DECENTRALIZED
AI-Assisted
Peer-to-Peer
Messaging Protocol
Using Fractal Propagation
with
LOW Footprint
&
Complete Privacy

I.F.P.P:

INTELLIGENT FRACTAL PROPAGATION PROTOCOL (IFPP) A PROTOCOL ORIGINATING FROM THE MAMAWMAIL PROJECT WHITEPAPER

SEPTEMBER 1, 2025 (SEPARATED FROM MAMAWMAIL WHITEPAPER)

ENGR JUAN CARLOS G AYENG,
BACOLOD CITY, THE PHILIPPINES

The Long Overdue Internet Protocol Revolution

I.F.P.P. PROTOCOL

OCTOBER 1, 2025

Good Day.

My name is Juan Carlos from Bacolod City, the Philippines. I have submitted the IFPP Proposal last June 20, 2025, and have developed it further from its original form.

The first form was mainly about a novel idea I had nursed for some time, in response to thinking about improvements in the current internet packet communications, from OSI using TCP/IP, to its sending packets.

As I continued to research, I have realized that the infrastructure was an incremental improvement of the mindset that is no longer true today—that of transmission limitations. The current reality and environment now is that the saturation of interconnected devices renders the initial assumptions in the 1970s to the 1990s are no longer true. The TCP/IP was designed to have intelligent Routers, and Dumb Networks. Once the packets are in the network, there is no memory, control, or guidance, except when it reaches a router with a table. And that table in a router is the bottleneck that, while provided a direction for the packet, is now used as a centralized and censorship bottleneck with packets transparent for manipulation.

It is about time to design the new internet based on current realities. This new internet is going to be about intelligent packets that are self-guiding, that hops between devices, carries an encrypted payload that the carriers of the packets cannot see—but only its intended destination device.

MAMAWMAIL will be the first application that can, but will no longer be dependent on OSI to encapsulate a packet designed for dumb networks.

And as path and message saturation increases between devices and the swarm can handle ‘pseudo-session’ multicasting, the current setup for a centralized server, centralized websites, centralized table digests will disappear. IFPP will truly become resilient, opaque, uncensorable communication networks for people using their devices.

Websites, crawlers for search, sniffing, forwarding, injections, will all be relegated to the past architecture.

It will be a totally different global communication network. It will be an unstoppable messaging connection between devices. And as saturation grows, it will be an unstoppable ‘session’ chat between multiple devices. And as saturation goes beyond the minimum latency for multicasting pseudo sessions for multiple devices—it will replace client server websites, news aggregation sites, and other centralized clearinghouses of both ‘pages’ and ‘messaging’.

IFPP is the first step for a new global communication infrastructure suited for the next form of the internet.

Thank you.

Engr, Juan Carlos G. Ayeng, RN
AUTHOR

MESSAGE:

It is a privilege to see JC's ideas on this whitepaper finally written down after having lived in his mind for some time.

While not a foregone conclusion, as an entrepreneurial spirit I have been inspired by these ideas - and want to see it in the world. As such, encouraging JC to share this and invite collaboration to explore wherever it leads.

He proposes a decentralized design that challenges some of the deepest assumptions baked into the internet's foundations and invites us to imagine how communication could evolve if the assumptions around its constraints were no longer true.

Whether or not IFPP and MAMAWMAIL ultimately collides with reality as envisioned, the thinking behind it deserves careful attention.

It is a reminder that breakthroughs often begin not as polished products, but as bold hypotheses put forward for others to test, refine, and build upon.

I hope you approach this work with curiosity and rigor, and that it sparks the kind of exploration and dialogue that ideas with potentially far-reaching implications deserve.

RAYMUND ED DOMINIC Y. BERMEJO
FRACTIONAL COO & FOUNDER ACCELERATOR

MESSAGE:

MODERN NETWORKS VERIFY MESSAGE DELIVERY BUT NOT THE PATH IT TOOK, LEAVING CRITICAL ISSUES OF ACCOUNTABILITY, AND TAMPER-RESISTANCE UNADDRESSED.

THIS CREATES A FUNDAMENTAL TRUST GAP IN SENSITIVE ENVIRONMENTS.

THIS WORK INTRODUCES A PROTOCOL THAT ESTABLISHES VERIFIABLE AND RESILIENT COMMUNICATION THROUGH HOP-BY-HOP CUSTODY TRANSFER, REVERSE TRACEBACK, AND SWARM-BASED OVERSIGHT.

