



SILENT DATA CORRUPTION DETECTION IN DISAGGREGATED NVME STORAGE USING END-TO-END INTEGRITY TELEMETRY

Gagan Vyas
Member of technical staff
Pure storage
Santa clara California, United States
gaganvyas@gmail.com

Abstract

When flash storage is disaggregated from compute and accessed via NVMe over Fabrics (NVMe-oF), the threat of silent data corruption (SDC) increases because no mechanism signals an error. To assess the ability of end-to-end integrity telemetry to detect and localize SDC, a systematic review was performed to collect evidence from the literature on field studies, device physics, integrity codes, and disaggregation architectures. Sources were categorized into four themes and combined with analytical models of the probability of an undetected error, verification overhead, and tail-latency amplification. The evidence shows that latent sector errors affected 8.5% of nearline disks, that checksum mismatches occurred on 0.86% of nearline disks, and that more than 8% of memory modules were affected by errors each year. A 32-bit checksum can lower the probability of a random undetected error to approximately 2.33 in 10 billion, at the cost of 0.098% metadata overhead for 4 KiB blocks. A layered verification architecture comprising five telemetry taps and a mismatch-attribution workflow is proposed. The synthesis identifies end-to-end verification at the host endpoint, together with layer-attached counters, as the approach that provides the most comprehensive coverage at low overhead.

Index Terms—Silent data corruption, disaggregated storage, NVMe over Fabrics, end-to-end integrity, integrity telemetry, checksums, cyclic redundancy check, flash reliability, tail latency, systematic review

I. INTRODUCTION

Silent data corruption (SDC) is the corruption of data or configuration without any error signal being returned or recorded by the system, causing downstream components to return or store incorrect data. Studies of large production fleets have revealed that such failures are not confined to rare, exotic failure modes; they occur in ordinary server infrastructure at scale and impair applications [1]. The fault produces no exception, log entry, or alert, so it must be detected through independent verification of results. The problem is more serious for storage, because corrupted data can remain undetected for a long time until the first read reveals it.



Storage media are not the only source of this exposure; compute elements are too. A few cores per several thousand machines have been reported to be defective, or “mercurial”, executing instructions incorrectly only under special circumstances and without any warning of failure [2]. If a processor computes a checksum, parity block, or compression result incorrectly, it can generate corruption that every subsequent protective layer accepts as valid. Verification mechanisms should therefore be independent, where possible, of the hardware used to generate the data.

Flash disaggregation is the separation of solid-state drives (SSDs) from compute servers so that capacity and throughput can be provisioned independently and utilization can be improved [3]. Remote access then traverses additional components that can corrupt payloads, such as network interface cards (NICs), switches, target-side processors, and buffers. Non-Volatile Memory Express (NVMe) over Fabrics (NVMe-oF) extends the NVMe command set over such networks. The goal of this review is to combine the evidence on the causes of corruption and the available detection methods, and to design a telemetry architecture that can localize SDC across the disaggregated data path.

II. LITERATURE REVIEW

The end-to-end argument holds that functions such as error detection cannot be provided fully and correctly until they are implemented at the communicating endpoints, with the lower layers contributing only performance optimizations; this argument forms the theoretical background for integrity verification in layered systems [4]. Applied to storage, the argument suggests that per-layer protections, however comprehensive, are not a substitute for checking between the application that produces the data and the application that reads it. The reviewed literature is presented below, organized by compute and memory errors, storage media and stack failures, and integrity mechanisms with disaggregated access.

A.Silent Errors in Compute and Memory

Baseline memory error rates have been obtained at scale from field measurements. A multi-year study of production servers reported dynamic random-access memory (DRAM) error rates of 25,000 to 70,000 failures in time (FIT) per megabit, with more than 8 percent of memory modules affected by errors in a single year [5]. These rates are many orders of magnitude higher than those proposed by earlier laboratory investigations. Errors were also strongly correlated within a module, suggesting that a faulty component tends to cause repeated errors rather than isolated ones.

A subsequent study of supercomputer deployments analyzed fault modes and the effectiveness of protection schemes. The results indicated that permanent faults dominated DRAM failures and that stronger error-correcting codes (ECC) reduced, but did not completely eliminate, uncorrected errors; performance varied with vendor and device design [6]. The study also showed that memory behavior differed with location and operating conditions. Such heterogeneity suggests that population-level averages are inadequate for risk estimation and that continuous per-device counters are preferable.



A large population of cloud processors was directly characterized for silent errors. A few processors per ten thousand were identified as defective, and susceptibility to faults was found to vary with the type of instruction and the operating conditions [7]. The faults were not apparent during normal use and were revealed only by dedicated testing. These observations confirm that computation can produce corrupt data and that the correctness of checksum generation cannot be assumed.

