V-Guard: An Efficient Client-Centric Distributed Ledger Safeguarding Critical Vehicular Data

Gengrui Zhang
Concordia University
gengrui.zhang@concordia.ca

Yunhao Mao
University of Toronto
yunhao.mao@mail.utoronto.ca

Shiquan Zhang
University of Toronto
shiquan.zhang@mail.utoronto.ca

Shashank Motepalli
University of Toronto
shashank.motepalli@mail.utoronto.ca

Yuqiu Zhang
University of Toronto
quincy.zhang@mail.utoronto.ca

Hans-Arno Jacobsen
University of Toronto
jacobsen@eecg.toronto.edu

ABSTRACT

As vehicles are increasingly equipped with vehicular automation technologies, automobile manufacturers have become centralized data monopolies, raising concerns about the transparency, integrity, and accountability of data management. This paper introduces V-Guard, a new blockchain solution designed to safeguard critical vehicle data. V-Guard introduces an innovative high-performance consensus algorithm that operates under dynamically changing networks and members of vehicles. V-Guard integrates membership agreement into the transaction consensus process, ensuring that consensus can be seamlessly achieved with shifting memberships. In addition, V-Guard separates the ordering of transactions from consensus, allowing concurrent ordering with high throughput. The evaluation results show that V-Guard's peak throughput is 22× higher than HotStuff [128], 9.5× higher than ResilientDB [66], and 1.8× higher than Narwhal [37]. Notably, under dynamic memberships, V-Guard outperforms state-of-the-art membership management mechanisms, achieving 8.7× and 3.7× higher throughput than MC [107] and Dyno [46], respectively.

PVLDB Reference Format:
Gengrui Zhang, Yunhao Mao, Shiquan Zhang, Shashank Motepalli, Yuqiu Zhang, and Hans-Arno Jacobsen. V-Guard: An Efficient Client-Centric Distributed Ledger Safeguarding Critical Vehicular Data. PVLDB, 14(1): XXX-XXX, 2026.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/vguardbc/vguardbft.

1 INTRODUCTION

The rapid advancement of vehicular automation technologies (VATs) has brought significant transformation to the automotive industry. However, it has also introduced substantial challenges in the realms of data management, as automobile manufacturers consistently amass vast amounts of vehicle data but grant limited access to users. This monopolization of data management has transformed manufacturers into centralized data monopolies, which present inherent risks to data transparency, integrity, and accountability.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

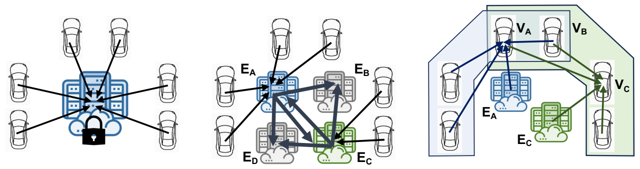
1.1 Motivation for V2X blockchains

Although VATs have seen fast developments in recent years, VATs are still not mature enough with rising reports of software malfunctions and investigations (e.g., Tesla crashes [5, 22, 101] and Waymo incidents [6, 11], driverless cars in San Francisco [113, 120]). Despite the immaturity of VATs, the manufacturers manage the vehicle in a centralized manner (shown in Figure 1a) and limit users' access to their own data [88]. The opaque data management of immature VATs has topped consumers' concerns in recent surveys [2, 70] and imposed raising challenges in legal systems in terms of liability and responsibility [12, 98, 99].

Data integrity and accountability. Recent leaks of customer complaints regarding Tesla's Full Self-Driving [122] show that there were more than 2400 self-acceleration issues and 1500 braking problems. Furthermore, in the event of incidents, users face significant challenges in accessing their own vehicle data and establishing its integrity [20, 51]. Unfortunately, these disputes have witnessed a notable surge in recent years, leaving customers skeptical about the integrity of the data at hand [50]. Users are left in the dark, unsure whether the provided data has been tampered [41, 61], especially when the manufacturer claims that certain data was either not recorded or failed to be recorded due to technical issues—a situation that can be perceived as *evidence burying*.

Data ownership and regulation compliance. The preservation of consumers' right to data ownership has become the focus of numerous regulations in recent years. For example, the European Union's General Data Protection Regulation (GDPR) and the California Privacy Rights Act (CPRA) share similar principles for managing data integrity and ownership [104], with a raising voice demanding the enforcement of these principles [54, 123].

Decentralized solutions. Blockchain systems, eliminating the reliance on a single source of truth, securely replicate data entries on an immutable ledger, effectively preventing the fabrication or concealment of evidence. This decentralized and transparent data management benefits both manufacturers and customers. With the data integrity guarantee, legal responsibility between the human drivers and VATs can be clearly delineated in the event of incidents [36, 106]. Furthermore, granting users unrestricted access to their own data provides them with greater flexibility in managing their driving-related expenses. For instance, both drivers and



(a) Centralized DM **(b) Semi-Decentral DM** **(c) Decentral DM (VG)**

Figure 1: Centralized, semi- and fully- decentralized models for VAT data management (DM). V-Guard (VG) enables a vehicular blockchain that achieves full decentralization.

insurers can benefit from the development of more precise pricing models [60, 114], enabling drivers to secure lower insurance quotes [85].

1.2 Challenges in V2X blockchains

While blockchain solutions have garnered significant attention in both academic and industrial circles [56, 129], technical challenges persist in blockchain applications for vehicular scenarios [38]. These unresolved challenges are impeding the practical deployment of vehicular blockchains.

Why existing blockchain solutions do not work? Existing permissioned blockchains often serve the purpose of *enterprise-centric* coalitions, enabling semi-decentralization where a few enterprises operating as consensus nodes (e.g., HyperLedger Fabric [8], CCF [108], and Diem [40]). Prominent automobile manufacturers, such as BMW [21], Mercedes [17, 53], Toyota [77], and Ford [14, 55] have embarked on blockchain initiatives to improve supply chain management, ridesharing, and carsharing applications [30, 32, 72]. When applying this traditional model to vehicle blockchains, manufacturers take the role of consensus nodes, and vehicles operate as clients (illustrated in Figure 1b).

However, existing blockchains are not a suitable vehicular solution. First, consensus nodes experience a large number of vehicular clients (e.g., millions of vehicle connections) requesting particular lock-heavy expensive consensus operations [9], which will result in poor performance, especially in leader-based blockchains [8, 40]. Second, scaling the traditional model requires additional enterprise participation (or backups), which will increase operational costs and further degrade performance, making it less practical for widespread adoption. Third, the traditional model primarily builds trust among enterprises rather than vehicles, leaving it vulnerable to protect data ownership and transparency. These limitations highlight the need for new vehicular-purpose blockchains.

Features and challenges of vehicular blockchains. Vehicular blockchains present unique features and challenges.

Dynamic connectivity. Vehicles can communicate with each other in Vehicle-to-Everything (V2X) networks [31], where vehicles communicate via wireless communication technologies [1], including Dedicated Short-Range Communication (DSRC) and Cellular Vehicle-to-Everything (C-V2X). However, V2X communication is often not reliable [7, 58, 103]. For example, in DSRC V2X networks, the vehicles in a running vehicle's vicinity can change dynamically on the road, leading to intermittent connections depending on the distance. Similarly, in C-V2X networks, drivers may power off their vehicles unpredictably, causing unavailable connections.

Independent transactions. Data generated by different vehicles is transactionally unrelated. Individual vehicles can establish autonomous blockchains for recording their data. As such, vehicles can participate in multiple blockchains and validate transactions initiated by other vehicles. This feature facilitates parallelism in transaction processing and significantly boosts system performance.

1.3 V-Guard and its contributions

V-Guard is the first vehicle-centric blockchain designed specifically for the automotive sector. It empowers decentralized vehicular data management, ensuring the preservation of data integrity, fostering accountability, and facilitating both evidence collection and regulatory compliance.

In V-Guard, as shown in Figure 1c, vehicles operate as consensus nodes, and each vehicle coordinates its own blockchain (a V-Guard instance). V-Guard proposes a new consensus algorithm that directly targets the problem of vehicles' arbitrary connectivity, enabling consensus to be achieved seamlessly with *dynamic members of vehicles.*

Addressing changing member configurations entails consensus algorithms to undergo reconfigurations. However, state-of-the-art Byzantine fault-tolerant (BFT) algorithms (e.g., PBFT [26] and HotStuff [128]) are primarily designed for *stable environments*, where the system operates under a static set of servers. When members change (e.g., new servers join or old servers leave), they rely on additional reconfiguration approaches (e.g., MC [107] and Lazarus [57]) to handle system reconfiguration. These approaches first suspend ongoing consensus processes and then update the membership profile on each server, which creates a "hard stop" on ongoing transaction consensus, drastically degrading performance [46, 89]. For example, MC [107] reconfigurations can impede transaction consensus for 5-10 seconds; under high system workloads and large scales, performance degradation is further pronounced [3, 18, 107].

In contrast, *V-Guard can seamlessly operate under dynamically changing memberships without disrupting ongoing consensus processes.* V-Guard incorporates membership configuration agreement into transaction consensus and provides a traceable and immutable ledger for both data transactions and their associated membership configurations. As such, the consensus target of V-Guard becomes:

$$\langle \textit{data transactions + membership configurations} \rangle$$

V-Guard employs a membership management unit (MMU) to handle dynamic membership configurations. The MMU keeps track of available vehicles and groups them into a queue of membership profiles, namely *booths.* During the consensus process, a data entry is paired with a designated booth in each phase. When an operating booth becomes unavailable, the MMU will provide a new booth from its queue. Thus, consensus results not only indicate what data is committed but also on what members committed the data.

In addition, *V-Guard introduces a lightweight consensus framework with high parallelism and minimal manufacturer involvement.* It separates the ordering of data entries from consensus, allowing ordering and consensus phases to take place in different booths. The ordering phase is conducted among only vehicles and concurrently appends data entries to a totally ordered log. The consensus phase, conducted by vehicles and manufacturers, periodically commits the ordered entries to an immutable ledger. The separation brings three benefits: ① it reduces message passing, leading to high performance; ② it minimizes the involvement of manufacturers, relieving the burden of consensus operations; and ③ it allows data entries to be ordered and committed in different booths, enabling more granular membership management.

Furthermore, *V-Guard achieves high performance under both static and dynamic memberships.* Under static membership, V-Guard peaks its throughput at 765, 930 TPS (transactions per second) with a latency of 143 ms; its throughput is 22× higher than HotStuff [128], 9.5× higher than ResilientDB [66], and 1.8× higher than Narwhal [37]. Under dynamic membership, V-Guard's membership management shows significant efficiency and peaks its throughput at 615, 328 TPS at 1324 ms; its throughput is 8.2× higher than MC [107] and 3.7× higher than Dyno [46] with the same number of changing members.

V-Guard is a feasible and practical solution for protecting vehicle critical data. **It is worth noting that V-Guard records critical decision data from VAT software not the raw sensor data.** Unlike the massive raw sensor data (GB/s), the results of VAT software only contain a relatively small amount of data (KB/s), including critical driving log information, such as speed, direction, acceleration, and object detection [125]. V-Guard's performance shows that it offers a feasible solution tailored to protecting the integrity, accountability, and ownership of vehicular data management.

2 V-GUARD OVERVIEW

V-Guard provides a blockchain service among vehicles and their manufacturers to avoid a single point of control of data. To apply this service, a vehicle creates a V-Guard instance and starts to operate as a *proposer*. Other vehicles joined in this instance operate as *validators*. The description of the roles is presented below.

Proposer (V_p). Each vehicle is the proposer of its own V-Guard instance, so *each instance has only one proposer*. The proposer operates as a leader in its V-Guard instance: it proposes data to validators and coordinates consensus for proposed data among validators.

Validator. In a V-Guard instance, except the proposer, other members operate as a validator. The automobile manufacturer always operates as a **pivot validator** (denoted by V_π), and the other vehicles operate as **vehicle validators** (denoted by V_i).

A vehicle can participate in multiple V-Guard instances: it operates as the proposer of its own V-Guard instance and as validators in other vehicles' V-Guard instances. We denote the number of instances a vehicle joins as *catering factor*, γ.

Catering factor (γ). The catering factor ($\gamma \in \mathbb{Z}^+$) of a vehicle describes the number of V-Guard instances this vehicle is currently participating in. For example, $\gamma = 2$ for V_A and V_B, and $\gamma = 1$ for V_C in Figure 1c. γ changes correspondingly when the vehicle joins and quits other vehicles' V-Guard instances.