BY HAVING EACH DEVICE RECORD ITS ROLE IN A TAMPER-PROOF LEDGER, THE SYSTEM EMBEDS INTEGRITY AND TRANSPARENCY BY DESIGN.

WHILE THERE ARE CERTAINLY CHALLENGES, THIS FRAMEWORK OFFERS A FOUNDATION FOR TRUST IN CRITICAL DOMAINS LIKE DEFENSE, AND, OR DISASTER RESPONSE.

JOHN HENRY MARSHAL G. MANALO
IT CONSULTANT/PROFESSIONAL

0.0 ABSTRACT

*The **Intelligent Fractal Propagation Protocol** (IFPP) is a protocol-level innovation designed to reimagine communication beyond the constraints of TCP/IP and classical networking models.*

Originating from the MAMAWMAIL project but designed for broader applicability, IFPP replaces passive packet delivery with autonomous, intelligent propagation.

*Each message packet carries a seven-member "Angelic Army: **Scout Ranger team**"—specialized roles (Recon, Point, Leader, Heavy, Radio, Intelligence, Sweeper) that collectively enable adaptive routing, security, and resilience in unstable environments.*

Propagation in IFPP follows a fractal model: packets explore multiple paths through controlled duplication, while "been-here" flags prevent endless loops. Every handoff is independently confirmed by Gabriel signals, which sweep the network to validate successful transfers, compile immutable path digests. *These digests become **shared swarm intelligence**, allowing the network to remember and evolve.*

---

*The result is a protocol where routing is no longer dictated by central tables or static addresses, but by the **collective memory** and **real-time decisions of the swarm itself**.*

In contrast to TCP/IP's deterministic, infrastructure-dependent pathfinding, IFPP operates as a digital guerrilla protocol: packets adapt like small units, leveraging consensus and swarm knowledge to ***find their way even in disrupted or resource-constrained*** *environments.*

This makes IFPP suitable for delay-tolerant networks, peer-to-peer swarms, IoT collectives, and post-disaster communications.

While MAMAWMAIL serves as the reference application, IFPP stands ***independently as a protocol framework*** *for a new era of communication: resilient, intelligent, and self-organizing at the edge of the network.*

1.0 INTRODUCTION

The Internet is built on the TCP/IP protocol stack, which was revolutionary in its time but is fundamentally limited for the challenges of today. TCP/IP was designed in an era where networks were fragile, nodes were scarce, and "best effort" was sufficient. Its assumptions are now outdated:

- ***TCP/IP today**:*

  - ✓ **Dumb packets, smart routers**. Packets blindly hop until TTL expiry, with reliability patched by retransmission.
  - ✓ **Stateless paths.** No memory of journeys; routes vanish once packets disappear.
  - ✓ **Passive envelopes.** Easy to intercept, trivial to drop, impossible to verify once lost.

Sending a packet in TCP/IP is like releasing a cowboy into the wild west—once gone, its fate is unknowable.

- ***IFPP** tomorrow: a device-centric, intelligent protocol. Packets are self-guiding couriers with memory, decision-making, and persistence. They do not expire but continue until delivery is achieved. The network itself becomes intelligent: headers and decision states guide routing at every hop.*

  IFPP re-centers intelligence at the packet level:

  - ✓ ***Packets as couriers.** Each carries memory, decision logic, and persistence.*
  - ✓ ***Sacred persistence.** Packets do not expire until delivery is achieved.*
  - ✓ ***Swarm intelligence.** Path digests and **Archangel Gabriel Messenger** Signals confirmations allow the swarm to retain collective memory of successful routes.*

Crucially, IFPP did not emerge in isolation. It was conceived within the MAMAWMAIL project, which remains its reference implementation. However,

IFPP is designed as a ***standalone, general-purpose protocol***, open to other systems and independent implementations.

IFPP IS DEVICE-AGNOSTIC

*Key Innovation: IFPP introduces concepts such as **SACRED PERSISTENCE**, **ANGELIC ARMY CRAWLERS**, and **ARCHANGEL GABRIEL SIGNALS** to transform packet propagation from a chaotic best-effort gamble into an intelligent, persistent, swarm-guided process.*

*MAMAWMAIL implements IFPP, but the protocol itself is meant to **EVOLVE** into a **STANDARD** that outlives any single application. The goal is to **evolve IFPP into a standard**, beyond any single app, as the foundation for next-generation resilient communication.*

2.0 CONCEPTUAL FOUNDATIONS

THIS SECTION OUTLINES THE PRINCIPLES THAT UNDERPIN IFPP. WHILE CONCEIVED WITHIN THE MAMAWMAIL PROJECT, THESE IDEAS STAND INDEPENDENTLY AS A PROTOCOL-LEVEL INNOVATION.

