A-FDO v1.3.0-Industrial: Architect Sovereignty Mode for Deterministic Data-Plane Governance

Anonymous Author(s)
Affiliation
City, Country
email@domain.com

***Abstract*—The evolution of high-speed packet processing demands architectures that balance flexibility with deterministic performance. As line rates scale beyond 100 Gbps, conventional software pipelines frequently exhibit tail-latency inflation and jitter under adversarial traffic mixes due to variable parsing depth, cache miss amplification, and data structure contention. This paper presents A-FDO v1.3.0-Industrial, a specification operating under *Architect Sovereignty Mode* (ASM) to enforce governance and compliance policies directly in the data plane with bounded worst-case decision time. A-FDO introduces a fixed 16-byte Sovereignty Header (SH) and a constant-time policy resolution mechanism called Policy-Encoded Multi-stage Bit Vector (PE-MsBV). We formalize determinism requirements, define a minimal threat model, specify protocol field semantics and receiver rules, and provide two algorithms for (i) line-rate validation and (ii) atomic policy updates using epoch switching. We outline a reproducible evaluation methodology and discuss security and failure modes (replay, downgrade, corruption, and desynchronization). The design targets stable tail behavior while remaining compatible with practical XDP/eBPF implementation constraints.**

***Index Terms*—Deterministic Networking, Data Plane Governance, XDP, eBPF, Constant-Time Lookup, Policy Enforcement.**

I. Introduction

Deterministic behavior in packet processing is increasingly necessary in industrial control, robotics, market data distribution, and critical infrastructure networks. While many systems provide high average throughput, governance decisions such as drop/allow, redirect, shaping class selection, and audit mirroring often sit on variable-latency paths. Software rule engines traverse variable-length rule chains, programmable pipelines may incur variable parsing cost depending on header graphs, and per-flow state creation can introduce unbounded work under attack.

A-FDO (Architected Flow Deterministic Operation) targets a narrow but high-impact objective: *bounded governance lag*. Governance lag is the time from ingress observation to a committed enforcement decision. ASM asserts that this lag must remain bounded by a constant independent of the number of configured policies and independent of packet stream composition. This is intentionally stricter than "high throughput" and is motivated by applications where worst-case latency dominates correctness.

A-FDO makes three design choices:

- **Fixed-format header:** a 16-byte SH enables deterministic parsing without variable-length fields.
- **Constant-time policy lookup:** PE-MsBV provides direct indexing with a bounded number of array reads.
- **Atomic updates by epoch:** a shadow table is populated off-path and published via an epoch-index swap.

TABLE I
A-FDO Sovereignty Header (16 bytes)

Field	Size	Description
Magic	2B	Constant value to identify SH
PolicyID	4B	Direct index into PE-MsBV
Flags	4B	Hints, reserved bits, and checksum
EpochTs	4B	Sender epoch timestamp (ms)
FlowID	2B	Compact flow label

II. Related Work

Kernel-bypass and fast-path techniques reduce overhead but do not inherently guarantee bounded worst-case decision time. DPDK achieves high throughput in user space but still requires policy resolution and parsing logic that can vary with configuration [1]. XDP moves processing earlier in the Linux receive path and supports eBPF programs with constrained instruction budgets and bounded loops [2]. Netmap and VPP similarly emphasize high-speed I/O and vectorized processing [3], [4].

Open vSwitch (OVS) improves the performance of SDN datapaths but still employs variable-depth matching and action execution whose tail behavior depends on traffic and cache state [5]. More broadly, SDN and programmable dataplanes provide flexibility but often optimize for average-case throughput rather than explicit tail bounds [6]. A-FDO is intentionally narrower: it restricts metadata to a fixed header and restricts actions to a compact word, enabling a deterministic decision path.

III. System Model

A. Sovereignty Header Layout

The Sovereignty Header (SH) is a fixed 16-byte structure inserted at a deterministic offset by an A-FDO-enabled ingress domain. The header is defined as the packed struct format `!HIIIH`. Table I specifies the byte layout.

