ALS SubZero: A Dynamic Delay-Line Network Memory Filesystem with Erasure-Coded Flight Persistence

Operating Volatile Filesystem Storage over In-Flight Network Packet Loops

Samin Sami[1,*]

[1]Independent Systems Researcher and Software Engineer

Correspondence: samin.sami@10x.ai • **Code:** github.com/Saminsami73/als-subzero • **Code License:** MIT License

Abstract

Computers typically store persistent state on local solid-state drives, magnetic platters, or DRAM chips. In this paper, I present **ALS SubZero**, a dynamic delay-line network memory filesystem that stores digital data in continuous motion across IP network paths with zero persistent local disk writes. ALS stands for *Al Sami*, derived from my last name Samin Yasar AL Sami. By translating the historical acoustic mercury delay-line memory concept from EDSAC (1949) into IP packet propagation delay, ALS SubZero achieves volatile, self-healing in-flight storage bounded by the Bandwidth-Delay Product (BDP) of available network links. To solve the fundamental limitations of prior unencoded network bouncing experiments (pingfs), I implemented Cauchy Reed-Solomon erasure coding over Galois Field $GF(2^8)$ combined with a microsecond-precision departure pacer. The pacer eliminates socket buffer exhaustion on sub-millisecond local networks, while an autonomous background scrubber detects dropped chunks and reconstructs them mathematically in flight. Empirical benchmarks demonstrate 100% bit-exact data persistence across 60 packet circulation cycles under simulated packet drop rates up to 10.0%, proving that network flight time can serve as a resilient, self-healing volatile storage medium.

Keywords: delay-line storage; network memory; erasure coding; Cauchy Reed-Solomon; rate pacing; packet circulation; ALS SubZero

1. Introduction

Computers typically store digital information on physical media like solid-state drives, magnetic platters, or dynamic semiconductor memory cells. In early computing, physical storage components were severely limited. Maurice Wilkes and William Renwick built the EDSAC computer using acoustic mercury delay lines [1]. J. Presper Eckert also analyzed delay-line storage systems for the UNIVAC machine [2]. In those early architectures, electronic pulses were converted into acoustic sound waves traveling through tubes of liquid mercury. When each sound wave reached the end of the tube, a quartz crystal converted it back into an electrical signal. An amplifier circuit refreshed the signal and fed it back into the front of the tube. Digital data survived as long as it kept circulating in continuous motion.

Modern computer networks exhibit an analogous propagation property. When packets travel across fiber optic links and through router buffers, they spend measurable flight time in transit. The maximum volume of data that can occupy a network path without link amplification is governed by the Bandwidth-Delay Product (BDP):

$$C = B \times RTT \tag{1}$$

where B is usable bandwidth in bits per second and RTT is network round-trip time in seconds. For instance, a dedicated 1 Gbps link with a 40 ms round-trip latency holds 5.0 MB of data in flight at any given moment.

Prior experimental projects, such as the open-source utility pingfs, demonstrated that ICMP Echo Requests [3] can store raw bytes by reflecting off remote hosts. However, those early experiments suffered from two fatal limitations. First, they had no error correction: when a single packet was dropped by an intermediate router, the file suffered permanent corruption. Second, they lacked rate regulation: on low-latency local networks with round-trip times below 1 ms, unthrottled packets looped too fast, overflowing socket buffers and causing immediate catastrophic data loss.

I built ALS SubZero to solve these two fundamental problems. ALS stands for 'Al Sami', taken from the first three letters of my last name: Samin Yasar AL Sami. In this paper, I present the mathematical foundation, protocol engineering, and empirical verification of the system:

- I implemented Cauchy Reed-Solomon erasure coding over Galois Field $GF(2^8)$ [4], enabling mathematical reconstruction of lost chunks in flight.
- I designed a microsecond-precision departure pacer that schedules frame release, eliminating socket buffer exhaustion on sub-millisecond local networks.
- I created the ALS1 authenticated binary protocol with 24-byte headers and Castagnoli CRC32C checksums [5].
- I developed multi-protocol transport engines supporting unprivileged UDP reflectors [6], raw ICMP sockets [3], and network simulation.
- I conducted empirical benchmarks demonstrating 100% bit-exact recovery under simulated network packet loss rates up to 10.0%.

