Android

IPsec/IKE Reliability

on (Mobile) Android

IPsec Workshop — Vienna, July 2026

Nathan Harold (nharold@google.com)

Yan Yan (evitayan@google.com)

Google

Proprietary and Confidential

Agenda

01 Background / Reminders

02 PMTUD with IKE and ESP Ping

03 Lossless MOBIKE

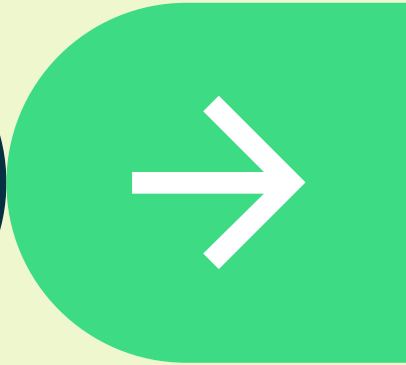
04 (Brief) ESP-based Link Error Detection

05 (Brief) Port Agility

Proprietary and Confidential

Why are we here?

Android IPsec/IKE



Proprietary and Confidential

Android Features using IKEv2/IPsec

01

Android/Pixel VPN

- Core AOSP VPN
- API via VpnManager
- Google Pixel devices have free access via the Pixel VPN

02

IMS/IWLAN (Cellular)

Pixel's IMS stack, much of which is open source, utilizes both transport mode (for SIP) and tunnel mode (for WiFi calling over IWLAN).

03

Virtual Carrier Network

Google Fi uses a network virtualization layer, the "VCN", which establishes one-or-more IPsec tunnels to 'virtualize' the cellular data connection.
Similarities with IWLAN, but security gateway is outside the cellular network, VPNs can run over the top.

Problem 1: Interoperability

Mobile ecosystem presents a variety of diverse (and broken) networks!

01

NAT & UDP

NAT and short UDP timeouts break connection persistence.

02

Firewalls

Strict firewall policies block IKEv2/IPsec traffic on key ports.

03

DNS64

DNS64 environments without ESP NAT translation disrupt tunnels.

04

MTU & Frag.

Varying MTUs and fragmentation behaviors lead to lost packets.

Problem 2: Variable Radio Link Conditions

Causes

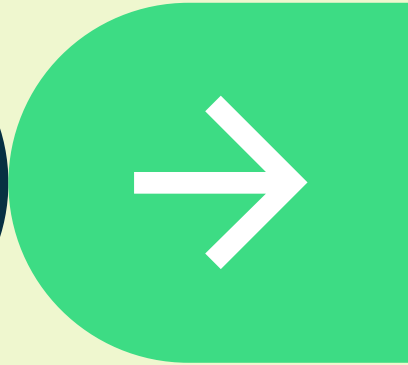
- Mobility — Cell <-> WiFi, Cell <-> Cell, WiFi <-> WiFi
- Power saving (constraints on NAT-T keepalives)
- Congestion

Effects

- Packet loss
- Out of order delivery
- Link failure

Proprietary and Confidential

PMTUD with IKE and ESP Echo



Proprietary and Confidential

MTU in the Wild: <Major ISP> WiFi Case Study

Android uses IPsec to securely offload carrier traffic to Wi-Fi. On <Major ISP> Wi-Fi, severe MTU issues/constraints discovered:

01

Restricted Link MTU

- Observed IPv4: 1396, IPv6: 1336, likely due to ISP network overhead
- Observed on specific sets of ISP-Managed “infra grade” APs

02

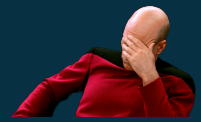
Fragment Drops

- Fragments are dropped silently
- Android's network validation fails to catch it because it uses small probe packets

03

Unreliable ICMP

The network returns inaccurate MTU values in ICMP packets: claiming 1400 v,s, actual 1396)



Practical Issues Encountered

Not specific to PMTUD but seem to stem from missing tools in a bigger “toolbox”

01 No standard “embedded” MTU detection even for IKE–relies on ICMP

02 Lack of fate sharing between ESP and IKE

03 Inability to recover from blackholes that cause IKE packet drops (after retries)

04 Time cost of repeated probing

05 Inability to synchronize MTU between Initiator and Responder

Proprietary and Confidential

Solution Approach - Hybrid of Fast/Cheap Tests

Composite Approach

Combining multiple strategies for MTU detection covers a variety of scenarios throughout the session, across varying link configurations.

