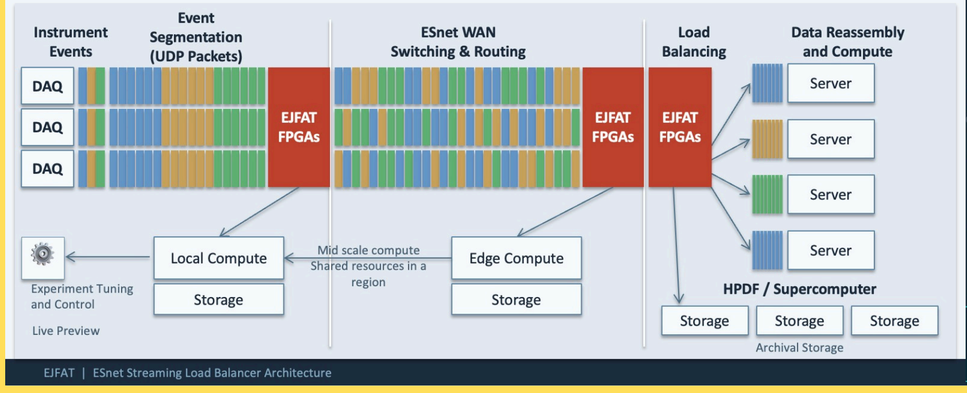
EJFAT: A Streaming Load Balancer

- Event aware data forwarding
- Separation of DAQ and Compute Ips
- Pre Integrated datapath with DOE supercomputers
- Nearly lossless packet transmission (RS Forward Error Correction)
- Scales to Tbps with parallel FPGA cards
- Active feedback load balancing , based on compute node consumption rates

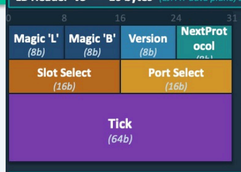


EJFAT | ESnet Streaming Load Balancer Architecture

EJFAT / E2SAR Data Plane Frame Formats

LB Header v3 (16 B) + RE Header v1 (20 B) — field definitions and on-wire layout

LB Header v3 — 16 bytes *(EJFAT data plane; stripped by load balancer hardware)*



- **Magic 'LB'** = 0x4C42 — LB header identifier
- **Version** = 3
- **NextProtocol** = 1 — RE header immediately follows
- **Slot Select** compute node select
- **Port Select** Selects LB UDP dst port
- **Tick** 64-bit event counter; repeats per event; must not wrap

RE Header v1 — 20 bytes *(Reassembly Engine; seen by receiver after LB strips LB header)*

- **Version** = 1 — 4-bit nibble in MSB of byte 0
- **Rsvd** 12 reserved bits — must be zero
- **Data-ID** Identifies the data source / sender
- **Buffer Offset** Byte position of this segment in reassembly buffer
- **Buffer Length** Total event size — pre-allocates the reassembly buffer
- **Event Number** Application event ID; independent from LB Tick

On-Wire Packet Structure:

Sender:	UDP	LB v3	FEC	RE v1	Payload
Receiver:	UDP	– stripped –	FEC	RE v1	Payload

Reed Solomon FEC to correct for packet erasures
RS(8,2) GF(4) x N

- Designed for 100 Gbps software implementation
- Designed for minimal FPGA gate utilization (one dot product per restored symbol)
- Systematic RS code: Receiver only processes missing packet data
- 4 bit symbols , 8 data nibbles + 2 parity nibbles: Trivial gate count
- Table lookup at decoder eliminates costly CHIEN search and Syndrome calculation
- Vector correction recognizing that for a packet, error locations are the same
- Bit level interleaving for full RS burst error protection
- Usable with any UDP streaming protocol (not EJFAT specific)