In general, as shown in Figure 2, V-Guard instances consist of three modules: consensus, storage, and gossiping. The consensus module is the most crucial component (§4). The proposer first uses

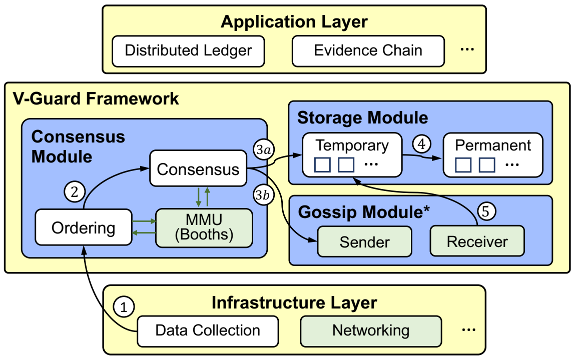


Figure 2: V-Guard architecture. Consensus is conducted by ordering and consensus instances interacting with the MMU.

its MMU to obtain a valid membership profile (§4.1). Then, ordering instances are created to secure a unique ordering ID for the data entries with vehicle validators and append ordered data entries to a totally ordered log (§4.2). Finally, the ordered entries are committed by a periodically scheduled consensus instance with both vehicle and pivot validators, reaching consensus for data entries and their membership profiles (§4.3). In addition, V-Guard uses two supplementary modules to support its vehicle-centric blockchain. The storage module temporarily stores all entries by default according to a predefined policy (e.g., for 24 hours) and provides an option for users to permanently store selected data. The gossip module aims to increase system robustness by further disseminating committed data entries to the network.

3 SYSTEM MODEL AND SERVICE PROPERTIES

V-Guard is a permissioned blockchain, which requires identity information from members; i.e., each vehicle can identify who they are communicating with. Since vehicles are manufactured with vehicle identification numbers (VIN) [4], where no two vehicles in operation have the same VIN, VINs can serve as vehicles' fingerprint, which enables the use of permissioned blockchains in practice.

Service properties. V-Guard's blockchain architecture provides evidence-based assurance that vehicles behave as intended. Each vehicle constructs its own blockchain, denoted as a V-Guard instance. Data entries are ordered and committed by quorums of independent vehicles with their signatures. After consensus is reached, data entries are traceable and immutable. Furthermore, neither party can bury evidence by claiming data loss because each committed data entry is consistently stored in vehicles in the consensus process.

Network assumptions. V-Guard adopts the partial synchrony network model introduced by Dwork et al. [47], where there is a known bound Δ and an unknown Global Stabilization Time (GST), such that after GST, all transmissions between two non-faulty servers arrive within time Δ. V-Guard does not require network synchrony to provide safety, but it requires partial synchrony to provide liveness.

Trust model. V-Guard assumes a Byzantine failure model where faulty members may behave arbitrarily, subject only to independent member failures. Each member, including vehicles and manufacturers, is an independent entity; that is, when f vehicles, regardless of their manufacturers, are faulty, they are counted as f failures. Note that a vehicle's behavior may not be universal when participating in multiple V-Guard instances ($\gamma > 1$); i.e., it could operate correctly in one instance and falsely in another one.

Note on pivot validator behavior. Vehicular blockchain systems uniquely monitor and record logs from vehicle manufacturers (pivot validator), who also participate directly in consensus. This dual role requires revisiting the traditional definition of Byzantine fault tolerance (BFT). We emphasize that BFT applies strictly to the consensus process—not to faults in the physical layer or hardware.

For instance, if a manufacturer installs a defective acceleration system, a vehicle might report a speed of 50 km/h while actually traveling at 70 km/h. While V-Guard can faithfully record this (potentially inaccurate) metered data, it cannot detect or correct faults originating from the physical world. Addressing such issues lies beyond the scope of blockchain protocols.

Within the consensus process, however, manufacturers are treated no differently than other participants. V-Guard tolerates arbitrary behavior from manufacturers—including equivocation, crashes, and other Byzantine faults—without compromising consensus integrity.

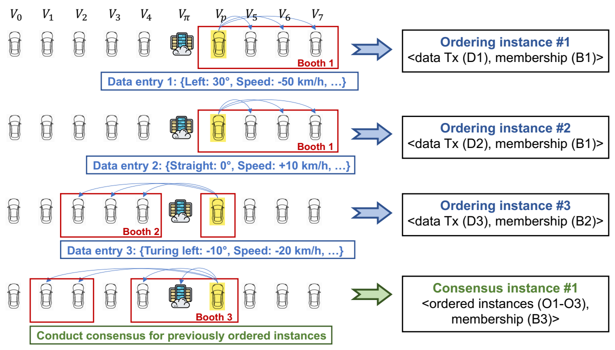
V-Guard guarantees the correctness of the consensus process, ensuring that outputs faithfully reflect the provided inputs. Even when the inputs are flawed, V-Guard still replicates them across vehicles and preserves them as verifiable evidence. By enforcing agreement between the proposer and the pivot validator, V-Guard ensures that neither party can fabricate or suppress data, thus maintaining integrity and transparency in vehicular data management.

Cryptographic primitives. V-Guard applies (t, n)-threshold signatures where t out of n members can collectively sign a message [84, 112]. Threshold signatures can convert t partially signed messages (of size $O(n)$) into one fully signed message (of size $O(1)$). The fully signed message can then be verified by all n members, proving that t members have signed it. The use of threshold signatures to obtain linearity has been common in state-of-the-art linear BFT protocols, such as HotStuff [128], SBFT [64], and Prosecutor [130].

Quorums in V-Guard. Similar to other BFT protocols, V-Guard requires a minimum quorum size of $2f + 1$ to tolerate f Byzantine failures in a total of $3f + 1$ members. Since V-Guard separates ordering from consensus, its ordering and consensus phase may operate in different quorums. Specifically, the ordering phase does not require the participation of a pivot validator; i.e., ordering can be achieved only by the proposer and vehicle validators. The consensus phase includes pivot validators in its quorums with vehicle validators. This feature ensures correctness as every data entry ordered by the ordering phase will still be examined by the consensus phase while minimizing the involvement of pivot validators (manufacturers). Thus, under membership changes, an ordering phase and a consensus phase can have up to $n - 1$ and $n - 2$ new members, respectively. V-Guard allows proposers and pivot validators to co-witness the consensus process while giving the maximum flexibility to membership changes.

4 THE V-GUARD CONSENSUS PROTOCOL

This section presents the V-Guard consensus system consisting of *event-driven* and *daemon* services (shown in Figure 4). Event-driven services are invoked when data is collected. Their orchestration achieves consensus for the collected data under their residing memberships. Daemon services handle network connections by dynamically connecting to and disconnecting from vehicles according to their availability.



V_0 V_1 V_2 V_3 V_4 V_π V_p V_5 V_6 V_7
Booth 1
Data entry 1: {Left: 30°, Speed: -50 km/h, ...}
Ordering instance #1
<data Tx (D1), membership (B1)>
Booth 1
Data entry 2: {Straight: 0°, Speed: +10 km/h, ...}
Ordering instance #2
<data Tx (D2), membership (B1)>
Booth 2
Data entry 3: {Turing left: -10°, Speed: -20 km/h, ...}
Ordering instance #3
<data Tx (D3), membership (B2)>
Booth 3
Conduct consensus for previously ordered instances
Consensus instance #1
<ordered instances (O1-O3), membership (B3)>

Figure 3: An example of MMU-embedded ordering and consensus procedures. When the current booth is no longer available, the proposer (V_p) retrieves a new booth from the MMU. Membership profiles are paired with data transactions in ordering and consensus instances.

4.1 Membership management unit (MMU)

The MMU keeps track of available vehicle connections in the network and prepares a queue of valid *booths*. A booth, denoted by $\mathcal{V}$, is a *membership configuration profile* that describes a set of participating members (vehicles). The profile contains a list of vehicle information on their network addresses (i.e., `v.conn`) and public keys (i.e., `v.pub`).

```
Booth:
    v []struct{ //v is a vehicle profile
        conn net.Conn //connection information
        pub  share.PubPoly //individual public key
        ...
    }
   thpub share.PubPoly //threshold signature public key
```

The size of each booth is a predefined parameter that can be defined by users. Since V-Guard assumes BFT failures, the minimum size of a booth is four ($3f + 1$) to tolerate $f = 1$ Byzantine failure. Our evaluation (see §5) shows that V-Guard supports large booth sizes with high performance, empowering vehicles to incorporate a greater number of validators in practice to tolerate more failures.

The MMU interface provides available booths for ordering and consensus instances (denoted by $\mathcal{V}_o$ and $\mathcal{V}_c$, respectively). The ordering booth consists of the proposer and vehicle validators, and the consensus booth consists of the proposer, vehicle validators, and the pivot validator. If there is no booth available in the queue, the ordering and consensus instances simply wait for new booths from the interface. In addition, the MMU dynamically manages booths based on their availability and efficiency. It removes a booth when f vehicles are unavailable and periodically pings the other vehicles, prioritizing the booth with the lowest network latency.

In each booth, the proposer first establishes the distributed key generation (DKG) of a (t, n)-threshold signature, where t is the quorum size, and n is the booth size. Once the DKG is completed, all vehicles within the same booth share the same public key for threshold signatures (`thpub`). It is important to mention that while vehicles from outside the booth cannot verify this threshold signature's public key, they can still verify the individual public key of a vehicle (i.e., `v.pub`) as all vehicles use the same V-Guard key generation service.

With the feature of MMU, V-Guard can issues ordering and consensus procedures in changing memberships. For example, Figure 3 shows a V-Guard instance of 10 participating vehicles, where V_p is the proposer, V_π is the pivot validator, and V_0 to V_7 are 8 vehicle validators. In this case, when the proposer issues an ordering instance for data entry 1, the procedure takes place in Booth 1 (the initial ordering booth). The next ordering instance (i.e., data entry 2) will reuse the current booth (Booth 1) if it is still available. When the current booth becomes unavailable (e.g., some vehicles go offline), the MMU will provide a new booth (Booth 2) for the next ordering instance (i.e., data entry 3). As such, each ordering instance produces a paired result of an ordered data transaction and its residing membership information (details in §4.2). When a scheduled consensus instance takes place, it interacts with the MMU to get a consensus booth. The consensus instance sustains the output of the ordering instances by committing the paired results, thereby ensuring the immutability of these results across different memberships. Equipped with MMU and ordering/consensus instances, V-Guard can achieve consensus seamlessly under changing members and produce an immutable ledger recording traceable data transactions with corresponding membership profiles.

4.2 Ordering – Form a totally ordered log

The ordering phase secures a unique sequence number (i.e., ordering ID) for each data batch on a totally ordered log. V-Guard separates the ordering of data batches from consensus. The separation enables a non-blocking ordering procedure in which the ordering of a data batch does not wait for the completion of the consensus of its prior batch. The separation also allows ordering and consensus to be conducted in different booths during membership changes. The two features make V-Guard more applicable and efficient in V2X networks where vehicles are connected over unstable networks.

The ordering phase instances append log entries to a totally ordered log showing that ① what data is agreed upon in ② what order and by ③ what vehicle membership. Specifically, first, the proposer starts an ordering instance when collected data reaches their predefined batch sizes (denoted by β). Then, the ordering instance distributes data batches collected from the proposer to all validators, including the pivot and vehicles ("what data"); the proposer assigns each data batch with a unique ordering ID (i.e., the sequence number) ("what order") and a membership configuration that indicates which validators agree on the ordering ID as a quorum ("what membership"). We describe the workflow of an ordering phase instance for a data batch as follows.

O1 The proposer V_p assigns a unique sequence number (denoted by i) as the ordering ID (denoted by O_i) to a data batch (denoted by $\mathcal{B}_i$) and starts an ordering instance.
   (a) V_p's MMU confirms that the ordering booth ($\mathcal{V}_o$) is available and up-to-date; if not, V_p requests a new $\mathcal{V}_o$.
   (b) V_p sends $\langle$`Pre-Order`$, \mathcal{B}_i, h_{\mathcal{B}_i}, \mathcal{V}_o, h_{\mathcal{V}_o}, i, \sigma_{V_p}\rangle$ to all members in $\mathcal{V}_o$, where $h_{\mathcal{B}_i}$ is the hash of $\mathcal{B}_i$; $h_{\mathcal{V}_o}$ is the hash of $\mathcal{V}_o$; and σ_{V_p} is the signature that V_p signs the combination of i, $h_{\mathcal{B}_i}$, and $h_{\mathcal{V}_o}$.
   (c) V_p creates a set $\mathcal{R}_{o_i}$, waiting for replies from the validators in $\mathcal{V}_o$.

O2 After receiving a `Pre-Order` message, V_i first verifies the message and then replies to V_p. The verification succeeds when ① h_B and $h_{\mathcal{V}_o}$ match the hashes of B and $\mathcal{V}_o$, respectively; ② σ_{V_p} is valid; and ③ i has not been used. If the verification succeeds, V_i then sends $\langle$`PO-Reply`$, i, \sigma_{V_i}\rangle$ to V_p, where σ_{V_i} is the signature that V_i signs the combination of i, h_B, and $h_{\mathcal{V}_o}$.