01

Test Link "Minimum Viability"

Must be fast (round trips) and cheap (consume minimal data) to establish base connectivity metrics rapidly.

02

Optimize After Setup / Handover

Minimize performance reduction time and prevent temporary connection disruptions during state shifts.

03

Leverage Existing Data Streams

Piggyback off other exchanges when feasible to reduce round-trips and minimize overhead.

Our Proposals

ESP Ping-based PMTUD

- Using the ESP Echo protocol to probe ESP path MTU

IKE-based PMTUD

- Using padded IKEv2 messages to probe path MTU

IKEv2 MTU Notification (RFC Comments)

- Suggesting a new Notify Payload to communicate and synchronize detected PMTU values between peers

Proprietary and Confidential

RFC Support - Compatibility

Goal: Delegate ESP decryption to **existing** kernel XFRM, routing the decrypted payload to an **unprivileged userspace** socket.

LIMIT 1

Dummy Packets Drop

Draft ESP Echo uses Next Header 59 (dummy packet); Linux XFRM drops by default for RFC 4303 compliance.

PROPOSALS:

[Proposal 1] Allow Inner UDP Header in the ESP payload with "SHOULD SUPPORT"

[Proposal 2] A New Dedicated Next Header Protocol (e.g., IPPROTO_ESP_ECHO)

LIMIT 2

ESP Padding Loses Information

Information regarding ESP padding thickness can be lost during standard kernel processing loops.

PROPOSAL:

[Proposal] Mandate that both entities apply only the **minimum** required ESP padding to maintain compatibility.

Proprietary and Confidential

RFC Support - Encapsulated ESP

Open question in the draft: *"Are ESP Echo/Reply valid when using UDP encapsulation?"*

Our Answer: Absolutely Yes, why?

HIGH PRIORITY

Path MTU Discovery (PMTUD)

To discover the PMTU, the device needs to send ESP Ping probes using the exact same protocol (UDP-encapsulated ESP) and port as the actual data traffic.

LOW PRIORITY

Reachability Testing

Assertion: it's highly unlikely that a network device will allow IKE while selectively dropping UDP-encapsulated ESP packets on the same port 4500.

Proprietary and Confidential

RFC Support - Comments

MTU Size Limitation

Due to the strict ESP 4-byte alignment requirement, there is a limitation imposed on supportable MTUs

CONSIDERATION:

Fully acceptable

Asymmetric MTU Detection

Handling situations where outbound and inbound path MTUs differ across the network route.

MANDATE:

The draft protocol mandates that the responder must copy the incoming data payload up to the detected MTU limit.

Note: If less is received, than sent, asymmetric MTU is detected.

Proprietary and Confidential

Active PMTUD using Padded IKE

Overview & Mechanism

HOW IT WORKS:

Sender transmits padded IKE request matching target test MTU with DF set

BENEFITS VS ICMP:

- **RFC Support:** Compliant servers must respond, unlike ICMP which is often dropped
- **Signalling Reduction:** Avoids round trips and even trims a few bytes when piggybacked
- **Route Parity:** IKE messages (sometimes) share the same routing path as ESP data traffic

Three Operational Modes

1. Probe-Only Mode (Max MTU)

Padded IKE INIT request with `unknown critical payload`. Server MUST reject with UNSUPPORTED_CRITICAL_PAYLOAD

2. Setup Mode (Min-viable MTU)

Padded IKE INIT request with `unknown non-critical payload`. Server ignores extra payload, proceeds with IKE setup.

3. MOBIKE Mode (Min-viable MTU)

INFORMATIONAL request padded with `unknown non-critical payload`. Server proceeds with MOBIKE.