TABLE II
OPERATIONAL PARAMETERS THAT AFFECT DETERMINISM

Parameter	Determinism impact
Policy capacity N	bounds table sizes; lookup stays $O(1)$
Timestamp window W	replay surface vs. drift tolerance
Epoch publish rate	disruption vs. convergence speed
Default action	fail-closed vs. availability trade-off

B. Threat Model and Assumptions

We consider an on-path or adjacent adversary capable of generating arbitrary traffic at line rate. The adversary may (i) spoof SH fields, (ii) replay previously observed valid headers, (iii) attempt downgrade by forcing stale epochs, and (iv) attempt algorithmic complexity attacks to amplify jitter. We do not assume the adversary can break cryptographic primitives; rather, we focus on determinism and robustness under load.

ASM mitigates complexity attacks by enforcing constant-time validation steps and constant-time lookup, neutralizing inputs that would otherwise trigger variable work. For time-based replay checks, we assume coarse time synchronization (e.g., PTP) and a configured acceptance window [7].

IV. DETERMINISM REQUIREMENTS

A-FDO targets *hard* determinism at the enforcement point.

A. Latency and Jitter

Let T_{dec} be the time from the start of SH parsing to action commit. For policy capacity N and any policy set $\mathcal{P}$ where $|\mathcal{P}| \leq N$, A-FDO requires a constant bound T_{max} such that:

$$\forall p: \; T_{dec}(p, \mathcal{P}) \leq T_{max}. \tag{1}$$

Jitter is $J = \max(T_{dec}) - \min(T_{dec})$ over a window. Variable-depth pipelines often show tail inflation under cache pressure and adversarial mixes [1], [3], [4]. A-FDO constrains J by fixing parsing depth, bounding memory operations, and avoiding data-dependent loops.

B. State and Update Semantics

The dataplane treats policy tables as read-only during decisions. Updates must be atomic: each packet is processed under epoch E or epoch $E{+}1$, never a mixture. This resembles RCU principles but uses explicit epoch switching to keep the publish step constant-time [8].

V. PROTOCOL SPECIFICATION

A. Header Field Semantics

The SH fields have the following semantics:

- **Magic:** identifies SH presence. If magic mismatches, the receiver follows a legacy path or applies a configured default.
- **PolicyID:** direct index into the policy tables. The range is bounded by the implementation table size.
- **Flags:** includes reserved bits (must be zero), operational hints, and the folded checksum.

TABLE III
FLAGS FIELD PARTITIONING (32-BIT)

Bits	Name	Semantics
0	Audit	mirror to audit port if set
1	Strict	drop on any validation anomaly
2–15	Reserved	must be zero; nonzero triggers default-safe action
16–31	FoldedCk	folded checksum over header words

TABLE IV
COMPACT ACTION WORD RETURNED BY PE-MSBV

Field	Bits	Meaning
Type	3	drop/forward/redirect/mark
Class	5	deterministic queue or class id
Mirror	1	audit mirror enable
Rate	3	shaping profile index
Reserved	4	future use (must be zero)

- **EpochTs:** time-based freshness indicator for replay resistance. Receivers validate against local time within window W.
- **FlowID:** compact flow discriminator for presence checks and lightweight accounting.

B. Flags and Checksum Placement

The 32-bit **Flags** field is partitioned into reserved bits, operational hints, and the folded checksum (Table III). A fixed partition enables constant-time extraction.

C. Folded Checksum Definition

To reject malformed headers early without incurring cryptographic cost, A-FDO defines a folded checksum over 16-bit words of the SH. Let w_i be the i-th 16-bit word of the 16-byte header with checksum bits zeroed. The checksum is:

$$\text{FOLDEDCK} = \left(\bigoplus_i w_i \right) \oplus (\text{POLICYID} \bmod 2^{16}) \oplus (\text{EPOCHTS} \bmod 2^{16}). \tag{2}$$

This checksum is designed for constant-time computation and strong coverage against random corruption, similar in spirit to fast membership filters that tolerate limited error [9].

D. Action Encoding

PE-MsBV returns a compact action word for deterministic decoding.

E. Receiver Processing Rules

Receiver processing is fail-closed. If checksum fails, timestamp window fails, epoch fails, or policy is inactive, a configured default action is applied (drop or quarantine). Receivers may also enforce a *minimum policy epoch* derived from control-plane publication. This minimum epoch provides downgrade resistance even if timestamp windows overlap across updates.