O3 V_p collects `PO-Reply` messages from validators and adds them to $\mathcal{R}_{o_i}$ until it receives $2f$ replies (i.e., $|\mathcal{R}_{o_i}| = 2f$); then, V_p takes the following actions.
   (a) V_p converts the $2f$ collected signatures (i.e., σ_{V_i} in $\mathcal{R}_{o_i}$) to a (t, n)-threshold signature, σ_{o_i}, where $t = 2f$.
   (b) V_p creates a subset Q_{o_i} including the $2f$ vehicles, the signatures of which are converted to σ_{o_i}, from $\mathcal{V}_o$ as a membership quorum; i.e., $|Q_{o_i}| = 2f \wedge Q_{o_i} \subset \mathcal{V}_o$.
   (c) V_p sends $\langle$`Order`$, i, Q_{o_i}, \sigma_{o_i}\rangle$ to all members in $\mathcal{V}_o$.

   V_p now considers batch $\mathcal{B}_i$ ordered with ordering ID O_i ($O_i = i$) by the quorum Q_{o_i} of the members from $\mathcal{V}_o$. It appends $\langle O_i, \mathcal{B}_i, Q_{o_i}, \mathcal{V}_o, \sigma_{o_i}\rangle$ to the totally ordered log.

O4 After receiving an `Order` message, V_i verifies it by three criteria: ① σ_{o_i} is valid with a threshold of $2f$ (correct quorum sizes); ② signers of σ_{o_i} match Q_{o_i} (valid validators); and ③ $\forall V_i \in Q_{o_i}, V_i \in \mathcal{V}_o$ (the same membership).
   If the criteria are met, V_i considers batch $\mathcal{B}_i$ ordered at O_i by quorum Q_{o_i} in $\mathcal{V}_o$ and appends $\langle O_i, \mathcal{B}_i, Q_{o_i}, \mathcal{V}_o, \sigma_{o_i}\rangle$ to the totally ordered log.

In each ordering instance, each data entry is paired up with a verifiable booth. Validators receive the booth information ($\mathcal{V}_o$) in **O1** and verify the quorum based on the received booth in **O4**. The paired information is ordered with a unique ordering ID and will be committed in a consensus instance.

4.3 Consensus – Commit the ordered log

Shuttled consensus. V-Guard periodically initiates consensus instances to commit the log entries appended to the totally ordered log. Each consensus instance, similar to a "shuttle bus", is initiated at regular time intervals, denoted by Δ (e.g., Δ = 100 ms), and is responsible for log entries appended in Δ. Formally, a consensus instance scheduled at time $ts + \Delta$ is responsible for log entries appended in $[ts, ts + \Delta)$, where initially $ts = 0$. Thus, consensus instances cover all the entries on the totally ordered log; i.e., **no two adjacent consensus instances leave uncommitted log entries in between.** We use the starting timestamp (i.e., ts in $[ts, ts + \Delta)$) as the ID of the corresponding consensus instance (i.e., $C_i = ts$).

Moreover, each consensus instance includes the pivot validator (V_π) in its booth. V_π serves as a witness to ensure that the proposer does not double commit the same ordered log entries to different consensus booths. In this way, V-Guard effectively minimizes the

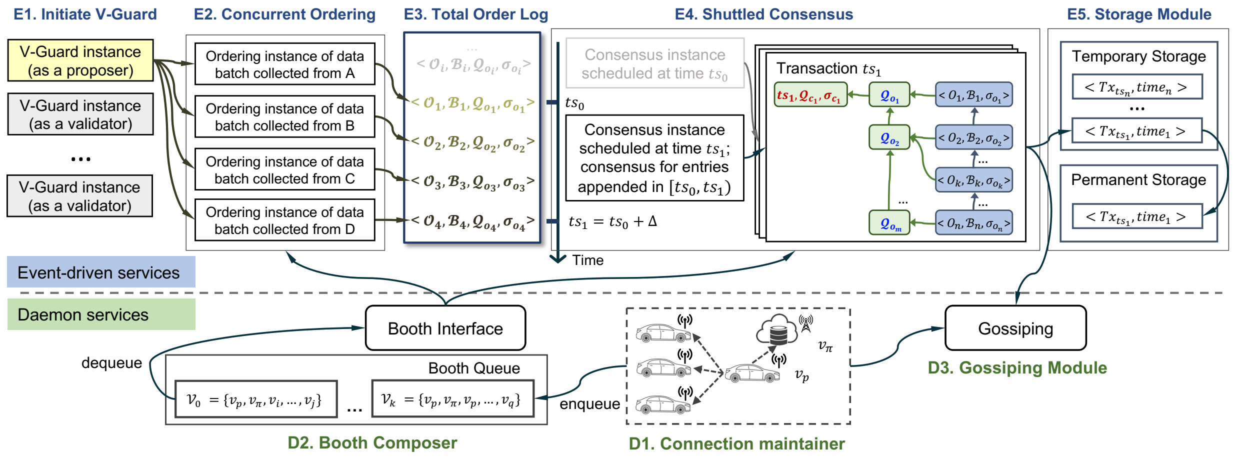

E1. Initiate V-Guard
E2. Concurrent Ordering
E3. Total Order Log
E4. Shuttled Consensus
E5. Storage Module
V-Guard instance (as a proposer)
V-Guard instance (as a validator)
V-Guard instance (as a validator)
Ordering instance of data batch collected from A
Ordering instance of data batch collected from B
Ordering instance of data batch collected from C
Ordering instance of data batch collected from D
Consensus instance scheduled at time ts_0
Consensus instance scheduled at time ts_1; consensus for entries appended in $[ts_0, ts_1)$
$ts_1 = ts_0 + \Delta$
Time
Transaction ts_1
Temporary Storage
Permanent Storage
Event-driven services
Daemon services
Booth Interface
Booth Queue
dequeue
enqueue
Gossiping
D1. Connection maintainer
D2. Booth Composer
D3. Gossiping Module

Figure 4: V-Guard workflow. The MMU prepares a queue of booths for ordering and consensus instances. Ordering instances concurrently append data entries to a totally ordered log. Consensus instances aggregate ordered entries into transactions and commit them periodically. Committed transactions can be further disseminated to other vehicles via gossiping.

participation of pivot validators (manufacturers) while upholding the integrity of the consensus process under changing memberships.

Membership pruning. We denote all the entries to be committed in a consensus instance C_i as a transaction (tx); i.e., $tx = \{entry_{ts}, ..., entry_{ts+\Delta}\}$. A transaction may contain multiple log entries whose orderings were conducted in the same membership. In this case, the transaction prunes away redundant membership and quorum information, linking entries ordered in the same booth to the same membership profile. In Figure 4 E4, if entries $\{\langle O_2, \mathcal{B}_2\rangle, ..., \langle O_k, \mathcal{B}_k\rangle\}$ are ordered by membership configuration $\langle Q_{O_2}, \mathcal{V}_{O_2}, \rangle$, they are linked to one membership profile. When membership changes infrequently, pruning can reduce the message size while maintaining the traceable membership feature. After applying pruning, a consensus instance with its ID $C_i = ts$ (i.e., scheduled at time $ts + \Delta$) works as follows.

C1 V_p prepares a pruned transaction (tx) for log entries appended in $[ts, ts + \Delta)$.

(a) V_p's MMU confirms that the booth for this consensus instance ($\mathcal{V}_c$) is available and up-to-date.

(b) V_p sends a `Pre-Commit` message to all members in $\mathcal{V}_c$. Since members in $\mathcal{V}_c$ may have changed, V_p checks if V_i was previously in the ordering quorums and has seen the entries in tx.

① If V_i has seen all entries in tx, V_p sends the starting and ending ordering IDs (O_{ts} and $O_{ts+\Delta}$) of entries in tx so that V_i can locate tx from its log. V_p sends $\langle$`Pre-Commit`$, ts, h_{tx}, O_{ts}, O_{ts+\Delta}, \mathcal{V}_c, h_{\mathcal{V}_c}, \sigma_{V_p}\rangle$, where h_{tx} and $h_{\mathcal{V}_c}$ are the hashes of tx and $\mathcal{V}_c$, respectively, and σ_{V_p} is the signature that V_p signs the combination of ts, h_{tx}, and $h_{\mathcal{V}_c}$.

② If V_i has not seen the entries in tx (e.g., V_i was not included in the ordering instance), V_p then sends $\langle$`Pre-Commit`$, ts, h_{tx}, tx, \mathcal{V}_c, h_{\mathcal{V}_c}, \{\mathcal{R}_{o_i}\}, \{Q_{o_i}\}, \sigma_{V_p}\rangle$, piggybacking the entries (tx) to be committed, where $\{\mathcal{R}_{o_i}\}$ is the set of the signed replies for every entry in tx, and $\{Q_{o_i}\}$ is the set of their corresponding signers (obtained from Step **O3** in ordering). This message is also sent to V_π, as V_π does not participate in the ordering instance.

(c) V_p creates a set $\mathcal{R}_c$, waiting for replies from validators.

C2 After receiving a `Pre-Commit` message, V_i first verifies the message based on the following criteria.

- If V_i has seen tx, V_i locates the log entries from O_{ts} to $O_{ts+\Delta}$ and calculates the hash of them. ① The hash must equal to h_{tx}, ② σ_{V_p} is valid, and ③ ts has not been used by other consensus instances.
- If V_i is not in the ordering booth (has not seen tx), then V_i is unable to verify the threshold signature because V_i did not participate in the DKG generation process in that booth (discussed in §4.1). Thus, to validate the correctness of the previous ordering phase, it must verify the replies in $\{\mathcal{R}_{o_i}\}$ of each entry included in tx. The public keys of the corresponding validators can be found in $\{Q_{o_i}\}$. In this case, ① V_i calculates the hash of tx, and the hash must equal to h_{tx}; ② for each data entry $\in tx$, $2f + 1$ signed replies can be found in $\{\mathcal{R}_{o_i}\}$ and the signatures are valid in accordance to $\{Q_{o_i}\}$; and ③ ts has not been used by previous consensus instances. In addition, V_π follows these criteria to verify a `Pre-Commit` message.

V_i/V_π sends $\langle$`PC-Reply`$, C_i, \sigma_{V_i/V_\pi}\rangle$ to V_p if the verification succeeds; σ_{V_i/V_π} is the signature that V_i/V_π signs the combination of ts, h_{tx}, and $h_{\mathcal{V}_c}$.

C3 V_p collects `PC-Reply` messages and adds them to $\mathcal{R}_c$ until it has received $2f$ replies and one of them is from V_π; i.e., $|\mathcal{R}_c| = 2f \wedge V_\pi \in \mathcal{R}_c$. Then, V_p takes the following actions.

(a) V_p converts the $2f$ collected signatures in $\mathcal{R}_c$ to a (t, n)-threshold signature, $\sigma_{c_{ts}}$, where $t = 2f$.

(b) V_p creates a subset $Q_{c_{ts}}$ including the $2f$ vehicles, the signatures of which are converted to $\sigma_{c_{ts}}$, from $\mathcal{V}_c$ as a membership quorum; i.e., $|Q_{c_{ts}}| = 2f \wedge Q_{c_{ts}} \subset \mathcal{V}_c$.

(c) V_p sends $\langle \texttt{Commit}, ts, Q_{o_i}, \mathcal{V}_c, \sigma_{c_{ts}} \rangle$ to members in $\mathcal{V}_c$ and considers tx committed.

C4 After receiving a `Commit` message, V_i verifies it by four criteria: ① $\sigma_{c_{ts}}$ is valid with a threshold of $2f$; ② signers of $\sigma_{c_{ts}}$ match $Q_{c_{ts}}$; ③ $V_\pi \in Q_{c_{ts}}$; and ④ $\forall\ V_i \in Q_{c_{ts}}, V_i \in \mathcal{V}_c$. If all criteria are met, V_i considers tx committed.

Consistency of memberships. V-Guard enforces a globally consistent state of consensus instance IDs across different changing memberships. Steps **C2**-**C4** establish a global-witness process by involving the pivot validator in each shuttled consensus instance. The pivot validator has a global view of the totally ordered log, so the proposer cannot cause a fork in memberships. Consequently, committed data batches form a *data chain*, and pruned membership profiles form a *membership chain* (illustrated in Figure 4). This feature of supporting dynamic memberships makes V-Guard applicable and efficient in V2X networks.

Booths are the smallest unit of membership. Despite data entries can be ordered and committed in different booths, the booth in each ordering/consensus instance is unchangeable; i.e., the same booth must persist through **O1**-**O4** or **C1**-**C4**. If a booth fails before an instance completes, the instance aborts the current process and invokes the MMU to obtain a new booth. Then, it retries the corresponding process.