2.1 **TCP/IP TODAY**: SMART ROUTERS, DUMB NETWORKS

TCP/IP treats packets as disposable, dumb envelopes.

They expire after TTL, routes remain opaque, interception is trivial, and there is no memory of prior journeys.

Sending a packet is like releasing a rider into the Wild West: once gone, its fate is unknown.

2.2 **IFPP TOMORROW**: INTELLIGENCE IN THE NETWORK

*IFPP transforms packets into **intelligent couriers** with **persistence** and **decision-making capacity**.*

*Headers and decision states guide routing. Messages never expire until delivery — **Sacred Persistence** replaces Time-To-Live.*

2.3 BUILT ON **MODERN CONNECTIVITY**

*Unlike TCP/IP's origins in fragile networks, IFPP is only possible today, when **devices are abundant**, **interconnectivity is dense**, and **AI/metadata layers can enrich routing**.*

*IFPP **leverages current infrastructure** rather than reinventing it from scratch.*

2.4 FROM CHAOS TO SWARM INTELLIGENCE

Legacy TCP/IP: expiring packets, hidden paths, untraceable routes.

IFPP Paradigm:

1. *Swarm-guided;*
2. *Memory-rich;*
3. *Persistent propagation;*
4. *Each hop is informed;*
5. *Each route is recorded, and*
6. *Each message learns from the network.*

2.5 DEFINING INNOVATIONS OF IFPP

- **DEVICE-CENTRIC IDENTITY***: network operates at the device level, not the user level, ensuring anonymity.*

- **Total Modularization & Encryption***: only the destination device can decrypt payloads; intermediaries see opaque couriers.*

- ***Three-Module Payload Model****: separation of payload, routing metadata, and integrity signals.*
    - Module A: **ROUTING DIGEST**
    - Module B: **INTEGRITY & AUTHENTICATION**
    - Module C: **PAYLOAD FRAGMENT**
    - Two are retained as “road markers” while the third moves forward.

- **Angelic Army** *(Crawlers/Scouts): proactive crawlers that map candidate paths, prevent loops via the Been-Here Flag, and guide payloads safely.*

- **ARCHANGEL GABRIEL SIGNALS***:*

    1. **SUCCESSFUL HANDOFF SIGNAL***: verifies safe transfer and prunes redundant copies.*

    2. **DELIVERY SUCCESS SIGNAL***: propagates back with complete route digest, enriching the swarm's memory.* This is the TRUMPET SIGNAL/TRIUMPHANT SIGNAL;

- **SACRED PERSISTENCE***: packets persist until delivery, preserving the Eternal Message Core.*

- **HORIZONTAL SINGULARITY***: saturation of route knowledge across the swarm, making successful delivery near-certain.*

- **VERTICAL SINGULARITY***: convergence toward the minimal number of required copies per message.*

- **FRACTAL HOP / SINGULAR HOP** *Adaptation: routing dynamically balances redundancy vs. efficiency based on swarm knowledge.*

2.6 IFPP: DEFINING ITS COMPONENTS

WHILE IFPP WAS CONCEIVED IN THE MAMAWMAIL PROJECT, IT STANDS ON ITS OWN AS A PROTOCOL. ITS FOUR PILLARS EXPLAIN ITS SCOPE AND DESIGN PHILOSOPHY:

A. INTELLIGENT

Packets are decision-making couriers. Multi-header metadata allows routing inside the network without a centralized authority. Copies are pruned on handoff, and success signals return mapped paths. Payloads are fully encrypted, invisible to intermediaries.

B. FRACTAL

Adaptive propagation: IFPP can replicate packets fractally across multiple hops or condense into singular precision routing. This mechanism regulates swarm load while ensuring delivery reliability.

C. PROPAGATION

Like biological systems or flocks of birds, IFPP spreads messages in a self-organizing way. Packets propagate intelligently, delete obsolete copies, and share successful pathways across devices. Over time, the network moves toward emergent equilibrium, defined mathematically by Horizontal (route certainty) and Vertical (copy efficiency) Singularities.