Detection strategies for processors rely on dedicated testing in production deployments. One approach performs periodic in-field tests to expose faulty devices, balancing the capacity used for testing against test coverage [8]. In this approach, detection is a statistical process, because a single pass does not guarantee that a faulty device will be exposed. The same holds for storage verification, where the scrub frequency and coverage define the window of undetected exposure.

B. Storage Media and Storage Stack Failures

A disk-level study quantified the unreadable sectors that were discovered only when they were accessed, known as latent sector errors (LSEs). Over a period of 32 months, 8.5 percent of nearline disks and 1.9 percent of enterprise disks experienced at least one LSE [9]. Errors exhibited both spatial and temporal locality: once an error occurred, the chance of another error nearby in location or time increased. Latent errors become dangerous if they are discovered during reconstruction, when the redundancy may already have been consumed. Proactive scanning was therefore identified as a key mitigation.

In a related study of the storage stack, corruption that is returned without any error status was investigated. Across 1.53 million disks observed for 41 months, more than 400,000 checksum mismatches were found, affecting 0.86% of the nearline disks and 0.065% of the enterprise disks [10]. Mismatches were grouped by type, including identity mismatches caused by misdirected writes, and were found to be concentrated on a few drives. The findings show that corruption in the storage stack is measurable and would remain undetectable without block-level checksums.

The response of file systems to partial disk failures, such as block failures and corruption, was also evaluated. Fault injection into a widely used file system showed inconsistent fault-detection and recovery policies, and an internal-robustness scheme (IRON), combining checksums, redundancy, and retry, was proposed [11]. The analysis introduced the concept of fail-partial behavior, a failure mode in which a disk fails in part but continues to operate. Integrity telemetry has to detect this behavior, because whole-device failure is easily observed.

A comprehensive study of error characterization at the flash level identified the main sources of raw bit errors in NAND flash (negative-AND flash, the most prevalent solid-state medium): program and erase cycling, cell-to-cell program interference, data retention loss, and read disturbance [12]. Controllers use ECC and error mitigation to mask these errors from the host. If the number of errors exceeds what the ECC can correct, the error is identified by a higher layer, which makes correction at the media level the first line of defense.

C. Disaggregated Access and Integrity Codes

Characterization of NVMe-oF established that remote access to flash adds at most about 10 microseconds of latency compared with local access, and that the protocol is more efficient than



earlier network storage protocols, which makes disaggregation feasible for latency-sensitive workloads [13]. Per-request processing costs at the initiator and the target were found to determine the achievable throughput. Any integrity function imposed on the target must therefore compete with data movement for the same resources, so the location of verification is a design parameter.

The capabilities of error-detection codes vary considerably. A study of cyclic redundancy check (CRC) polynomials showed that detection performance depends on the choice of polynomial and on the message length (characterized by Hamming distance), and that some polynomials used in popular standards are outperformed by other polynomials for specific message lengths [14]. For a CRC with a generator polynomial $G(x)$ of degree r , the remainder of the polynomial division is computed as in Equation 1, where $M(x)$ is the message polynomial.

$$C(x) = [M(x) \cdot x^r] \bmod G(x) \quad (1)$$

III. METHODS

A fixed sample of 27 sources, including peer-reviewed journal articles, conference papers, and preprints, was used to conduct a systematic review covering memory, processors, storage, networks, and disaggregated flash. Each source was coded for the corruption mechanism, the population measured, the detection method, and the overhead reported. The coding is similar to the approach used in resilience surveys that combine detection and recovery mechanisms throughout the system stack [15]. Four themes emerged and are shown in Table I; the distribution of sources across publication periods is shown in Fig. 1.

TABLE I. LITERATURE-TO-CONCEPT MAPPING OF THE REVIEW CORPUS

Theme	Sources	n	Span	Core concepts
Compute and memory errors	[1], [2], [5]–[8], [15], [16]	8	2009–2023	DRAM field error rates; defective cores; in-field testing; redundancy-based detection; exascale resilience
Storage media and stack	[9]–[12], [17], [18], [20], [22], [26], [27]	10	1988–2017	Latent sector errors; checksum mismatches; fail-partial behavior; NAND error mechanisms; SSD field failures; parity repair
Codes and principles	[4], [14], [19], [21]	4	1984–2009	End-to-end argument; CRC polynomials; checksum effectiveness; CRC and transport checksum disagreement
Disaggregation and tail latency	[3], [13], [23]–[25]	5	2013–2021	Flash disaggregation; NVMe-oF overhead; remote-flash dataplane; SmartNIC multi-tenancy; tail amplification
Total	[1]–[27]	27	1984–2023	

Source: Coded from the reviewed sources; bracketed numbers refer to the reference list.