Message complexity. V-Guard has a message complexity of $O(n)$. In Steps **O1**-**O2** and **C1**-**C2**, messages of size $O(1)$ flow only between the proposer and $(n-1)$ validators, so they have a message complexity of $O(n)$. In Steps **O3**, the proposer converts $2f$ signatures (in total of size $O(n)$) to one threshold signature of size $O(1)$ and broadcasts it to $n-1$ validators, so the message complexity remains $O(n)$. Therefore, V-Guard obtains linear message complexity.

Failure handling. In contrast to traditional BFT algorithms, view changes, selecting a new leader upon a leader's failure, are unnecessary in V-Guard. Since data from different vehicles is transactionally unrelated, each vehicle is the only proposer of its V-Guard instance, and different V-Guard instances operate independently. To detect a proposer's failure of a particular V-Guard instance, a vehicle validator employs a timer to restrict the consensus completion time of the proposer. If the timer expires, the vehicle simply withdraws from the instance. Thus, when a proposer fails, all validators will trigger timeouts and quit its instance. Note that a failed instance does not affect other instances on a validator, as they operate independently.

Malicious pivot validators cannot affect V-Guard's safety, but they may jeopardize liveness. For example, if a malicious pivot validator stops responding, the proposer cannot proceed in **C3**. In each V-Guard instance, its proposer uses an eventually strong failure detector (denoted by $\diamond S$), introduced by Malkhi and Reiter [90], to detect quiet pivot validators. $\diamond S$ guarantees that eventually a quiet node is permanently suspected by correct nodes.

After a pivot validator is detected, V-Guard yields the replacement of the faulty pivot validator to applications. Applications can implement their own replacement mechanisms associated with tagging and reporting such failures. For example, manufacturers may examine their operating cluster for possible intrusion attacks or assign other manufacturers as a pivot validator. For simplicity, we assume that a new pivot validator will eventually be provided to V-Guard instances.

4.4 Correctness discussion

We now sketch the correctness of V-Guard in terms of validity, safety, and liveness.

Theorem 1 (Validity). *Every data entry committed by correct members in the consensus phase must have been proposed in the ordering phase.*

Proof. Only a vehicle can be the proposer of its own V-Guard instance. In **O2**, validators sign the same content as the proposer, which includes the hash of a proposed data entry. In **O3** and **O4**, a valid threshold signature is converted from a quorum of signatures that sign the same hash of the proposed data entry. In addition, the proposer signs the hash of the transaction that includes the data entry in **C1**. If a transaction includes non-proposed data entries, the verification process in **C2** fails. Thus, a committed transaction must include data entries proposed in the ordering phase. □

V-Guard's membership management separates the ordering of data batches from consensus and achieves high performance with more granular control of dynamic memberships. The separation sill ensures safety, where no two non-faulty members commit the same data entry with conflicting ordering or consensus IDs.

Lemma 1. *No two non-faulty members in the same ordering booth agree on conflicting ordering IDs for the same data entry.*

Proof. We prove this lemma by discussing the two following scenarios. First, when the proposer is correct, it will not send conflicting ordering IDs, and f faulty vehicle validators cannot impact the process that the proposer collects sufficient `PO-Replys` from the remaining $2f$ correct vehicle validators in **O2**.

Second, when the proposer is faulty, it can send conflicting ordering IDs in two cases: ① it assigns the same ordering ID (i) to two data entries, or ② it assigns different ordering IDs to the same data entry. In ①, correct validators will not send a `PO-Reply` to the proposer if i has been used (Criterion 3 in **O2**), so the proposer cannot collect sufficient replies. In ②, although validators will reply to the proposer because they are blindsided, their signatures are signed with different i in the combination of $<i, h_B, h_{\mathcal{V}_o}>$ in **O2**, so the proposer cannot convert collected replies to a valid (t, n)-threshold signature with $t = 2f$. Therefore, no two non-faulty members in the same ordering booth agree on conflicting ordering IDs for the same data entry. □

Lemma 2. *No two non-faulty members in different consensus booths agree on conflicting consensus IDs for the same entry appended on the totally ordered log.*

Proof. The pivot validator participates all the consensus instances and has a global view of the paired consensus process.

If the proposer performs equivocations by sending different booths with conflicting consensus IDs, the pivot validator will not send a PC-Reply message to support a conflicting consensus instance. Thus, the proposer cannot form a valid threshold signature that can be examined by members (**C4**). Since consensus instances are continuous and do not leave uncommitted data entries in between, the formed chain do not permit a data entry ordered and committed with conflicting IDs. □

Before presenting the safety and liveness properties of V-Guard, we reiterate its core purpose. **The primary objective of vehicular blockchains is to resolve disputes over data between the user (proposer) and the manufacturer (pivot validator).** Specifically, V-Guard is designed to ensure that neither party can suppress evidence (e.g., delete data) nor fabricate it (e.g., manipulate records). For instance, consider a scenario where the user suspects a vehicle malfunction, while the manufacturer denies any fault—or both parties attempt to shift blame onto each other. V-Guard offers an efficient and secure mechanism to preserve and present verifiable evidence in such cases.

Although, in theory, a user and manufacturer could collude at the system level, such behavior undermines the very motivation for adopting a vehicular blockchain: to resolve potential disputes between users and manufacturers by establishing mutual accountability and tamper-proof data integrity.

Now, with Lemma 1 and 2, we discuss V-Guard's safety.

Theorem 2 (Safety). *All non-faulty members agree on a total order for proposed data entries in the presence of no more than f failures in each booth.*

Proof. We prove safety by discussing four cases: the proposer and the pivot validator are both correct (Case ①); the proposer is correct, but the pivot validator is faulty (Case ②); the pivot validator is correct, but the proposer is faulty (Case ③); and the proposer and pivot validator are both faulty (Case ④).

In Case ①, with a correct proposer and pivot validator, f faulty vehicle validators in a booth cannot form a valid quorum certificate of size $2f + 1$. We prove safety by contradiction. We claim that two correct members commit the two data entries in the same order. Say if two data entries are committed in consensus instances with the same order, with Lemma 2, there must exist two quorums constructed in **C3** agreeing on the two entries, which have been appended to the totally ordered log; otherwise, the proposer cannot receive sufficient votes in **C3**. In this case, the two entries must have been ordered with the same ordering ID in an ordering booth, which contradicts Lemma 1. Therefore, each entry is committed with a unique ordering ID. This is similar to the condition where the leader is correct in traditional BFT algorithms such as PBFT [26] and HotStuff [128].

In Case ②, since the consensus process involves the pivot validator only in consensus instances, a faulty pivot validator may not send a correctly signed PC-Reply message to the proposer in **C2**, which will result in the proposer cannot proceed in **C3**. This may lead to a temporary suspension in consensus but has no impact on safety. No data entries with conflicting orders will be committed.

In Case ③, since the pivot validator is involved in all consensus instances, it has a globally consistent view of ordering IDs for date entries. When the proposer is faulty and trying to cause a fork among different booths, the pivot validator will not send a PC-Reply message in **C2**; thus, the consensus cannot be achieved.

In Case ④, by Lemma 1, conflicting ordering IDs cannot be confirmed, as the proposer cannot produce $2f$+1 valid signatures to form a quorum. While this may stall the consensus process until the pivot validator is replaced, it does not compromise safety. As previously noted, it does not make practical sense for the user and manufacturer to collude, since the primary purpose of vehicular blockchains is to establish mutual accountability and prevent such collusion. If the collusion takes place, it has no effect over other instances as each proposer's instance operates independently.

To conclude, the joint participation of the proposer and pivot validator ensures that data entries with conflicting ordering IDs cannot be committed. As a result, all non-faulty members across booths reach agreement on a consistent total order of data entries. □

Theorem 3 (Liveness). *After GST, a non-faulty proposer eventually commits a proposed data entry.*

Proof. V-Guard assumes partial synchrony for liveness. After GST, the message delay and processing time are bound. When the pivot validator is correct, since no two adjacent consensus instances leave uncommitted log entries in between, every data entry appended on the totally ordered log will be committed.

When the pivot validator is faulty, it has no impact on ordering, but consensus instances will temporarily suspend. The failure detector $\diamond S$ will eventually detect and report a quiet pivot validator. After a correct pivot validator is assigned, the consensus instances resume. Thus, during sufficiently long periods of synchrony, a correct proposer eventually receives replies from validators and completes the consensus for a proposed data entry. □

4.5 Storage and gossiping

V-Guard stores committed data batches in a *storage master*. Unlike server-centric blockchains (e.g., HyperLedger Fabric [8] and Diem [40]), V-Guard operates among vehicles, which usually have limited storage capability. V-Guard stores temporary and permanent records, aiming to flexibly maximize storage usage.

In addition, V-Guard provides a gossiping module that strengthens system robustness by disseminating committed transactions across the network, increasing redundancy and fault tolerance. This module does not affect the correctness of consensus and can be applied based on preferences.

Due to space limitations, the details of the storage and gossiping modules are presented in an online appendix [10].

5 EVALUATION

We compared the end-to-end performance of our V-Guard implementation (vg) against three state-of-the-art baseline approaches (using their open-source implementations):

- HotStuff [68, 128] (hs), a linear BFT protocol, whose variant is used in Libra/Diem blockchains [40].
- ResilientDB [66, 105] (rdb), a BFT key-value store using PBFT [26] as its consensus protocol.
- Narwhal [37, 94] (nw), a DAG-based mempool protocol that distributes transactions before consensus.

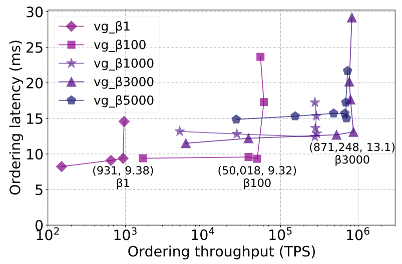


Figure 5: Ordering throughput vs. latency of V-Guard under varying batch sizes, where $\beta = 3000$ obtains the best performance under $n = 4$, $\delta = 0$, and $m = 32$.

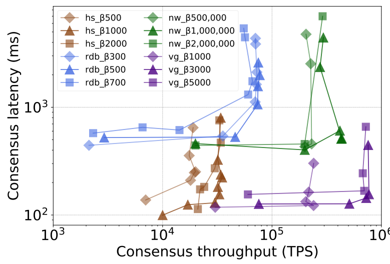


Figure 6: Consensus throughput vs. latency comparison of V-Guard and its baselines with varying batch sizes under $n = 4$, $\delta = 0$, and $m = 32$.

	Batch size	Throughput (TPS)	Latency (ms)
HotStuff	(β=1000)	34, 015	155
ResilientDB	(β=500)	80, 580	4368
Narwhal	(β=10^{6*})	423, 058	516
V-Guard (o)	(β=3000)	871, 248	13
(c)		765, 930	143

* Narwhal uses fixed-bytes buffers as batches, so its buffer size = $m \times \beta_b$.

Figure 7: A summary of the peak performance of V-Guard (including ordering (o) and consensus (c)) and its baselines under their best batch sizes, where $n = 4$, $\delta = 0$, and $m = 32$.

We deployed them on 4, 16, 31, 61, and 100 VM instances on Compute Canada Cloud [29]. Each instance includes a machine with 2 vCPUs supported by 2.40 GHz Intel Xeon processors (Skylake) with a cache size of 16 MB, 7.5 GB of RAM, and 90 GB of disk space running on Ubuntu 18.04.1 LTS. The TCP/IP bandwidth measured by `iperf` and the raw network latency between two instances are around 400 Megabytes/s and 2 ms, respectively.

We use the following notation for reporting the results:

m	The message size (bytes); i.e., the size of a data entry sent to achieve consensus, excluding any message header (other parameters in the message).
β	The batch size; i.e., the number of data entries (transactions) in a data batch sent to validators.
n	The number of nodes (the membership size).
δ	Emulated network delay implemented by `netem`.

Network delays. We used `netem` to implement additional network delays of δ = 0, 10±5, 50±10, 100±20 ms in normal distribution at all scales. The performance change under the emulated network delays implies the performance in more complicated networks (e.g., wireless networks).

Workloads. V-Guard is a versatile blockchain that enables users to define their own message types. The size of messages is a critical factor in replication as all messages need to be serialized for transmission over the network, regardless of their type. To ensure a fair comparison with other baselines, the evaluation presented in the paper focuses on the replication performance of plain messages with different message sizes of m = 32, 64, and 128 bytes.

The rest of this section is organized as follows. § 5.1 shows the evaluation result of V-Guard comparing against its baselines in a static membership; § 5.2 reports the performance of V-Guard against two state-of-the-art membership reconfiguration approaches under dynamic memberships; and § 5.3 summarizes the evaluation results.