D. PROTOCOL

A formal, extensible convention that supports diverse implementations. IFPP is not bound to Mamawmail alone; any system can adopt it. Enterprise systems may customize internal routing optimizations; community editions rely on distributed swarm intelligence. Payload modules are extensible to future applications.

3.0 GLOSSARY OF TERMS

BUILDING THE SHARED MENTAL MODEL OF IFPP

This section provides precise definitions of core terms and symbolic references used throughout IFPP and MAMAWMAIL. It serves two purposes:

To prevent misinterpretation by aligning readers to a common vocabulary.

To act as a conceptual springboard for the more technical sections that follow, such as architectures, propagation mathematics, and protocol comparisons.

Because IFPP introduces novel constructs that differ from conventional networking terminology (e.g., multicast, persistence, hop semantics), this glossary ensures evaluators approach the system with a coherent and unified frame of reference.

3.1 CORE PROTOCOL CONSTRUCTS

ETERNAL MESSAGE CORE [SOUL]
The immutable, encrypted payload at the heart of IFPP communication. Unlike headers, which evolve, the Eternal Message Core persists unchanged end-to-end. Only the destination device can decrypt it.

HEADER
The adaptive metadata envelope carried alongside the Eternal Message Core. Contains routing data, hop history, flags, and state markers (such as the Been-Here Flag).

HOP
A single relay event where a message moves from one device to the next. Hops are the atomic units of propagation.

HANDOFF
The act of successfully transferring responsibility for a message to a new device. Once confirmed, the prior copy is deleted, preventing uncontrolled duplication.

BEEN-HERE FLAG
A lightweight marker stored in local device state to indicate prior receipt of a packet. Prevents looping and stabilizes propagation.

3.2 PROPAGATION DYNAMICS

FRACTAL PROPAGATION / FRACTAL HOPS
Recursive branching in which a message spreads to multiple devices simultaneously during early hops. Growth is exponential but later transitions to singular hops to conserve resources.

SINGULAR HOPS
One-to-one propagation after fractal expansion stabilizes. Singular hops maintain reliability without flooding the network.

PROPAGATION CYCLE
The full sequence of hops, acknowledgments, and cleanup events for a single Eternal Message Core.

MULTICAST PATHING (PSEUDO-SESSION)
Distinct from TCP/IP multicasting: in IFPP, multicasting refers to creating a pseudo-session that supports two-way flows (forward + backward) along the same discovered path.

SINGULARITY EVENT
A critical convergence concept in IFPP that frames delivery assurance in mathematical terms. Two types are defined:

1. **HORIZONTAL SINGULARITY** – Models path optimization. It is the limit equation describing the number of discovered and active routes required such that effective message latency approaches zero.

2. **VERTICAL SINGULARITY** – Models redundancy optimization. It is the limit equation describing the number of message replications required such that delivery reliability approaches one.

Unlike simple termination conditions (e.g., when no further hops are available), singularity events define predictive thresholds that IFPP's adaptive AI seeks to approach. These thresholds ensure the protocol dynamically balances efficiency (minimal latency) and resilience (maximal reliability).

3.3 DELIVERY ASSURANCE AND CLEANUP

HANDOFF SUCCESS SIGNAL
Confirmation from the next device that a hop has succeeded. Triggers deletion of the prior payload copy.

DESTINATION DELIVERY TRUMPET SIGNAL
Final confirmation from the recipient that the Eternal Message Core has arrived intact. Propagates backward along the route, initiating cleanup and mapping.

ARCHANGEL GABRIEL
Metaphor for the delivery herald. Represents the trumpet of success signaling that the Eternal Message Core has safely reached its destination.

ACKNOWLEDGMENT SIGNAL
A reverse-path message confirming payload arrival. Powers cleanup operations and route learning.

DELETE SIGNAL
Back-propagated instruction that redundant local copies must be deleted. Ensures Sacred Persistence is upheld without uncontrolled storage growth.

SACRED PERSISTENCE
Principle that at least one Eternal Message Core survives until confirmed delivery. Opposite of TCP/IP "time-to-live."

REDUNDANCY WINDOW
Temporary state during which multiple payload copies coexist, ensuring robustness against disconnections. Collapses once delivery is confirmed.

3.4 NETWORK ENVIRONMENT CONCEPTS

CRAWLER / DAEMON

Subcomponents that discover neighbors, maintain neighborhood maps, and score candidate devices. They illuminate safe paths for propagation.