Pre-Computed H Matrix: Erasure Recovery Worked Example

Supplemental

1 Receive codeword — identify erasures from sequence numbers

Erasures detected at: d_2, d_4, p_6, p_7

2 Index erasure pattern → retrieve pre-computed H matrix

$\{d_2, d_4, p_6, p_7\}$ → Pre-computed H table → $H_{2,4}$

3 Substitute surviving parity symbols for erased positions

m_sub = 1 2 4 4 7 (d0 d1 p5→d2 d3 p8→d4)

H × m_sub = m_corrected

m_sub:	1	2	4	4	7			
d_0	1	0	0	0	0	× 1 =	1	NOP
d_1	0	1	0	0	0		2	2 NOP
d_2	12	3	14	9	5		4	3 GF(16) dot product ◀
d_3	0	0	0	1	0		4	4 NOP
d_4	12	4	9	12	10		7	5 GF(16) dot product ◀

Only the 2 correction rows (d_2, d_4) require computation. The 3 identity rows are a pass-through — no multiply needed. For RS(10,8) with T=1, only 9 H matrices cover all possible single-packet erasure patterns.

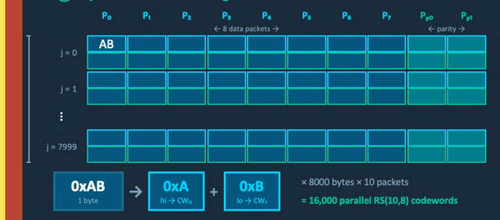
m_corrected = [1 2 3 4 5] ✓ Voilà!

EJFAT Packet-Based FEC

Parallel RS Operations & H Matrix Storage

RS(10,8) · GF(16) · 8000-byte packets

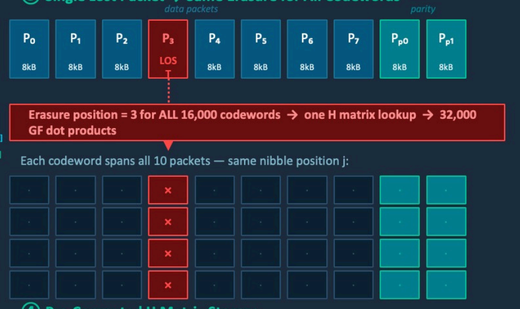
① Byte → Nibble Packing → 2 Parallel Codewords



0xAB (1 byte) → 0xA (hi → CW_0) + 0xB (lo → CW_1)

× 8000 bytes × 10 packets = 16,000 parallel RS(10,8) codewords

③ Single Lost Packet → Same Erasure for All Codewords



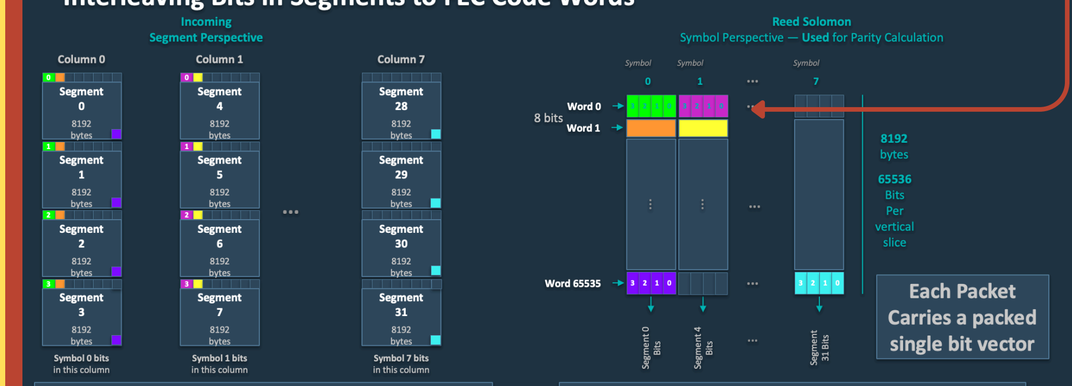
④ Pre-Computed H Matrix Storage

16,000	× 10	2x8000	32,000
codewords per packet	packet positions (8+2)	GF(16) ops / per erassure	dot products for 2 erasures
8000 bytes × 2 nibbles/byte	8 data + 2 RS parity packets	2 correction rows (T=1)	16,000 CWs × 2 rows each