5.1 Performance in static memberships

We set V-Guard's consensus interval time to Δ = 100 ms; i.e., a consensus instance is scheduled every 100 ms. Performance was reported in throughput and latency, where throughput is calculated as transactions per second (TPS) (i.e., the number of entries committed per second), and latency was measured as the time elapsed for committing a data entry.

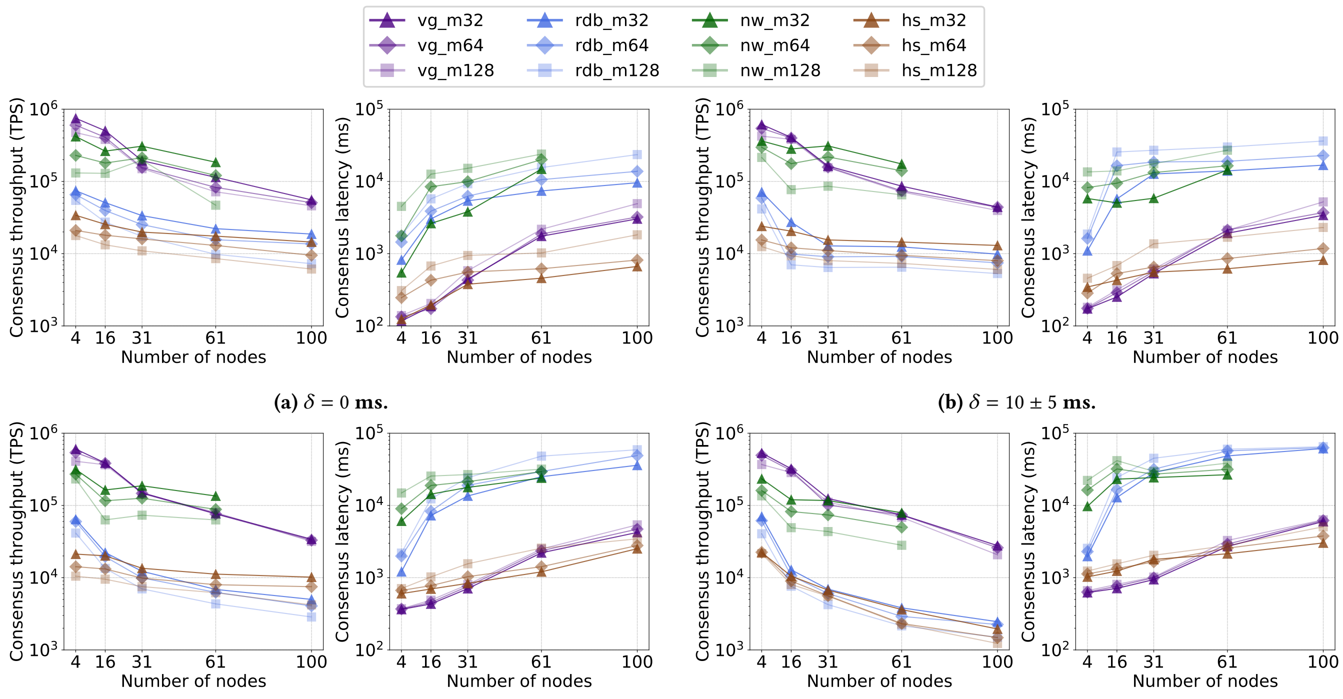
Batching is applied to measure performance. Under increasing batch sizes, throughput gains a diminishing marginal increase and peaks at a particular (the best) batch size; after this, throughput decreases as batching becomes more costly than other factors in the consensus process.

To make fair comparisons, V-Guard operates one instance (i.e., $\gamma = 1$) when comparing against its baselines in §5.1.1 and 5.1.2; the performance of multiple simultaneously running V-Guard instances is shown §5.1.3.

5.1.1 Peak performance. The peak performance of all approaches occurred at $n = 4$ under a message size of $m = 32$ bytes without implemented network delay (i.e., $\delta = 0$ ms). To obtain V-Guard's peak performance, we kept increasing its batch size and monitoring the result (shown in Figure 5): the ordering throughput first increases with diminishing marginal returns (e.g., $931 > \frac{50{,}018}{100} = 500 > \frac{871{,}248}{3000} = 290$) and peaks at a throughput of 871, 248 TPS with a latency of 13 ms under $\beta = 3000$. After that, under a higher β, the ordering throughput decreases while the latency increases. We used this method to measure the peak performance of consensus throughput and latency for all approaches (shown in Figure 6). Under each batch size, we kept increasing clients to saturate the network (until an elbow of a curve occurs). We show the result of V-Guard and its baselines under three batch sizes that result in near-optimal (e.g., `vg_1000`), optimal (e.g., `vg_3000`), and over-optimal performance (e.g., `vg_5000`).

Figure 7 summarises the peak performance of all approaches, where V-Guard outperforms its baselines both in throughput and latency. V-Guard's peak throughput is 22× higher than HotStuff [128], 9.5× higher than ResilientDB [66], and 1.8× higher than Narwhal [37]. Since consensus instances periodically commit ordered data entries, V-Guard's ordering and consensus have similar throughput. V-Guard's high performance benefits from its implementation and design. Compared with the implementations of HotStuff [68] and ResilientDB [105], V-Guard makes use of threshold signatures and obtains a linear message complexity. In addition, the separated ordering reduces latency by issuing ordering instances concurrently, and shuttled consensus significantly reduces message passing, which saves network bandwidth and increases throughput.

5.1.2 Scalability. We then measured the consensus performance of V-Guard and its baselines at increasing message sizes and system



(a) $\delta = 0$ **ms.** **(b)** $\delta = 10 \pm 5$ **ms.**

(c) $\delta = 50 \pm 10$ **ms.** **(d)** $\delta = 100 \pm 20$ **ms.**

Figure 8: Performance comparisons of V-Guard and its baselines using their best batch sizes with varying message sizes of $m = 32, 64,$ and 128 bytes under implemented network delays of $\delta = 0, 10 \pm 5, 50 \pm 10,$ and 100 ± 20 ms (normal distribution) at increasing system scales of $n = 4, 16, 31, 61,$ and 100 nodes. (Narwhal's mempool protocol was unable to initialize because its required memory space exceeded our system resources under $n = 100$.[1])

scales under varying network delays (shown in Figure 8). We set each approach's batch size to their best (optimal) batch sizes under $n = 4$. The results show that when the message size and network delay increase, all approaches experience a decline in throughput and an increase in latency; in particular, message sizes have a more pronounced effect on throughput while network delays have a more direct effect on consensus latency.

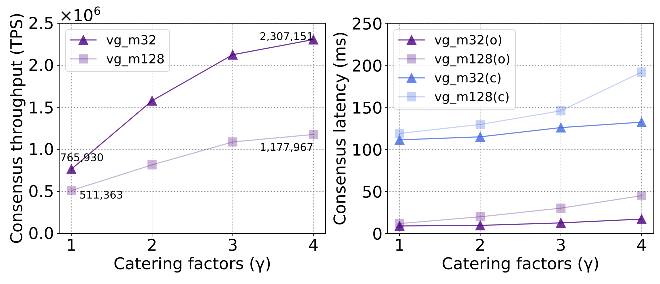
Under increasing system scales, V-Guard, HotStuff, and ResilientDB, which use a single leader to manage the consensus and transaction distribution, witness a decrease in throughput. In contrast, Narwhal observes an increase in throughput when the system scales up from 16 to 31 nodes. Narwhal develops a mempool protocol that distills the distribution of transactions from consensus. Each node in Narwhal runs as a worker and a validator, where the worker disseminates transactions by the mempool, while the validator concentrates on achieving consensus on the hash of transactions. Consequently, Narwhal's throughput increases (from 16 to 31 nodes) when more transactions are fed into the network but then starts to decline when the transaction distribution becomes a heavier burden than the consensus process. While Narwhal exhibits impressive throughput, its DAG structure results in high latency, making Narwhal one of the highest latency algorithms.

V-Guard manages to achieve the highest throughput at small scales. This is due to the separation of the ordering and consensus processes, which significantly reduces message passing overhead. As the system scales up, V-Guard, despite only having one node (the proposer) to disseminate data entries, still maintains a comparable throughput to Narwhal, even though Narwhal uses a DAG structure and has multiple nodes disseminating data entries. Overall, V-Guard's separation of ordering and consensus processes allows it to achieve high throughput with low latency, even at large scales.

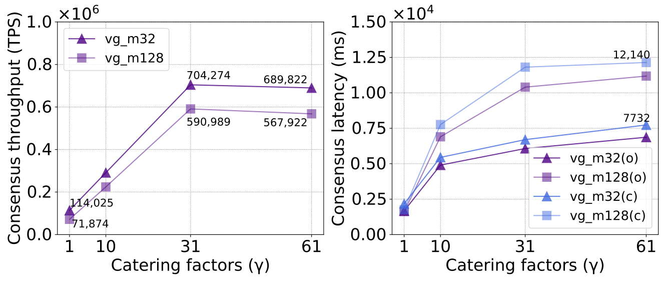
In addition, V-Guard shows the mildest performance drop under increasing message workloads, system scales, and network delays among all approaches. In particular, compared to its baselines, V-Guard's consensus throughput and latency are the least affected by higher network delays. For example, when $n = 4$ and $m = 32$, its throughput drops from 765, 930 TPS under $\delta = 0$ ms to 531, 594 TPS under $\delta = 100 \pm 20$ ms while latency rises from 143 to 622 ms. V-Guard's ordering instances take place in a non-blocking manner, without waiting for the completion of their prior instances, and shuttled consensus instances amortize the network delay among data entries included in Δ. This feature makes V-Guard a strong fit for V2X applications, where higher network latency is the norm.

5.1.3 Performance under increasing catering factors. We also evaluated V-Guard's performance when each node operates multiple V-Guard instances (i.e., $\gamma > 1$). For example, in a 4-node system

[1]Narwhal's mempool protocol requires static memory allocations based on the system and batch sizes; when n=100, it requires each worker to allocate 5.7 Gigabytes of static memory for the protocol alone before the program runs, which cannot be accommodated by our system resources.

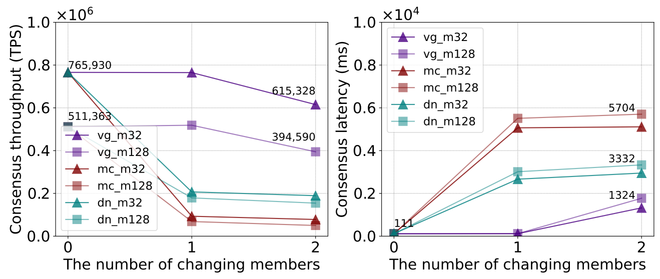


(a) $n = 4$.

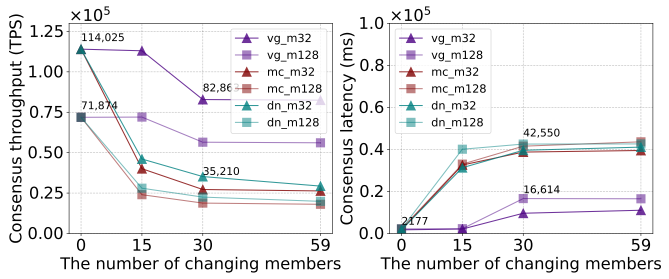


(b) $n = 61$.

Figure 9: V-Guard's performance with varying message sizes under increasing catering factors, where $\delta = 0$ and $\beta = 3000$.



(a) $n = 4$.



(b) $n = 61$.

Figure 10: Performance comparison of V-Guard and its baselines (i.e., MC and Dyno (dn)) under dynamic memberships, where $\delta = 0$ and $\beta = 3000$. V-Guard's integrated transaction consensus and membership management shows significant advantages in both throughput and latency.

(n_1 to n_4) with $\gamma = 2$, n_1 and n_2 start their individual V-Guard instances and operate as the proposer; n_1 also operates as a validator in n_2's instance and vice versa. The remaining two nodes, n_3 and n_4, operate as a validator in both n_1's and n_2's instances.

We measured V-Guard's throughput and latency at two system scales under increasing catering factors from $\gamma = 1$ to $\gamma = n$ (shown in Figure 9). When n=4, the throughput increases with diminishing gains and peaks at a throughput of 2,307,151 TPS, while both the ordering and consensus latencies experience a slight increase. When n=61 and $m = 32$, throughput peaks at $\gamma = 31$ with 704,274 TPS and decreases at $\gamma = 61$ with 689,822 TPS. With a higher catering factor, the performance of each instance drops as system resources are amortized among all instances.

