International Journal of Scientific Research and Modern Technology (IJSRMT) ijsrmt.com
Volume 5, Issue 10, 2026
DOI: https://doi.org/10.38124/ijsrmt.v5i10.1737

Proxy ARP at XDP: Answering and Dropping ARP Before skb Allocation

Azif Abdul Salam Kolisam beevi[1]; Anil Kumar Reddy[2]

[1]Director of Enterprise Solutions Engineering, San Jose, CA, USA
[2]Principal Engineer, Bangalore, Karnataka, India

Publication Date: 2026/11/02

Abstract

Proxy ARP is usually implemented after the network stack has already paid for the packet: whether the logic lives in a kernel module or a userspace daemon, every ARP request has been received into an skb before anyone decides to answer, forward or discard it. An eBPF program attached at the XDP hook runs on the driver's receive buffer before any skb exists. Using BPF hash maps as binding tables and as sets, it can answer known requests in place with XDP_TX, drop unwanted ones with XDP_DROP, and pass only the remainder to the stack. This paper describes that design vendor-neutrally: the conventional implementations and their per-packet cost, the XDP datapath, the drop policy built from sets and maps, operational failure modes, and how to evaluate the result. It also states the main limit plainly: the skb saving requires native (driver-mode) XDP.

Keywords: *Proxy ARP, eBPF, XDP, XDP_TX, XDP_DROP, BPF Maps, LPM trie, skb Allocation, Wi-Fi, IEEE 802.11 Proxy ARP Service, Hotspot 2.0, DHCP Snooping, ARP Storm, Bridged Datapath.*

I. INTRODUCTION

Broadcast ARP is cheap on a switched LAN and expensive on a WLAN. Three properties of 802.11 make the difference:

- *Rate*

Group-addressed frames go out at a basic rate so every station can decode them, so one broadcast can cost many times the airtime of the same frame sent unicast.

- *No Acknowledgement*

Broadcast frames are not retried, so a request can be lost silently and the requester retries the whole exchange.

- *Power Save*

Group-addressed traffic is buffered and delivered after the DTIM beacon. Every dozing station in the BSS wakes to receive an ARP request that almost never concerns it, which costs battery on phones and IoT devices.

In a dense deployment, most ARP requests target either the gateway or another station whose address the AP already knows from association and DHCP. Answering those at the AP removes them from the air entirely. The IEEE 802.11 Proxy ARP service (used by Hotspot 2.0) [4] standardizes this behaviour; this paper is about implementing it, and the policy around it, in eBPF.

There is also a cost inside the AP. Every ARP frame that reaches a proxy ARP implementation has already been allocated an skb, run through the receive path and, in many products, copied to userspace. In a large L2 domain, the wired uplink carries a steady stream of ARP broadcasts for addresses the AP will never answer, and an ARP storm or scan can make that stream the dominant packet load on a small AP CPU. Handling ARP at XDP [5] removes both costs at once: the air is spared the broadcast, and the host is spared the skb.

- *Scope*

IPv4 ARP on a Linux-based AP or gateway, independent of how a given vendor implements proxy ARP today. Hardware offload is out of scope.

This paper makes the following contributions:

- It describes the conventional proxy ARP implementations and breaks down their per-packet cost, showing that they all decide after the packet has become an skb.
- It presents an XDP datapath in which userspace decides what is true and the datapath only answers from that

Kolisam beevi, A. A. S., & Reddy, A. K. (2026). Proxy ARP at XDP: Answering and Dropping ARP Before skb Allocation. *International Journal of Scientific Research and Modern Technology*, *5*(10), 86–94. https://doi.org/10.38124/ijsrmt.v5i10.1737

truth, with a binding table written only by a control-plane agent.
- It specifies the implementation: map schema, parsing and validation, in-place rewrite, transmit, VLAN handling, and a drop policy built from BPF maps used as sets and policy tables.
- It lists the operational failure modes with their mitigations, defines an evaluation methodology, and compares conventional implementations with XDP.

Section 2 reviews ARP, proxy ARP, how proxy ARP is usually implemented and what that costs. Section 3 presents the design and Section 4 the implementation. Section 5 covers operational concerns, Section 6 the evaluation methodology and Section 7 the comparison of conventional implementations with XDP. Section 8 concludes.