CANDIDATE DEVICE

A peer device identified as a possible next hop. Selection depends on scoring metrics from the crawler.

NEIGHBORHOOD MAP

The dynamic graph of nearby devices, continuously updated as nodes appear and vanish. Serves as substrate for fractal and singular hops.

ANGELIC ARMY

Metaphor for the swarm of crawlers and propagated payloads working in concert to light safe routes through the network.

4.0 IMPLEMENTATION

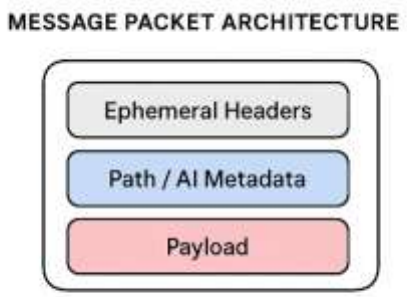
THIS SECTION DEMONSTRATES HOW THE INTELLIGENT FRACTAL PROPAGATION PROTOCOL (IFPP) INTEGRATES INTO THE SEVEN ARCHITECTURAL PILLARS OF THE MAMAWMAIL SYSTEM.

The Intelligent Fractal Propagation Protocol (IFPP) is not a standalone system but the propagation intelligence that binds decentralized communication networks together.
Its implementation is expressed through seven architectural interfaces, where IFPP provides the propagation logic, decision-making, and singularity optimization that higher-level systems (such as MAMAWMAIL) can build upon.

4.1. MESSAGE ARCHITECTURE

IFPP defines the structural rules for message construction:

1. EPHEMERAL HEADERS – carry routing metadata, been-here flags, and deletion rules.
2. PATH / AI METADATA – store path digests, propagation state, and swarm-learning hooks.
3. PAYLOAD ANCHOR – the immutable "eternal message core," which IFPP guarantees is preserved across all hops.



The message is structured in three modular parts: ephemeral headers, path / AI metadata, and the payload

By embedding fractal handoff rules into headers, IFPP ensures each packet is simultaneously traceable, ephemeral, and deletion-compliant.

4.2. PROPAGATION ARCHITECTURE

This is the core of IFPP.

1. FRACTAL EXPANSION – exploratory bursts across multiple candidates, creating redundancy and ensuring initial resilience.

2. SINGULAR COLLAPSE – narrowing propagation once a reliable path emerges, reducing latency and network overhead.

3. ADAPTIVE TUNING – machine learning balances exploration vs. exploitation in real time.

Machine learning within IFPP tunes the balance between EXPLORATION (fractals) and EXPLOITATION (singular paths).

4.3. CRAWLER ARCHITECTURE (THE ANGELIC ARMY)

IFPP consumes crawler outputs rather than *operating discovery directly*.

a. Crawlers:
   i. *map candidate devices,*
   ii. *evaluate hop viability, and*
   iii. *feed this intelligence into IFPP.*

b. IFPP then decides whether to branch fractally or commit singularly.

In practice, IFPP acts as the propagation brain interpreting crawler data, turning discovery into optimized delivery actions.

4.4. INTELLIGENCE LAYER (SWARM MEMORY)

IFPP is powered by ***distributed reinforcement learning***:

1. SUCCESSFUL PATHS → reinforced and shared across peers.
2. FAILED PATHS → decayed from memory to prevent retry loops.

The intelligence layer is where IFPP's runtime parameters are tuned:

1. max hop depth,
2. fractal-to-singular thresholds,
3. redundancy coefficients.

This ensures the swarm evolves dynamically, approaching optimal states without centralized orchestration.

4.5. SELF-HEALING / SELF-PRUNING ARCHITECTURE (ARCHANGEL GABRIEL)

IFPP guarantees controlled replication by embedding *handoff deletion and return-path confirmation* into every message cycle.

1. On successful delivery*, upstream copies are pruned*.
2. On failure, alternative routes are *explored until success*.
3. Reverse signals carry ***success digests*** that *reinforce swarm knowledge*.

In this role, IFPP is the execution layer of pruning logic, ensuring no uncontrolled duplication persists.

