampling protocols. In contrast to BMDecay, which ensures that frequency estimation remains viable over streams reaching 10 million elements, enabling AUPE to continuously mitigate Byzantine identifier overrepresentation. Furthermore, BMDecay achieves nearly optimal resilience for Byzantine fractions up to 20% while using 12% of the memory required by exact counting.

The remainder of this chapter is organized as follows. Section 6.1 presents background on unbounded stream summarization using sketches and the specific challenges of merging nonlinear, fingerprint-coupled sketches in distributed settings. In section 6.2, we describe our system model and problem statement. Section 6.3 describes our BMDecay sketch, designed to handle unbounded streams. Section 6.4 analyzes the merging procedure. Section 6.5 presents the experimental methodology used to evaluate the decay and merge strategies in terms of sketch accuracy and their integration with AUPE, across various configurations. Section 6.6 discusses the results. Finally, Section 6.7 concludes the chapter and outlines directions for future work.

6.1 Background and Motivation

This section provides the necessary background on unbounded stream summarization using sketches and motivates the need for an adaptive mechanism for Byzantine fault-tolerant peer sampling. We first review the fundamental challenges of summarizing infinite streams in fixed memory by surveying the main families of solutions proposed in the literature (6.1.1). We then discuss the additional difficulties introduced by sketch merging in distributed settings, with a particular focus on non-linear sketches (6.1.2).

6.1.1 The Challenge of Unbounded Stream Summarization

In large-scale distributed systems, nodes continuously receive streams of identifiers (IDs) from their peers. Unlike bounded datasets that can be stored and processed in their entirety, these streams are potentially infinite: new IDs arrive indefinitely, and the set of distinct IDs may grow without bound. Maintaining exact frequency counts for every observed ID would require memory proportional to the number of distinct elements, which is impractical.

This motivates the use of sketches to provide approximate frequency estimates using fixed-size memory, regardless of the stream length or the number of distinct elements. However, the fixed-size nature of sketches introduces a fundamental tension when processing unbounded streams. First, the more IDs are received, the more collisions occur, leading to more errors in the estimated counts. Second, as more IDs are observed, sketch counters accumulate larger values, eventually approaching or exceeding their maximum capacity. Therefore, a decay strategy [DSHK08] must be applied to the counters so that older elements gradually lose influence and free space for future elements.

The literature has proposed several strategies to handle the temporal dimension of unbounded streams. These approaches can be broadly categorized into four families: temporal decay functions, sliding-window mechanisms, Hierarchical aggregation, and counter-aging strategies.

Temporal Decay Functions. Temporal decay-based approaches assign weights to observations that decrease over time, ensuring that recent data contribute more significantly to frequency estimates than older data. Two main paradigms exist. In *backward decay*, the weight of an observation depends on its age relative to the current query time. This approach is particularly used to detect heavy hitters (high-frequency IDs) [CM14]. For an item arriving at time t_i and queried at time T, the decayed weight is computed as $g(T - t_i)$, where g is a decreasing function such as $g(x) = e^{-\lambda x}$. Each cell in the sketch stores both a decayed count and a timestamp. While effective, backward decay poses implementation

challenges. In fact, each query requires recomputing all weights as of the current time, and out-of-order arrivals necessitate complex adjustment procedures. *Forward decay* [CSSX09] offers an elegant alternative. Rather than computing weights relative to the current time, forward decay measures the distance from a fixed landmark time L (typically the start of observation). The weight of an item arriving at time t_i is $g(t_i - L)$, where g is a non-decreasing function. This eliminates the need to recompute past weights at query time, since each observation's weight is determined once at insertion.

Sliding Window Mechanisms. Sliding windows [ZCQ+23] restrict analysis to the most recent observations (count-based) or to observations within the last τ time units (time-based). Observations inside the window contribute fully to frequency estimates, while those outside contribute nothing. The Exponential Histogram technique [DGIM02] enables approximate counting over sliding windows by grouping arrivals into buckets whose sizes grow geometrically, merging the two oldest buckets when their count exceeds a threshold. Building on this primitive, the ECM-Sketch [PGD12] attaches an Exponential Histogram to each cell of a Count-Min Sketch, enabling per-item frequency estimation over both time-based and count-based. Although sliding windows offer clear semantics, they require additional bookkeeping to track which observations fall within the current window, which necessitates storing timestamps or sequence numbers. For time-based windows, expired observations must be identified and removed, which increases both memory overhead and computational complexity, particularly in high-throughput streaming scenarios. Moreover, frequencies can exhibit abrupt discontinuities at window boundaries when observations suddenly transition from full contribution to zero contribution.

Hierarchical Aggregation. Rather than decaying individual counters, hierarchical approaches maintain multiple sketches covering different temporal granularities. The simplest instance is *epoch-based rotation* (or double buffering) [ZYL+21], which maintains two sketches: an active sketch that receives insertions and a passive sketch that serves queries. Their roles are periodically swapped, with the old active sketch cleared. The mechanism requires only twice the memory of a single sketch and involves no per-element bookkeeping. Similarly to sliding windows, it provides a coarse temporal boundary as only the last full epoch contributes. Hokusai [MSA12] generalizes this idea to a full hierarchy of sketches for time intervals of geometrically increasing size: the current interval, the previous interval of equal size, then intervals covering 2, 4, 8, and so on time units. When an interval boundary is crossed, adjacent sketches are merged, and a new sketch is allocated for the current interval. This design enables point-in-time queries with logarithmic

space overhead in the time horizon, which is impractical for lightweight nodes with fixed memory budgets. Finally, the rigid interval structure provides predictable patterns that an adversary could exploit by concentrating malicious traffic around the transition points.

Counter Aging. When explicit temporal tracking is infeasible, counter manipulation provides implicit aging. Several strategies have been proposed, differing in granularity, overhead, and behavior under adversarial conditions. The simplest mechanism is *multiplicative aging* which applies a factor $\alpha \in (0, 1)$ at regular intervals, setting each counter c to $\lfloor \alpha \cdot c \rfloor$. Counter halving is the special case $\alpha = 1/2$, whose correctness and truncation error bounds were formally analyzed in the context of cache admission [EFM17]. *Lossy Counting* [MM02] takes a different approach by coupling aging with element deletion. For these two strategies, the stream is divided into conceptual buckets. At each bucket boundary, counter values are reduced (divided by two or subtracted by one), and any element whose count reaches zero is removed from the data structure. This provides a deterministic guarantee that any item whose true frequency exceeds a threshold is retained, while infrequent items are progressively pruned. An alternative to synchronous aging is *probabilistic decrement* [DR06]. To avoid the abrupt synchronous drop of multiplicative aging at fixed global intervals, each counter is independently decremented with some probability p upon each insertion or time step. However, it introduces variance in the decay rate across counters, and the stochastic nature of the decrement makes it harder to reason about worst-case guarantees under adversarial conditions. Additive Error Estimators [BEMV20] (AEE) maintains a global sampling probability p, and whenever *any* counter overflows, p is halved and simultaneously every counter $C[i, j]$ is replaced with $\lfloor C[i, j]/2 \rfloor$ (deterministic) or $\text{Bin}(C[i, j], 1/2)$ (binomial probability).

Moreover, most decay designs assume linear sketches, such as the Count-Min sketches [CM05]. Non-linear or fingerprint-coupled sketches such as HeavyKeeper [GYZ+18] or BitMatcher [SJL+24] introduce item-dependent states that interact with decay in complex ways. They employ decay techniques specific to the complexity of their structure. For example, HeavyKeeper introduced *count-with-exponential-decay*. Rather than applying a global aging step, decay occurs per insertion and only upon hash collisions. In fact, when a new item maps to a bucket already occupied by a different fingerprint, the existing counter is decremented with probability b^{-C}, where C is the current counter value and $b > 1$ is a decay parameter. Similarly, BitMatcher uses *replacement strategy* to decrease the count of low counter items when a new item does not find an empty entry in the bucket during its insertion in the sketch. This makes large counters unlikely to be evicted, while small counters are quickly displaced. The decay is thus

flow-dependent and does not prevent counters from overflowing in the long term.

In addition, temporal decay and sliding windows require per-element bookkeeping (timestamps and floating-point decay computations), which is unsuitable for resource-constrained settings with scarce memory and minimal computational overhead. Hierarchical aggregation multiplies the memory footprint and accumulates error over long horizons. Counter-aging strategies offer the lowest overhead and are thus the most natural candidates for integration into a lightweight peer-sampling protocol. These observations motivate the design of a decay mechanism specifically tailored for non-linear fingerprint-coupled sketches.

6.1.2 The Challenge of Sketch Merging

In distributed architectures, sketches are deployed on multiple measurement nodes and must be merged to obtain a global view. This merging operation is fundamental to peer sampling protocols, in which each trusted node maintains a local sketch summarizing its observations and periodically exchanges sketches with neighbors to build a system-wide picture of ID frequencies. Agarwal, Cormode et al. [ACH+13] formalized the mergeability of data summaries, defining it as the ability to combine two summaries on two data sets into a single summary on their union while preserving both error and size guarantees.

Linear Sketches. A sketch is said to be *linear* if it computes a linear function of the input frequency vector [CM05]. Formally, given a stream σ that induces a frequency vector $a \in \mathbb{R}^n$ (where a_i counts the occurrences of element i), a linear sketch is a data structure whose internal state can be expressed as $S = M \cdot a$ for some fixed matrix M determined by the hash functions it uses. The key consequence of linearity is *additivity*: given two sketches $S_1 = M \cdot a$ and $S_2 = M \cdot b$ built with the same hash functions on streams inducing frequency vectors a and b respectively, their element-wise sum $S_1 + S_2 = M \cdot (a + b)$ is exactly the sketch one would have obtained by processing the concatenated stream. Linear sketches are *trivially mergeable* with element-wise addition, which is associative, commutative, and preserves all error bounds. This class includes the Count-Min Sketch and the Count Sketch [CCFC02] among others.

Non Linear sketches. Not all useful frequency estimation data structures are linear. For example, the Count-Min Sketch with Conservative Update (CM-CU) [EV03] update function is not a linear function of the frequency vector. Nevertheless, CM-CU remains mergeable via element-wise addition, although the merged result may have higher error than a single CM-CU sketch built on the concatenated stream, since the conservative property is lost during merging.