Proprietary and Confidential

RFC Support - IKEv2 MTU Notification

Proposal Overview

`draft-liu-ipsecme-ikev2-mtu-dect-10`

Additional capability to support using IKEv2 to actively communicate detected Path MTU (PMTU) values.

KEY ASSUMPTIONS

- ➔ Viability of ICMP to detect PMTU
- ➔ Path symmetry is assumed? Reflects most common network scenarios.

Sync Mechanism

Detection

- **Active Discovery:** Node A detects outbound PMTU (A ⟶ B).
- **Provider Context:** Tunnel MTU is capped via network knowledge.

Notification

Node A sends `DETECTED_PMTU` notification to Node B.

Adjustment

Node B proactively limits outbound transit packets:

`TMAP(B → A) = PMTU(A → B) - overhead`

Node B runs its own PMTUD, treating notification as a maximum upper bound.

Live Demo!

IKE & ESP Ping

Watch me sweat while I pray that the demo app works to show various mechanisms of MTU verification!

Demo Setup & Verification

Client Setup

An Android client that sets up a `transport` mode SA via IpSecManager and encap ESP Ping in `UDP` instead of Next Header 59

Server Setup

A deamon on `a Google Cloud Server` that filters out incoming ESP packets and redirects them from the kernel to userspace via `NFQUEUE`.

```
# Redirect ESP packets
nft add rule inet esp_ping prerouting esp spi 7 queue num 100

# Redirect UDP-encapsulated ESP (Port 4500)
nft add rule inet esp_ping prerouting udp dport 4500 @th,64,32 7 queue num 100
```

Demo Setup & Verification

Client Setup

An Android client sends padded IKE requests

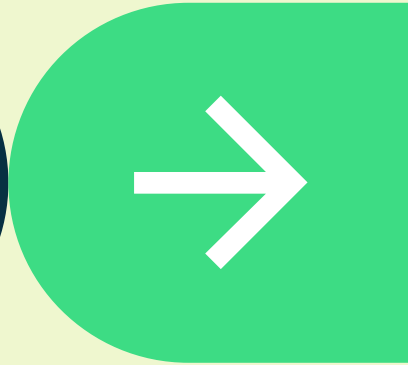
For MOBIKE case, the requested target MTU is `rounded down` to the nearest cipher block boundary to determine the actual tested MTU.

Server Setup

A vanilla “SWAN” server on GCP

Proprietary and Confidential

IPsec Lossless MOBIKE



Proprietary and Confidential

Marginal Link Handover Dilemma

01 OPTION A: STAY ON LINK

Stay on Marginal Connection

The device remains attached to a deteriorating Wi-Fi or Cellular path.

Key Risks & Costs:

- High latency & packet loss
- Poor link reliability during degraded state

02 OPTION B: SWITCH LINK

Blind Path Switchover

The device switches to an unused alternate link (e.g. Wi-Fi → Cell).

Key Risks & Costs:

- Target link quality is unverified while unused
- High transition costs without quality guarantees

CORE DILEMMA: Neither option guarantees path quality without pre-switchover probing and validation.

Proprietary and Confidential

General Approach: Soft Handover

TENET 01

Minimize Processing Overhead

Leverage existing IPsec architecture with “only SPI” identification.

Utilize only a single ESP SA.

TENET 02

Broad Usability

- Server handovers / failovers
- IP address rotation
- NAT expiry
- Road warriors

TENET 03

Flexibility

Flexible signalling to support varying levels of support.

No need for one-size-fits-all approach

TENET 04

Client Control

Process is primarily initiated and controlled by the initiator/client to manage soft handover smoothly.

Server-supported, options for server steering.

Proposal

STEP 01

Negotiation

Peers exchange `BONDED_DYNAMIC_UPDATE` in IKE_INIT—if unsupported, falls back to standard MOBIKE.

STEP 02

Client Trigger

Valid uplink ESP packets from a new path trigger the server to duplicate downlink traffic.