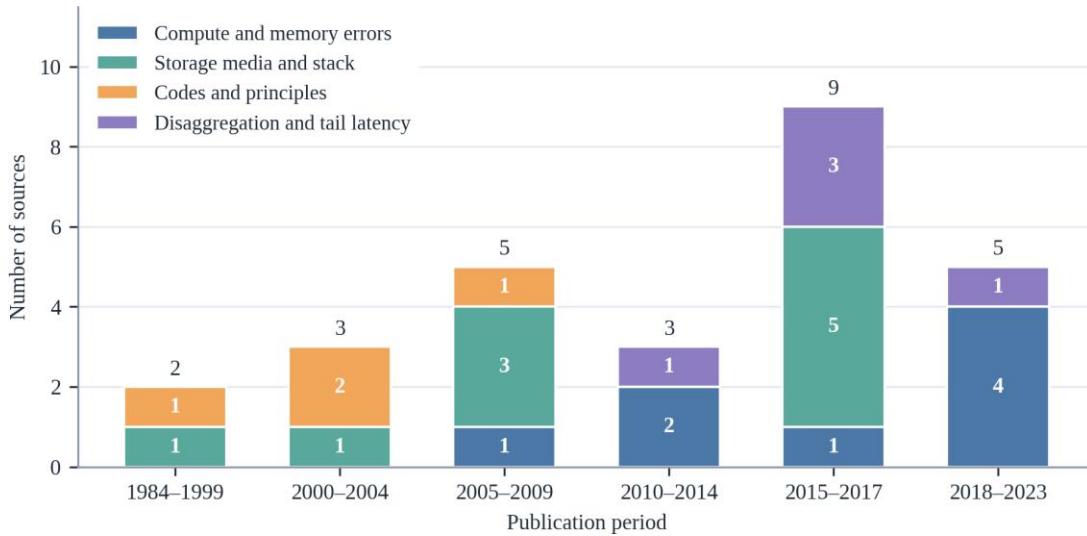


Fig. 1. Distribution of reviewed sources by publication period and theme.

Synthesis was conducted using analytical models rather than pooled statistics, because the populations and measurements in the sources differ. A coverage-based formulation was motivated by the observation, for message-passing applications, of redundant executions of the same code [16]. To treat the verification layers as independent checks, each layer i is assumed to detect corruption with probability c_i , which yields Equation 2, where L is the number of layers and P_{esc} is the probability that corruption escapes all layers. The result is an optimistic bound, because shared hardware can couple failures and independence is an idealization.

$$P_{\text{esc}} = \prod_{i=1}^L (1 - c_i) \quad (2)$$

IV. RESULTS AND DISCUSSION

A. Field Evidence on Corruption Frequency

Reliability testing for NAND-based SSDs is concerned with the raw bit error rate (RBER) and retention, and the results are expressed as the uncorrectable bit error rate (UBER) after ECC [17]. Table II summarizes the quantitative field evidence collected in this review for media, memory, processors, networks, and fabrics. The reported figures vary by more than two orders of magnitude, but none is negligible at fleet scale: when thousands of devices run side by side, even an event that is rare per device becomes a common occurrence.



TABLE II. QUANTITATIVE FIELD EVIDENCE ON CORRUPTION AND FAILURE FREQUENCY

Phenomenon	Population or setting	Reported measure	Source
Latent sector errors, nearline disks	1.53 million disks, 32 months	8.5% of disks	[9]
Latent sector errors, enterprise disks	1.53 million disks, 32 months	1.9% of disks	[9]
Checksum mismatches, nearline disks	1.53 million disks, 41 months	0.86% of disks	[10]
Checksum mismatches, enterprise disks	1.53 million disks, 41 months	0.065% of disks	[10]
Total checksum mismatches	1.53 million disks, 41 months	More than 400,000 events	[10]
DRAM error rate	Production servers	25,000–70,000 FIT per Mbit	[5]
DRAM modules affected	Production servers	More than 8% per year	[5]
Mercurial processor cores	Production fleet	A few per several thousand machines	[2]
Silent-error processors	Large cloud processor population	Order of a few per ten thousand	[7]
Transport checksum failure after link CRC pass	Real network traffic	About 1 in 1,100 to 1 in 32,000 packets	[19]
Block checksum protection	Distributed file system	32 bits per 64 KB block	[20]
Remote flash dataplane throughput	Per processor core	About 850,000 IOPS	[24]

Source: Values as reported in the cited studies; ranges and rounding follow the original reports. FIT: failures per billion device hours.