SALSA [BEMV21a] proposes merging two sketches by traversing their counters and performing sum-merging. Each counter in the merged sketch is at least as large as it was in the previous sketches. If the sum exceeds the capacity of the resulting counter, a local merge with adjacent counters is triggered to obtain sufficient bits, which progressively reduces the capacity of the sketch.

Elastic Sketch [YJL+18b] is a two-level structure designed from the outset for distributed merging and bandwidth adaptation. It comprises a heavy part (a small hash table with a voting/eviction mechanism to capture high-frequency items) and a light part (a standard Count-Min Sketch for low-frequency items), with a configurable threshold between the two, similar to Cold Filter [YJZ+19b]. The paper recommends using *Maximum Merging* (element-wise maximum) for merging the light parts of two given sketches, especially when they receive disjoint traffic. This strategy avoids amplifying noise from independent sketches. For the heavy, non-linear part, it operates on the key union, increasing the memory footprint.

In fact, non-linear fingerprint-coupled sketches that process different traffic will generally have different item sets in their counters. In this case, naive element-wise merging operations such as sum, maximum, or average are undefined, especially when a fixed memory boundary is involved. This motivates the design of a distributed merging algorithm for nonlinear sketches, particularly for BitMatcher, which is among the most accurate in peer sampling.

6.2 System model and problem statement

We consider a system composed of N active nodes, each with a unique identifier (ID). Each node has a list of v neighbors, called a view, with which it communicates via a routed protocol. Nodes run the AUPE peer sampling protocol and receive streams of IDs through push and pull answers. We assume that the system is at time T_0, when no nodes join or leave.

A fraction $f < 1$ of the nodes in the system are Byzantine and do not respect the protocol. These Byzantine nodes attempt to overrepresent themselves in the views of correct nodes by executing a balanced attack [BGK+08], starting at time T_a. The remaining fraction $h = 1 - f$ of correct nodes follow the AUPE protocol. In addition, a fraction t of nodes, referred to as trusted nodes, operate within a trusted execution environment and collaboratively track the spread of identifiers within the system.

In the AUPE peer sampling protocol, nodes track the frequencies of received IDs from the beginning of the protocol, and use these frequencies to downweight overrepresented IDs (which are likely Byzantine) and restore uniformity to the sampling process. In the original design, the tracking component is an array that stores the exact frequencies of the received IDs. However, due to the $O(N)$ memory

footprint, an alternative approach is to provide frequency estimates with a fixed memory budget using sketches.

According to the analyses in Chapter 5, the sketch that achieves the best accuracy is the BitMatcher. Under the condition of a sufficient memory budget, the estimated frequency distribution of received IDs, obtained by querying the BitMatcher, exhibits a high degree of similarity with the true frequency distribution. In the face of intense attacks, BitMatcher has been shown to perform well in classifying high- and low-represented IDs with high precision and recall. Furthermore, BitMatcher can estimate the bias factor of a received stream of IDs, that is, the extent to which the class of overrepresented identifiers dominates in occurrence over the class of underrepresented identifiers, thus with high accuracy.

However, as the total and distinct numbers of IDs increase, its performance declines. In fact, the design has limitations with unbounded streams. As insertions are performed, small counters are sacrificed to increase larger ones. While this delays future overflow, it also reduces the number of distinct IDs that the sketch can track. In turn, this increases the estimation error of untracked fingerprints, resulting in a degradation of accuracy over time. Furthermore, the possible transitions between bucket types eventually become exhausted, causing future insertions to be missed. This problem highlights the need for an adaptive strategy to handle unbounded streams.

6.3 BitMatcher decay

To address these limitations of BitMatcher for unbounded stream summarization, we propose the BMDecay sketch, which adds two components in BitMatcher design: (1) *controlled transitions* that limit bucket state changes to preserve fingerprint capacity and (2) *counter halving* that periodically halves all counters and reorganize them in the sketch, to prevent overflow while preserving as much information as possible.

6.3.1 Controlled transitions

Let us consider a BitMatcher sketch composed of two tables as in the former design. The sketch constantly receives new identifiers, yet it operates within a fixed space. Eventually, the space fills up, resulting in continuously worse estimates. An overflowing counter, which is a full counter, must be reallocated either by increasing it (swapping its entry for a larger one) or by moving it into the corresponding bucket in the other array. In the former design, if this reallocation fails, the entry with the smaller counter bit width must be sacrificed, which could cause the bucket

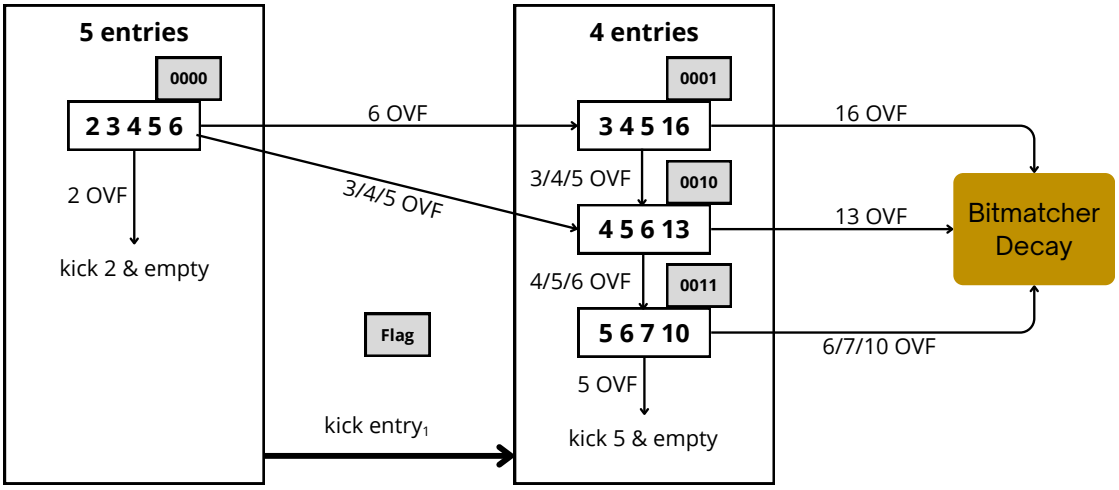


Figure 6.1: BMDecay bucket transition from state 0 to 3.

to shrink and reduce the number of entries. This strategy allows high-frequency items to gradually occupy the largest entries in buckets.

However, due to bucket shrinkage, fewer identifiers can be mapped into the bucket. In a scenario where the input stream contains many overrepresented items (as in the balanced peer sampling attack), some high-frequency items in the lowest-bit-capacity entries are sacrificed during shrinkage. Or their count faces underrepresentation errors as the insertion of new items in the sketch triggers the replacement strategy (counter value minus one). This problem is even more acute for an infinite stream. As insertions are made, the possible transitions are exhausted, and the buckets eventually reach the maximum state. This leads to overflow counters being blocked and future insertions being ignored.

To address the decline in sketch capacity, we propose limiting bucket transitions to state 3 only (flag `0011` in Figure 6.1). In fact, a transition from state 0 to any state beyond state 3 causes the bucket to lose at least two of its five entries, reducing fingerprint capacity by at least 40%. This ensures that each bucket contains at least four entries, which maintains a minimal sketch capacity throughout the protocol. An overflow that reduces a bucket to three entries triggers the decay procedure.

6.3.2 Counter halving

Due to controlled transitions, several counters will experience unresolvable overflows. These overflows cannot be resolved through reallocations or state transitions. To enable future insertions into these counters, we apply a decay procedure to the entire sketch. This reduces the space required to store count values while

preserving the sketch's relative frequency distribution, which is essential for the sampling algorithm to function properly.

One simple decay procedure that ensures this property is to halve each sketch counter. Assuming there are no rounding errors, if an item P appeared twice as frequently as an item Q in the initial sketch, it would appear at twice the frequency after decay. As counter values decrease through decay, fewer bits are required to represent them. Then, freed bits can be physically recovered and reallocated to other entries within the same bucket. This motivates our *extraction-reconstruction* strategy, which extracts the fingerprints and their associated counts, then reconstructs the entire sketch.

When a counter drops to zero, both its counter bits and fingerprint bits are freed, making an entire slot available again. Moreover, as items are received, the decay is continuously applied to the sketch, highlighting overrepresented items. Using this counter halving frees up half the memory and postpones the need for future decay operations.

Decay procedure. Algorithm 2 describes this decay procedure, which operates in two phases following the extraction-reconstruction scheme. During the extraction phase (lines 1-10), all non-empty slots across both hash tables A_1 and A_2 of the sketch are scanned. Fingerprints fp are extracted with their canonical bucket index h_1 (which uniquely references item positions in table A_1), and counter values c, which are halved by a single right bit-shift ($\lfloor c/2 \rfloor$). Items residing in table A_2 require recovering h_1 from the relation $h_1 = h_2$ XOR $hash(fp)$, where h_2 is the item bucket index in A_2.

Then, the extracted items are sorted in descending order by their halved counts. This ordering ensures that high-frequency items, which carry the most important frequency information, are re-inserted first and thus receive priority access to bucket slots with sufficient bit capacity. After that, both sketch hash tables are cleared by resetting all their buckets to zero. This way, the sketch regains its initial capacity (five entries per bucket), which is beneficial for the reconstruction phase.

The reconstruction phase is described in Algorithm 3. Items are re-inserted one by one in sorted order. For each item (fp, h_1, c), the two candidate buckets are $A_1[h_1]$ and $A_1[h_2]$. The insertion proceeds in two steps. First, both candidate buckets are scanned in order to find an empty slot whose bit width is sufficient to represent c. If such a slot exists, the item is placed there immediately (lines 4-7). This is the common case after a decay, since halving all counters frees the sketch considerably, and most items find a compatible slot without further effort.