STEP 03

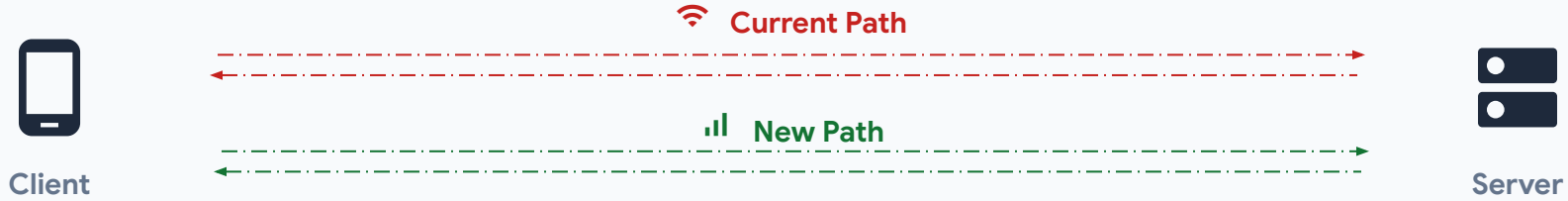
Handoff

Client completes handover by sending IKEv2 `UPDATE_SA_ADDRESSES` on preferred link.

STEP 04

Flow Cleanup

Upon receiving the MOBIKE request, the server stops packet duplication.



Proposal

Bi-directional Bonding

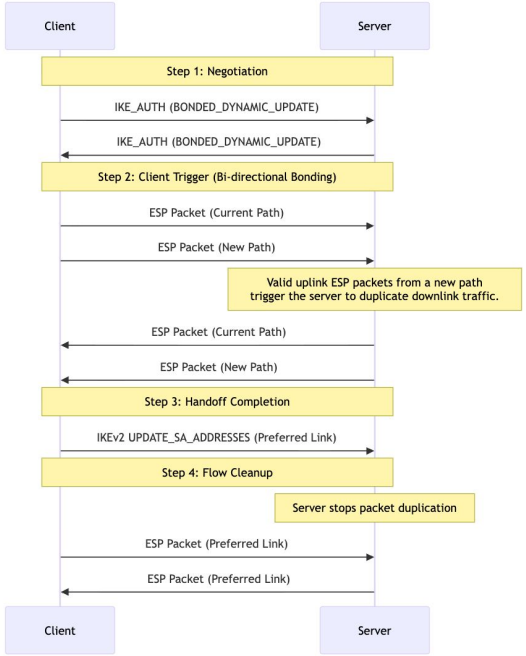
Client Bonding Supported

Client duplicates uplink traffic, enabling bi-directional packet duplication across both links.

Downlink-Only Bonding

Client TX Unsupported

Client is bonding-aware but cannot duplicate uplink. Duplication is handled entirely by the server on the downlink.



Sequence Diagram of Bi-directional Bonding

Proprietary and Confidential

Linux Kernel Changes - Input

Proposal Overview

Allow RX SAs without DADDR Match

There is a mandatory DADDR match in __xfrm_state_lookup().

Proposal Options

- ➔ **[PREFERRED]** Add a new SA flag for input SAs like `XFRM_STATE_DADDR_WILDRCV or XFRM_STATE_SPI_ONLY`.
- ➔ **[ALTERNATE]** Reused the existing `XFRM_STATE_WILDRCV`.
- ➔ **[ALTERNATE 2]** Define a special value for the DADDR, (e.g. 0.0.0.0 and ::); then do a lower-priority match if the initial lookup fails.

RFC Compliance

"For inbound processing, the SPI alone (or SPI + Protocol) MAY be used to look up the SA in the SAD, provided the receiving system ensures that SPIs are unique across all SAs." - RFC4301

While this language is idealistic, it lacks the pragmatism of a receiver with multiple interfaces, each with multiple IP addresses, support for firewall marks, etc.

Suggest not to not change the default level of permissiveness.

Kernel Changes - Output ???????

Implementation Options Evaluation

1. [PREFERRED] XFRM Integration

Leverages native IPsec processing framework for optimal performance and cleaner state management.