II. BACKGROUND

➢ *ARP and Proxy ARP*

ARP (RFC 826) [1] maps an IPv4 address to a link-layer address with a broadcast request and a unicast reply. Proxy ARP (RFC 1027) [2] lets a node answer for an address it does not own. Classic router proxy ARP answers with the router's own MAC; the Wi-Fi variant answers with the *target station's* MAC, so traffic still flows directly to that station.

➢ *How Proxy ARP Is Usually Implemented*

- *Kernel Space*

A module or stack hook (a netfilter or bridge hook, a packet-type handler, or a vendor datapath module) inspects ARP after the driver has built the skb and handed it up the receive path, looks up a binding table and either transmits a reply or lets the frame continue.

- *Userspace*

A daemon receives ARP through a raw or packet socket (often with a classic BPF socket filter), or via a punt path from a kernel module, and answers from its own table. Each frame is an skb plus a copy across the kernel boundary, plus a wake-up and a context switch.

- *Hybrid*

The kernel answers known bindings and punts misses or policy decisions to userspace.

These differ in where the table lives, but they share one property: the decision is made after the packet has become an skb.

➢ *The Cost: An skb for Every ARP*

- *Allocation and Receive Path*

skb and data allocation, protocol demux and any hooks in front of the proxy logic run for every ARP, including the ones that will be thrown away.

- *Userspace Overhead*

A punt to userspace adds a copy and scheduling latency per packet, and a daemon that falls behind loses requests or delays replies.

- *No Cheap Discard*

ARP that should simply be dropped (scans, storms, requests for addresses outside the served subnets, sources that are not associated) still costs the full receive path before it is discarded.

- *Load Under Attack*

Because the cost is per packet, an ARP flood on the uplink turns directly into CPU load on the AP, competing with the traffic it exists to forward.

XDP moves the decision in front of all of this. The program runs on the driver's receive buffer before skb allocation, so a reply costs a rewrite and a transmit from the same buffer, and a drop costs almost nothing. Only ARP that genuinely needs the stack is passed on. Figure 1 breaks that cost down per frame. Each row is one implementation, and each column is a stage of work the frame may pay for.