Second, if no candidate bucket contains a suitable empty slot, the item is handed to the standard InsertByFp routine, which handles overflows. Moreover,

Algorithm 2: Decay Procedure

Input: Sketch S with two hash tables A_1, A_2, each with w buckets
Output: Sketch with halved counters, high-count items prioritized

```
/* Phase 1: Extract and Divide                                    */
```
$\mathcal{I} \leftarrow \emptyset$;
foreach *table* $t \in \{1, 2\}$ **do**
  foreach *bucket index* i *in* A_t **do**
    foreach *non-empty slot* j *in bucket* i **do**
      $fp \leftarrow A_t[i][j].fp$;
      $c \leftarrow A_t[i][j].count$;
      if $c > 1$ **then**
        if $t = 1$ **then** $h_1 \leftarrow i$;
        else $h_1 \leftarrow i$ XOR $hash(fp)$;
        $\mathcal{I} \leftarrow \mathcal{I} \cup \{(fp,\ h_1,\ \lfloor c/2 \rfloor)\}$;
Sort $\mathcal{I}$ in descending order by count;
foreach *table* $t \in \{1, 2\}$ **do**
  Reset all buckets in A_t to zero;

```
/* Phase 2: Reconstruct with Capacity Optimization                */
```
S.ReinsertItems($\mathcal{I}$); // See Algorithm 3

as the reconstruction phase is executed on a known list of items, we removed the replacement strategy. Because the list of items is sorted, overflows are handled only via bucket transitions.

The decay mechanism operates transparently within the BitMatcher tracking component and is triggered automatically once all possible transitions for a bucket have been exhausted. No changes to the Aupe protocol logic are required as the tracking component simply exhibits consistent behaviour over extended execution periods.

6.4 Merging BMDecay sketches

Merging tracking components has been shown to improve local knowledge of node ID frequencies in the Aupe peer sampling protocol. With an array as Aupe tracking component, the merging operation is the mean of the corresponding entries. Similarly, when using a sketch such as the Count-Min Sketch, two sketches can be merged by summing their corresponding counters, provided they are the

Algorithm 3: Sketch reconstruction

Input: Sorted list $\mathcal{I} = \{(fp, h_1, c)\}$ in decreasing order of c; cleared sketch tables A_0, A_1

Output: Sketch repopulated with items from $\mathcal{I}$

```
1 foreach (fp, h_1, c) ∈ I do
2     h_2 ← (h_1 XOR hash(fp));
3     inserted ← false;
4     foreach t ∈ {1, 2} do
5         if ∃ empty slot j in A_t[h_t] with c ≤ 2^CountBits(A_t[h_t], j) − 1 then
6             Place (fp, c) in slot j;   inserted ← true;
7             break;
8     if ¬ inserted then InsertByFp(fp, h_1, c);
```

same size and use identical hash functions. However, merging two BMDecay sketches is significantly more challenging due to their bit-level counter adjustments and fingerprint management.

Suppose two trusted nodes aggregate their information by merging their local sketches. We designed a merging procedure that combines the information from the two sketches, helping them to acquire more information about the global representation of the nodes in the system To ensure consistent merging, the hash functions used for the bucket and fingerprint calculations in the two sketches must be identical. We use a maximum-based merging procedure, which minimises overestimation and underestimation errors compared to sum-based merging. As recommended in Elastic Sketch [YJL+18b], merging by maximum is most suitable for mitigating the fact that an element common to both sketches may represent different node IDs observed by the two merging nodes. Furthermore, as maximum merging operations are performed over time, overrepresented identifiers in the global system are highlighted in the sketch.

Merge procedure. The merge procedure, summarized in Algorithm 4, followed the same extraction-reconstruction scheme as the decay. During the extraction phase (lines 1-12), both input sketches S_A and S_B are scanned exhaustively. For each item encountered, its fingerprint fp and canonical bucket index h_1 (as indexed in table A_1 or recomputed from table A_2) are used as an item composite key. The counts from both sketch items are merged into a hash map $\mathcal{M}$ using element-wise maximum.

Then the merged items are converted to a list and sorted in descending order by their resulting counts. As in the decay procedure, this ordering ensures that high-

Algorithm 4: Merging Two BMDecay Sketches

Input: Two sketches S_1 and S_2, each with hash tables A_1, A_2
Output: Merged sketch S_{12} containing the element-wise maximum of both sketches

```
/* Phase 1: Extract and merge counts from both sketches */
```
$\mathcal{M} \leftarrow$ empty map from (fp, h_1) to count;
foreach *sketch* $S \in \{S_1, S_2\}$ **do**
  foreach *table* $t \in \{1, 2\}$ **do**
    foreach *bucket index* i *in* $S.A_t$ **do**
      foreach *non empty slot* j *in bucket* i **do**
        $fp \leftarrow S.A_t[i][j].fp$;
        $c \leftarrow S.A_t[i][j].count$;
        if $t = 1$ **then**
          $h_1 \leftarrow i$;
        else
          $h_1 \leftarrow i$ XOR hash(fp)
        $\mathcal{M}[(fp, h_1)] \leftarrow \max\big(\mathcal{M}[(fp, h_1)],\ c\big)$;
$\mathcal{I} \leftarrow \big[(fp, h_1, c) \mid \big((fp, h_1), c\big) \in \mathcal{M}\big]$;
Sort $\mathcal{I}$ in descending order by c;
```
/* Phase 2: Reconstruct the new sketch */
```
S_{12}.ReinsertItems($\mathcal{I}$); // See Algorithm 3

frequency items receive priority during reconstruction, maximizing their chances of being placed in a slot with sufficient bit capacity. After that, the sorted items are re-inserted using the same procedure described in Algorithm 3. The merge-by-maximum strategy is commutative, as merge(A, B) produces the same result as merge(B, A), and associative, as merge(merge(A, B), C) equals merge(A, merge(B, C)).

Finally, the merge strategy integrates seamlessly with Aupe's trusted-node collaboration mechanism. When a trusted node receives a BMDecay instance from a peer, it invokes the merge procedure to combine the received sketch with its own. This serves as the basis for subsequent debiasing operations, incorporating frequency information from both local and peer observations. After sufficient rounds of exchange between trusted nodes, their sketch instances converge towards a shared understanding of identifier frequencies. This enables more effective debiasing than would be possible through isolated operation, benefiting other correct nodes.

6.5 Evaluations and results

We evaluate the effectiveness of the decay and merge strategies through simulation experiments conducted on the AUPE protocol. Our evaluation addresses four research questions. First, how accurately does the BMDecay sketch estimate frequencies over extended stream processing? Second, does AUPE with BMDecay provide effective debiasing performance over the course of extended protocol execution? Third, how quickly the merge strategy enables the two merging sketches to learn the node propagation. Finally, how does merging their sketches help trusted nodes achieve less-biased views more quickly and benefit the system during peer sampling?

Experimental protocol. We implemented BMDecay support in C++ by modifying the BitMatcher's official code [Wen24]. Then, we integrated it with the AUPE simulator infrastructure implemented in Rust.

To run simulations and evaluate the decay within a reasonable timeframe, our experiments consider networks of $N = 1,000$ nodes, each with a unique identifier. For the peer sampling experiment, each node maintains a view of $v = 20$ entries. We varied the Byzantine fraction f from 10% to 30% of the network population. For the merge strategy, trusted nodes constitute a fraction $t \in \{10\%, 20\%\}$ of the network. The remaining fraction of nodes $(1 - f - t)$ represents honest nodes.

6.5.1 BMDecay Evaluation

We evaluate BMDecay in two scenarios. First, in a streaming scenario, to isolate the effect of decay from the dynamics of the peer sampling protocol. Second, we integrate BMDecay into AUPE to evaluate its impact on resilience, which is the average proportion of Byzantine samples in correct node views, under an attack.

Streaming scenario. To evaluate the accuracy of BMDecay, we compare it to the former BitMatcher sketch over extended stream processing. We stored the stream in an array to obtain the exact frequency distribution of IDs. Then, we fed each sketch the stream and queried it to obtain an output frequency distribution of the stream IDs. After querying all sketches, we compared their output frequency distributions with the exact frequency distribution provided by the array, using the metrics defined in Chapter 5. The Kullback–Leibler (KL) divergence estimates how closely the output distribution resembles the exact distribution. The F1 score estimates a sketch's ability to preserve the classes of overrepresented (Byzantine) and underrepresented (correct) items, as determined by K-means clustering with K = 2. The bias factor error determines the discrepancy between the original and

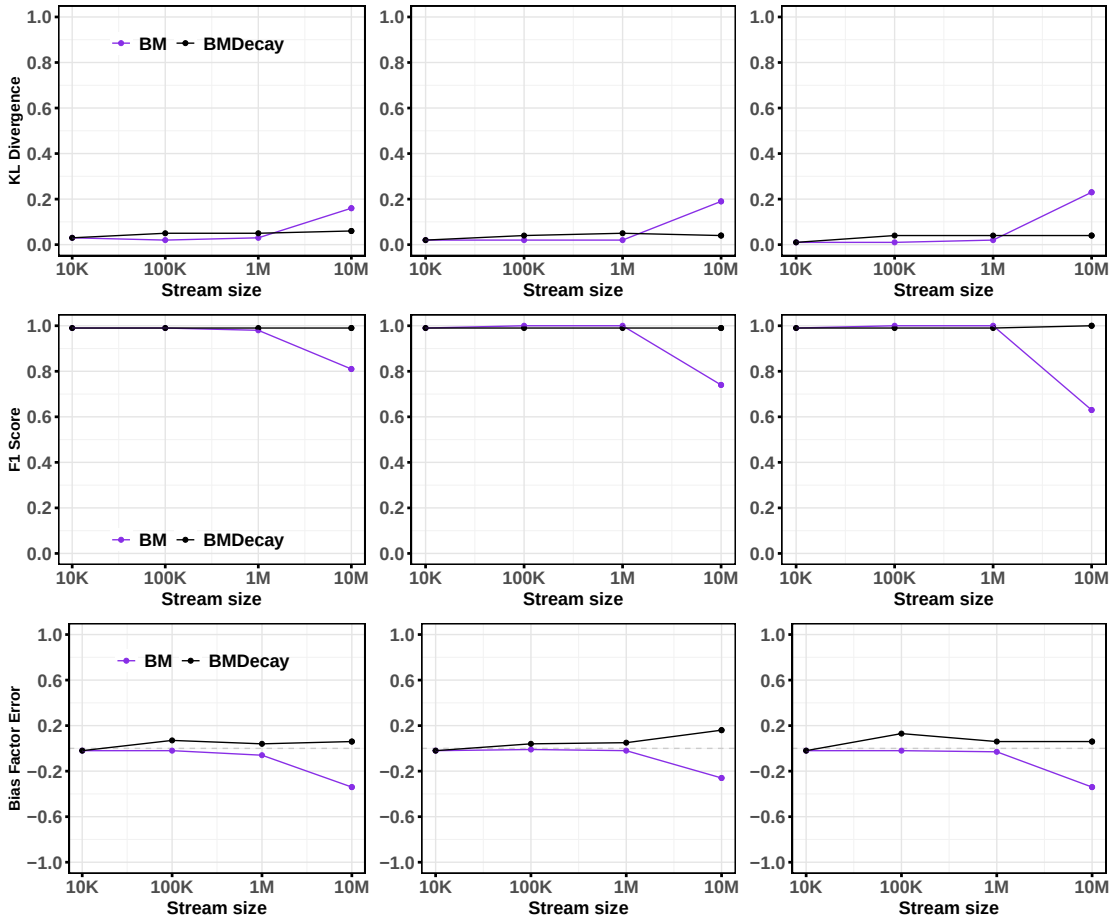


Figure 6.2: Metrics vs. stream size for the malicious case. $N = 1,000$, $f = 10\%$ (left), 20% (middle), 30% (right) and $\gamma = 10$. Budget is 500 bytes.

approximate bias factors. These factors indicate the extent to which Byzantine IDs are overrepresented relative to correct IDs within a given frequency distribution.