4.6. SINGULARITY ARCHITECTURE (CHAOS ATTRACTOR EQUATIONS)

IFPP formalizes delivery optimization as TWO limit equations that function as attractors for swarm learning:

1. HORIZONTAL SINGULARITY (Path Saturation):

$$\lim_{\{P \to P^*\}} L(P) \to 0$$

Latency L(P) approaches zero as path count P approaches the optimal saturation point P^*

2. VERTICAL SINGULARITY (Message Unit Saturation):

$$\lim_{\{M \to M^*\}} R(M) \to 1$$

Reliability R(M) approaches certainty as message redundancy M approaches the optimal saturation point M*

In IFPP, these attractors serve as *machine learning objectives:* the swarm constantly adjusts replication and routing *until the system **approaches** these thresholds*.

4.7. PRIVACY ARCHITECTURE

While IFPP itself is transport-agnostic, it enforces payload integrity through strict rules:

1. The ETERNAL MESSAGE CORE is encrypted and opaque to IFPP.
2. Headers and digests ***alone*** are available for routing intelligence.
3. Deletion rules prevent *stale payloads* from persisting in the swarm.

Thus, IFPP integrates seamlessly with higher-level cryptographic guarantees, ensuring privacy while still enabling swarm learning.

SUMMARY:

IN MAMAWMAIL, THE SEVEN LAYERS ARE SYSTEM ARCHITECTURE.

IN IFPP, THE SEVEN LAYERS ARE PROTOCOL INTEGRATION POINTS,

WHERE IFPP:

1. INJECTS PROPAGATION LOGIC,
2. PRUNING INTELLIGENCE, AND
3. SINGULARITY-DRIVEN OPTIMIZATION.

5.0 MESSAGE ARCHITECTURE

THE MESSAGE ARCHITECTURE DEFINES THE INTERNAL STRUCTURE, GUARDIANSHIP, AND PERSISTENCE OF EACH MESSAGE PACKET IN IFPP AND MAMAWMAIL.

UNLIKE TRADITIONAL INTERNET PACKETS, WHICH ARE INERT CARRIERS, THESE PACKETS EMBODY A LIVING TACTICAL ESCORT UNIT THAT ENSURES DISCOVERY, VALIDATION, PROPAGATION, AND DELIVERY.

EACH MESSAGE IS NOT JUST DATA — IT IS A PERSISTENT ENTITY, CARRYING WITHIN IT AGENTS, MEMORY, AND SAFEGUARDS DESIGNED FOR ASSURED ARRIVAL.

5.1. CORE PRINCIPLE

At the heart of IFPP and MAMAWMAIL is the idea that a message is not a disposable packet but a self-guarding unit with its own escort microarchitecture. Every device in the swarm holds dormant templates of these escorts, which are awakened when a message arrives.

Together, they preserve the Eternal Message Core and ensure safe delivery under the doctrine of Sacred Persistence.

5.1.1 ETERNAL MESSAGE CORE

THE ETERNAL MESSAGE CORE IS THE UNALTERABLE PAYLOAD. IT IS ENCRYPTED, IMMORTAL, AND PERSISTS UNTIL DELIVERY. IT CARRIES:

1. ***THE MESSAGE PAYLOAD.***
2. ***ANGELIC SHARDS THAT ACTIVATE ESCORT AGENTS IN EACH DEVICE.***
3. ***A BINDING IDENTITY ENSURING CONTINUITY ACROSS HOPS.***

5.1.2 MESSAGE COMPONENTS

THE MESSAGE IS COMPOSED OF THREE MAIN PARTS:

1. **EPHEMERAL HEADERS**
    A. *Carry routing metadata.*
    B. *Contain been-here flags and deletion instructions.*
    C. *Adapt dynamically with each hop.*
2. **AI METADATA**
    A. *Encodes adaptive heuristics for swarm routing.*
    B. *Contains device-neighbor scoring, path learning, and swarm-level memory digests.*
    C. *Helps the escort team decide efficient next hops.*
3. **PAYLOAD** ***(ETERNAL MESSAGE CORE)***
    A. *The immortal, encrypted data to be delivered.*
    B. *Custodied by the last man agent.*
    C. *Surrounded by the angelic army, ensuring it cannot be lost.*

It is ***within this payload that the angelic army is instantiated*** *— the tactical guardian unit that guides, protects, defends, directs and delivers the eternal message core.*

*5.2 THE **PHILIPPINE SCOUT RANGER** ANALOGY*

The **MESSAGE ARCHITECTURE** *draws inspiration from the tactical doctrine of the Philippine Scout Rangers, renowned for small-unit guerilla warfare. IFPP "…in honor of the Philippine Scout Rangers, whose 7-man rifle teams inspired this microarchitecture…"*