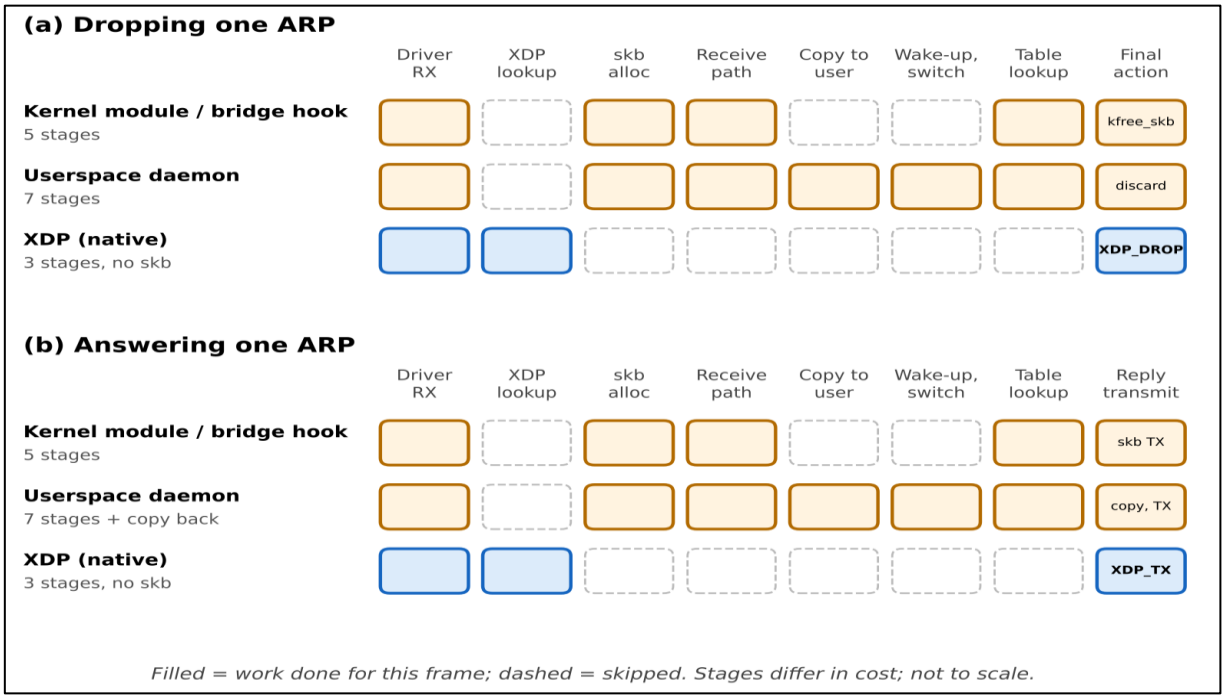


Fig 1 Per-Packet Work for (a) A Dropped ARP and (b) An Answered ARP in Three Implementations. Filled = Work Done for This Frame; Dashed = Skipped. Stages Differ in Cost; Not to Scale.

A dropped frame shows the difference most clearly. The kernel module pays for allocation and the receive path only to free the skb, and the daemon additionally pays a copy and a wake-up for a frame it will ignore. XDP reads the buffer, checks its maps and returns.

For an answered frame, the conventional paths transmit from an skb, and the daemon also copies the reply back into the kernel. XDP rewrites the received buffer and transmits it from the same ring.

The diagrams show which work is done, not how long each stage takes. Allocation and the receive path dominate in the kernel case, and the copy and context switch dominate in the userspace case. Section 6 describes how to measure the actual per-frame cost on a given AP.

➢ *Packet Path: Conventional vs XDP*

The two paths reach the same three outcomes. What differs is whether the packet has already cost an skb when the decision is made (Figure 2).

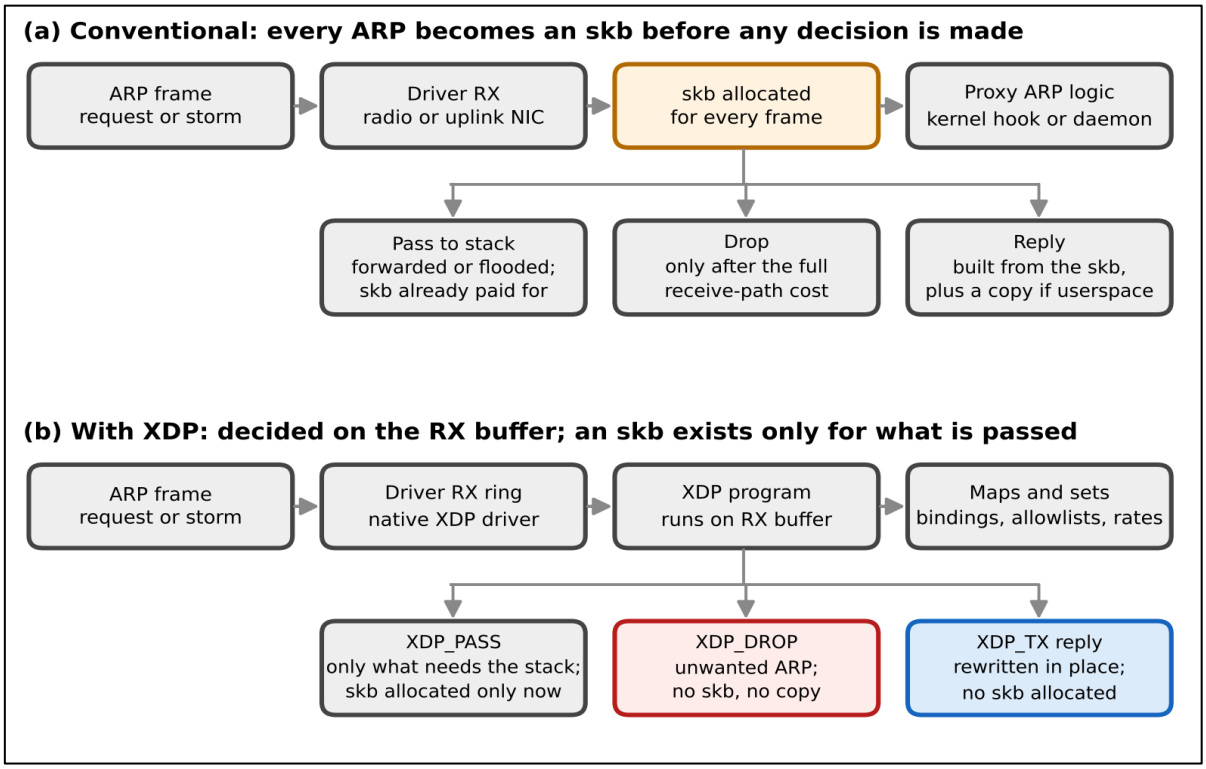


Fig 2 (a) Conventional Proxy ARP, in the Kernel or in Userspace, After skb Allocation. (b) Proxy ARP at XDP, Decided Before skb Allocation.