The system comprises $N = 1,000$ distinct IDs. We generated three streams of $10,000,000$ IDs drawn from the identifier space, one per Byzantine fraction f, following a distribution in which Byzantine IDs appear more frequently than correct IDs, with a bias factor $\gamma = 10$. BitMatcher-based strategies use a 500 bytes budget (12% of an array's space), corresponding to 32 buckets per sketch table. We compute these metrics as the stream is received at various checkpoints, 10K, 100K, 1M, and 10M.

Figure 6.2 shows the evolution of the KL divergence, F1 score, and bias factor error as the stream size increases. The graph illustrates that BitMatcher performance degrades as stream elements are received, especially when the number of

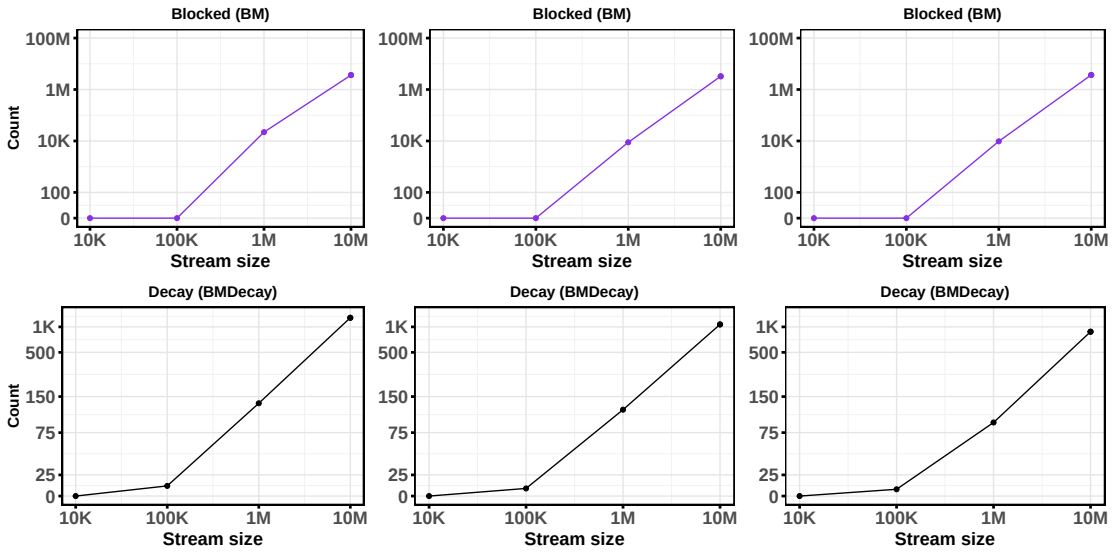


Figure 6.3: At the top, the number of blocked insertions in BitMatcher sketch. At the bottom, the number of decays in BMDecay. Budget is 500 bytes, $N = 1,000$, and $f = 10\%, 20\%, 30\%$.

insertions exceeds 1M. For instance, at a byzantine fraction of 30%, the KL divergence of BitMatcher increases beyond 0.2 while the one of BMDecay remains quite stable. The F1 score drops by nearly 40% (from 100% to 60%), and the bias factor error increases to 35% as the number of IDs the BM sketch receives increases from 1M to 10M. Moreover, the BitMatcher sketch progressively underestimates the proportion of overrepresented IDs in the received stream, potentially facilitating a Byzantine attack. This degradation in accuracy metrics observed in the previous figure can be explained by counters saturating within the BitMatcher sketch. We observe that this degradation increases with the number of Byzantine nodes, since their counter values become more important. In contrast, the decay strategy maintains quite stable performance throughout stream processing. This demonstrates that BMDecay is suitable for extended stream processing.

Figure 6.3 shows the evolution of the number of blocked insertions in the BitMatcher sketch and the number of decays triggered in the BMDecay sketch. After processing 10K items, for all configurations, there are neither blocked insertions nor decay operations. For higher stream sizes, the number of blocked insertions and decay increase progressively. Blocked insertions occur when possible transitions are exhausted. As one might expect, this impacts the sketch's accuracy because these insertions are ignored. BMDecay solves this issue by triggering the decay operation when the maximum possible transition is insufficient to handle future overflows.

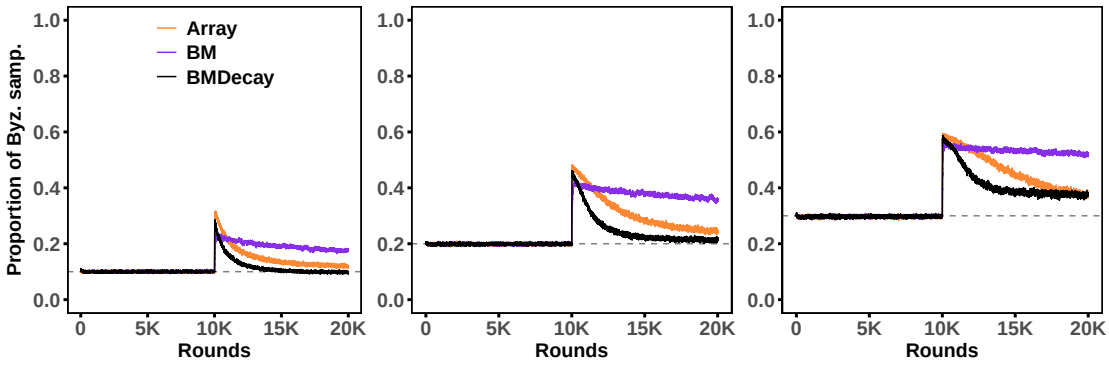


Figure 6.4: AUPE peer sampling: Evolution of the proportion of Byzantine samples across different tracking components. $N = 1,000, v = 20$, $f = 10\%, 20\%, 30\%$. Budget for BitMatcher strategies is 500 bytes.

Evaluation with AUPE. We evaluate the impact of the BMDecay sketch in the AUPE peer-sampling context, compared to BitMatcher and the Array as tracking components. We consider a balanced attack initiated by Byzantine nodes after round $T_a = 10,000$, which corresponds to approximately 200K IDs received by each node via gossiping, and measure how these strategies recover from the attack over a $20,000$ round experiment. The Array baseline represents the theoretical optimum achievable with unlimited memory.

The system comprises $N = 1,000$ nodes, each with a view of $v = 20$ entries. We varied the Byzantine fraction, f, from 10% to 30% of the network population. To capture a long-term experiment, we executed AUPE over 10,000 rounds. Similarly, BitMatcher-based strategies use a 500 bytes budget to accelerate sketch saturation and speed up the simulation.

Figure 6.4 depicts the evolution of the average proportion of Byzantine samples in the views of non-Byzantine nodes for each tracking component. Each curve represents a different tracking component used with AUPE. Before round 10,000, since the attack hasn't started yet, all strategies produce optimal Byzantine samples in the correct node views (matching the real system's Byzantine proportion). After round 10,000, all these strategies experience a peak in the number of Byzantine samples in the view of correct nodes. As expected, AUPE with Array converges to the optimal proportion of Byzantine samples. AUPE with BMDecay converges faster than AUPE array, thanks to the counter-halving procedure, which progressively prunes low-frequency items. Conversely, AUPE with BitMatcher is worse, and its debiasing component struggles to debias the views. This can be explained by an underestimation of the bias factor and a drop in the F1 score, as observed in previous streaming analyses.

6.5.2 BMDecay Merge Evaluation

Similarly, we evaluate the effectiveness of the merge in two scenarios. First, in a streaming scenario, to demonstrate how quickly the merge strategy helps two merging sketches learn from node propagation. Second, we integrate the merging of BMDecay sketches in the collaborative debiasing procedure of AUPE to evaluate how trusted nodes achieve less-biased views more quickly and how it benefits the system.

Streaming scenario. To validate the merge strategy, we conducted experiments in which two sketches, (BM_N1) and (BM_N2), received independent streams sampled from the same underlying distribution. At each time step, each sketch is fed a fixed portion of 20 identifiers drawn from its own stream, then merged with the other sketch to produce ($BMMerge$). In parallel, an exact frequency table ($Oracle$) is fed both stream portions at each step and serves as an oracle with full knowledge of the true global distribution. Both BMDecay sketches use a 1 KB memory budget, corresponding to 64 buckets per sketch table. The identifier space comprises $N = 1,000$ distinct IDs, and each stream is drawn from an unbalanced distribution in which a fraction $f = 30\%$ of IDs appear more frequently than the rest, with a bias factor of $\gamma = 10$. Each stream contains $4,000$ identifiers.

We assess how quickly the merged sketch converges to a reliable estimate of the global distribution using three metrics, all computed relative to the oracle. The first is the fraction of known IDs, which is the fraction of the N identifiers for which the sketch reports a non-zero count. The second is the number of correctly identified overrepresented identifiers, obtained via K-means clustering ($K = 2$, separating high- from low-frequency IDs) applied to the sketch estimates, counted as true positives against the oracle classification. The third is the bias factor error, measuring how accurately the sketch estimates the ratio of frequencies between the two classes relative to the oracle. We compare $BMMerge$ with the two isolated sketches BM_N1 and BM_N2 to quantify the gain from merging.