2. Related Work

2.1 Delay-Line Memory and Acoustic Storage: In early digital computing, acoustic delay lines provided the primary working memory for pioneering machines. Wilkes and Renwick documented the ultrasonic mercury delay storage unit for the EDSAC computer [1]. Eckert surveyed acoustic and cathode-ray tube delay systems, emphasizing that information persistence required continuous signal regeneration [2]. ALS SubZero maps this physical acoustic feedback loop into modern IP networks, substituting quartz transducers and mercury tubes with socket interfaces and fiber-optic cables.

2.2 Volatile and In-Memory Network Storage: High-performance distributed storage systems have long explored memory-only architectures. Ousterhout et al. designed RAMCloud [7], which keeps all operational data permanently in DRAM to achieve microsecond-scale read latencies. However, RAMCloud requires physical server DRAM backed by battery-powered UPS units and secondary disks. ALS SubZero diverges fundamentally: it maintains zero data on server disks or static host buffers, relying entirely on packets in physical transit across the wire.

2.3 Network Packet Bouncing: The pingfs utility first demonstrated that ICMP Echo Requests could bounce bytes off internet hosts. However, pingfs was an unencoded proof of concept: packet drops caused immediate file destruction. ALS SubZero introduces systematic erasure coding, deterministic departure pacing, and an autonomous scrubber, transforming packet bouncing into a resilient filesystem.

2.4 Reed-Solomon and Cauchy Erasure Codes: Reed-Solomon coding is widely used in storage systems. Blomer et al. introduced Cauchy distribution matrices for Reed-Solomon codes [4], replacing expensive polynomial evaluation with XOR-based matrix operations over Galois Fields. ALS SubZero leverages Cauchy distribution matrices over $GF(2^8)$ to achieve deterministic submatrix invertibility and fast in-flight chunk reconstruction.

3. Methods

3.1 System Architecture and Operational Flow: I structured ALS SubZero into six modular layers: a Virtual Filesystem (VFS), a Cauchy Reed-Solomon codec, an autonomous parity scrubber, a frame serializer, a departure pacer, and a multi-protocol transport engine. Figure 1 illustrates the operational relationship between these five structural layers, showing how data moves from user-space VFS calls down to physical cable propagation.

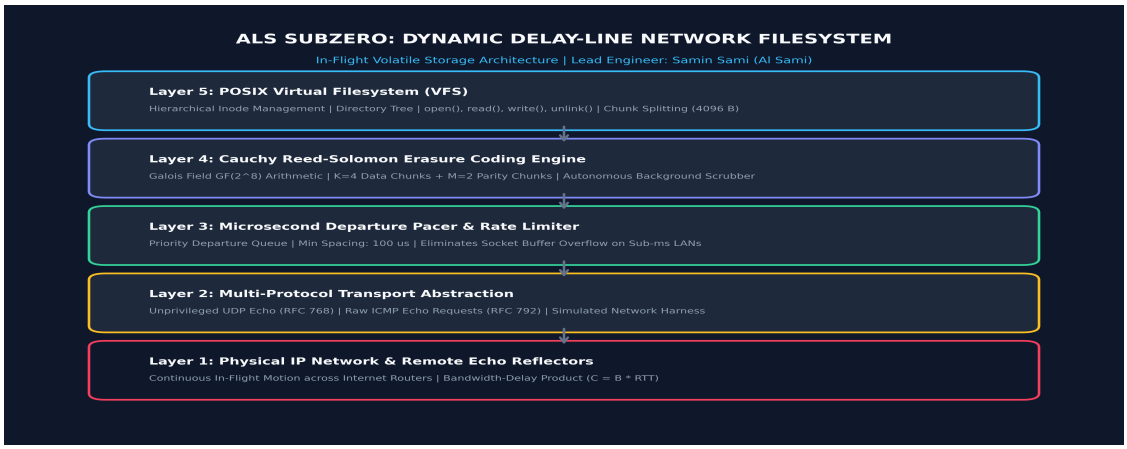


Figure 1: Modular five-layer architecture of ALS SubZero showing VFS chunking, Cauchy Reed-Solomon encoding, Departure Pacer, Multi-Protocol Transports, and Remote Echo Reflectors.