In a conventional implementation, every ARP frame is received into an skb before the proxy logic sees it, so even a frame that is dropped has paid the full receive cost, and a userspace implementation adds a copy on top.

At XDP, the program decides on the driver's receive buffer: known requests are answered with XDP_TX, unwanted ones are dropped with XDP_DROP, and an skb is allocated only for frames passed to the stack. This holds with native (driver-mode) XDP; in generic mode the skb already exists, and only the policy benefit remains.

III. DESIGN

The design splits cleanly: userspace decides what is true, and the datapath only answers from that truth.

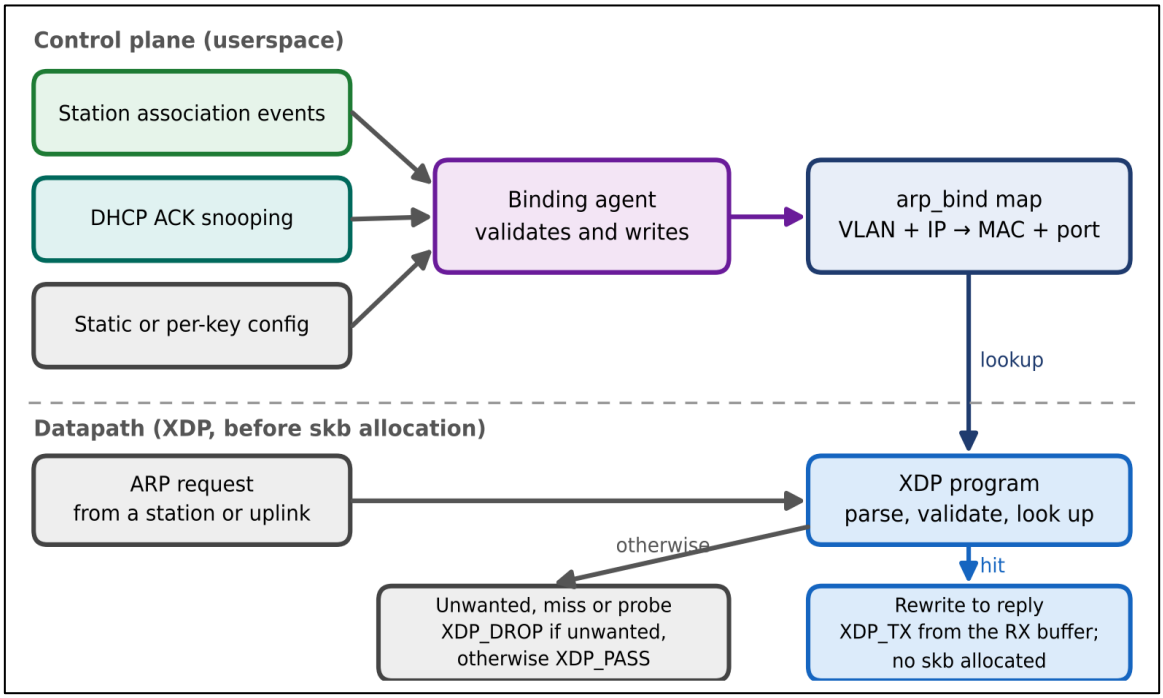


Fig 3 Architecture: Three Feeders, One Map, One Datapath Program. The Control Plane Owns the Table; the Datapath Only Answers from it.

DOI: https://doi.org/10.38124/ijsrmt.v5i10.1737

Trusted feeders write the binding map through one agent; the datapath program reads it and answers in place, drops what is unwanted, or passes the frame to the stack unchanged.

➢ *The Binding Table as Source of Truth*

The map is written only by the control-plane agent, never by the datapath and never from ARP seen on the air. That one rule closes the spoofing hole of learned neighbor tables and makes every answer traceable to a source event.