Figure 6.5 shows how these metrics evolve as the stream is received. Globally, at any given stream size, $BMMerge$ knows a higher fraction of the system nodes than BM_N1 and BM_N2. Similarly, $BMMerge$ learns the system's overrepresented IDs and their factor of this overrepresentation faster than the individual sketches do. This shows that merging the sketches accelerates the knowledge of the node propagation, which is beneficial for collaborative debiasing. However, due to the sketch capacity limitation (maximum of 640 counters with 1 KB), the $BMMerge$ fails to reach the final knowledge of the $Oracle$.

Evaluation with AUPE. To evaluate the merge strategy, we compared AUPE with BMDecay across various trusted-node configurations: $t = 10\%$, $t = 20\%$, and

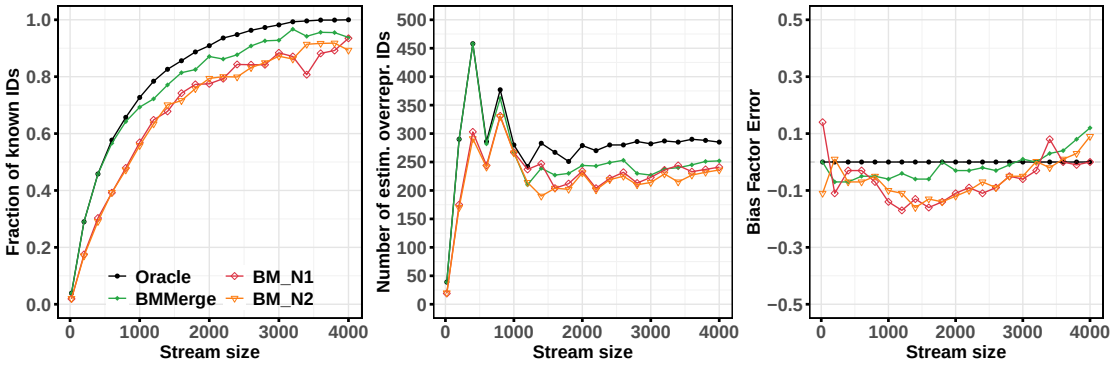


Figure 6.5: Evolution of metrics with the merge $N = 1,000$, $f = 30\%$. Budget is 1 KB.

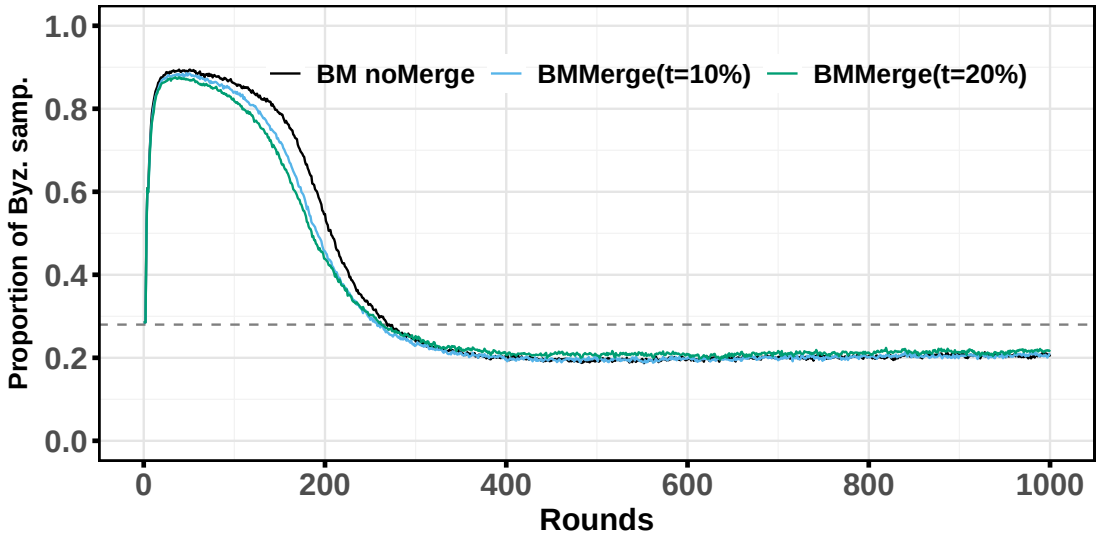


Figure 6.6: Evolution of merge for 200 rounds. $N = 1,000, v = 20$. $f = 28\%$. Budget is 1 KB.

NoMerge ($t = 0\%$). As described in the AUPE protocol, each trusted node maintains and renews a trusted peer list of 10 entries and performs merge operations with the peers on this list. We examine various Byzantine node configurations, ranging from 10% to 40%. Byzantine nodes launch a balanced attack starting at the start of the protocol ($T_a = 0$), and the experiments run for 1,000 rounds.

Figure 6.6 illustrates the evolution of the average proportion of Byzantine IDs in the views of non-Byzantine nodes for various trusted configurations and a fixed fraction $f = 28\%$ of Byzantine nodes. Each non-Byzantine node has a BMDecay

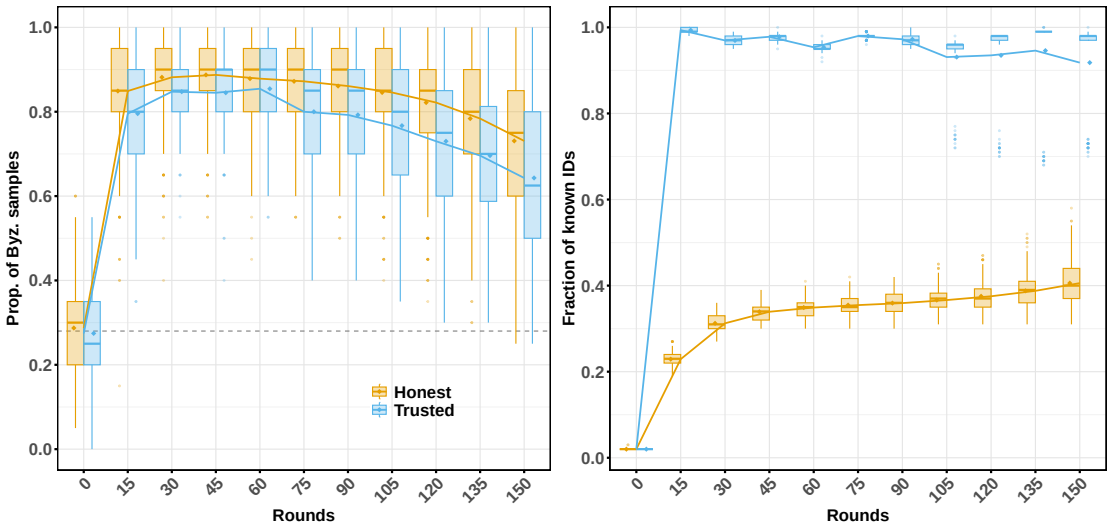


Figure 6.7: Evolution of merge for 150 rounds. $N = 1,000, v = 20$. $f = 28\%, t = 10\%$. Budget is 1 KB.

sketch that occupies 1 KB. The figure shows that having more trusted nodes helps mitigate bias in the views of correct nodes more quickly. At round 150, there is a 15% resilience gap between the configuration without merge and the one with $t = 20\%$. After a few rounds, all the BMDecay Merge variants converge on roughly the same result, suggesting that the merge accelerates debiasing.

In addition, we separate the group of trusted nodes from the rest of the correct nodes (identified as honest nodes) to assess how quickly the merge strategy enables the trusted node sketches to learn from node propagation and mitigate the attack. Figure 6.7 shows the evolution of the average proportion of Byzantine Ids for the group of trusted and the group of honest separately. It shows that, at a given time, most trusted nodes have a lower proportion of Byzantine IDs in their views than other correct nodes. Moreover, trusted nodes can learn the system IDs faster than correct nodes, thanks to their collaboration, and their sketches exhibit this knowledge.

Figure 6.8 shows the impact of collaborative debiasing of AUPE with BMDecay sketches. It illustrates how the average proportion of Byzantine IDs in the views of non-Byzantine nodes evolves as a function of the system's Byzantine ID proportion, f, and for each trusted proportion. Overall, for values of $f < 20\%$, the gains of the merging are negligible. For values between 22 and 34%, it is more detectable with a gain of up to 15% for $f = 28\%$, between the NoMerge and the merge with $t = 20\%$ configurations. For higher proportions of Byzantine nodes ($f > 34\%$), the gains of merging are also negligible.

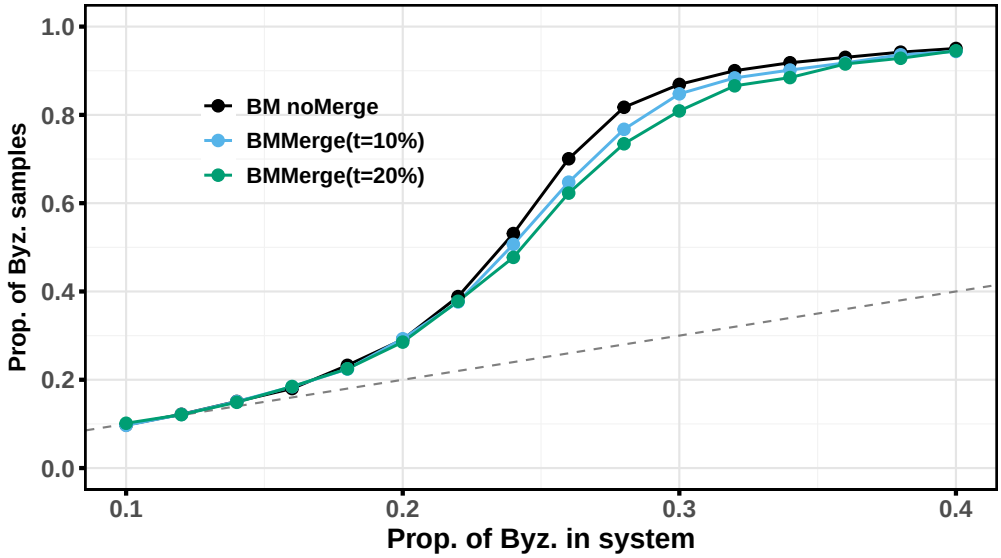


Figure 6.8: Impact of merge for 150 rounds. $N = 1,000, v = 20$. Budget is 1 KB.