2. [ALTERNATE] eBPF Implementation

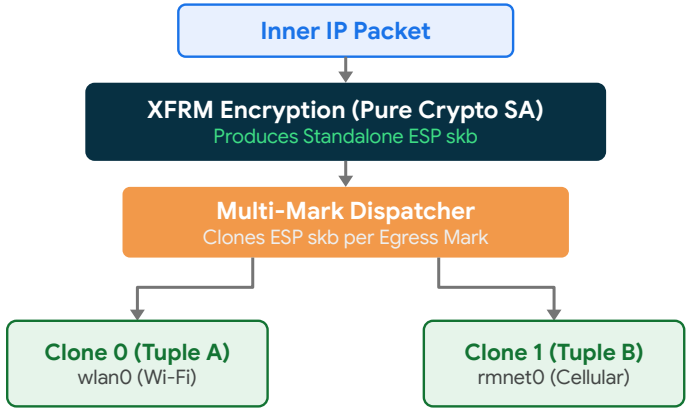
Complicated to implement and maintain within the existing socket-level filtering framework.

3. [ALTERNATE2] New Network Device

Slower performance due to repetitive removal and re-addition of headers. Highly redundant architecture.

XFRM: Outbound Duplication with Pure Crypto SA

IPsec payloads are encrypted once into standalone ESP packets. The kernel clones the packet for each assigned egress mark, routing independently.



Decoupled IPsec SA

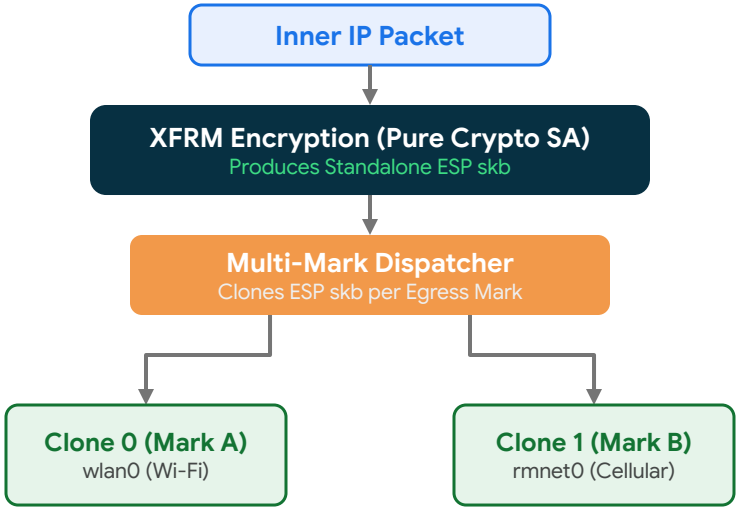
Outbound Egress Pipeline

Early ESP Encryption: The XFRM subsystem encrypts the inner IP packet using the pure-crypto SA, producing a standalone ESP packet.

Single Sequence Numbering: All duplicated copies of an ESP packet share the exact same ESP Sequence Number (SQN).

Multi-Mark Cloning: The XFRM output dispatcher inspects egress marks and clones the ESP skb for each path.

Policy Routing & Encap: Standard policy routing (ip rule) routes clones independently across physical interfaces (e.g., wlan0, rmnet0).



Decoupled IPsec SA

UAPI & SA Configuration

1. Pure Crypto SA (XFRM_STATE_UNBOUND)

SAs use wildcard IP addresses (`0.0.0.0`) with decoupled outer tunnel details.

2. Multiple Netlink Attributes

Attach multiple `XFRMA_SET_MARK_EXTENDED` attributes dynamically at runtime.

3. Policy Template Configuration

Wildcard SPD matches inner traffic against Policy Selector, Mark, and `if_id`.

Path-Specific Encapsulation

Userspace-Configured per Egress Mark

```
struct xfrm_set_mark_extended {
  struct xfrm_mark mark;
  struct xfrm_encap_tmpl encap;
  xfrm_address_t saddr, daddr;
  __u16 family;
};
```

Egress Execution

- XFRM inspects encap.encap_type and prepends the UDP 4500 header if needed

Problem: Risk of Replay Window Expiry

MECHANISM

How Replay Expiry Occurs

Latency gaps between bonded links trigger server-side packet loss during bi-directional bonding.