The behavior of flash in the field differs from what is expected in the laboratory. A large study of production SSDs found that the failure rate did not increase monotonically with wear exposure, that temperature affected the failure rate, and that read disturbance was not a significant problem in the field [18]. Fig. 2 shows the prevalence of the two error types (latent errors and checksum mismatches) at the disk level for different drive classes. Nearline drives showed a prevalence approximately 4.5 times (latent errors) and 13 times (checksum mismatches) that of enterprise drives, respectively, indicating that device class is an important modulator of exposure.



Fig. 2. Disk-level prevalence of latent errors and checksum mismatches.

B. Detection Layers and Verification Architecture

The integrity of one layer does not guarantee the integrity of the next. Experiments with real traffic showed that, for between about 1 in 1,100 and 1 in 32,000 packets, the transport-layer checksum failed even though the link-level CRC passed, indicating corruption in hosts, routers, and buffers rather than on the wire [19]. Table III summarizes the detection layers that are relevant to disaggregated NVMe, the mechanism used, and the principal strength and blind spot of each.

TABLE III. DETECTION LAYERS IN A DISAGGREGATED NVME DATA PATH

Layer	Mechanism	Principal strength	Principal blind spot	Source
NAND media and controller	ECC with mitigation	Raw bit errors from wear, retention, and interference	Errors beyond correction capability	[12], [22]
Link and transport	Link CRC; transport checksum	Wire-level bit errors	Corruption in hosts, routers, and buffers after link verification	[19]
Storage stack	Per-block checksum verified on read	Media errors; misdirected writes	Corruption introduced before checksum generation; latent until read	[10], [20]
Redundancy and scrubbing	Parity; replicas; idle scans	Repair of identified blocks; early discovery of latent errors	Faulty copy cannot be identified without a checksum	[9], [27]
Redundant computation	Comparison of replicated executions	Compute-induced corruption	Capacity cost; limited to replicable work	[16]
End-to-end verification	Tag generated at source, verified at destination	Corruption at any intermediate layer	Residual risk if tag generation is itself faulty	[4], [7]

Source: Synthesized from the reviewed sources; bracketed numbers refer to the reference list.



A common pattern for storage-stack protection is block-level checksumming. In a large distributed file system, each 64 KB block is protected by a 32-bit checksum that is verified before data are returned to a requester, and rarely used data are scanned during idle periods [20]. The metadata cost of this scheme is the ratio ρ of the checksum size s_c to the block size s_b , as given in Equation 3; for the sizes listed, it is approximately 0.006%.

$$\rho = \frac{s_c}{s_b} \quad (3)$$

Fig. 3 shows the proposed layered architecture, in which verification points and telemetry taps are placed along the NVMe-oF data path.

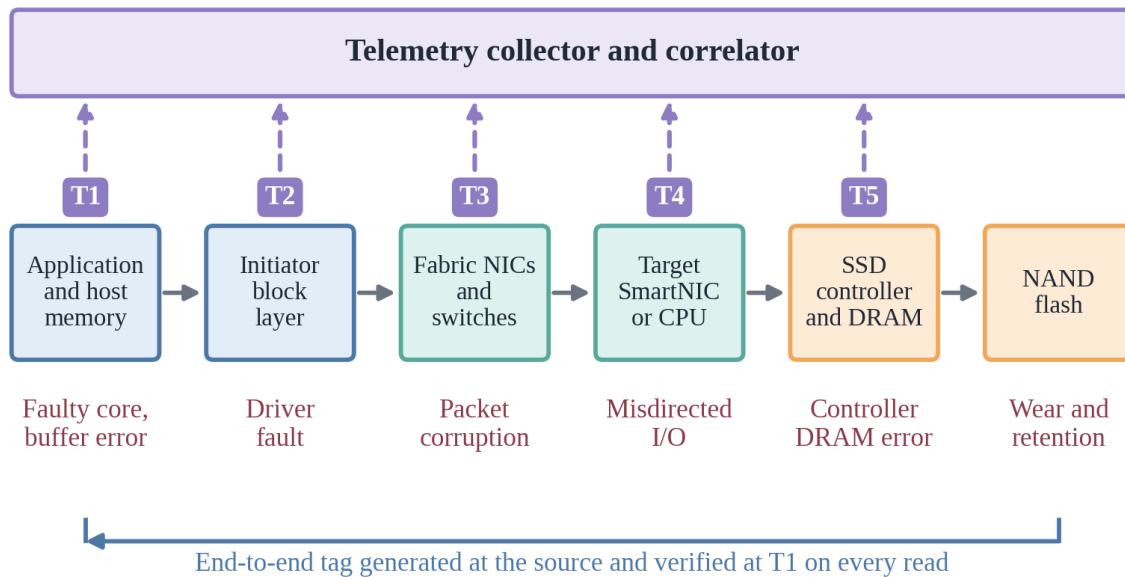


Fig. 3. Layered NVMe-oF data path with verification points and telemetry taps.