6.6 Discussion

The results confirm that counter saturation is a critical obstacle for sketch-based debiasing in long-running peer sampling protocols. The standard BitMatcher sketch progressively underestimates the overrepresentation of Byzantine identifiers as counters reach their maximum values, with F1 score degrading by up to 40% between 1M and 10M stream elements. BMDecay solves this problem by triggering counter-halving when bucket transitions are exhausted. It maintains stable accuracy throughout the entire stream, at a cost of only 500 bytes of memory, which represents 12% of the memory required for exact counting with the Array. Although the 500-byte memory experiments for decay showed that BMDecay converges faster than the Array baseline, larger budgets would simply delay saturation and reduce the convergence gap between BMDecay and the Array.

The merge experiments show that collaborative debiasing accelerates knowledge acquisition. Merged sketches detect overrepresented identifiers and estimate bias factors faster than isolated sketches. However, the bounded capacity of sketches imposes a ceiling that merging cannot overcome. Within AUPE, the benefit of merging is concentrated in the range $f \in [22\%, 34\%]$, with up to 15% improvement at $f = 28\%$.

6.7 Conclusion

This paper presented two complementary strategies that extend BitMatcher to support Byzantine-tolerant peer sampling protocols: a decay mechanism (BMDecay) for handling unbounded streams and a merge procedure for collaborative frequency tracking among trusted nodes. The decay strategy addresses the fundamental limitation of fixed-size sketches in long-running protocols. By globally halving counter values, BMDecay maintains stable accuracy over arbitrarily long execution periods. Our experiments demonstrate that BMDecay preserves classification accuracy and low KL divergence even after processing 10 million stream elements, whereas standard BitMatcher degrades significantly beyond 1 million elements. When integrated with the Aupe peer sampling protocol, BMDecay enables sustained debiasing performance over 20,000 rounds, matching the effectiveness of exact array-based tracking while using orders-of-magnitude less memory. The merge strategy enables trusted nodes to combine frequency information. Our results show that collaborative tracking with 10–20% trusted nodes improves Aupe's Byzantine resilience by accelerating debiasing and reducing the proportion of Byzantine samples in node views, even for Byzantine fractions up to 34%.

Chapter 7

Conclusion and perspectives

7.1 Conclusion

In this thesis, we proposed solutions to improve the resilience of gossip-based peer sampling in large-scale distributed systems subject to Byzantine adversaries. The central challenge was twofold: mitigating the bias that adversarial nodes inject into the identifier stream received by correct nodes, and doing so within the memory constraints inherent to large-scale deployments. We made three contributions: we developed AUPE, a collaborative Byzantine fault-tolerant peer sampling protocol; we conducted a systematic evaluation of sketch-based frequency estimation techniques for adversarial peer sampling; and we designed decay and merge strategies to integrate compact data structures into AUPE for unbounded stream processing. In more detail, we have the following contributions:

Protocol-level defense with AUPE. The AUPE protocol equips each node with a Set Cleaner that combines two mechanisms: a tracking component that records how often each identifier appears in the received gossip stream, and a debiasing component that transforms the biased stream into one that more closely approximates a uniform distribution. High-frequency identifiers are downweighted when producing local samples. The distinctive feature of AUPE is its collaborative debiasing mechanism. A subset of nodes, running inside trusted execution environments such as Intel SGX, periodically exchange and merge their frequency-tracking components. Because each trusted node observes a different slice of the gossip traffic, the merged information provides a more complete picture of the global identifier distribution than any individual node could obtain. We evaluated AUPE using simulations with up to 10,000 nodes. Our evaluations show that AUPE achieves near-perfect resilience when up to 26% of nodes are adversarial, meaning that the adversary gains no advantage from its flooding strategy. This substan-

tially outperforms Brahms and Basalt under equivalent attack scenarios.

Systematic evaluation of sketch-based frequency estimation. We conducted a systematic study of six sketching strategies for frequency estimation in the specific context of adversarial peer sampling: Count-Min Sketch, Count-Mean-Min Sketch, Lossy Conservative Update, Cold Filter, BitMatcher, and Probabilistic Sketch. These strategies were evaluated on synthetic streams modeling adversarial injection scenarios with varying attack intensities. We proposed the bias factor, a new metric that quantifies the extent to which occurrences of high-frequency identifiers exceed those of low-frequency ones. Unlike classical aggregate metrics such as Average Absolute Error, the bias factor error directly captures the information that matters for debiasing. Our evaluation identified BitMatcher as the most promising candidate for integration into peer sampling protocols, due to its adaptive bit allocation mechanism and strong performance on unbalanced distributions characteristic of Byzantine attacks.

Integration of Aupe with BitMatcher: decay and merge strategies. We designed BMDecay, a controlled decay procedure for BitMatcher that enables its use in unbounded peer sampling streams. BMDecay periodically reduces counter values via a halving procedure triggered by bucket-type transitions, which are constrained to maintain sufficient fingerprint capacity. We also proposed a merge procedure that combines two BMDecay instances by matching fingerprints and taking the maximum of corresponding counters, exploiting the fact that trusted nodes using identical hash functions maintain structurally compatible layouts. We integrated both mechanisms into Aupe and evaluated the complete system under adversarial scenarios. Our experiments show that without decay, BitMatcher's F1 score degrades by up to 40% after processing streams exceeding 1 million elements, rendering it unsuitable for long-running protocols. In contrast, BMDecay ensures that frequency estimation remains viable over streams reaching 10 million elements. Furthermore, Aupe equipped with BMDecay achieves nearly optimal resilience for Byzantine fractions up to 20% while using 12% of the memory required by exact counting.

7.2 Perspectives

Formal analysis of Aupe. While the experimental evaluation of Aupe provides strong empirical evidence of its effectiveness, a formal analysis of its convergence properties and resilience guarantees is an important next step. In particular, establishing provable bounds on the fraction of adversarial identifiers in the views of correct nodes as a function of the adversarial attack force, the number of trusted

nodes, and the sketch parameters would strengthen the theoretical foundations of our approach. Such an analysis could draw on techniques from the study of random processes on graphs and the theory of streaming algorithms.

Deployment and real-world evaluation. Moving from simulation to deployment on real distributed testbeds or production networks is a necessary step to validate our contributions in practice. For example, we have assumed no churn to simplify the analysis and evaluation of our protocols. In practice, departed node counters would be reset in the tracking component. We expect AUPE to remain effective under moderate churn, as the Set Cleaner adapts to the evolving stream; however, a quantitative analysis under churn is left for future work. Moreover, deploying AUPE with BMDecay on a cryptocurrency testnet would provide valuable insights into the practical trade-offs and operational challenges of resilient peer sampling at scale.

Extension to hub-based topologies. In this thesis, we have mainly focused on peer sampling protocols that target uniform random views. Although hub-based topologies, such as those produced by Elevator and LIFT, are not widely adopted, they are promising alternatives for applications such as decentralized federated learning. Investigation of the applicability of frequency-based debiasing techniques to the hub election process in such topologies remains as future work.

Acknowledgment

The research leading to these results has received funding from the French National Research Agency (ANR) under grant ANR-21-CE25-0021-03.

Appendix A

Résumé étendu en français

Les protocoles d'échantillonnage de pairs par gossip sont largement utilisés dans les systèmes distribués à grande échelle pour fournir à chaque nœud un sous-ensemble de voisins sélectionnés de manière uniformément aléatoire dans la population globale du réseau. Ce service est fondamental car il conditionne directement la qualité de la construction d'overlays, de la dissémination d'information et du consensus distribué dans des systèmes tels que les blockchains, les plateformes d'apprentissage fédéré et les réseaux pair-à-pair. Dans ces protocoles, les nœuds échangent périodiquement des portions de leurs listes de voisins, appelées *vues*, avec des pairs choisis aléatoirement. Ce mécanisme simple produit des vues qui convergent rapidement vers une connaissance statistiquement représentative du réseau.

Cependant, dans les environnements ouverts, ces protocoles sont vulnérables aux fautes byzantines. Des nœuds malveillants peuvent biaiser la dissémination des identifiants en surreprésentant leurs propres identifiants dans les échanges de gossip. Ce biais empoisonne progressivement les vues des nœuds corrects, ouvrant la voie à des attaques par éclipse et à la manipulation de protocoles de niveau supérieur. Des attaques concrètes contre Bitcoin ont démontré la faisabilité et l'impact de ces stratégies adversariales.

Deux familles de défenses ont été proposées dans la littérature. Les approches par détection, telles que Secure Peer Sampling et SecureCyclon, cherchent à identifier et exclure les nœuds malveillants en utilisant des techniques cryptographiques. Les approches par tolérance, telles que Brahms, Raptee et Basalt, visent à atténuer l'influence des nœuds byzantins sans les identifier explicitement, en produisant des flux d'identifiants plus uniformes. Ces dernières représentent l'état de l'art, mais elles restent vulnérables à des attaques byzantines mobilisant une grande fraction des nœuds du système (ex. : 20%). Une condition nécessaire pour produire des flux d'identifiants uniformes dans un contexte adversarial est d'avoir une connaissance globale et exacte des nœuds du système et de leur fréquence d'apparition. Cepen-

dant, en raison du caractère dynamique et du nombre potentiellement élevé de nœuds dans les systèmes à grande échelle, le maintien d'un compteur de fréquence exact pour chaque identifiant distinct rencontré par un nœud nécessite une mémoire qui augmente de manière linéaire avec le nombre d'identifiants dans le système. Cette tension entre l'efficacité du débiaisage et les contraintes de ressources constitue le fil conducteur de cette thèse.

Les contributions présentées dans cette thèse peuvent être classées en trois propositions complémentaires : (1) un protocole de peer sampling tolérant aux fautes byzantines par débiaisage collaboratif, (2) une évaluation systématique de structures de données compactes pour l'estimation de fréquences en contexte adversarial, et (3) l'intégration de ces structures dans le protocole proposé via des stratégies de décroissance et de fusion adaptées aux flux non bornés. Les contributions sont détaillées ci-dessous.

AUPE : Protocole collaboratif d'échantillonnage de pairs tolérant aux fautes byzantines

La première contribution de cette thèse est AUPE, un nouveau protocole d'échantillonnage de pairs tolérant aux fautes byzantines. AUPE équipe chaque nœud d'un composant appelé *Set Cleaner* qui combine deux mécanismes : un composant de suivi qui enregistre la fréquence d'apparition de chaque identifiant reçu dans le flux de gossip, et un composant de débiaisage qui transforme le flux biaisé en un flux plus proche d'une distribution uniforme. Le principe est que les identifiants apparaissant plus fréquemment que ce qui est attendu sous une distribution uniforme sont sous-pondérés lors de la production d'échantillons locaux.