➢ *Feeders*

- *Station association events*

Association, disassociation and roam give the MAC, the BSS and, with MLO, the MLD address. Disassociation is the primary delete signal. Any implementation's association handler can produce these.

- *DHCP ACK snooping*

On the trusted (uplink) side only, gives the IP-to-MAC binding and the lease lifetime.

- *Static and per-key configuration*

Fixed devices, gateways, served subnets, and per-key groups that need their own VLAN or policy.

The agent reconciles these: an IP is bound only when a DHCP lease or static entry agrees with an associated station, and it is removed as soon as either side disappears.

➢ *Hook Point: XDP or TC*

Table 1 Hook Points for the Proxy ARP Program

Hook	Runs	skb Allocated Before the Program?	Where it Fits
Native XDP (driver mode)	On the driver RX buffer	No	Wired uplink and any radio driver with XDP support; the target of this design
Offloaded XDP	On the NIC	No	SmartNICs; out of scope here
Generic XDP (skb mode)	After the driver builds the skb	Yes	Fallback on drivers without XDP; same program, policy benefit only
TC ingress	After the skb, before the stack	Yes	Paths that need skb metadata, such as hardware-stripped VLAN tags

The skb saving exists only in native mode. Many Wi-Fi radio drivers support only generic XDP today, so the largest gain is usually on the wired uplink, which is also where broadcast ARP from a large L2 domain and ARP storms arrive. The same object file loads in either mode, so a design can start with generic XDP on the radios and gain the allocation saving as drivers add native support.

IV. IMPLEMENTATION

The datapath returns one of three verdicts for every ARP frame, all before an skb exists: answer it in place (XDP_TX), discard it (XDP_DROP), or hand it to the stack (XDP_PASS).

➢ *Map Schema*

The binding map is keyed by VLAN and IPv4 address, so the same address can exist in different tenants. The value carries the station MAC, the port it lives behind and flags (static, DHCP-learned, roaming).

Listing 1 Binding Map Schema

```
struct bind_key { __u16 vlan; __u16 pad; __be32 ip; };
struct bind_val { __u8 mac[ETH_ALEN]; __u32 ifindex; __u32 flags; };

struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __uint(max_entries, 4096);
    __type(key, struct bind_key);
    __type(value, struct bind_val);
} arp_bind SEC(".maps");
```

A per-CPU array of counters (hit, miss, probe passed, gratuitous passed, malformed) sits alongside it.

➢ *Parse and Validate*

The program accepts a frame only if it is Ethernet/IPv4 ARP with hlen == 6, plen == 4 and op ==

REQUEST. Every header access is bounds-checked against data_end to satisfy the verifier. Two cases are passed through untouched:

- *Probe (spa == 0):*

An RFC 5227 [3] duplicate-address probe. Answering it would make the prober believe its address is taken.

- *Gratuitous (spa == tpa):*

An announcement, which peers and the control plane may need to see.

➢ *In-Place Rewrite*

Listing 2 In-Place Rewrite of the Request into a Reply

```
__be32 req_ip = arp->spa;
__u8 req_mac[6];
__builtin_memcpy(req_mac, arp->sha, 6);

arp->op  = bpf_htons(ARPOP_REPLY);
__builtin_memcpy(arp->tha, req_mac, 6);
arp->tpa = req_ip;
__builtin_memcpy(arp->sha, v->mac, 6);
arp->spa = k.ip;

__builtin_memcpy(eth->h_dest,   req_mac, 6);   /* ARP sha, not eth src */
__builtin_memcpy(eth->h_source, v->mac,  6);
```

The Ethernet destination comes from the ARP sender hardware address rather than the Ethernet source. They normally match, but a requester behind a relay or a bridged repeater may differ, and the ARP field is the one the requester's stack will act on.

➢ *Transmit*

- *XDP:*

Return XDP_TX; the reply leaves the same interface.

- *TC Ingress:*

Return bpf_redirect(skb->ifindex, 0); hairpins it to the port's egress.

A bare ARP reply is 42 bytes, under the 60-byte Ethernet minimum. Most drivers pad on transmit; where a path does not, grow the frame with bpf_xdp_adjust_tail (XDP) or bpf_skb_change_tail (TC).