C. Selection of Integrity Codes

For short messages, an evaluation of checksum families found that CRC is usually more effective at detecting burst and multi-bit errors, at a higher software implementation cost [21]. If errors occur independently of the structure of the code, that is, if there is no correlation between the errors and the payload, the probability that an error goes undetected approaches the probability that a random value matches the r -bit check value, as given in Equation 4, where r is the checksum width in bits. The resulting probabilities and metadata overheads for common widths are shown in Table IV.

$$P_{ud} \approx 2^{-r} \quad (4)$$



TABLE IV. UNDETECTED-ERROR PROBABILITY AND METADATA OVERHEAD BY CHECKSUM WIDTH

Checksum width r (bits)	Undetected probability	Overhead, 4 KiB block (%)	Overhead, 64 KiB block (%)
8	3.91×10^{-3}	0.024	0.002
16	1.53×10^{-5}	0.049	0.003
32	2.33×10^{-10}	0.098	0.006
64	5.42×10^{-20}	0.195	0.012
128	2.94×10^{-39}	0.391	0.024

Source: Computed from Equations 3 and 4 under a random-error model, with $s_c = r/8$ bytes; code-specific guarantees such as Hamming distance are not represented.

Detection latency, that is, the time during which errors remain undetected, matters as much as code strength. A characterization of multi-level cell NAND found that the major source of error was retention loss, which increased with program and erase wear, and this led to NAND recovery techniques such as retention-aware read-voltage adjustment [22]. If errors occur uniformly in time at a rate λ (errors per unit time) and all blocks are scrubbed every τ , the expected exposure to errors before discovery is $\tau/2$, and Equation 5 gives the expected number of latent errors, N_{lat} . The undetected-error probability and the metadata overhead are plotted against the checksum width in Fig. 4.

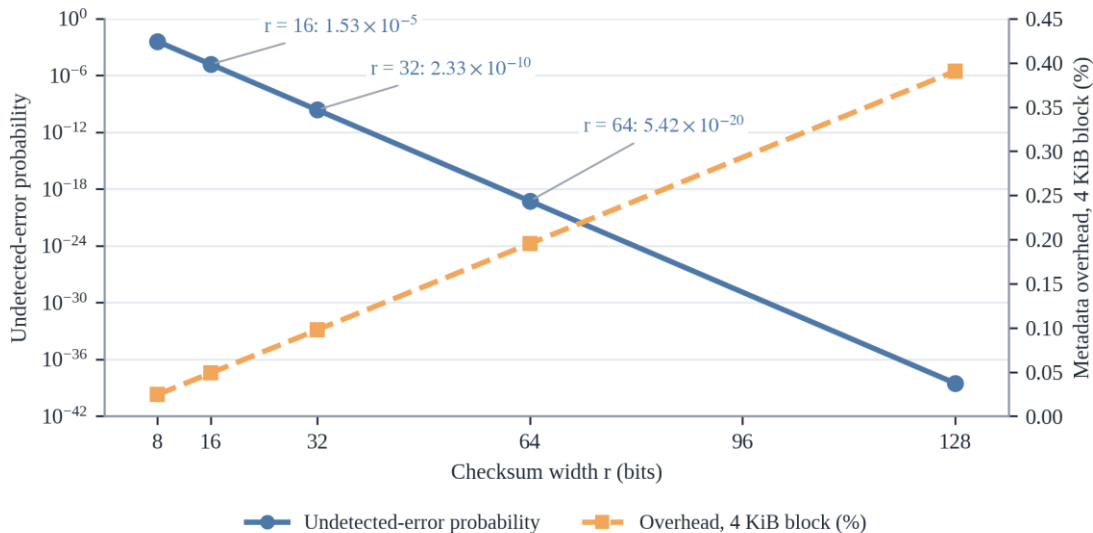


Fig. 4. Undetected-error probability and metadata overhead versus checksum width.

D. Tail Latency and Verification Overhead

Verification increases the amount of work for each request, and its effect on response-time percentiles needs to be measured. If a request fans out into n components, each of which is slow



with probability p , then Equation 6 gives the probability that at least one of the n components is slow. For $p = 0.01$ and 100 components, this probability is approximately 63% in large-scale services [23]. Table V and Fig. 5 quantify the amplification for several fan-outs, which means that verification should not add any delay, even an infrequent one, to the critical path.

$$P_{\text{tail}} = 1 - (1 - p)^n \quad (6)$$

TABLE V. TAIL-LATENCY AMPLIFICATION BY FAN-OUT

Fan-out n	$p = 0.001$ (%)	$p = 0.005$ (%)	$p = 0.01$ (%)
1	0.10	0.50	1.00
10	1.00	4.89	9.56
50	4.88	22.17	39.50
100	9.52	39.42	63.40
200	18.14	63.30	86.60

Source: Computed from Equation 6; probabilities denote requests with at least one slow component.