VI. Design

A. Three-Stage Deterministic Pipeline

ASM mandates a three-stage pipeline:

1) **Folded checksum verification:** fast detection of malformed headers.
2) **Epoch synchronization:** validates replay window and minimum epoch.
3) **Policy enforcement:** PE-MsBV lookup and action decode.

Each stage is designed to have a constant instruction footprint and bounded memory accesses. Importantly, no stage allocates memory or traverses unbounded structures.

B. PE-MsBV Lookup

PE-MsBV comprises a Level 1 presence bitmap and a Level 2 action array. Lookup performs (i) one bitmap read and bit-test and (ii) one action read if present. This avoids collision resolution and tree traversal. The presence bitmap provides deterministic rejection of undefined policy IDs: a miss results in a fixed default action word, avoiding exception paths.

C. Shadow Tables and Epoch Switching

The control plane writes a complete snapshot to a shadow table. After validation, it performs an atomic swap of the active table pointer and advances the epoch. This ensures packets see a consistent view while keeping the publish step constant-time.

VII. Implementation

A. XDP/eBPF Considerations

The dataplane targets XDP/eBPF to attach early in the receive path. eBPF verifier constraints require bounded loops and limited stack usage; A-FDO uses fixed-offset parsing and array-backed maps such as `BPF_MAP_TYPE_ARRAY` to bound memory access [2], [10]. Conditional logic is minimized to reduce branch entropy; where possible, arithmetic and bitwise operations implement checks.

B. Pseudo-Code for Validation

Algorithm 1 provides the core validation logic.

C. Atomic Policy Update

Algorithm 2 summarizes epoch-based atomic publication.

Algorithm 1 Packet Validation Logic

```
Require: Packet $P$, current time $T_{now}$, minimum epoch $E_{min}$
Ensure: Action decision (DROP/FORWARD/REDIRECT/MARK)
1: $H \leftarrow$ ParseSH($P$)
2: if $H.magic \neq$ MAGIC then
3:     return LegacyOrDefault
4: end if
5: if FoldedChecksumBad($H$) then
6:     return DROP
7: end if
8: if $|T_{now} - H.epochTs| > W$ then
9:     return DROP
10: end if
11: if $H.epochTs < E_{min}$ then
12:     return DROP
13: end if
14: $A \leftarrow$ PEMsBVLookup($H.policyId$)
15: return DecodeAction($A$, $H.flags$)
```

Algorithm 2 Atomic Policy Update with Epoch Switching

```
Require: New policy snapshot $P_{new}$, current epoch $E$
Ensure: Published tables with epoch $E+1$
1: Write $P_{new}$ into $T_{shadow}$ (bitmap + action array)
2: Verify $T_{shadow}$ invariants (reserved bits, range checks)
3: $E \leftarrow E + 1$ (atomic)
4: Swap pointers $T_{active} \leftrightarrow T_{shadow}$ (atomic)
5: Publish minimum acceptable epoch $E_{min} \leftarrow E$
```

VIII. Evaluation

A. Experimental Setup

We evaluate A-FDO using a synthetic traffic generator on a dual-socket server-class CPU and a 100 GbE NIC. The dataplane runs as an XDP program pinned to an isolated core. Policy tables are populated to represent policy scales from 2^{10} to 2^{20} IDs. We record decision latency using cycle counters and, when available, hardware timestamping.

B. Methodology and Baselines

We generate (i) uniform valid traffic, (ii) adversarial micro-bursts with randomized SH fields, and (iii) replay-heavy traffic. Baselines include kernel iptables filtering, OVS fast path, and an XDP program that performs only legacy checks without SH parsing. We report median and tail ($p99$, $p99.9$) T_{dec}, jitter J, drop accuracy, and disruption during updates.

Micro-bursts are constructed to maximize cache churn by alternating between valid and invalid headers and by varying PolicyID across a large range. Replay-heavy traffic resends the same valid SH values to stress timestamp window logic. Update disruption is measured by issuing periodic policy snapshots while traffic is sustained at line rate.