- Client replicates traffic on **current_link** (**fast**, high-loss) and **new_link** (**slower**, stable).
- **Faster path** packets arrive first, advancing the server's IPsec anti-replay window.
- **Slower path** duplicate packets arrive outside this window and are discarded.

IMPACT & CONDITIONS

Linux Replay Window Limits

Rare on mobile; primarily affects enterprise site-to-site VPN rekeys. With Android's **4096-packet** window, failure requires extreme scenarios:

Scenario A: High Throughput

400 Mbps+ at **100 ms** latency difference.

Scenario B: Moderate Throughput

100 Mbps+ at **500 ms** latency difference.

Mitigation: Replay Windows per Port & Address

To avoid single-window limits, track replay windows individually for each `<saddr, sport>` tuple.

ENTRY MECHANICS

Trigger & Entry Condition

Track a new path only if its initial packet on `<saddr, sport>` decodes successfully and falls within the current receive window of the forward-most active path.

CLEANUP MECHANICS

Exit Condition (Cleanup)

Discard tracking state based on:

- Inactivity timers
- Explicit `UPDATE_SA_ADDRESSES` exchange
- Successful receipt of all expected packets

There are three ways to implement this multi-window tracking.

Proposed Solution: Overlapping Windows

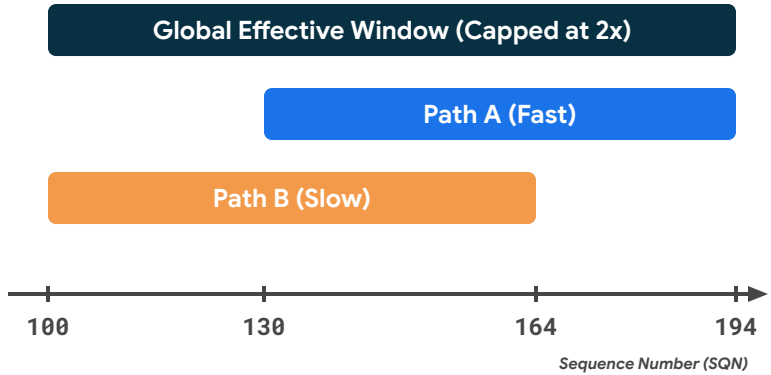
KEY MECHANICS

Simplified Tracking State

- **Mechanism:** Tracks global window from slowest path's bottom to fastest path's top.
- **Size Limit:** Strictly capped at `N * configured_size`.
- **Benefit:** Lagging slower paths automatically discarded without keeping multiple windows.
- **Drawback:** Potential packet loss on extremely slow paths if outrun.

VISUALIZATION (2 links)

Overlapping Windows Model



Alternative 1: Independent Windows with a Combined Tracker

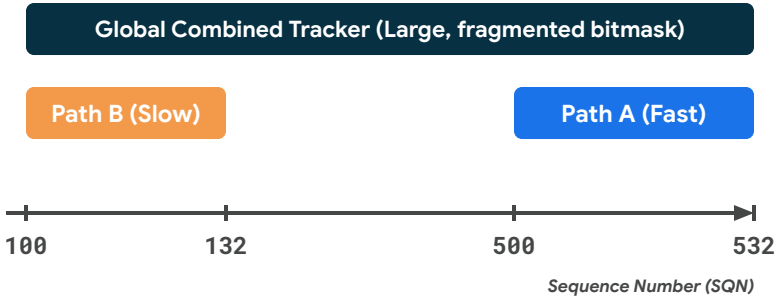
KEY MECHANICS

Independent Windows with a Combined Tracker

- **Mechanism:** Packets are validated against their **local path window** before updating the **global tracker**.
- **Cleanup Constraint:** No automatic **boundary-based cleanup** when windows become disjoint. Relies on **timers**.
- **Benefits:** Allows slow paths to lag significantly without packet drops.
- **Drawback:** Extremely **high engineering complexity** to maintain a massive, fragmented **global bitmask**.