5.2 Performance in dynamic memberships

To demonstrate V-Guard's performance under dynamic environments, we compared it against two membership reconfiguration approaches, MC [107] and Dyno [46], under frequently changing memberships. MC keeps track of system membership and periodically notifies the nodes of membership changes, whereas Dyno handles membership changes more actively. When members join or leave, it starts a consensus process between an old membership and a new membership to agree on the new configuration.

To make a fair comparison, we implemented MC and Dyno and paired them with V-Guard's replication mechanism. Thus, all three protocols exhibit identical performance with a static set of members. This allows the evaluation to purely compare the performance of the membership reconfiguration when members start to join and leave each system.

We measured the performance of all protocols under dynamic memberships at two scales: $n = 4$ and $n = 61$. Since V-Guard always includes the proposer and pivot validator in membership, the maximum number of changing members supported is 2 and 59 members (nodes) to change under $n = 4$ and $n = 61$, respectively.

The results show that V-Guard outperforms MC and Dyno in both throughput and latency with changing members. V-Guard's dynamic memberships do not affect performance when the number of changing members is less than f. Since each ordering and consensus instance takes place in different booths, when a consensus instance includes new members that have not previously seen the ordered data entries, new members must undergo a time-consuming task to validate all signatures of data entries included in a transaction. Thus, old members reply to the proposer faster than the new members; when the remaining old members can still form a quorum ($2f + 1$), the system performance is not affected. For example, 1 and 15 changing members in Figure 10a and Figure 10b, respectively. In contrast, if the number of remaining old members in a consensus quorum drops below $2f + 1$, the quorum will have to wait for new members to join. This significantly impacted performance, as new members must complete the validation task and respond to the proposer, leading to a drop in overall performance. When $m = 32$, V-Guard's throughput drops by 19% and 27% under

$n = 4$ and $n = 61$, respectively. With 30 new members in $n = 61$, the consensus latency increases from 2, 177 to 16, 614 ms.

Dyno and MC do not integrate membership changes into transaction consensus; they both use designated consensus operations to reconfigure the system when members join and leave. Compared to MC, Dyno actively conducts consensus to reach an agreement on the new membership with old members and exhibits a higher performance. Compared to V-Guard, both Dyno and MC must reconfigure the system when a member joins or leaves, even if the number of newly joined members is less than f. With 1 changing member in $n = 4$, V-Guard's throughput is 8.2× higher than MC and 3.7× higher than Dyno. With more changing members in $n = 61$, the performance gap gradually diminishes, as all protocols start to suffer from the time cost of validating the signatures of previously ordered data entries.

5.3 Summary of results

Under static membership, V-Guard outperforms its baselines in terms of throughput and latency (§ 5.1.1). When the system scales up, V-Guard shows a similar performance decrease as its baselines under varying workloads; under increasing network delays, V-Guard exhibits the lowest decrease in throughput at all scales (§ 5.1.2). Under dynamic membership, V-Guard outperforms its baseline membership reconfiguration approaches, with its performance remaining unchanged when the number of new members is no more than f (§ 5.2).

6 RELATED WORK

Blockchains in V2X networks. V2X blockchains should be able to achieve consensus in real-time for a large volume of data, such as vehicle status data generated by Telsa Autopilot [119] and Cadillac SuperCruise [24], and cope with unstable connections among vehicles on the roads [71, 86]. Depending on whether membership management is required, blockchains can be categorized as permissionless and permissioned blockchains. Permissionless blockchains, applying the Proof-of-X protocol family (e.g., Proof-of-Work [95] and Proof-of-Stake [59]) to reach probabilistic agreement, allow participants to join anonymously. However, due to their limited performance, permissionless blockchains are often not deployed in latency-critical V2X blockchain applications coping with a large volume of data in real-time [49, 67, 73, 129], despite some privacy-critical V2X use cases because they require anonymous participation [52, 100].

In contrast, permissioned blockchains, registering participants with identities, achieve high throughput and low latency where participants can be verified by their signatures. They often apply BFT consensus algorithms (e.g., PBFT [26], SBFT [64] and HotStuff [128]) to achieve deterministic agreement among all participants and have been favored by pioneering blockchain testbeds initiated by major automobile manufacturers [17, 21, 77, 124]. However, they are not adaptive to dynamic memberships; when membership changes, consensus has to temporarily stop and facilitates for updating configuration profiles [107].

BFT consensus algorithms. Consensus algorithms are at the core of providing safety and liveness for state machine replication (SMR) [110]. BFT SMR provides an abstraction of consensus services that agree on a total order of requests in the presence of Byzantine failures [13, 79, 81, 109]. Byzantine failures are becoming more common in blockchains as participating users may intentionally break the protocol (launching Byzantine attacks) to gain more profit [35, 83, 131].

Leader-based BFT algorithms have been widely used by permissioned blockchains, such as HyperLedger Fabric [8] and Diem [40]. After PBFT [26] pioneered a practical BFT solution, numerous approaches have been proposed for optimizations from various aspects. They use speculative decisions to reduce workloads for the single leader [44, 65, 76], develop high performance implementations [19, 23, 48, 62, 102, 116], reduce messaging costs [43, 87, 91, 96, 115, 127], obtain linear message complexity by using threshold signatures [64, 128, 130], distill transaction distribution from consensus [37], limit faulty behavior using trusted hardware [15, 27, 39, 75, 82], improve fault tolerance thresholds [69, 126], offer confidentiality protection dealing with secret sharing [121], and apply accountability for individual participants [28, 97, 111].

In addition to leader-based algorithms, leaderless BFT algorithms avoid single points of failure and single leader bottlenecks [34, 45, 78, 92, 118]. Without a leader, leaderless BFT algorithms often utilize binary Byzantine agreement [93] to jointly form quorums [16] but suffer from high message and time costs for conflict resolutions.

Membership reconfiguration. Membership changes involve reconfiguration of consensus systems [80], where the agreement protocol needs to switch to the updated replica-group configuration [42]. Under loosely-consistent models, some approaches implement peer-to-peer lookups that determine a system membership as the neighborhood of a certain identifier (e.g., [25, 117]). Under strong-consistent models, a sequence of consistent views of the system memberships is required to ensure the safety of consensus processes [33, 63, 74]. For example, MC [107] proposes a membership reconfiguration approach for PBFT [26], keeping track of system membership and periodically notifying other system nodes of membership changes. However, transaction consensus must wait for each membership reconfiguration to be completed, which makes it inefficient during frequent membership changes. Furthermore, Dyno [46] handles membership changes more efficiently by actively conducting consensus on a membership reconfiguration; it coordinates servers in an old membership (M_1) to agree on a new membership (M_2). However, Dyno is not suitable for V2X blockchains, because M_1 can become unavailable unpredictably before agreeing on the new M_2 given vehicles' intermittent connectivity.

7 CONCLUSIONS

V-Guard efficiently conducts consensus when system memberships constantly change (dynamic environment). It binds each data entry with a system membership profile during the consensus process. In addition, it separates ordering from consensus and allows data entries to be ordered and committed in different configurations, achieving high performance in both static and dynamic memberships. Our evaluation results show that V-Guard outperforms traditional BFT algorithms, including HotStuff [128], ResilientDB [66], and Narwhal [37]. Under changing memberships, V-Guard's performance is the least affected compared to state-of-the-art membership reconfiguration mechanisms including MC [107] and Dyno [46].

REFERENCES

[1] Khadige Abboud, Hassan Aboubakr Omar, and Weihua Zhuang. 2016. Interworking of DSRC and cellular network technologies for V2X communications: A survey. *IEEE transactions on vehicular technology* 65, 12 (2016), 9457–9470.

[2] Hillary Abraham, Chaiwoo Lee, Samantha Brady, Craig Fitzgerald, Bruce Mehler, Bryan Reimer, and Joseph F Coughlin. 2016. Autonomous vehicles, trust, and driving alternatives: A survey of consumer preferences. *In Transportation Research Board 96th Annual Meeting* (2016).

[3] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. 2016. Solida: A blockchain protocol based on reconfigurable byzantine consensus. *arXiv preprint arXiv:1612.02916* (2016).

[4] National Highway Traffic Safety Administration. [n.d.]. Vehicle Identification Numbers (VINs). http://www.ibc.ca/qc/.

[5] National Highway Traffic Safety Administration. 2021. Summary Report: Standing General Order on Crash Reporting for Automated Driving Systems. https://www.nhtsa.gov/sites/nhtsa.gov/files/2022-06/ADS-SGO-Report-June-2022.pdf.

[6] National Highway Traffic Safety Administration. 2022. Waymo tops NHTSA's list for autonomous vehicle crashes. https://www.freightwaves.com/news/waymo-tops-nhtsas-list-of-autonomous-vehicle-crashes/.

[7] 5G Americas. 2021. Vehicular Connectivity: C-V2X & 5G. https://www.5gamericas.org/wp-content/uploads/2021/09/Vehicular-Connectivity-C-V2X-and-5G-InDesign-1.pdf.

[8] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, 30.

[9] Karolos Antoniadis, Rachid Guerraoui, Dahlia Malkhi, and Dragos-Adrian Seredinschi. 2018. *State machine replication is more expensive than consensus*. Technical Report.

[10] V-Guard Online Appendix. 2024. V-Guard BFT. https://github.com/vguardbc/vguardbft/blob/main/appendix.

[11] CBS BAY AREA. 2023. Driverless robotaxi crashes with fire truck in San Francisco; Passenger injured. https://www.cbsnews.com/sanfrancisco/news/cruise-driverless-car-crash-san-francisco-fire-truck/.

[12] American Bar Association. 2018. Legal implications of driverless cars. https://www.americanbar.org/news/abanews/publications/youraba/2018/december-2018/legal-implications-of-driverless-cars/.

[13] Hagit Attiya and Jennifer Welch. 2004. *Distributed computing: fundamentals, simulations, and advanced topics*. Vol. 19. John Wiley & Sons.

[14] Ford Authority. 2021. Ford Blockchain Initiative Aims To Identify Unethically Sourced Materials. https://fordauthority.com/2021/12/ford-blockchain-initiative-aims-to-identify-unethically-sourced-materials/.

[15] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. 2017. Hybrids on steroids: SGX-based high performance BFT. In *Proceedings of the Twelfth European Conference on Computer Systems*. 222–237.

[16] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. 1994. Asynchronous secure computations with optimal resilience. In *Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing*. 183–192.

[17] Mercedes benz Group. 2020. Once round the block, please! https://group.mercedes-benz.com/innovation/blockchain-2.html.

[18] Alysson Bessani, Eduardo Alchieri, João Sousa, André Oliveira, and Fernando Pedone. 2020. From byzantine replication to blockchain: Consensus is only the beginning. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 424–436.

[19] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. 2014. State machine replication for the masses with BFT-SMART. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 355–362.

[20] Bloomberg. 2022. Tesla Denies Malfunction to Blame for Deadly Crash Caught on Video in China. https://www.bloomberg.com/news/articles/2022-11-14/tesla-to-assist-investigation-into-fatal-chinese-car-accident.

[21] BMW. 2021. How blockchain automotive solutions can help drivers. https://www.bmw.com/en/innovation/blockchain-automotive.html.

[22] National Transportation Safety Board. 2021. A driverless Tesla crashed and burned for four hours, police said, killing two passengers in Texas. https://www.washingtonpost.com/nation/2021/04/19/tesla-texas-driverless-crash/.

[23] Ethan Buchman. 2016. *Tendermint: Byzantine fault tolerance in the age of blockchains*. Ph.D. Dissertation.

[24] GM Cadillac. [n.d.]. Cadillac SuperCurise. https://www.cadillac.com/world-of-cadillac/innovation/super-cruise.

[25] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S Wallach. 2002. Secure routing for structured peer-to-peer overlay networks. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 299–314.

[26] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[27] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. 2007. Attested append-only memory: Making adversaries stick to their word. *ACM SIGOPS Operating Systems Review* 41, 6 (2007), 189–204.

[28] Pierre Civit, Seth Gilbert, and Vincent Gramoli. 2021. Polygraph: Accountable byzantine agreement. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 403–413.

[29] Compute Canada Cloud. [n.d.]. Arbutus Cloud Platform. https://arbutus.cloud.computecanada.ca/.

[30] CNBC. 2018. Four of the world's largest automakers want to bring the 'blockchain' to your car. https://www.cnbc.com/2018/05/02/four-of-the-worlds-largest-automakers-want-to-bring-the-blockchain-to-your-car.html.

[31] Riccardo Coppola and Maurizio Morisio. 2016. Connected car: technologies, issues, future trends. *ACM Computing Surveys (CSUR)* 49, 3 (2016), 1–36.

[32] Forbes Technology Council. 2021. Blockchain In The Automotive Sector: Three Use Cases And Three Challenges. https://www.forbes.com/sites/forbestechcouncil/2021/12/22/blockchain-in-the-automotive-sector-three-use-cases-and-three-challenges/?sh=5dcd164f2508.