45	2 × 8	360 B
H matrices	symbols per matrix	total storage
Choose (10,2) erasure patterns	2 correction rows × 8 cols	45 × 8 bytes (720 nibbles)

EJFAT Packet-Based FEC | Confidential

Interleaving Bits in Segments to FEC Code Words



Complete FEC Block contains
32 segments × 8192 bytes/segment
= 256 KiB (max)
= 2097152 bits (max)

Complete FEC Block contains:
65536 words × 8 symbols/word × 4 bits/symbol
= 256 KiB (max)
= 2097152 bits (max)

Burst Error Protection: Now we can correct up to 16 packets lost out of 48 total as long as they corrupt no more than 2 symbols.

E2SAR Reassembler — Receive Path Architecture

Network / Socket
UDP from Load Balancer
Kernel SO_RCVBUF 3 MB
per-thread UDP sockets

Recv Thread ×N
select() + recvfrom()
Parse RE Header
Reassemble into event buf

LOCK-FREE
Event Queue
boost::lockfree::queue
capacity: 1000 events
ptr handoff — no copy

Application
getEvent()
recvEvent() — blocking
Owns buffer: delete[]

per-packet processing

Recv Thread: per-packet receive + reassembly (N threads, each owns a subset of UDP ports)

select() wakes thread · recvfrom() into malloc'd recvBuffer · parse RE header · memcpy(event + bufOffset, payload, n) · check complete

recvBuffer	COPY 1	Event Buffer	COPY 2	eventQueue.
9 KB · malloc per pkt free'd after memcpy	kernel→user	new uint8_t[bufferLength] segments placed at bufferOffset	memcpy()	lock-free, cap 1000 ptr handoff to app

eventsInProgress: boost::unordered_map<(eventNum, dataId), EventState> · mutex-protected · shared with GC thread
When curBytes == bufferLength: remove from eventsInProgress → enqueue ptr to eventQueue → notify recvThreadCond

free(recvBuffer) per packet · Event Buffer ownership transfers to application · app must call delete[] when done

Copies & Buffers

Copies
① kernel → recvBuffer recvfrom() — 9 KB / pkt
② recvBuffer → Event Buffer memcpy(event)
③ metadata only on enqueue ~48 B (ptr, no payload)

Buffers
Kernel SO_RCVBUF 3 MB / socket
recvBuffer (transient) 9 KB
Event Buffer new uint8_t[len]
eventQueue ptrs cap 1000

GC Thread ×1
Sweeps every 500 ms
Free timed-out events
Mutex on eventsInProgress

NO ZERO COPY
malloc/free per packet
3 copies per event
No DPDK / io_uring / recvmmsg

SendState Thread ×1
PID controller on queue
gRPC → control plane
Every 100 ms

Threads
- Recv (×N) — select() + recvfrom() + reassemble
- GC (×1) — timeout sweep
- SendState (×1) — gRPC PID feedback
- App — getEvent() / recvEvent()

EJFAT / E2SAR | Reassembler Data Path: UDP → Recv Threads → Event Assembly → Application

E2SAR Segmenter — Send Path Architecture

Application
addToSendQueue(buf, len)
sendEvent(buf, len)
Owns event buffer

LOCK-FREE
Lock-Free Queue
capacity: 2047 items
Ptr only. No copy

Send Thread ×1
Busy-wait poll
Dequeues events
Posts to pool

Thread Pool ×N
N= # sockets

sendEvent(): direct call
dispatch_send()

Worker Thread: _send() (N pool threads, one event each)

Per seg: malloc LBREHdr (34B), fill headers, iovec scatter-gather

Application Event Buffer (original — unmodified) — ZERO COPY

iov[0] Hdr 34B	iov[1] Payload ptr	iov[0] Hdr 34B	iov[1] Payload ptr	...
Segment 0		Segment 1		Seg N-1