➢ *VLAN Handling*

On XDP, the program parses 802.1Q (and 802.1ad if used) itself and keys the lookup on the VID. On TC, a tag stripped by hardware or bridge VLAN filtering appears in skb->vlan_tci instead of the packet. The program must read the VID from wherever the path in use puts it, and the reply must leave with the same tag.

➢ *Dropping Unwanted ARP with Sets and Maps*

The drop path is where XDP pays off most: a frame dropped at XDP costs a parse and a few map lookups, with no allocation and no copy. BPF maps serve as sets (a key with an empty or one-byte value) and as policy tables.

Table 2 Maps Used as Sets and Policy Tables

Map	Type	Used as	Drops
served_subnets	LPM_TRIE, key = prefix + VLAN + IPv4	Set of subnets this AP serves	Requests whose target is outside every served subnet
known_src	HASH, key = VLAN + MAC	Set of associated stations and known gateways	ARP from an unknown source MAC on a radio port
vlan_policy	ARRAY, index = VLAN	Per-VLAN miss action and suppression flags	Misses on VLANs marked drop-on-miss
src_rate	LRU_PERCPU_HASH, key = source MAC	Token bucket per sender	Senders above their ARP rate (scans, storms)
arp_bind	HASH, key = VLAN + IPv4	Binding table (Section 4.1)	Not a drop source; a hit is answered

The checks run cheapest and most decisive first (Figure 4):

- Malformed or non-Ethernet/IPv4 ARP: XDP_DROP.
- Probe (spa == 0) or gratuitous (spa == tpa): rate-check, then XDP_PASS.
- Sender over its rate in src_rate: XDP_DROP.
- On radio ports, sender not in known_src: XDP_DROP.
- Target not in served_subnets: XDP_DROP.
- Target in arp_bind: rewrite, XDP_TX.
- Otherwise apply vlan_policy: XDP_DROP or XDP_PASS.