[33] James Cowling, Dan RK Ports, Barbara Liskov, Raluca Ada Popa, and Abhijeet Gaikwad. 2009. Census: Location-aware membership management for large-scale distributed systems. (2009).

[34] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. 2018. Dbft: Efficient leaderless Byzantine consensus and its application to blockchains. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 1–8.

[35] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 910–927.

[36] The Daily. 2023. Who's liable in a 'self-driving' car crash? https://thedaily.case.edu/whos-liable-in-a-self-driving-car-crash/.

[37] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 34–50.

[38] Tooska Dargahi, Hossein Ahmadvand, Mansour Naser Alraja, and Chia-Mu Yu. 2021. Integration of blockchain with connected and autonomous vehicles: vision and challenge. *ACM Journal of Data and Information Quality (JDIQ)* 14, 1 (2021), 1–10.

[39] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. 2022. DAMYSUS: streamlined BFT consensus leveraging trusted components. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 1–16.

[40] Diem. 2020. The Diem blockchain. https://developers.diem.com/main/docs/the-diem-blockchain-paper.

[41] Murat Dikmen and Catherine Burns. 2017. Trust in autonomous vehicles: The case of Tesla Autopilot and Summon. In *2017 IEEE International conference on systems, man, and cybernetics (SMC)*. IEEE, 1093–1098.

[42] Tobias Distler. 2021. Byzantine fault-tolerant state-machine replication from a systems perspective. *ACM Computing Surveys (CSUR)* 54, 1 (2021), 1–38.

[43] Tobias Distler and Rüdiger Kapitza. 2011. Increasing performance in Byzantine fault-tolerant systems with on-demand replica consistency. In *Proceedings of the sixth conference on Computer systems*. 91–106.

[44] Sisi Duan, Sean Peisert, and Karl N Levitt. 2014. hBFT: speculative Byzantine fault tolerance with minimum cost. *IEEE Transactions on Dependable and Secure Computing* 12, 1 (2014), 58–70.

[45] Sisi Duan, Michael K Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2028–2041.

[46] Sisi Duan and Haibin Zhang. 2022. Foundations of dynamic BFT. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1317–1334.

[47] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)* 35, 2 (1988), 288–323.

[48] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. 2019. BlockchainDB: A shared database on blockchains. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1597–1609.

[49] Vasiliy Elagin, Anastasia Spirkina, Mikhail Buinevich, and Andrei Vladyko. 2020. Technological aspects of blockchain application for vehicle-to-network. *Information* 11, 10 (2020), 465.

[50] ELECTREK. 2021. Several Tesla owners claim Model X accelerated/crashed on its own but everything points to user error. https://electrek.co/2016/09/30/tesla-model-x-owners-claim-crash-sudden-acceleration/.

[51] ELECTREK. 2021. Tesla publicly shares data logs of vehicle involved in crash that led owner to protest at auto show. https://electrek.co/2021/04/23/tesla-shares-publicly-data-logs-of-vehicle-involved-in-crash-led-owner-protest-auto-show/.

[52] Qi Feng, Debiao He, Sherali Zeadally, Muhammad Khurram Khan, and Neeraj Kumar. 2019. A survey on privacy protection in blockchain system. *Journal of Network and Computer Applications* 126 (2019), 45–58.

[53] Forbes. 2020. Mercedes Leveraging Blockchain As Part Of Its Ambition2039 Initiative. https://www.forbes.com/sites/darrynpollock/2020/01/31/mercedes-leveraging-blockchain-as-part-of-its-ambition2039-initiative/?sh=79f5c9464b8a.

[54] Forbes. 2022. Privacy Battle Over Connected Cars Takes An Interesting Turn In California. https://www.forbes.com/sites/stevetengler/2022/05/17/privacy-battle-over-connected-cars-takes-an-interesting-turn-in-california/?sh=23c35e164d72.

[55] Ford. 2020. FORD STUDY SHOWS BLOCKCHAIN, DYNAMIC GEOFENCING AND PLUG-IN HYBRID VANS CAN HELP IMPROVE URBAN AIR QUALITY. https://media.ford.com/content/fordmedia/feu/en/news/2020/12/17/ford-study-shows-blockchain--dynamic-geofencing-and-plug-in-hybr.html.

[56] Yuchuan Fu, Fei Richard Yu, Changle Li, Tom H Luan, and Yao Zhang. 2020. Vehicular blockchain-based collective learning for connected and autonomous vehicles. *Ieee wireless communications* 27, 2 (2020), 197–203.

[57] Miguel Garcia, Alysson Bessani, and Nuno Neves. 2019. Lazarus: Automatic management of diversity in bft systems. In *Proceedings of the 20th International Middleware Conference*. 241–254.

[58] Mario H Castañeda Garcia, Alejandro Molina-Galan, Mate Boban, Javier Gozalvez, Baldomero Coll-Perales, Taylan Şahin, and Apostolos Kousaridas. 2021. A tutorial on 5G NR V2X communications. *IEEE Communications Surveys & Tutorials* 23, 3 (2021), 1972–2026.

[59] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 51–68.

[60] SP Global. 2022. Data access among top concerns as insurers await autonomous cars. https://www.spglobal.com/marketintelligence/en/news-insights/latest-news-headlines/data-access-among-top-concerns-as-insurers-await-autonomous-cars-69760098.

[61] The Guardian. 2017. The customer is always wrong: Tesla lets out self-driving car data – when it suits. https://www.theguardian.com/technology/2017/apr/03/the-customer-is-always-wrong-tesla-lets-out-self-driving-car-data-when-it-suits.

[62] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. 2010. The next 700 BFT protocols. In *Proceedings of the 5th European conference on Computer systems*. 363–376.

[63] Rachid Guerraoui and André Schiper. 2001. The generic consensus service. *IEEE Transactions on Software Engineering* 27, 1 (2001), 29–41.

[64] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. SBFT: A scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 568–580.

[65] Lachlan J Gunn, Jian Liu, Bruno Vavala, and N Asokan. 2019. Making speculative BFT resilient with trusted monotonic counters. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 133–13309.

[66] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. [n.d.]. ResilientDB: Global Scale Resilient Blockchain Fabric. *Proceedings of the VLDB Endowment* 13, 6 ([n. d.]).

[67] Vikas Hassija, Vinay Chamola, Guangjie Han, Joel JPC Rodrigues, and Mohsen Guizani. 2020. DAGIoV: A framework for vehicle to vehicle communication using directed acyclic graph and game theory. *IEEE Transactions on Vehicular Technology* 69, 4 (2020), 4182–4191.

[68] HotStuff. 2019. Libhotstuff: A general-puropse BFT state machine replication library with modularity and simplicity. https://github.com/hot-stuff/libhotstuff.

[69] Ruomu Hou, Haifeng Yu, and Prateek Saxena. 2022. Using throughput-centric byzantine broadcast to tolerate malicious majority in blockchains. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1263–1280.

[70] Earl W Huff Jr, Siobahn Day Grady, and Julian Brinnkley. 2021. Tell Me What I Need To Know: Consumers' Desire for Information Transparency in Self-Driving Vehicles. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 65. SAGE Publications Sage CA: Los Angeles, CA, 327–331.

[71] IBM. 2020. Blockchain-powered autonomous automobiles can be the answer. https://www.ibm.com/blogs/blockchain/2020/04/blockchain-powered-autonomous-automobiles-can-be-the-answer/.

[72] QAD Inc. 2020. Five ways blockchain is changing the automotive industry. https://www.qad.com/blog/2020/03/5-ways-blockchain-is-changing-the-automotive-industry.

[73] Tigang Jiang, Hua Fang, and Honggang Wang. 2018. Blockchain-based internet of vehicles: Distributed network architecture and performance analysis. *IEEE Internet of Things Journal* 6, 3 (2018), 4640–4649.

[74] Håvard Johansen, André Allavena, and Robbert Van Renesse. 2006. Fireflies: scalable support for intrusion-tolerant network overlays. *ACM SIGOPS Operating Systems Review* 40, 4 (2006), 3–13.

[75] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. 2012. CheapBFT: Resource-efficient Byzantine fault tolerance. In *Proceedings of the 7th ACM european conference on Computer Systems*. 295–308.

[76] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2007. Zyzzyva: speculative Byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. 45–58.

[77] Toyota Blockchain Lab. 2020. Accelerating Blockchain Technology Initiatives and External Collaboration. https://global.toyota/en/newsroom/corporate/31827481.html.

[78] Leslie Lamport. 2011. Brief announcement: Leaderless Byzantine paxos. In *International Symposium on Distributed Computing*. Springer, 141–142.

[79] Leslie Lamport and Michael Fischer. 1982. *Byzantine generals and transaction commit protocols*. Technical Report. Technical Report 62, SRI International.

[80] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. 2010. Reconfiguring a state machine. *ACM SIGACT News* 41, 1 (2010), 63–73.

[81] Leslie Lamport, Robert Shostak, and Marshall Pease. 2019. The Byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*. 203–226.

[82] Dave Levin, John R Douceur, Jacob R Lorch, and Thomas Moscibroda. 2009. TrInc: Small Trusted Hardware for Large Distributed Systems.. In *NSDI*, Vol. 9. 1–14.

[83] Michael Lewis. 2014. *Flash boys: a Wall Street revolt*. WW Norton & Company.

[84] Benoît Libert, Marc Joye, and Moti Yung. 2016. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theoretical Computer Science* 645 (2016), 1–24.

[85] Hazel Si Min Lim and Araz Taeihagh. 2018. Autonomous vehicles for smart and sustainable cities: An in-depth exploration of privacy and cybersecurity implications. *Energies* 11, 5 (2018), 1062.

[86] LimeChain. [n.d.]. Automotive industry problems today. https://limechain.tech/blockchain-use-cases/blockchain-automotive-industry/.

[87] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolić. 2016. XFT: Practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 485–500.

[88] David Lopez and Bilal Farooq. 2020. A multi-layered blockchain framework for smart mobility data-markets. *Transportation Research Part C: Emerging Technologies* 111 (2020), 588–615.

[89] Jacob R Lorch, Atul Adya, William J Bolosky, Ronnie Chaiken, John R Douceur, and Jon Howell. 2006. The SMART way to migrate replicated stateful services. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*. 103–115.

[90] Dahlia Malkhi and Michael Reiter. 1997. Unreliable intrusion detection in distributed computations. In *Proceedings 10th Computer Security Foundations Workshop*. IEEE, 116–124.

[91] J-P Martin and Lorenzo Alvisi. 2006. Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing* 3, 3 (2006), 202–215.

[92] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 31–42.

[93] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. 2014. Signature-free asynchronous Byzantine consensus with t< n/3 and O (n2) messages. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*. 2–9.

[94] MystenLabs. [n.d.]. Narwhal and Tusk. https://github.com/MystenLabs/narwhal.

[95] Satoshi Nakamoto. 2019. *Bitcoin: A peer-to-peer electronic cash system*. Technical Report. Manubot.

[96] Ray Neiheiser, Miguel Matos, and Luís Rodrigues. 2021. Kauri: Scalable BFT consensus with pipelined tree-based dissemination and aggregation. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 35–48.

[97] Joachim Neu, Ertem Nusret Tas, and David Tse. 2021. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 446–465.

[98] National Conference of State Legislatures. 2020. Self-Driving Vehicles Enacted Legislation. https://www.ncsl.org/research/transportation/autonomous-vehicles-self-driving-vehicles-enacted-legislation.aspx.

[99] Jo-Ann Pattinson, Haibo Chen, and Subhajit Basu. 2020. Legal issues in automated vehicles: critically considering the potential role of consent and interactive digital interfaces. *Humanities and Social Sciences Communications* 7, 1 (2020), 1–10.

[100] Li Peng, Wei Feng, Zheng Yan, Yafeng Li, Xiaokang Zhou, and Shohei Shimizu. 2021. Privacy preservation in permissionless blockchain: A survey. *Digital Communications and Networks* 7, 3 (2021), 295–307.

[101] The Washington Post. 2022. Teslas running Autopilot involved in 273 crashes reported since last year. https://www.washingtonpost.com/technology/2022/06/15/tesla-autopilot-crashes/.

[102] Ji Qi, Xusheng Chen, Yunpeng Jiang, Jianyu Jiang, Tianxiang Shen, Shixiong Zhao, Sen Wang, Gong Zhang, Li Chen, Man Ho Au, et al. 2021. Bidl: A high-throughput, low-latency permissioned blockchain framework for datacenter networks. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating

Systems Principles. 18–34.

[103] Qualcomm. 2021. Connecting vehicles to everything with C-V2X. . https://www.qualcomm.com/research/5g/cellular-v2x.

[104] The General Data Protection Regulation. 2022. GDPR: Principles relating to processing of personal data. https://gdpr-info.eu/art-5-gdpr/.