VISUALIZATION

Independent Windows with a Combined Tracker



Alternative 2: Independent Windows without a Combined Tracker

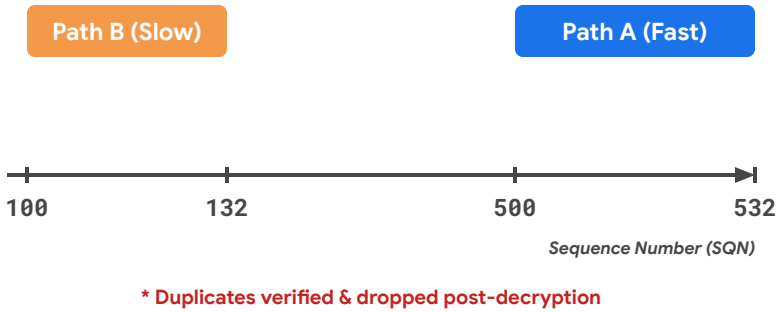
KEY MECHANICS

Independent Windows

- **No Global Tracker:** Independent receive windows per path.
- **Cleanup:** Relies strictly on timers or MOBIKE exchanges.
- **Benefits:** Slow paths lag without packet drops.
- **Drawback:** **Double-decryption overhead** (no deduplication).
- **Security:** Duplicates discarded post-ICV verification.

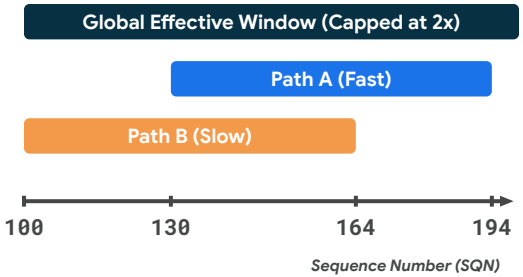
VISUALIZATION

Independent Windows Model

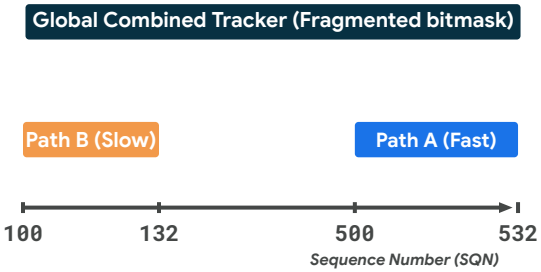


Compare Three Options

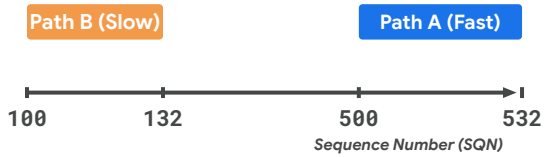
Overlapping Windows Model



Independent with Combined Tracker

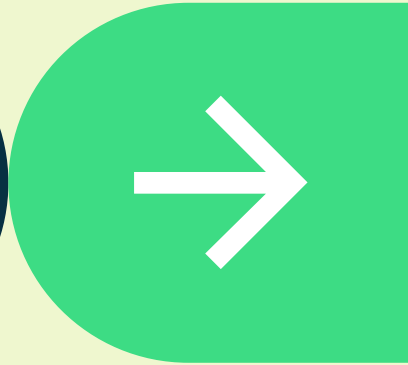


Independent Windows Model



Proprietary and Confidential

ESP-Based Link Error Detection



ESP-based Link Error Detection

Versus connection-based detection and tracking:

01

Ordered Stream

Tunneled packets all flow through a single ordered stream.

02

Easy Detection

Detects aggregate/link level failures much more easily.