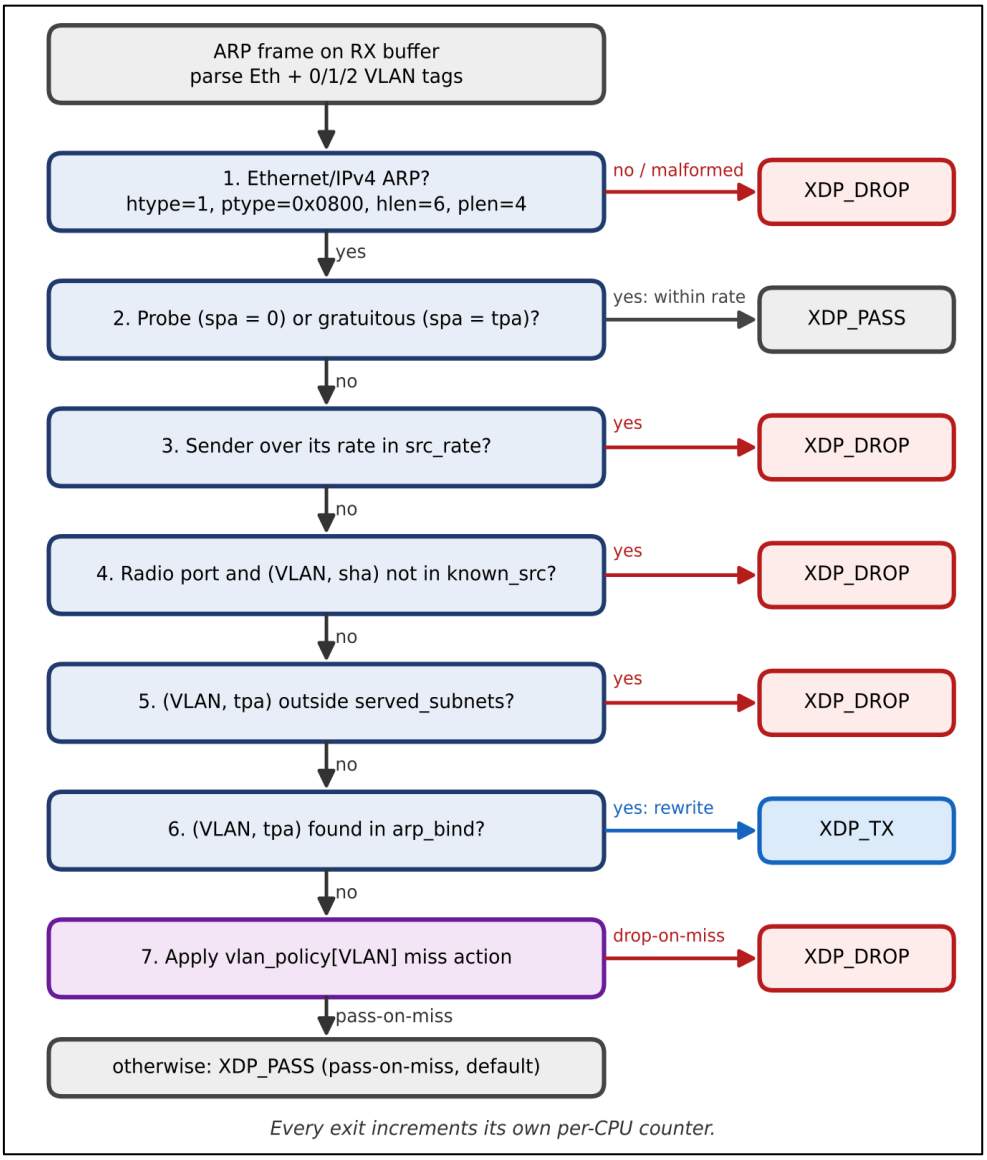


Fig 4 Order of the Datapath Checks. The Checks Run Cheapest and Most Decisive First; Every Drop Reason has its Own Counter.

Listing 3 Served-Subnet Set and the Subnet Check (Step 5)

```
struct lpm_key { __u32 prefixlen; __u16 vlan; __u16 pad; __be32 ip; };

struct {
    __uint(type, BPF_MAP_TYPE_LPM_TRIE);
    __uint(max_entries, 256);
    __uint(map_flags, BPF_F_NO_PREALLOC);
    __type(key, struct lpm_key);
    __type(value, __u8);              /* set: value unused */
} served_subnets SEC(".maps");

/* step 5 */
struct lpm_key lk = { .prefixlen = 64, .vlan = vid, .ip = arp->tpa };
if (!bpf_map_lookup_elem(&served_subnets, &lk)) {
    count(STAT_DROP_SUBNET);
    return XDP_DROP;
}
```

The rate limiter is per CPU, so its limit is approximate: a sender spread across N receive queues can reach up to N times the configured rate. That is acceptable for storm control; where an exact limit matters, use a shared map with atomic updates and accept the contention.

Every drop reason has its own counter. A drop path without counters is indistinguishable from a broken network, and the first question in any incident will be why ARP is being discarded.

V. OPERATIONAL CONCERNS

The rewrite is the easy part. Deployments fail on table correctness, and almost every failure below is a stale or wrong binding being answered confidently.

Table 3 Operational Concerns and Mitigations

Concern	What Goes Wrong	Mitigation
Roaming and staleness	The AP keeps answering for a station that left; after DHCP reuses the address, the new owner is blackholed	Delete on station disassociation events, move on roam notifications, and treat timeouts only as a backstop
Client isolation	The reply points the requester at a peer MAC, but L2 isolation drops the unicast that follows	Per SSID, either answer with the gateway/AP MAC or hairpin through the uplink; choose explicitly
MLO (802.11be)	Answering with a link address breaks when the station switches links	Bind and reply with the MLD address; the receiving link is irrelevant
Duplicate address detection	Answering an RFC 5227 probe tells the prober its address is taken	Pass probes (spa == 0); answer only as a deliberate defence of a known binding
Gratuitous ARP	Suppressing announcements hides address changes from peers and the control plane	Pass them; optionally let the control plane validate them against the table
Spoofing	A table learned from air traffic can be poisoned by any associated client	Write the map only from trusted sources: association events, DHCP ACK from the trusted port, static config
Map capacity	A full hash map silently refuses new bindings	Size per AP station limit and VLAN count; alarm on insert failure in the agent
Frame size	A 42-byte reply is under the Ethernet minimum on paths that do not pad	Extend the tail before transmit

- *Miss Policy*

A miss means the AP does not know the target. Passing it preserves today's behaviour and is the safe default. Dropping it is stronger suppression and is correct only where the table is complete by construction, for example a guest SSID where every client must have obtained its address through snooped DHCP. Make this a per-VLAN setting, and count misses either way: a rising miss rate is the earliest sign that a table feeder is broken.