[105] ResilientDB. [n.d.]. ResilientDB: A scalable permissioned blockchain fabric. https://github.com/resilientdb/resilientdb.

[106] Cassandra Burke Robertson. 2023. Litigating Partial Autonomy. *Case Legal Studies Research Paper* 4 (2023), 109.

[107] Rodrigo Rodrigues, Barbara Liskov, Kathryn Chen, Moses Liskov, and David Schultz. 2010. Automatic reconfiguration for large-scale reliable storage systems. *IEEE Transactions on Dependable and Secure Computing* 9, 2 (2010), 145–158.

[108] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Cédric Fournet, Matthew Kerner, Sid Krishna, Julien Maffre, Thomas Moscibroda, Kartik Nayak, Olya Ohrimenko, Felix Schuster, Roy Schwartz, Alex Shamis, Olga Vrousgou, and Christoph M. Wintersteiger. 2019. *CCF: A Framework for Building Confidential Verifiable Replicated Services*. Technical Report MSR-TR-2019-16. Microsoft. https://www.microsoft.com/en-us/research/publication/ccf-a-framework-for-building-confidential-verifiable-replicated-services/

[109] Fred B Schneider. 1984. Byzantine generals in action: Implementing fail-stop processors. *ACM Transactions on Computer Systems (TOCS)* 2, 2 (1984), 145–154.

[110] Fred B Schneider. 1990. The state machine approach: A tutorial. *Fault-tolerant distributed computing* (1990), 18–41.

[111] Alex Shamis, Peter Pietzuch, Burcu Canakci, Miguel Castro, Cédric Fournet, Edward Ashton, Amaury Chamayou, Sylvan Clebsch, Antoine Delignat-Lavaud, Matthew Kerner, et al. 2022. IA-CCF: Individual Accountability for Permissioned Ledgers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 467–491.

[112] Victor Shoup. 2000. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 207–220.

[113] SLATE. 2023. Self-Driving Taxis Are Causing All Kinds of Trouble in San Francisco. https://slate.com/technology/2022/12/san-francisco-waymo-cruise-self-driving-cars-robotaxis.html.

[114] Lorraine Sommerfeld. 2021. Autonomous cars are on their way, and insurance companies aren't ready. https://driving.ca/column/lorraine/autonomous-cars-are-on-their-way-and-insurance-companies-arent-ready.

[115] Yee Jiun Song and Robbert van Renesse. 2008. Bosco: One-step Byzantine asynchronous consensus. In *International Symposium on Distributed Computing*. Springer, 438–450.

[116] Joao Sousa, Alysson Bessani, and Marko Vukolic. 2018. A Byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 51–58.

[117] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM computer communication review* 31, 4 (2001), 149–160.

[118] Florian Suri-Payer, Matthew Burke, Zheng Wang, Yunhao Zhang, Lorenzo Alvisi, and Natacha Crooks. 2021. Basil: Breaking up BFT with ACID (transactions). In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 1–17.

[119] Tesla. [n.d.]. Autopilot. https://www.tesla.com/autopilot.

[120] The New York Times. 2023. Driverless Car Gets Stuck in Wet Concrete in San Francisco. https://www.nytimes.com/2023/08/17/us/driverless-car-accident-sf.html.

[121] Robin Vassantlal, Eduardo Alchieri, Bernardo Ferreira, and Alysson Bessani. 2022. COBRA: Dynamic Proactive Secret Sharing for Confidential BFT Services. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 1528–1528.

[122] The Verge. 2023. Tesla points to 'insider wrongdoing' as cause of massive employee data leak. https://www.theverge.com/2023/8/21/23839940/tesla-data-leak-inside-job-handelsblatt.

[123] Peter Virk. 2021. Connected cars require data transparency. https://www.autonews.com/commentary/connected-cars-require-data-transparency.

[124] Volkswagen. 2019. Protecting people and the environment with blockchain. https://www.volkswagenag.com/en/news/stories/2019/04/protecting-people-and-the-environment-with-blockchain.html#.

[125] Yuning Wang, Zeyu Han, Yining Xing, Shaobing Xu, and Jianqiang Wang. 2023. A Survey on Datasets for Decision-making of Autonomous Vehicle. *arXiv preprint arXiv:2306.16784* (2023).

[126] Zhuolun Xiang, Dahlia Malkhi, Kartik Nayak, and Ling Ren. 2021. Strengthened fault tolerance in Byzantine fault tolerant replication. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 205–215.

[127] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. 2021. DispersedLedger: High-Throughput Byzantine Consensus on Variable Bandwidth Networks. *arXiv preprint arXiv:2110.04371* (2021).

[128] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM symposium on principles of distributed computing*. 347–356.

[129] Yong Yuan and Fei-Yue Wang. 2016. Towards blockchain-based intelligent transportation systems. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)*. IEEE, 2663–2668.

[130] Gengrui Zhang and Hans-Arno Jacobsen. 2021. Prosecutor: An efficient BFT consensus algorithm with behavior-aware penalization against Byzantine attacks. In *Proceedings of the 22nd International Middleware Conference*. 52–63.

[131] Yunhao Zhang, Srinath T. V. Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. 2020. Byzantine Ordered Consensus without Byzantine Oligarchy. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 633–649.

A GOSSIPING

The gossiping module strengthens system robustness by further disseminating committed transactions to the network. This module does not affect the correctness of consensus and can be applied based on preferences. When gossiping is enabled, after the consensus module commits a transaction, the proposer sends a `Gossip` message piggybacking the transaction to other connected vehicles, which will keep propagating this message to their connected vehicles. Therefore, the transmission forms a *propagation tree* where the root node is the *proposer* and other nodes are *propagators*.

Each `Gossip` message has a *lifetime* (denoted by λ) that determines the number of propagators it traverses in a transmission link; i.e., the height of the propagation tree. For example, in Figure 11, V_1 is the proposer and has direct connections with V_2, V_3, and V_4. The gossip message has a lifetime of 2, and its propagation stops when a path has included 2 propagators; e.g., path <V_1, V_2, V_5>. Specifically, we describe the gossiping workflow as follows.

Proposer: The proposer (V_p) starts to disseminate committed transactions by taking the following actions.

(1) V_p creates a lifetime (e.g., $\lambda = 3$) for a `Commit` message and a set $\mathcal{G}$ for recording the traverse information of the message's transmission path.

(2) V_p hashes the `Commit` message and signs the combination of the hash result (h_c) and λ to obtain a signature σ_{V_p}. Then, it adds a *traverse entry* containing λ, σ_{V_p}, and C_{V_p} to $\mathcal{G}$; i.e., $\mathcal{G}.add([\lambda, \sigma_{V_p}, C_{V_p}])$, where C_{V_p} is V_p's address and public key.

(3) V_p sends $\langle$`Gossip`, $\langle$`Commit`$\rangle$, $h_c, tx, \mathcal{G}\rangle$ to connected vehicles that are not included in the consensus process of the `Commit` message and then creates a list for receiving `ack` messages from propagators.

Propagator: A propagator (V_i) stores a valid `Gossip` message and may further disseminate the message if there is lifetime remaining.

(1) V_i verifies a received `Gossip` message by four criteria: ① it has not previously received this message; ② signatures in $\mathcal{G}$ are valid; ③ λ in $\mathcal{G}$ is strictly monotonically decreasing; and ④ the message still has remaining lifetime; i.e., $\lambda_{min} = min\{\mathcal{G}.\lambda\} > 0$. If the verification succeeds, V_i sends an `ack` message to this message's proposer and the pivot validator, registering itself on the propagator list.

(2) V_i decrements this gossip message's lifetime; i.e., $\lambda_{new} = \max\{\lambda_{min} - 1, 0\}$. If $\lambda_{new} > 0$, then V_i prepares to further disseminate the message. It signs the combination of h_c and λ_{new} to obtain a signature σ_{V_i} and adds a new traverse entry into $\mathcal{G}$; i.e., $\mathcal{G}.add([\lambda_{new}, \sigma_{V_i}, V_i])$.

(3) V_i sends $\langle$`Gossip`, $\langle$`Commit`$\rangle$, $h_c, tx, \mathcal{G}\rangle$ to other connected vehicles, excluding those it receives this `Gossip` message from (in (1)).

The gossip module in V-Guard is an additional mechanism that provides an additional layer of redundancy. Since each propagator is registered to the proposer, the proposer can connect to propagators and read its own portion of data. This is especially important when consensus is conducted in small booth sizes, as the failure of vehicles can significantly affect the system's overall resilience. Although gossiping does not guarantee the delivery of all transactions to all participants in the network, it can help to increase the probability of transaction delivery even in the face of network partitions or failures, strengthening the system's robustness without interfering with the core consensus process.

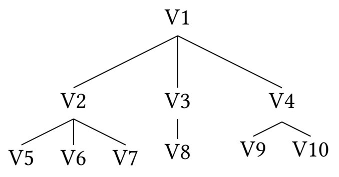


Figure 11: A propagation tree of a gossip message with $\lambda = 2$. V_1 is the proposer and others are propagators.

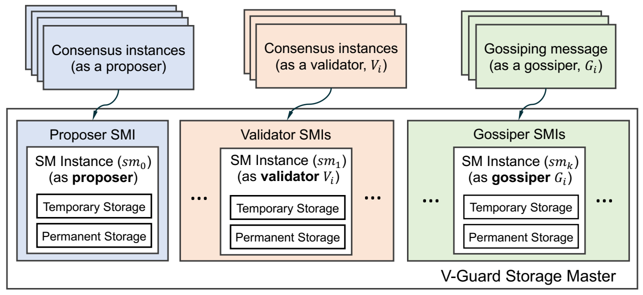
B STORAGE

V-Guard stores committed data batches in a *storage master*. The storage master creates storage master instances (SMIs) for transactions from different vehicles, as a vehicle may operate in different roles in different booths (i.e., $\gamma > 1$). For example, in Figure 12, the storage master has three instance zones containing proposer, validator, and gossiper SMIs (if the gossiping module is applied). A vehicle's storage master has at most one proposer SMI because it is the only proposer of its own V-Guard instance. When the vehicle joins other booths (operating as a validator), its storage master creates validator SMIs. The number of proposer and validator SMIs equals the vehicle's catering factor (γ). When gossiping is enabled, a vehicle may receive gossiping messages, and its storage master creates gossiper SMIs storing consensus results disseminated from other vehicles.

Each storage master instance has two layers: temporary and permanent, with the purpose of maximizing the usage of vehicles' limited storage space. Unlike the blockchain platforms working on servers (e.g., HyperLedger Fabric [8], CCF [108], and Diem [40]), V-Guard operates among vehicles, which usually have only limited storage capability.

The temporary storage temporarily stores transactions based on a predefined policy. V-Guard's implementation uses a timing policy that defines a time period (denoted by τ) that the storage master stores a transaction (e.g., τ = 24 hours). A transaction is registered in the temporary storage by default and is deleted after τ time if the user issues no further command. In addition, temporarily stored transactions can be moved to permanent storage and kept permanently per user request. This option hands over the control of storage to users. For example, when users suspect malfunctions of their vehicles, they may want to keep related transactions as evidence and move them to permanent storage. We introduce four SMI APIs for the layered design.

- `sm.RegisterToTemp(&tx, time.Now())` registers a transaction to the temporary storage layer.
- `sm.CleanUpTemp(time.Now())` is a daemon process that calculates the remaining time for transactions in the temporary storage and deletes expired ones.
- `sm.MoveToPerm(&tx)` is called by users, moving selected transactions from temporary to permanent storage.
- `sm.DeletePerm(&tx)` is called by users. A user may delete a permanently stored transaction after it has served the user's purpose.



Consensus instances (as a proposer)
Consensus instances (as a validator, V_i)
Gossiping message (as a gossiper, G_i)
Proposer SMI
SM Instance (sm_0) (as **proposer**)
Temporary Storage
Permanent Storage
Validator SMIs
SM Instance (sm_1) (as **validator** V_i)
Temporary Storage
Permanent Storage
Gossiper SMIs
SM Instance (sm_k) (as **gossiper** G_i)
Temporary Storage
Permanent Storage
V-Guard Storage Master

Figure 12: Storage master instances (SMIs). A vehicle operates a proposer SMI when it conducts consensus for itself, validator SMIs when it participates in other vehicles' consensus, and gossiper SMIs when it enables the gossip module.

The policy of temporary storage can be implemented differently. For example, a policy that defines a fixed size of storage space may apply. When the temporary storage exceeds the predefined size, it clears up old transactions in a first-in-first-out (FIFO) manner.

The policy-based storage module can significantly reduce the use of storage space as vehicles often have limited storage capability. Note that the pivot validator can permanently store all data as automobile manufacturers often operate on their cloud platforms with scalable storage devices.