UDP Send
sendmsg()
sendmmsg() batch
io_uring async
4 UDP sockets
Random src ports
3 MB send

On Wire
IP Hdr 20B
UDP Hdr 8B
LB Hdr 16B
RE Hdr 18B
Payload

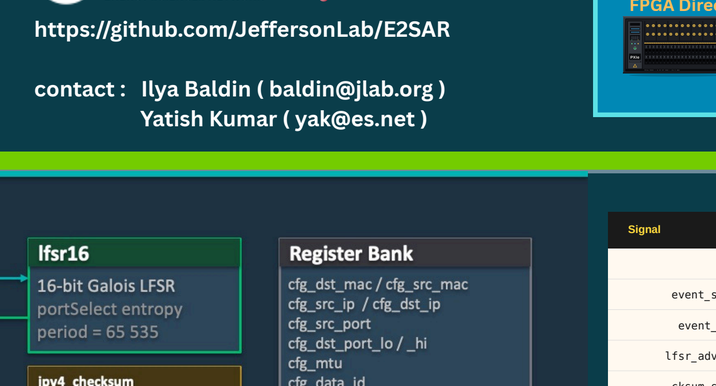
Sync Thread ×1
Sends SyncHdr every ~1s
Separate UDP socket
→ Control Plane

ZERO COPY
addToSendQueue() stores ptr
iov[1] → caller's buffer
No memcpy ever

Threads
- App (×1)
- Send (×1)
- Workers (×N)
- Sync (×1)

EJFAT / E2SAR | Segmenter Data Path: Application → Queue → Workers → UDP

- E2SAR Software library. Fully open source.
  - Incorporated in dozens of projects
  - Tested at upto 100G per compute node
  - Python and C++ distributions
    - Docker / Conda / Debian Packages
- Multi threaded with CPU pinning
- Optimized memory to memory handoff
- 100% Claude compatible



- In production and deployed at ESnet and Jefferson Lab
  - 800 Gbps WAN capacity
  - Instant availability from any facility
- Early adoption and integration:
  - Advanced Light Source
  - Advanced Photonic Source
  - Jefferson Lab (CEBAF / EIC)
  - SLAC (LCLS)
  - FRIB
  - NSLS-II
- HPC Facilities:
  - NERSC (Perlmutter)
  - Oakridge OLCF (Defiant / Lux)
  - Jefferson lab (HPC)
  - UCSD (NRP / OSG)
  - NSF (FABRIC)

EJFAT
ESnet Jefferson Lab FPGA Acclerated Transport

Real Time streaming to multiple supercomputer sites simultaneiously !
Dynamic load balancing, with uSecond granularity
Dynamic node connection / removal
Terabit scale

ESnet ENERGY SCIENCES NETWORK — Jefferson Lab

https://github.com/JeffersonLab/E2SAR

contact : Ilya Baldin (baldin@jlab.org)
Yatish Kumar (yak@es.net)

EJFAT Framer — RTL Data Path Architecture

External BRAM
512-bit wide
2-cycle rd latency
← bram_en
← bram_addr
→ bram_dout

Event Interface
event_start
event_num [63:0]
event_bytes [31:0]
tick [63:0]
bram_base_addr
← event_done
← busy

event_ctrl
Event segmentation FSM
8-state: IDLE → CALC → CKSUM → EMIT → WAIT_PKT → NEXT_SEG → DONE
Segment arithmetic
ip_id counter (per pkt)

lfsr16
16-bit Galois LFSR
portSelect entropy
period = 65 535

ipv4_checksum
2-cycle pipeline
RFC 791 checksum
partial static sum
(TTL, IPs cached)