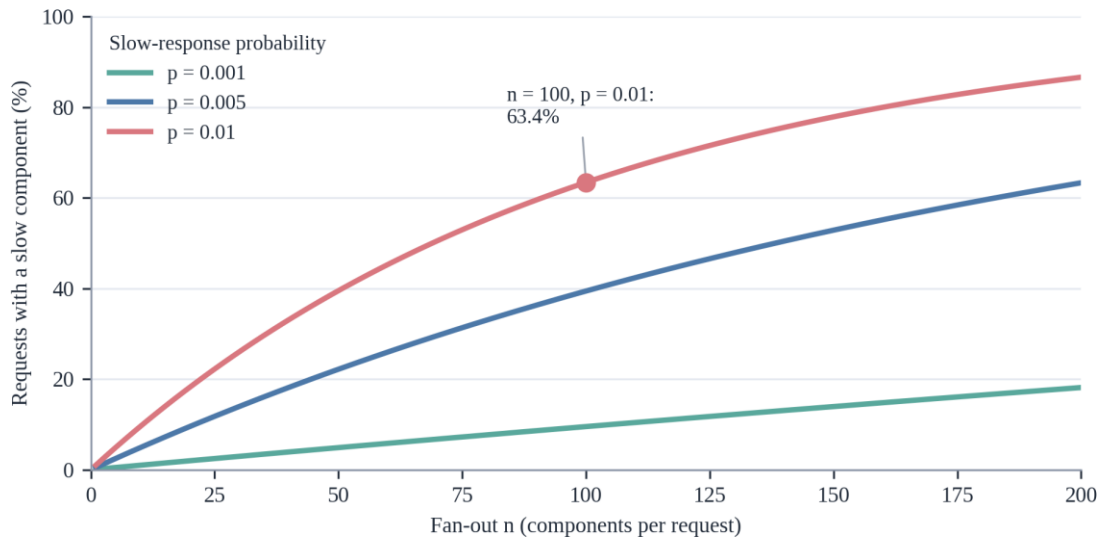


Fig. 5. Probability of a slow component versus request fan-out.

Remote flash access can approximate local performance with a lean software path. A dataplane for remote flash was reported to deliver approximately 850,000 input/output operations per second (IOPS) per core, in comparison with standard networking, while supporting tenant service-level objectives [24]. The end-to-end latency can be modeled by Equation 7, in which L_{local} is the device latency, L_{net} is the network and protocol latency, and the verification term is the block size s_b divided by the checksum throughput B . For a block size of 4 KiB and an assumed throughput of 4 GB/s, the verification term is approximately 1 microsecond.



$$L_{e2e} = L_{\text{local}} + L_{\text{net}} + \frac{s_b}{B} \quad (7)$$

With multi-tenant disaggregation, contention arises at the storage target. A design for flash arrays on programmable network interface cards (SmartNICs) provided explicit scheduling of target resources under mixed workloads, addressing per-tenant fairness and latency-isolation requirements [25]. In this way, host cores are not consumed by verification, and the control path remains free for telemetry export. Offload also transfers trust, however: checksums are calculated only at the target, so corruption that occurs before the data reach the target cannot be revealed.

E. Telemetry Design and Fault Attribution

Operational telemetry has been shown to help account for device failures. To study SSD failures, a set of failures from a large datacenter was selected, and the time and type of each failure were determined along with correlated factors such as device age and workload [26]. For integrity monitoring, the most important metric is the verified-read mismatch rate, calculated as in Equation 8, where N_{mis} is the number of verification mismatches and V_{read} is the number of terabytes read. Table VI specifies the proposed metric set, and Fig. 6 shows the attribution workflow that maps a mismatch to a localized fault.

$$R_{\text{mis}} = \frac{N_{\text{mis}}}{V_{\text{read}}} \quad (8)$$

TABLE VI. PROPOSED INTEGRITY TELEMETRY METRIC SET

Metric	Definition	Tap	Purpose	Basis
Verified-read mismatch rate	Failed verifications per terabyte read	T1	Primary indicator of silent corruption	[10], [26]
Layer-attributed mismatch count	Mismatches whose tag first fails at a given tap	T2-T4	Localizes the fault to host, fabric, or target	[19]
Retry recovery ratio	Share of mismatches resolved by re-read	T1	Separates transient from persistent faults	[11]
Scrub-detected latent errors	Errors found by background verification	T4	Measures exposure before first read	[9], [20]
Corrected-bit counters	ECC-corrected errors per read	T5	Tracks wear and retention trends	[18], [22]
Verification latency percentile	Added latency at the 99th percentile	T1-T4	Bounds the tail impact of verification	[23]
Compute probe outcome	Result of periodic tests on hosts computing tags	T1	Detects faulty cores that compute checksums	[8]

Source: Proposed by synthesis of the reviewed sources; the mismatch rate is defined in Equation 8. T1-T5 denote the telemetry taps in Fig. 3.