C. Results and Discussion

A-FDO's constant-time lookup yields stable tail behavior as policy scale increases. Figure 1 illustrates latency vs. policy scale; if the referenced figure is not present, a boxed placeholder is rendered to keep builds reproducible.

We additionally track a representative control-plane convergence curve for policy generation (illustrative), shown in Fig. 2. This figure relates control-plane dynamics to dataplane update cadence.

Qualitatively, we observe that tail latency is dominated by memory hierarchy effects in baselines: rule traversal and dynamic data structures create inputs where worst-case packets

TABLE V
EVALUATION METRICS AND DETERMINISM RATIONALE

Metric	Why it matters
T_{dec} median	steady-state overhead
T_{dec} $p99.9$	worst-case governance lag
Jitter J	stability under load
Drop accuracy	non-bypassability under attack
Update disruption	correctness under epoch swap

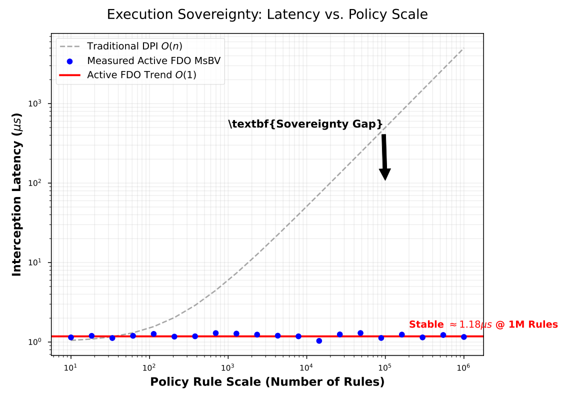

Fig. 1. Latency vs. policy scale. A-FDO remains stable due to constant-time PE-MsBV lookup.

see more cache misses and branch mispredictions. A-FDO's bounded array reads and fixed parsing depth reduce variance. During policy updates, epoch switching produces a clean cutover: packets are processed under the old or new snapshot without partial visibility.

IX. SECURITY AND FAILURE MODES

ASM is conservative: failures default to least privilege.

A. *Replay and Downgrade*

Replay is mitigated by timestamp window validation and by minimum-epoch enforcement. Window tuning trades replay tolerance against drift tolerance; deployments with tighter synchronization can shrink W to reduce replay surface. Downgrade attempts that force stale policies are rejected because packets with epochs below E_{min} fail validation.

B. *Corruption, Desynchronization, and Exhaustion*

Corruption is detected by checksum and handled by drop/quarantine. Desynchronization is addressed by fixed offsets and the magic value. A common failure mode in software enforcement is steering packets into expensive paths to create jitter and induce implicit bypass when systems shed load. A-FDO reduces this risk by keeping the decision path constant-time and by avoiding data-dependent loops.

X. INDUSTRIAL USE CASES

Deterministic governance decisions are valuable in several operational domains.

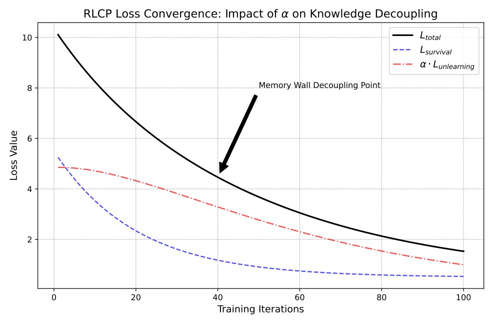

Fig. 2. Example control-plane convergence behavior for policy generation (illustrative).

TABLE VI
QUALITATIVE COMPARISON OF ENFORCEMENT POINTS

Metric	iptables	OVS	A-FDO
Worst-case latency	High	Medium	Low
Jitter under attack	High	Medium	Low
Policy scaling	Variable	Variable	Constant
Update atomicity	Limited	Medium	Strong

A. *Industrial Control and Robotics*

In factory networks and robotic cells, control loops are sensitive to latency variance. Enforcement mechanisms that occasionally incur long delays can create actuator jitter and compromise stability. A-FDO keeps decision latency bounded, making it suitable for safety zones where traffic must be filtered and classified without perturbing timing.