The design should be measured on two fronts: host cost, where XDP's advantage over a conventional implementation shows, and the air, where proxy ARP of any kind shows. Correctness under mobility is the risk to test. No results are reported here; the following is a test plan, run against the same AP with the conventional implementation and with the XDP program.

VI. EVALUATION METHODOLOGY

Table 4 Evaluation Test Plan

Metric	How to Measure	What to Look for
CPU per ARP frame	Traffic generator at fixed ARP rates on the uplink; softirq and total CPU on the AP	Lower CPU at every rate with native XDP
Saturation point under ARP flood	Raise the ARP rate until forwarding throughput of user traffic degrades	Flood rate the AP absorbs before user traffic suffers
skb allocations	Kernel allocation counters or tracing during the flood	Allocations only for passed frames
Reply latency	Time from request to reply at the requester	No worse than the conventional path
Drop counters by reason	Program per-CPU counters	Each drop explainable by subnet, source, rate or policy
Group-addressed ARP on air	Over-the-air sniffer per BSS	Requests for known bindings no longer reach the air
Correctness under roam	Scripted roams between two APs while a peer pings the roaming station	No reply names a station that has left
Address reuse	Release and re-lease an address to a different client	Old binding gone before the new lease is answered
DAD safety	Client joins with a duplicate address	Conflict still detected by the client

Generic vs native mode	Repeat the CPU tests with the program in generic mode	Quantifies how much of the gain is the skb saving

Power-save impact can be measured on a dozing test client by counting wake-ups at DTIM for group-addressed frames over a fixed window.

VII. CONVENTIONAL IMPLEMENTATIONS VS XDP

A conventional implementation is simpler to build and debug; XDP earns its complexity where ARP volume, ARP storms or a small AP CPU make the per-packet cost matter, and where unwanted ARP should be discarded rather than processed.

Table 5 Conventional Implementations Compared with eBPF at XDP

Aspect	Kernel-Space Module or Hook	Userspace Daemon	eBPF at XDP (Native)
Decision point	After skb allocation and receive-path hooks	After skb allocation plus a copy to userspace	On the driver RX buffer, before any skb
Cost of a drop	Full receive path	Full receive path, copy and wake-up	Parse plus map lookups
Cost of a reply	skb transmit	Copy in, copy out, transmit	In-place rewrite, XDP_TX
Behaviour under ARP flood	CPU load grows per packet	Daemon backlog, delayed or lost replies	Dropped at line rate the driver can sustain
Policy changes	Module update or sysctl	Daemon restart or reload	Map update from userspace, no reload
Safety of change	Kernel bug can crash the AP	Isolated from kernel	Verifier-checked; cannot crash the kernel
Driver dependency	None	None	Native XDP support for the skb saving
Complex logic	Easy	Easiest	Constrained by the verifier; keep complex logic in userspace
Engineering cost	Module to maintain across kernels	Daemon to maintain	Program, loader and feeder agent; CO-RE eases portability

VIII. CONCLUSION

Proxy ARP is usually implemented after the packet has become an skb, which means every ARP frame is paid for before anyone decides it should not have been. An XDP program reverses that order: it answers known requests from the receive buffer, drops unwanted ARP using maps as sets and policy tables, and allocates an skb only for what the stack actually needs. The gain depends on native XDP support, which today is strongest on the wired uplink, where most ARP volume and storms arrive. The datapath itself stays small; the engineering effort belongs in the control plane that keeps the bindings and sets correct through roaming, address reuse and multi-link operation, because a wrong answer or a wrong drop delivered at line rate is worse than a slow one.

REFERENCES

[1]. D. C. Plummer, "An Ethernet Address Resolution Protocol," RFC 826, IETF, Nov. 1982.

[2]. S. Carl-Mitchell and J. S. Quarterman, "Using ARP to Implement Transparent Subnet Gateways," RFC 1027, IETF, Oct. 1987.

[3]. S. Cheshire, "IPv4 Address Conflict Detection," RFC 5227, IETF, Jul. 2008.

[4]. IEEE 802.11, Proxy ARP service (Wi-Fi Passpoint / Hotspot 2.0).

[5]. T. Høiland-Jørgensen et al., "The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel," in *Proc. ACM CoNEXT*, 2018.

[6]. Linux kernel documentation, "BPF Documentation," https://docs.kernel.org/bpf/.