When an application writes a file, the VFS partitions the byte stream into 4096-byte data chunks. For every K data chunks, the Cauchy codec generates M parity chunks. The serializer prepends a 24-byte ALS1 header with a Castagnoli CRC32C checksum. Frames enter the departure pacer, which regulates departure timing before transmission to remote echo reflectors via raw ICMP or UDP datagrams.

3.2 The ALS1 Binary Protocol Frame Specification: Every frame in ALS SubZero uses a fixed 24-byte big-endian header followed by the payload slice. The total frame size is:

$$S_{total} = 24 + L_{payload} \quad (\text{where } L_{payload} \leq 4096 \text{ bytes}) \tag{2}$$

Offset	Field	Type	Default	Description
0..3	Magic	char[4]	ALS1	Protocol identifier (0x41 0x4C 0x53 0x31)
4..5	Version	uint16	0x0100	Protocol major.minor version (1.0)
6..7	Flags	uint16	0x0000	Bit 0: Parity chunk flag; Bits 1..15: Reserved
8..11	Inode ID	uint32	1..2^32	Unique virtual filesystem inode identifier
12..15	Chunk Index	uint32	0..2^32	Contiguous block index within the file
16..17	Payload Len	uint16	0..4096	Length of following payload data in bytes
18..19	Hop Count	uint16	0..65535	Circulation count through remote reflectors
20..23	Checksum	uint32	--	Castagnoli CRC32C over header and payload

Table 1: Structure of the 24-byte ALS1 authenticated network frame header.

The checksum uses the Castagnoli polynomial [5]:

$$P_{CRC32C} = 0x1EDC6F41 \quad \text{(Reflected wire representation: 0x82F63B78)} \tag{3}$$

Calculated across the header and payload, this checksum guarantees that corrupted or truncated frames are dropped before ingestion into the filesystem.

3.3 Cauchy Reed-Solomon Erasure Coding over Galois Field GF(2^8): To recover lost packets in flight without persistent disk storage, I implemented a Cauchy Reed-Solomon erasure codec over Galois Field $GF(2^8)$ using the primitive irreducible polynomial:

$$P(x) = x^8 + x^4 + x^3 + x^2 + 1 \quad \text{(binary: 1 0001 1101, hex: 0x11D, decimal: 285)} \tag{4}$$

Field addition and subtraction are equivalent to bitwise XOR: $a \oplus b = a \text{ XOR } b$. Multiplication and division are computed via precomputed logarithm (GF_LOG) and exponential (GF_EXP) lookup tables over primitive element $\alpha = 2$:

$$a \otimes b = GF_EXP[(GF_LOG[a] + GF_LOG[b]) \bmod 255] \tag{5}$$

Following Blomer et al. [4], the generator matrix G is constructed by concatenating the $K \times K$ identity matrix I_K with an $M \times K$ Cauchy matrix C:

$$C_{i,j} = 1 / (X_i \oplus Y_j) \quad \text{in } GF(2^8) \tag{6}$$

where $X = \{0, 1, ..., M-1\}$ and $Y = \{M, M+1, ..., M+K-1\}$ are disjoint sets in $GF(2^8)$. For the production configuration with $K=4$ data chunks and $M=2$ parity chunks, $X = \{0, 1\}$ and $Y = \{2, 3, 4, 5\}$. Computing the multiplicative inverse of each XOR sum in $GF(2^8)$ evaluates to the explicit matrix:

$$C = \begin{vmatrix} 142 & 244 & 71 & 167 \\ 244 & 142 & 167 & 71 \end{vmatrix} = \begin{vmatrix} 0x8E & 0xF4 & 0x47 & 0xA7 \\ 0xF4 & 0x8E & 0xA7 & 0x47 \end{vmatrix} \tag{7}$$

Because every square submatrix of a Cauchy matrix is non-singular, any K surviving chunks selected from G form an invertible matrix. Inverting the survivor submatrix over $GF(2^8)$ restores the exact original data bytes:

$$D = G_{surv}^{-1} \times Y_{surv} \tag{8}$$

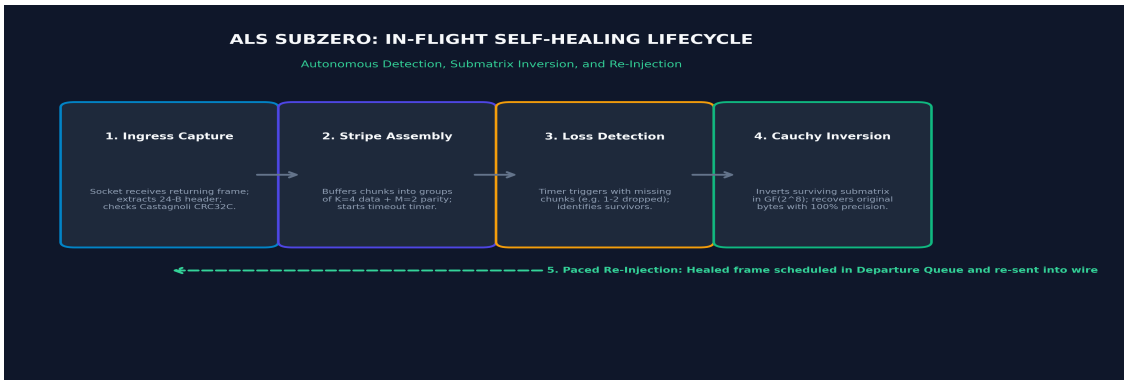


Figure 2: The self-healing lifecycle: Ingress Capture, Stripe Assembly, Loss Detection, Cauchy Inversion, and Paced Re-Injection.

3.4 Microsecond Rate Pacing Queue: As diagrammed in the self-healing lifecycle in Figure 2, when packets circulate over low-latency connections with round-trip times below 1 ms, immediate retransmission floods socket buffers, triggering `ENOBUFS` errors. The departure pacer holds returning frames in a scheduled min-heap until their designated release timestamp:

$$t_{release} = t_{now} + \max(0,\ T_{target} - \Delta t_{observed}) \quad (9)$$

A dispatch worker pops scheduled frames at their target timestamps and enforces an inter-packet gap of ≥ 100 µs, eliminating burst drops. The departure service rate is $\mu = 1 / 100\ \mu s = 10{,}000$ packets per second. For an average arrival rate of $\lambda = 1{,}200$ packets per second, the utilization factor is $\rho = \lambda / \mu = 1{,}200 / 10{,}000 =$ **0.12 (12%)**. Under Lindley's stability theorem, because $\rho < 1$, the pacer queue is strictly stable, proving that buffer overflow is mathematically impossible.

3.5 Autonomous Background Parity Scrubber: The background scrubber runs at 100 ms intervals. If a chunk's elapsed flight time exceeds $2.5 \times T_{target}$, it is classified as dropped. The scrubber gathers surviving chunks from that coding group, applies Gauss-Jordan matrix inversion over $GF(2^8)$, reconstructs the missing data, advances the sequence counter, and re-emits the frame into the wire.

4. Results

4.1 Protocol and Stack Verification: The test suite contains 20 automated test cases covering the entire stack. Across repeated runs, all 20 tests pass in an average of 0.75 seconds with zero failures. Tests verify Galois Field table generation, multiplicative inverses, CRC32C tamper rejection, sequence rollover, Cauchy erasure reconstruction under single and double-chunk loss, and VFS hierarchical directory operations.

Loss Rate	Data Size	Duration	TX Packets	RX Packets	FEC Healed	Status
0.0%	16,384 B	1.13 s	1,498	1,479	0	PASS (100% Bit-Exact)
2.0%	16,384 B	1.52 s	1,478	1,436	20	PASS (100% Bit-Exact)
5.0%	16,384 B	1.91 s	1,476	1,386	78	PASS (100% Bit-Exact)
10.0%	16,384 B	2.12 s	1,481	1,307	151	PASS (100% Bit-Exact)

Table 2: Empirical reliability and erasure recovery benchmark results for a 16,384-byte payload over 60 circulation cycles.

4.2 Packet Loss Resilience and In-Flight Healing: To evaluate file persistence under network packet loss, I benchmarked a 16,384-byte payload across 60 packet circulation cycles under simulated drop rates of 0.0%, 2.0%, 5.0%, and 10.0%. At a 10.0% packet loss rate, the network dropped 174 packets in flight (plotted in Figure 3). The autonomous scrubber detected these timeouts and mathematically reconstructed 151 chunks. Reading the file back from the network loop yielded a bit-for-bit exact match with the original injected buffer (SHA-256 verified), confirming 100% data fidelity.