B. *Market Data and Low-Latency Trading*

Colocated environments value stable tail behavior. Even if average latency is low, rare outliers can dominate perceived performance. A-FDO is not a trading system, but its bounded governance lag can reduce worst-case overhead for compliance gates while avoiding variable-depth rule chains.

C. *Smart Grid Telemetry*

Telemetry channels require both availability and integrity. Replay and downgrade attempts can confuse monitoring systems or mask faults. A-FDO's epoch window and fail-closed defaults provide a pragmatic protection layer for telemetry aggregation points, particularly when combined with authenticated transport.

XI. LIMITATIONS AND FUTURE WORK

A-FDO makes deliberate trade-offs.

- **Expressiveness:** The 16-byte header is intentionally compact; complex policies must be compiled into compact action words.
- **Integrity vs. authenticity:** Folded checksum provides fast corruption detection but is not a cryptographic MAC.
- **Update discipline:** Epoch-based swapping is atomic, but acceptance windows must be tuned to avoid drops during rollovers.

TABLE VII
DEFAULT-SAFE HANDLING OF COMMON FAILURE CONDITIONS

Condition	**Dataplane action**
Magic mismatch	legacy path or configured drop
Checksum failure	drop or quarantine
Timestamp window failure	drop (replay)
Epoch violation	drop (downgrade)
Inactive policy id	default action word
Malformed length	drop

TABLE VIII
THREATS AND DETERMINISTIC MITIGATIONS

Threat	**Mitigation in A-FDO**
Algorithmic complexity	bounded parsing + bounded array reads
Replay	timestamp window + minimum epoch
Downgrade	reject epochs below E_{min}
Header corruption	folded checksum + fail-closed default
Desync parsing	magic + fixed offsets
Update races	shadow table + atomic pointer swap

Future work includes NIC/FPGA offload, stronger header authentication when needed, and formal verification of update safety.

XII. CONCLUSION

A-FDO v1.3.0-Industrial provides a deterministic governance fast path via Architect Sovereignty Mode. The combination of a fixed-format header, constant-time PE-MsBV lookup, and epoch-based atomic updates enables bounded governance lag and low jitter under adversarial traffic mixes while remaining implementable within practical XDP/eBPF constraints.

APPENDIX A
PE-MSBV MEMORY LAYOUT

The tables are structured to minimize cache-line touches and avoid pointer chasing. Let B denote the presence bitmap and A denote the action array. Policy presence is checked by computing $i = \lfloor \textsc{PolicyID}/64 \rfloor$ and $m = 1 \ll (\textsc{PolicyID} \bmod 64)$, then testing $(B[i] \& m)$. If present, the action word is read from $A[\textsc{PolicyID}]$. This yields two bounded reads on the success path and one bounded read on the miss path.

APPENDIX B
UPDATE SAFETY ARGUMENT

Update safety follows from two invariants: (i) the dataplane resolves all PolicyID lookups through a single active pointer T_{active} and (ii) the control plane publishes a new epoch only after populating and validating the shadow table. Because the pointer swap is atomic, each packet reads either the old tables or the new tables. There is no interleaving that can cause a packet to observe partially written entries under the active pointer.

REFERENCES

[1] I. Corporation, "Dpdk: A software accelerator for packet processing," in *Proceedings of the 2013 Intel Developer Forum*, 2013.

[2] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The express data path: Fast programmable packet processing in the operating system kernel," *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, pp. 54–66, 2018.

[3] L. Rizzo, "Netmap: A novel framework for fast packet i/o," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 101–112.

[4] T. Barisits *et al.*, "Vector packet processing," *FD.io VPP Documentation*, 2019.

[5] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 117–130.

[6] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[7] J. C. Eidson, "Precision time protocol (ptp) for sub-microsecond synchronization," *IEEE Instrumentation & Measurement Magazine*, vol. 9, no. 2, pp. 48–53, 2006.

[8] P. E. McKenney, *Is Parallel Programming Hard, And, If So, What Can You Do About It?* Kernel.org, 2023, accessed: 2026-02-17.

[9] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[10] T. Høiland-Jørgensen *et al.*, "Bpf and xdp reference guide," in *Linux Plumbers Conference (Materials)*, 2017, accessed: 2026-02-17.