L'originalité d'AUPE réside dans son mécanisme de débiaisage collaboratif. Un sous-ensemble de nœuds, exécutés dans des environnements d'exécution sécurisés (TEE) tels qu'Intel SGX, échange périodiquement et fusionne leurs informations de suivi de fréquences. Chaque nœud de confiance observe une portion différente du trafic de gossip ; la fusion de ces informations fournit une connaissance plus complète et plus précise de la distribution globale des identifiants que n'en obtiendrait un nœud isolé. Les nœuds de confiance utilisent ensuite cette information enrichie pour produire des vues moins biaisées, qui bénéficient à l'ensemble du système via le processus de gossip.

La méthodologie d'évaluation a consisté à simuler des réseaux de 10 000 nœuds dans divers scénarios d'attaque. Le modèle adversarial considéré est celui des fautes byzantines : une fraction f des nœuds du réseau peut dévier arbitrairement du protocole. Les nœuds byzantins exploitent le mécanisme de gossip pour inonder les nœuds corrects de leurs propres identifiants, afin de maximiser leur représen-

tation dans les vues des nœuds corrects. Nous avons évalué AUPE face à des attaques où les nœuds adversariaux promeuvent équitablement leurs identifiants. La fraction de nœuds adversariaux varie entre 8% et 50% pour caractériser le seuil de résilience du protocole.

Les résultats montrent qu'AUPE maintient une distribution d'échantillonnage quasi-uniforme même lorsque 26% des nœuds sont adversariaux. La proportion moyenne d'identifiants byzantins dans les vues des nœuds corrects converge vers la proportion réelle de nœuds malveillants dans le système. Cela signifie que l'adversaire ne tire aucun avantage de son attaque. Ce résultat surpasse substantiellement les protocoles de l'état de l'art, tels que Brahms et Basalt, dans des scénarios d'attaque équivalents. De plus, les résultats montrent que l'effet de la collaboration entre nœuds de confiance est significatif : même un petit nombre de nœuds de confiance améliore considérablement la qualité du débiaisage pour l'ensemble du réseau.

Estimation robuste des fréquences pour l'échantillonnage de pairs en contexte adversarial

Bien qu'AUPE démontre l'efficacité du débiaisage par suivi de fréquences, sa dépendance au comptage exact par identifiant pose un problème de passage à l'échelle. En effet, la mémoire nécessaire à ce suivi croît linéairement avec le nombre d'identifiants distincts dans le système. La deuxième contribution de cette thèse aborde cette limitation par une étude systématique des techniques d'estimation de fréquences à partir de sketches dans le contexte spécifique de l'échantillonnage de pairs en contexte adversarial.

Nous avons évalué six stratégies de sketching : le Count-Min Sketch, le Count-Mean-Min Sketch, le Lossy Conservative Update, le Cold Filter, le BitMatcher et un sketch probabiliste. Ces stratégies ont été évaluées sur des flux synthétiques modélisant des scénarios d'injection adversariale à différentes intensités d'attaque. Un apport méthodologique important est la proposition du *facteur de biais*, une nouvelle mesure qui quantifie dans quelle mesure les occurrences d'identifiants à haute fréquence (souvent adversariaux) dépassent celles des identifiants à basse fréquence (généralement corrects). Contrairement aux métriques classiques telles que l'erreur absolue moyenne, l'erreur sur le facteur de biais capture directement l'information pertinente pour le débiaisage, à savoir la capacité du sketch à distinguer les identifiants adversariaux des identifiants corrects avec une précision suffisante. Les résultats de cette étude ont révélé que BitMatcher constitue le candidat le plus prometteur pour l'intégration dans les protocoles d'échantillonnage de pairs, en raison de son mécanisme d'allocation de bits adaptatif et de ses per-

formances sur des distributions biaisées, caractéristiques des attaques byzantines.

Stratégies de vieillissement et de fusion de sketch en contexte adversarial

Pour finir, nous allons plus loin dans les capacités de passage à l'échelle d'AUPE en intégrant le sketch BitMatcher au protocole. Cette intégration soulève deux défis propres au contexte de l'échantillonnage de pairs.

Le premier défi concerne le caractère non borné du flux. Un protocole d'échantillonnage de pairs fonctionne indéfiniment, traitant un flux croissant d'identifiants. Un sketch de taille fixe finit par saturer ; ses compteurs atteignent leur valeur maximale et il ne peut plus distinguer les identifiants fréquents des autres. Pour résoudre ce problème, nous proposons BMDecay, une procédure de vieillissement pour BitMatcher. BMDecay réduit périodiquement les valeurs des compteurs par une procédure de division par deux déclenchée par des transitions de type de bucket, elles-mêmes contraintes à maintenir une capacité d'empreintes suffisante.

Le deuxième défi concerne la fusionnabilité. BitMatcher est un sketch non linéaire couplé à des empreintes, c'est-à-dire que chaque position de compteur stocke à la fois un compteur de fréquence et une empreinte compacte de l'identifiant associé. Deux instances de BitMatcher construites indépendamment contiennent généralement des empreintes différentes aux positions correspondantes, ce qui rend les opérations de fusion élément par élément naïves sémantiquement invalides. Nous proposons une procédure de fusion qui combine deux instances de BMDecay en appariant les empreintes et en prenant le maximum des compteurs correspondants, en exploitant le fait que les nœuds de confiance utilisant les mêmes fonctions de hachage maintiennent des structures compatibles.

La méthodologie d'évaluation a consisté à valider nos propositions en deux temps. Dans un premier temps, nous avons évalué les stratégies de vieillissement et de fusion de manière isolée, en mesurant leur impact sur la précision d'estimation des fréquences au fil du temps. Nos résultats montrent que sans stratégie de vieillissement du sketch BitMatcher, sa précision se dégrade de 40% en F1 F1-score après le traitement de flux dépassant 1 million d'éléments. Avec BMDecay, l'estimation des fréquences reste viable pour des flux allant jusqu'à 10 millions d'éléments.

Dans un second temps, nous avons intégré BMDecay dans AUPE et évalué le système complet dans divers scénarios d'attaque. Les résultats montrent qu'AUPE équipé de BMDecay atteint une résilience quasi-optimale pour des fractions byzantines allant jusqu'à 20%, tout en utilisant seulement 12% de la mémoire requise par le comptage exact. Ce résultat démontre qu'il est possible de combiner le débi-

aisage au niveau du protocole et l'estimation efficace des fréquences en mémoire en une solution pratique et scalable pour l'échantillonnage de pairs résilients.

Perspectives

Ce travail de thèse ouvre plusieurs perspectives de recherche à court et à long terme. En premier lieu, une analyse formelle des propriétés de convergence d'AUPE et de ses garanties de résilience en fonction du nombre de nœuds byzantins et de confiance constituerait un apport théorique important. En particulier, établir des bornes prouvables sur la proportion d'identifiants adversariaux dans les vues des nœuds corrects, en fonction des paramètres du sketch, renforcerait les fondements théoriques de notre approche. Une telle analyse pourrait s'appuyer sur les techniques issues de l'étude des processus aléatoires sur les graphes et de la théorie des algorithmes de streaming.

Passer de la simulation au déploiement sur des réseaux distribués réels de test ou sur des réseaux de production est une étape nécessaire pour valider nos contributions en pratique. Des facteurs tels que la latence du réseau, des capacités de traitement hétérogènes et une dynamique réaliste des nœuds qui rejoignent et quittent le système peuvent révéler des comportements que les simulations ne permettent pas de saisir. Le déploiement d'AUPE avec BMDecay sur un réseau de test de cryptomonnaie fournirait des informations précieuses sur les compromis pratiques et les défis opérationnels liés à l'échantillonnage résilient à grande échelle.

Enfin, dans cette thèse, nous nous sommes principalement concentrés sur les protocoles d'échantillonnage par les pairs visant à obtenir des vues aléatoires uniformes. Bien que les topologies basées sur des hubs, telles que celles produites par Elevator et LIFT, ne soient pas largement adoptées, elles constituent des alternatives prometteuses pour des applications telles que l'apprentissage fédéré décentralisé. L'étude de l'applicabilité des techniques de débiaisement basées sur la fréquence au processus d'élection des hubs dans de telles topologies reste à accomplir.

References

[ABF+23] Alex Auvolat, Yérom-David Bromberg, Davide Frey, Djob Mvondo, and François Taïani. Basalt: A rock-solid byzantine-tolerant peer sampling for very large decentralized networks. In *Proceedings of the 24th International Middleware Conference*, Middleware '23, page 111–123, New York, NY, USA, 2023. Association for Computing Machinery.

[ABG13] Emmanuelle Anceaume, Yann Busnel, and Sébastien Gambs. *On the Power of the Adversary to Solve the Node Sampling Problem*, pages 102–126. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[ABS13] Emmanuelle Anceaume, Yann Busnel, and Bruno Sericola. Uniform node sampling service robust against collusions of malicious nodes. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, 2013.

[ACH+13] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Trans. Database Syst.*, 38(4), December 2013.

[Auv20] Alex Auvolat. Basalt rps simulator. GitHub, 2020.

[AV23a] Alexandros Antonov and Spyros Voulgaris. Securecyclon: Dependable peer sampling. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 1–12, Los Alamitos, CA, USA, jul 2023. IEEE Computer Society.

[AV23b] Alexandros Antonov and Spyros Voulgaris. Securecyclon: Dependable peer sampling. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 1–12. IEEE, 2023.

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[BCFM98] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336, 1998.

[BCFM00] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-Wise Independent Permutations. *Journal of Computer and System Sciences*, 60(3):630–659, June 2000.

[BEMV20] Ran Ben Basat, Gil Einziger, Michael Mitzenmacher, and Shay Vargaftik. Faster and more accurate measurement through additive-error counters. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 1251–1260, 2020.

[BEMV21a] Ran Ben Basat, Gil Einziger, Michael Mitzenmacher, and Shay Vargaftik. SALSA: Self-Adjusting Lean Streaming Analytics . In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 864–875, Los Alamitos, CA, USA, April 2021. IEEE Computer Society.