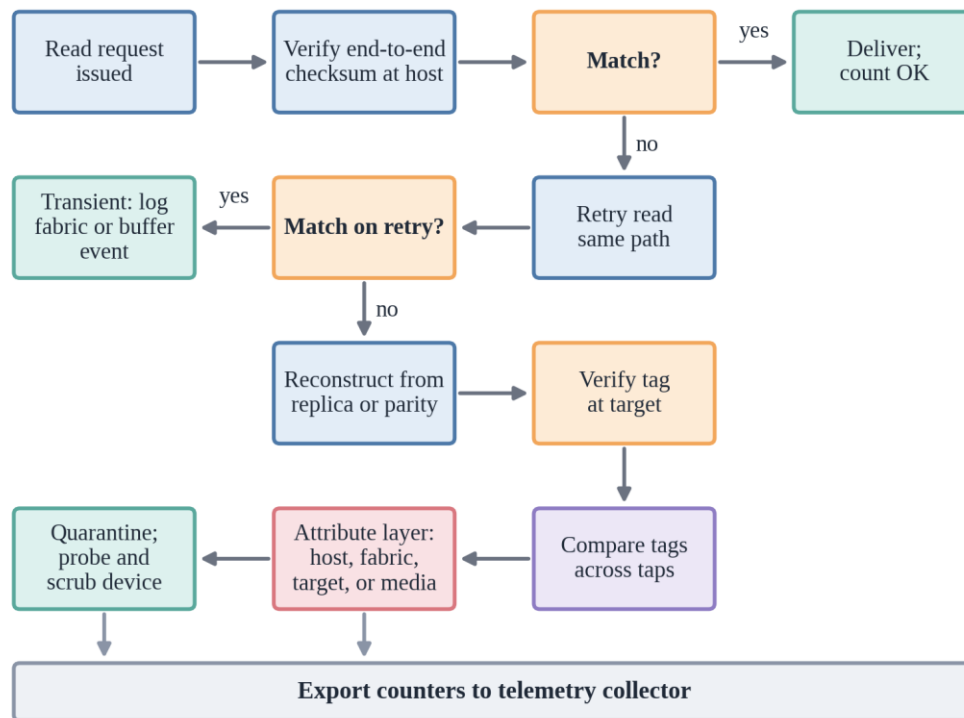


Fig. 6. Mismatch attribution and repair workflow.

Redundancy converts detection into correction. Redundant arrays of inexpensive disks (RAID) showed that lost or corrupted blocks can be rebuilt, using parity or mirroring, once the failing block is discovered [27]. However, parity does not identify which block is bad, so an end-to-end tag with a checksum is necessary to select the correct source. The sequence of telemetry-guided repair is therefore verify, retry, reconstruct, attribute, and quarantine.

V. CONCLUSION

This review compiled 27 sources and concludes that silent corruption is a measurable, regular occurrence that can affect memory, processors, storage media, networks, and disaggregated flash paths. The evidence shows that verification points cannot be placed arbitrarily: a 32-bit checksum adds only 0.098% metadata overhead for 4 KiB blocks and gives a probability of approximately 2.33×10^{-10} of a random undetected error, while tail-latency amplification limits where verification can be placed. At relatively low cost, the most complete coverage is achievable with an end-to-end tag that is verified at the host, together with layer-attributed telemetry counters.

Future work should replace the analytical bounds presented here with measurements from production NVMe-oF deployments, validate the proposed metric set layer by layer, and report observed mismatch rates.