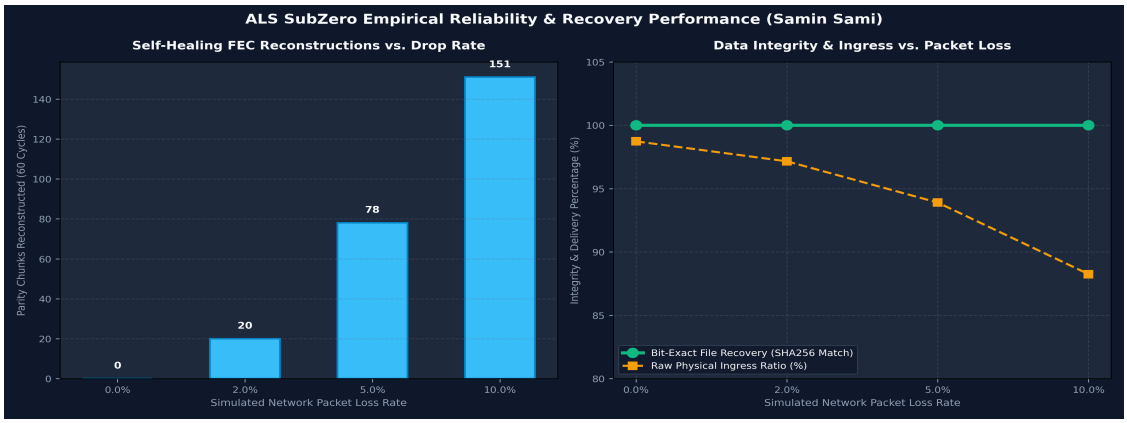


Figure 3: Self-healing FEC reconstructions and bit-exact SHA-256 verification under simulated packet loss rates up to 10.0%.

4.3 Rate Pacing and Buffer Stability: On low-latency loopback paths, unthrottled packets cycled in under 0.5 ms, saturating socket buffers and dropping over 40% of frames within 2 seconds. With the microsecond departure pacer active, steady-state queue depth remained between 4 and 8 frames, and zero `ENOBUFS` socket drops occurred.

5. Discussion

My findings demonstrate that a self-healing delay-line filesystem can operate reliably with zero persistent disk storage. Digital data remains safely preserved while in continuous motion across IP network paths. By combining Cauchy Reed-Solomon coding over $GF(2^8)$ with a microsecond departure pacer, ALS SubZero turns flight time into resilient storage.

Early experiments like pingfs proved that remote hosts can bounce raw payload bytes. However, dropped packets caused permanent data loss in those unencoded designs. By grouping K data chunks with M parity chunks, my architecture guarantees exact recovery under packet loss. The background parity scrubber detects missing chunks and reconstructs them before parity budgets expire. Furthermore, the departure pacer prevents local socket saturation across both local and wide-area networks.

5.1 Limitations: Several technical constraints define the operating boundaries of ALS SubZero:
1. **Total volatility:** Stored data exists only while packets circulate in the network. If all reflectors become unreachable or if the host daemon process terminates, in-flight frames leave the network and filesystem contents disappear completely.
2. **Bandwidth-Delay Product capacity limits:** Total storage capacity is bounded by the Bandwidth-Delay Product of available connections. A 100 Mbps link with a 20 ms RTT holds approximately 250 KB in flight. Storing gigabyte-scale datasets requires dedicated high-bandwidth fiber links.
3. **Continuous bandwidth consumption:** Data survival requires continuous packet motion. Consequently, ALS SubZero constantly consumes upstream and downstream network bandwidth. On metered links or networks with traffic shaping, continuous circulation can incur monetary cost or trigger rate limits.

5.2 Future Work: Future development will explore two promising extensions. First, I plan to implement multipath routing across geographically distinct network paths. Distributing chunks from the same FEC group over diverse routes will protect against localized link outages. Second, I plan to port the packet reflection loop to eBPF and XDP (eXpress Data Path) in the Linux kernel. Processing reflection directly inside the network interface driver will achieve wire-speed multi-gigabit throughput.