03

Coordinated Recovery

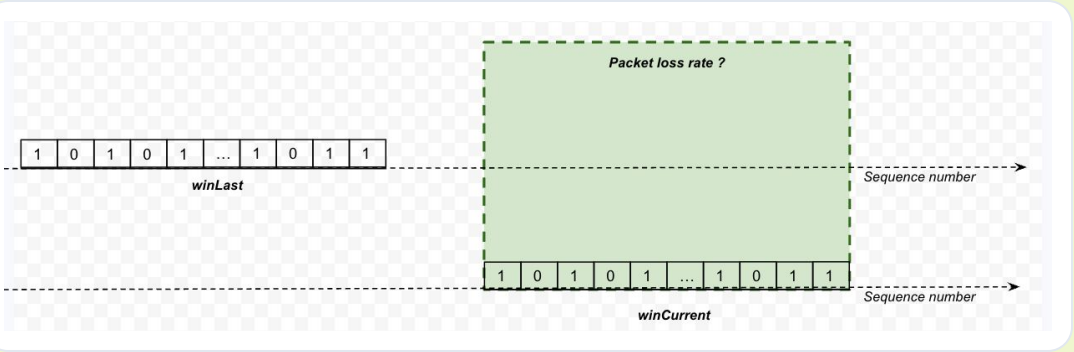
Recovery of global/link-level events is coordinated in a single component (IKE).

ESP-based Link Error Detection

Replay-window-based detection algorithm

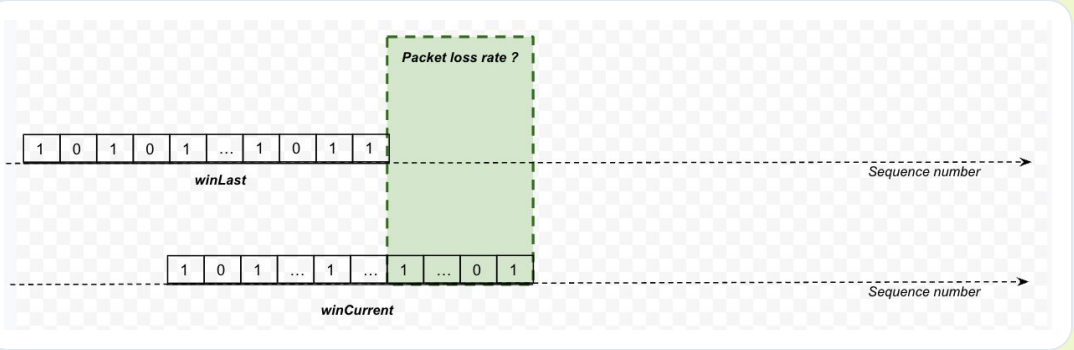
Non-Overlapping Windows

- **Expected Packet Count:** Window size.
- **Actual Received Count:** Total packets in the current window.



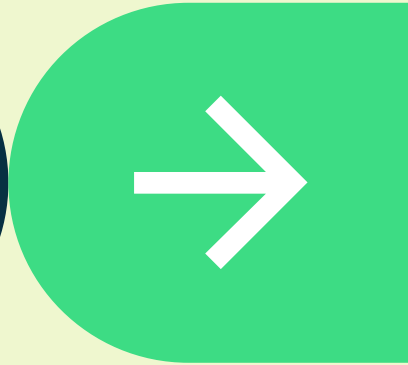
Overlapping Windows

- **Expected Packet Count:** Change in highest sequence number between last and current snapshots.
- **Actual Received Count:** Subset of current window between last and current highest sequence numbers.



Proprietary and Confidential

IPsec Port Agility



Proprietary and Confidential

IKEv2/IPsec Port Agility

Problem Statement

- **Firewalls block IPsec ports 500/4500**
- Power saving encourages switching between TCP & UDP; TCP ports like 443 tend to be optimized

Alternative Option

Limited side-channel for IKEv2 "wakeups" ONLY.

E.g., only used for Dead Peer Detection (DPD)?

Proposed RFC7296 Language Changes

IKE **typically** runs over UDP ports 500 and 4500, and implicitly sets up ESP/AH associations...

An implementation MUST accept incoming requests even if source port is not 500/4500, **MAY accept connections over another port if negotiated out-of-band**, and MUST respond to the address, **protocol**, and port received.

It MUST specify the address and port at which the request was received as the source address/port in response.

Proprietary and Confidential

Thank you