REFERENCES

1. H. D. Dixit *et al.*, "Silent data corruptions at scale," arXiv:2102.11245, 2021, doi: 10.48550/arXiv.2102.11245.
2. P. Hochschild *et al.*, "Cores that don't count," in *Proc. Workshop Hot Topics Oper. Syst. (HotOS '21)*, 2021, pp. 9–16, doi: 10.1145/3458336.3465297.
3. Klimovic, C. Kozyrakis, E. Thereska, B. John, and S. Kumar, "Flash storage disaggregation," in *Proc. 11th Eur. Conf. Comput. Syst. (EuroSys '16)*, 2016, Art. no. 29, doi: 10.1145/2901318.2901337.
4. J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, 1984, doi: 10.1145/357401.357402.
5. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: A large-scale field study," in *Proc. 11th Int. Joint Conf. Meas. Model. Comput. Syst. (SIGMETRICS '09)*, 2009, pp. 193–204, doi: 10.1145/1555349.1555372.
6. Sridharan *et al.*, "Memory errors in modern systems: The good, the bad, and the ugly," in *Proc. 20th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS '15)*, 2015, pp. 297–310, doi: 10.1145/2694344.2694348.
7. S. Wang, G. Zhang, J. Wei, Y. Wang, J. Wu, and Q. Luo, "Understanding silent data corruptions in a large production CPU population," in *Proc. 29th ACM Symp. Oper. Syst. Princ. (SOSP '23)*, 2023, pp. 216–230, doi: 10.1145/3600006.3613149.
8. H. D. Dixit, L. Boyle, G. Vunnam, S. Pendharkar, M. Beadon, and S. Sankar, "Detecting silent data corruptions in the wild," arXiv:2203.08989, 2022, doi: 10.48550/arXiv.2203.08989.
9. L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," in *Proc. 2007 ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2007, pp. 289–300, doi: 10.1145/1254882.1254917.
10. L. N. Bairavasundaram, G. R. Goodson, B. Schroeder, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "An analysis of data corruption in the storage stack," *ACM Trans. Storage*, vol. 4, no. 3, Art. no. 8, 2008, doi: 10.1145/1416944.1416947.
11. V. Prabhakaran, L. N. Bairavasundaram, N. Agrawal, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "IRON file systems," in *Proc. 20th ACM Symp. Oper. Syst. Princ.*, 2005, pp. 206–220, doi: 10.1145/1095810.1095830.
12. Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proc. IEEE*, vol. 105, no. 9, pp. 1666–1704, 2017, doi: 10.1109/JPROC.2017.2713127.
13. Z. Guz, H. Li, A. Shayesteh, and V. Balakrishnan, "NVMe-over-fabrics performance characterization and the path to low-overhead flash disaggregation," in *Proc. 10th ACM Int. Syst. Storage Conf. (SYSTOR '17)*, 2017, doi: 10.1145/3078468.3078483.
14. P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," in *Proc. Int. Conf. Dependable Syst. Netw. (DSN 2004)*, 2004, pp. 145–154, doi: 10.1109/DSN.2004.1311885.
15. F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir, "Toward exascale resilience: 2014 update," *Supercomput. Front. Innov.*, vol. 1, no. 1, pp. 5–28, 2014, doi: 10.14529/jsfi140101.



16. D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC '12)*, 2012, doi: 10.1109/SC.2012.49.
17. N. R. Mielke, R. E. Frickey, I. Kalastirsky, M. Quan, D. Ustinov, and V. J. Vasudevan, "Reliability of solid-state drives based on NAND flash memory," *Proc. IEEE*, vol. 105, no. 9, pp. 1725–1750, 2017, doi: 10.1109/JPROC.2017.2725738.
18. J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," in *Proc. 2015 ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2015, pp. 177–190, doi: 10.1145/2745844.2745848.
19. J. Stone and C. Partridge, "When the CRC and TCP checksum disagree," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM '00)*, 2000, pp. 309–319, doi: 10.1145/347059.347561.
20. S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proc. 19th ACM Symp. Oper. Syst. Princ.*, 2003, pp. 29–43, doi: 10.1145/945445.945450.
21. T. C. Maxino and P. J. Koopman, "The effectiveness of checksums for embedded control networks," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 1, pp. 59–72, 2009, doi: 10.1109/TDSC.2007.70216.
22. Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, "Data retention in MLC NAND flash memory: Characterization, optimization, and recovery," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2015, pp. 551–563, doi: 10.1109/HPCA.2015.7056062.
23. J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013, doi: 10.1145/2408776.2408794.
24. Klimovic, H. Litz, and C. Kozyrakis, "ReFlex: Remote flash $\approx$ local flash," in *Proc. 22nd Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS '17)*, 2017, pp. 345–359, doi: 10.1145/3037697.3037732.
25. J. Min *et al.*, "Gimbal: Enabling multi-tenant storage disaggregation on SmartNIC JBOFs," in *Proc. 2021 ACM SIGCOMM Conf.*, 2021, pp. 106–122, doi: 10.1145/3452296.3472940.
26. Narayanan *et al.*, "SSD failures in datacenters: What? When? And why?," in *Proc. 9th ACM Int. Syst. Storage Conf. (SYSTOR '16)*, 2016, Art. no. 7, doi: 10.1145/2928275.2928278.
27. D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proc. 1988 ACM SIGMOD Int. Conf. Manage. Data*, 1988, pp. 109–116, doi: 10.1145/50202.50214.