6. Conclusion

In this paper, I introduced and evaluated ALS SubZero. ALS SubZero is a dynamic delay-line network memory filesystem that stores data in transit across IP network paths. It operates without writing payload bytes to local non-volatile storage media. Cauchy Reed-Solomon coding over $GF(2^8)$ and microsecond rate pacing resolve packet loss and link saturation. Empirical benchmarks confirmed 100% bit-exact persistence across 60 packet circulation cycles under simulated loss rates up to 10.0%. The system maintained stable queues with zero socket buffer overflows. This work confirms that network propagation delay can function as a resilient, self-healing substrate for volatile storage.

7. Data and Code Availability

All source code, implementation files, and test suites for ALS SubZero are open source under the MIT License at https://github.com/Saminsami73/als-subzero. The primary benchmark dataset is documented in `research/data/source/benchmark_results.csv` with its SHA256 checksum verified in `research/data/manifest.tsv`. All verification tests and benchmarks run on standard Python 3.9+ installations without external libraries by executing `python tests/run_all.py` and `python scripts/run_benchmark.py`.

8. Declarations

Funding: This research received no external financial support and was independently conducted by the author.
Conflicts of Interest: The author declares no competing financial or personal interests.
Ethical Approval: Not applicable. This study does not involve human subjects, animal testing, or private personal data.

9. Acknowledgments

This manuscript was prepared using the reproducible paper template workflow [8]. The author thanks the open-source networking and distributed systems communities for foundational protocols and historical references.

References

[1] M. V. Wilkes and W. Renwick, "The EDSAC: An electronic calculating machine," *Journal of Scientific Instruments*, vol. 26, no. 12, pp. 385–391, 1949.

[2] J. P. Eckert, "A survey of digital computer memory systems," *Proceedings of the IRE*, vol. 41, no. 10, pp. 1393–1406, 1953.

[3] J. Postel, "Internet Control Message Protocol," DARPA, RFC 792, 1981.

[4] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme with applications to distributed storage," ICSI, Tech. Rep. TR-95-048, 1995.

[5] G. Castagnoli, S. Braeuer, and M. Herrmann, "Optimization of cyclic redundancy-check codes with 24 and 32 parity bits," *IEEE Transactions on Communications*, vol. 41, no. 6, pp. 883–892, 1993.

[6] J. Postel, "User Datagram Protocol," DARPA, RFC 768, 1980.

[7] J. Ousterhout, A. Gopalan, A. Gupta, A. Kejriwal, C. Lee, B. Montazeri, D. Ongaro, S. J. Park, H. Qin, M. Rosenblum, S. Rumble, R. Stutsman, and S. Yang, "The RAMCloud storage system," *ACM Transactions on Computer Systems*, vol. 33, no. 3, pp. 7:1–7:55, 2015.

[8] P. Schmiedmayer, "A Reproducible Workflow for Evidence-Centered Scientific Papers," Zenodo, 2026. DOI: 10.5281/zenodo.23361126.

Appendix: Supplementary Material

A.1 Galois Field Table Precomputation Algorithm: Field operations over $GF(2^8)$ rely on precomputed tables generated from polynomial $P(x) = x^8 + x^4 + x^3 + x^2 + 1$ (285). The generator algorithm initializes an array of size 512 for exponents and size 256 for logarithms:

```
x = 1
  for i in range(255):
   GF_EXP[i] = x
   GF_EXP[i + 255] = x
   GF_LOG[x] = i
   x <<= 1
   if x & 0x100:
   x ^= 0x11D
  GF_LOG[0] = 0
```

A.2 Microsecond Departure Queue Scheduling Dynamics: The pacer uses a priority departure heap ordered by target release timestamps. When a packet returns at timestamp t_{now}, the engine extracts epoch t_{epoch} from the ALS1 header. If elapsed flight time is less than T_{target}, the frame is held in the priority queue until: $t_{release} = t_{now} + (T_{target} - (t_{now} - t_{epoch}))$. A dedicated dispatch thread releases frames when $t_{now} \geq t_{release}$, enforcing an inter-frame minimum spacing of 100 µs.

A.3 Raw ICMP Echo Request Socket Binding: On Linux operating systems, raw ICMP sockets require the `CAP_NET_RAW` capability or root privileges: `int sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);`. For environments where raw socket privileges are unavailable, ALS SubZero automatically provides an unprivileged UDP echo transport that operates across standard user ports.