Register Bank
cfg_dst_mac / cfg_src_mac
cfg_src_ip / cfg_dst_ip
cfg_src_port
cfg_dst_port_lo / _hi
cfg_mtu
cfg_data_id
cfg_ttl
cfg_lfsr_seed / cfg_lfsr_load

rd_start / rd_addr · *seg_* fields* · *pkt_done* · *cfg_* signals*

bram_reader
BRAM read controller
2-bit in-flight pipe
2-entry skid buffer
Back-pressure safe
rd_data[511:0]
rd_valid / rd_last
rd_byte_count[6:0]

hdr_builder (comb.)
Assembles 78-byte header
Eth/IPv4/UDP/LBHdrV3/RE
hdr_beat0[511:0] B0–63
hdr_beat1_14B[111:0] B64–77 (hdr residue)

hdr_beat0/1

pkt_assembler
Header + payload merge
14-byte barrel shift
Beat 0: hdr_beat0 (64B)
Beat 1: hdr_beat1 + pay[0:49]
Beat N: residue + payload
→ pkt_done → event_ctrl

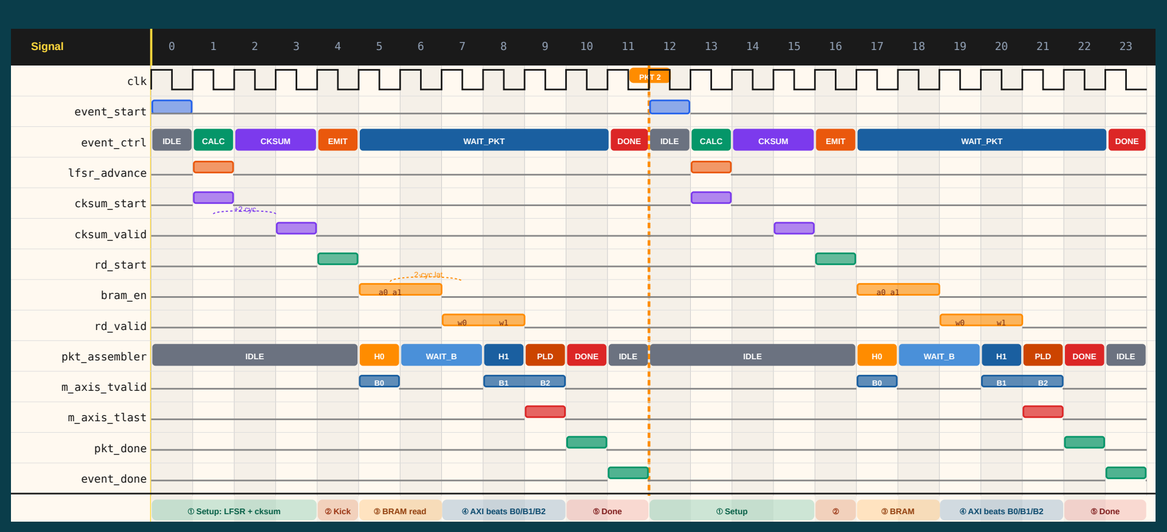
AXI-S beats

tx_fifo
Store-and-forward
256-deep × 512-bit
Holds until cksum_done
Patches UDP checksum
→ beat 0 bytes 40–41
→ 100G CMAC (AXI-S)

rd_data tap · *cksum_done/value*

udp_cksum_onthefly
Passively taps bram_reader rd_data bus · accumulates RFC 768 checksum in parallel with transmission (no extra BRAM reads) · pseudo-hdr + static hdr sum + payload → cksum_done / cksum_value → tx_fifo patches beat 0 bytes 40–41

EJFAT Framer | Event → MTU Segments → Eth/IPv4/UDP/LBHdrV3/REHdr → 100 Gbps CMAC



	LUTs	REG	BRAM
Framer	7000	4000	1
FEC encoder	11000	6400	64
FEC decoder	23480	14000	144

- VHDL framer developed and tested at JLab
- Verilog framer developed and tested at ESnet
- FEC Verilog and SW modules in development
- 100 Gbps design