[BGK+08] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine resilient random membership sampling. In *Proceedings of the Twenty-Seventh ACM Symposium on Principles of Distributed Computing*, PODC '08, page 145–154, New York, NY, USA, 2008. Association for Computing Machinery.

[bit25a] bitcoincore. Bitcoin Core 0.11 (ch 4): P2P Network. https://en.bitcoin.it/wiki/Bitcoin_Core_0.11_(ch_4):_P2P_Network, 2025. Accessed: 2025-07-17.

[Bit25c] Bitnode.io. snapshot of reachable nodes. https://bitnodes.io/nodes/, 2025. Accessed: 2025-07-07.

[BKK+03] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. *Commun. ACM*, 46(2):43–48, February 2003.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[CCFC02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, page 693–703, Berlin, Heidelberg, 2002. Springer-Verlag.

[CM05] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[CM14] Ling Chen and Qingling Mei. Mining frequent items in data stream using time fading model. *Inf. Sci.*, 257:54–69, February 2014.

[CSSX09] Graham Cormode, Vladislav Shkapenyuk, Divesh Srivastava, and Bojian Xu. Forward decay: A practical time decay model for streaming systems. In *2009 IEEE 25th International Conference on Data Engineering*, pages 138–149, 2009.

[DGIM02] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[Dou02] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[DR06] Fan Deng and Davood Rafiei. Approximately detecting duplicates for streaming data using stable bloom filters. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, page 25–36, New York, NY, USA, 2006. Association for Computing Machinery.

[DR07] Fan Deng and Davood Rafiei. New estimation algorithms for streaming data: Count-min can do more. *Webdocs. Cs. Ualberta. Ca*, 2007.

[DSHK08] Xenofontas Dimitropoulos, Marc Stoecklin, Paul Hurley, and Andreas Kind. The eternal sunshine of the sketch data structure. *Computer Networks*, 52(17):3248–3257, 2008.

[EFM17] Gil Einziger, Roy Friedman, and Ben Manes. TinyLFU: A highly efficient cache admission policy. *ACM Trans. Storage*, 13(4):35:1–35:31, 2017.

[EV03] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, August 2003.

[FAKM14] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, page

75–88, New York, NY, USA, 2014. Association for Computing Machinery.

[GDC12] Amit Goyal, Hal Daumé, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, page 1093–1103, USA, 2012. Association for Computational Linguistics.

[GDI11] Amit Goyal and Hal Daumé III. Approximate scalable bounded space sketch for large data NLP. In Regina Barzilay and Mark Johnson, editors, *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 250–261, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.

[GI11] Amit Goyal and Hal Daumé III. Lossy conservative update (lcu) sketch: succinct approximate count storage. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, page 878–883. AAAI Press, 2011.

[GYZ+18] Junzhi Gong, Tong Yang, Haowei Zhang, Hao Li, Steve Uhlig, Shigang Chen, Lorna Uden, and Xiaoming Li. HeavyKeeper: An accurate algorithm for finding top-k elephant flows. In *Proc. USENIX ATC*, pages 909–921, 2018.

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, page 129–144, USA, 2015. USENIX Association.

[JGKvS04] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In Hans-Arno Jacobsen, editor, *Middleware 2004*, pages 79–98, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[JMB04] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In Giovanna Di Marzo Serugendo, Anthony Karageorgos, Omer F. Rana, and Franco Zambonelli, editors, *Engineering Self-Organising*

Systems, pages 265–282, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[JMB05a] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, aug 2005.

[JMv10a] Gian Paolo Jesi, Alberto Montresor, and Maarten van Steen. Secure peer sampling. *Computer Networks*, 54(12):2086–2098, 2010. P2P Technologies for Emerging Wide-Area Collaborative Services and Applications.

[JMv10b] Gian Paolo Jesi, Alberto Montresor, and Maarten van Steen. Secure peer sampling. *Computer Networks*, 54(12):2086–2098, 2010. P2P Technologies for Emerging Wide-Area Collaborative Services and Applications.

[JVG+07a] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8–es, August 2007.

[JVG+07b] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8–es, aug 2007.

[KDG03] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 482–491, 2003.

[KMG03] A.-M. Kermarrec, L. Massoulie, and A.J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258, 2003.

[KS04] Valerie King and Jared Saia. Choosing a random peer. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '04, page 125–130, New York, NY, USA, 2004. Association for Computing Machinery.

[KSW21] Jing Huey Khor, Michail Sidorov, and Peh Yee Woon. Public blockchains for resource-constrained iot devices—a state-of-the-art survey. *IEEE Internet of Things Journal*, 8(15):11960–11982, 2021.

[LPBTF24] Mohamed Amine Legheraba, Maria Potop-Butucaru, Sebastien Tixeuil, and Serge Fdida. Emergent Peer-to-Peer Multi-Hub Topology . In *2024 22nd International Symposium on Network Computing and Applications (NCA)*, pages 219–226, Los Alamitos, CA, USA, October 2024. IEEE Computer Society.

[LRPBT25] Mohamed Amine Legheraba, Nour Rachdi, Maria Potop-Butucaru, and Sébastien Tixeuil. Lift: Byzantine resilient hub-sampling. In *2025 Thirteenth International Symposium on Computing and Networking (CANDAR)*, pages 1–9, 2025.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[LX21] Yongqiang Liu and Xike Xie. Xy-sketch: on sketching data streams at web scale. In *Proceedings of the Web Conference 2021*, WWW '21, page 1169–1180, New York, NY, USA, 2021. Association for Computing Machinery.

[LX23] Yongqiang Liu and Xike Xie. A probabilistic sketch for summarizing cold items of data streams. *IEEE/ACM Trans. Netw.*, 32(2):1287–1302, October 2023.

[Mer78] Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, April 1978.

[MIF02] R. Matei, A. Iamnitchi, and P. Foster. Mapping the gnutella network. *IEEE Internet Computing*, 6(1):50–57, 2002.

[MLP+15] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Discovering Bitcoin's Public Topology and Influential Nodes. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS'15)*, pages 123–136, Denver, CO, USA, 2015. ACM.

[MM02] Gurmeet Singh Manku and Rajeev Motwani. Chapter 31 - approximate frequency counts over data streams. In Philip A. Bernstein, Yannis E. Ioannidis, Raghu Ramakrishnan, and Dimitris Papadias, editors, *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, pages 346–357. Morgan Kaufmann, San Francisco, 2002.

[Mon04] Alberto Montresor. A robust protocol for building superpeer overlay topologies. In *Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, P2P '04, page 202–209, USA, 2004. IEEE Computer Society.

[MSA12] Sergiy Matusevych, Alexander J. Smola, and Amr Ahmed. Hokusai — sketching streams in real time. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, UAI'12, page 594–603, Arlington, Virginia, USA, 2012. AUAI Press.

[PBQY+22] Matthieu Pigaglio, Joachim Bruneau-Queyreix, Bromberg Yerom, Davide Frey, Etienne Riviere, and L. Réveillère. Raptee: Leveraging trusted execution environments for byzantine-tolerant peer sampling services. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pages 603–613, Los Alamitos, CA, USA, jul 2022. IEEE Computer Society.

[PGD12] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *Proc. VLDB Endow.*, 5(10):992–1003, June 2012.

[Pit87] Boris Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, 1987.

[PS19] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Comput. Surv.*, 51(6), jan 2019.

[RD07] Florin Rusu and Alin Dobra. Statistical analysis of sketch estimators. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, page 187–198, New York, NY, USA, 2007. Association for Computing Machinery.

[RKA16] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 1449–1463, New York, NY, USA, 2016. Association for Computing Machinery.

[SGG03] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multim. Syst.*, 9(2):170–184, 2003.

[SJL+24] Qilong Shi, Chengjun Jia, Wenjun Li, Zaoxing Liu, Tong Yang, Jianan Ji, Gaogang Xie, Weizhe Zhang, and Minlan Yu. Bitmatcher:

Bit-level counter adjustment for sketches. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 4815–4827, 2024.

[SMLN+03] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[SNDW06] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–12, 2006.

[TJ09] Norbert Tölgyesi and Márk Jelasity. Adaptive peer sampling with Newscast. In *European Conference on Parallel Processing*, EuroPar, pages 523–534. Springer, 2009.

[VGVS05a] Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and systems Management*, 13(2):197–217, 2005.

[VGvS05b] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13:197–217, 2005.

[Wen24] WenjunLi. Bitmatcher implementation. https://www.wenjunli.com/BitMatcher, 2024. Accessed: 2025-05-07.

[YJL+18a] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 561–575, New York, NY, USA, 2018. Association for Computing Machinery.

[YJL+18b] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 561–575, New York, NY, USA, 2018. Association for Computing Machinery.

[YJZ+19b] Tong Yang, Jie Jiang, Yang Zhou, Long He, Jinyang Li, Bin Cui, Steve Uhlig, and Xiaoming Li. Fast and accurate stream processing by filtering the cold. *The VLDB Journal*, 28(5):735–763, October 2019.

[YZJ+17] Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. Pyramid sketch: a sketch framework for frequency estimation of data streams. *Proc. VLDB Endow.*, 10(11):1442–1453, August 2017.

[ZCQ+23] Zijie Zeng, Lin Cui, Mimi Qian, Zhen Zhang, and Kaimin Wei. A survey on sliding window sketch for network measurement. *Computer Networks*, 226:109696, 2023.

[ZLT+21] Bohan Zhao, Xiang Li, Boyu Tian, Zhiyu Mei, and Wenfei Wu. Dhs: Adaptive memory layout organization of sketch slots for fast and accurate data stream processing. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 2285–2293, New York, NY, USA, 2021. Association for Computing Machinery.

[ZYL+21] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, and Nicholas Zhang. LightGuardian: A full-visibility, lightweight, in-band telemetry system using sketchlets. In *Proc. USENIX NSDI*, pages 991–1010, 2021.

Publications and Awards

Articles in Peer-reviewed Conferences

[MBQR24] Augusta Mukam, Joachim Bruneau-Queyreix, and Laurent Réveillère. Aupe: Collaborative byzantine fault-tolerant peer-sampling. In *2024 22nd International Symposium on Network Computing and Applications (NCA)*, pages 17–24, 2024.

Awards

[bpa24] The best paper award. Aupe: Collaborative byzantine fault-tolerant peer-sampling. In *2024 22nd International Symposium on Network Computing and Applications (NCA 2024)*, 2024.