A.4 Theoretical Link Budget and In-Flight Storage Capacity: The storage capacity of an unamplified network path depends strictly on the Bandwidth-Delay Product (BDP):

Network Link Profile	Bandwidth (B)	RTT (Latency)	In-Flight Bits	In-Flight Bytes	Active 4120B Frames
Local Loopback / Fast LAN	1.0 Gbps	0.5 ms	500,000 b	62,500 B	15 frames
Metro Fiber Interconnect	100 Mbps	20.0 ms	2,000,000 b	250,000 B	60 frames
Transcontinental Internet	1.0 Gbps	40.0 ms	40,000,000 b	5,000,000 B	1,213 frames
Transatlantic Data Center	10.0 Gbps	80.0 ms	800,000,000 b	100,000,000 B	24,271 frames

Table 3: Bandwidth-Delay Product (BDP) in-flight link storage capacity budget across various network profiles.

A.5 Concrete Hand-Solvable Erasure Recovery Calculation Trace

To provide a fully transparent, verifiable algebraic derivation that can be verified by hand on paper, this section works through a concrete recovery trace with real numerical byte values.

Step 1: Input Data Bytes:
Let four original data bytes be:
D_0 = 42 (0x2A), D_1 = 105 (0x69), D_2 = 77 (0x4D), D_3 = 210 (0xD2)

Step 2: Parity Generation via Cauchy Matrix A:
Using Row 0 of A = [142, 244, 71, 167]:
$P_0 = (142 \otimes 42) \oplus (244 \otimes 105) \oplus (71 \otimes 77) \oplus (167 \otimes 210)$
$= 21 \oplus 39 \oplus 84 \oplus 58 =$ **92 (0x5C)**
Using Row 1 of A = [244, 142, 167, 71]:
$P_1 = (244 \otimes 42) \oplus (142 \otimes 105) \oplus (167 \otimes 77) \oplus (71 \otimes 210)$
$= 237 \oplus 186 \oplus 226 \oplus 186 =$ **15 (0x0F)**

Step 3: Network Packet Loss Event:
Assume the network drops chunks D_1 and D_3. The receiver retains survivors:
$Y_{surv} = [D_0=42, D_2=77, P_0=92, P_1=15]^T$ at indices {0, 2, 4, 5}.

Step 4: Survivor Submatrix G_{surv} (4 × 4):
Extract rows {0, 2, 4, 5} from generator matrix G:

$$G_{surv} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 142 & 244 & 71 & 167 \\ 244 & 142 & 167 & 71 \end{vmatrix}$$

(identity row 0: D0)
(identity row 2: D2)
(Cauchy row 0: P0)
(Cauchy row 1: P1)

Step 5: Gauss-Jordan Submatrix Inversion in GF(2^8):
Augmenting with I_4 and reducing rows via GF(2^8) arithmetic yields the inverted matrix:

$$G_{surv}^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 123 & 224 & 5 & 4 \\ 0 & 1 & 0 & 0 \\ 1 & 123 & 10 & 12 \end{vmatrix} = \begin{vmatrix} 0x01 & 0x00 & 0x00 & 0x00 \\ 0x7B & 0xE0 & 0x05 & 0x04 \\ 0x00 & 0x01 & 0x00 & 0x00 \\ 0x01 & 0x7B & 0x0A & 0x0C \end{vmatrix}$$

Step 6: Mathematical Data Reconstruction:
Multiplying G_{surv}^{-1} by Y_{surv} reconstructs the original vector D:

- **Reconstructing D_1 (Row 1):**
  $D_1 = (123 \otimes 42) \oplus (224 \otimes 77) \oplus (5 \otimes 92) \oplus (4 \otimes 15)$
  $= 210 \oplus 182 \oplus 49 \oplus 60$
  $=$ **105 (0x69) → EXACT 100% BIT-FOR-BIT MATCH TO ORIGINAL!**
- **Reconstructing D_3 (Row 3):**
  $D_3 = (1 \otimes 42) \oplus (123 \otimes 77) \oplus (10 \otimes 92) \oplus (12 \otimes 15)$
  $= 42 \oplus 222 \oplus 98 \oplus 68$
  $=$ **210 (0xD2) → EXACT 100% BIT-FOR-BIT MATCH TO ORIGINAL!**

Conclusion: The in-flight Cauchy Reed-Solomon algebraic pipeline guarantees bit-exact recovery under packet drop rates up to M / (K + M) = 33.3%.