A typical 7-man Scout Ranger team I was taught and briefly a part of, in ROTC at the University of the Philippines, Diliman in 1991-1992 included:

1. **SCOUT** – *advances to locate threats and paths.*
2. **POINT MAN** – *first in contact, stabilizes engagements.*
3. **LEADER/OFFICER** – *makes tactical decisions.*
4. **HEAVY WEAPONS MAN** – *provides supporting firepower.*
5. **INTEL/RADIO MAN** – *maintains situational awareness and communication.*
6. **LIAISON** – *connects with external allies.*
7. **LAST MAN** – *secures the rear and ensures no loss of unit integrity.*

"The IFPP operates according to principles of small-unit tactics — autonomous maneuver, adaptive pathfinding, and self-defense at every step. Like a guerilla team, it analyzes its environment, decides its route dynamically, and ensures survival until the objective is secured.

***PHILIPPINE SCOUT RANGERS* COMPLETES ITS MISSION UNDER ANY CONDITION.**

***In honor of the Philippine Scout Rangers**, whose 7-man rifle teams embody these tactics, this microarchitecture is modeled as a living escort unit for each message packet."*

*5.3 **ANGELIC GUARDIANS** AND **GUIDES** (MESSAGE EQUIVALENT)*

Each message instantiates an Angelic Army, a 7-agent escort team, directly mapped from the Scout Ranger doctrine. These guardians embody specialized functions:

- *They scout, validate, replicate, log, and coordinate.*
- *They operate autonomously yet cohesively.*
- *They dissolve and re-form with each hop, carrying forward their mission context.*

Thus, every message is not just a packet but a living tactical unit, reborn at each step of its journey.

1. *Recon – passive scout, instantiates to candidate devices.*
2. *Point – inseparable escort, first into contact.*
3. *Leader – commander and decision-maker.*
4. *Intel – link to Gabriel, swarm awareness.*
5. *Heavy Weapons – security specialist.*
6. *Liaison – communicator with the host device.*
7. *Last Man – final custodian of the Eternal Message Core.*

They dissolve and reform with each hop, ensuring continuity and custody of the payload.

*5.4 THE SEVEN **ANGELS***

1. ***RECON** — Passive Scout*
    - *Acts only under orders from the Leader.*
    - *Instantiates to candidate devices (identified via Intel/Gabriel).*
    - *Checks been-here flags, logs, and traces.*
    - *Reports findings back to the Leader.*

2. ***POINT** — Payload Escort & Decoy*
    - *Never leaves the Eternal Message Core.*
    - *Goes in first during handoff, absorbing attacks or errors.*
    - *Protects, Manages, and Validates the payload transfer.*

3. ***LEADER** — Commander & Decision-Maker*
    - *Receives candidate lists (from Intel) and reports (from Recon, Liaison).*
    - *Chooses the next hop and issues orders.*
    - *Directs Recon where to instantiate, Point when to engage, and Liaison what information to exchange.*
    - *Does not communicate with Gabriel.*
4. ***INTEL** — System Link to Gabriel*
    - *One of only two angels allowed to talk to Archangel Gabriel.*
    - *Requests candidate device lists from Gabriel and hands them to the Leader.*
    - *Provides global awareness of swarm opportunities.*

5. ***HEAVY WEAPONS** — Security Specialist*
    - *Provides defense during hostile encounters.*
    - *Modular by design for independent security updates.*
    - *Protects the escort unit without affecting other modules.*

6. ***LIAISON** — Device Communicator & Knowledge Broker*
    - *Talks directly with the current host device.*
    - *Exchanges intelligence: paths, logs, metadata.*
    - *Reports device-level information to the Leader.*
    - *Shares back swarm-approved data to strengthen the node as instructed by the Leader*
7. ***LAST MAN** — Custodian of the Eternal Message Core*
    - *Sole custodian of the payload:[Eternal Message Core].*
    - *Waits for Gabriel's confirmation before deleting old copies.*
    - *Ensures only one living instance exists at all times.*
    - *Reinstantiates with the payload on the new device.*

Figure 5.1: Data Flow Diagram (DFD) *showing the forward path of a packet through IFPP.*

Circles represent process roles (agents), rectangles represent external entities, and open-ended rectangles represent data stores.

Arrows denote the flow of the Eternal Message Core and its adaptive headers.

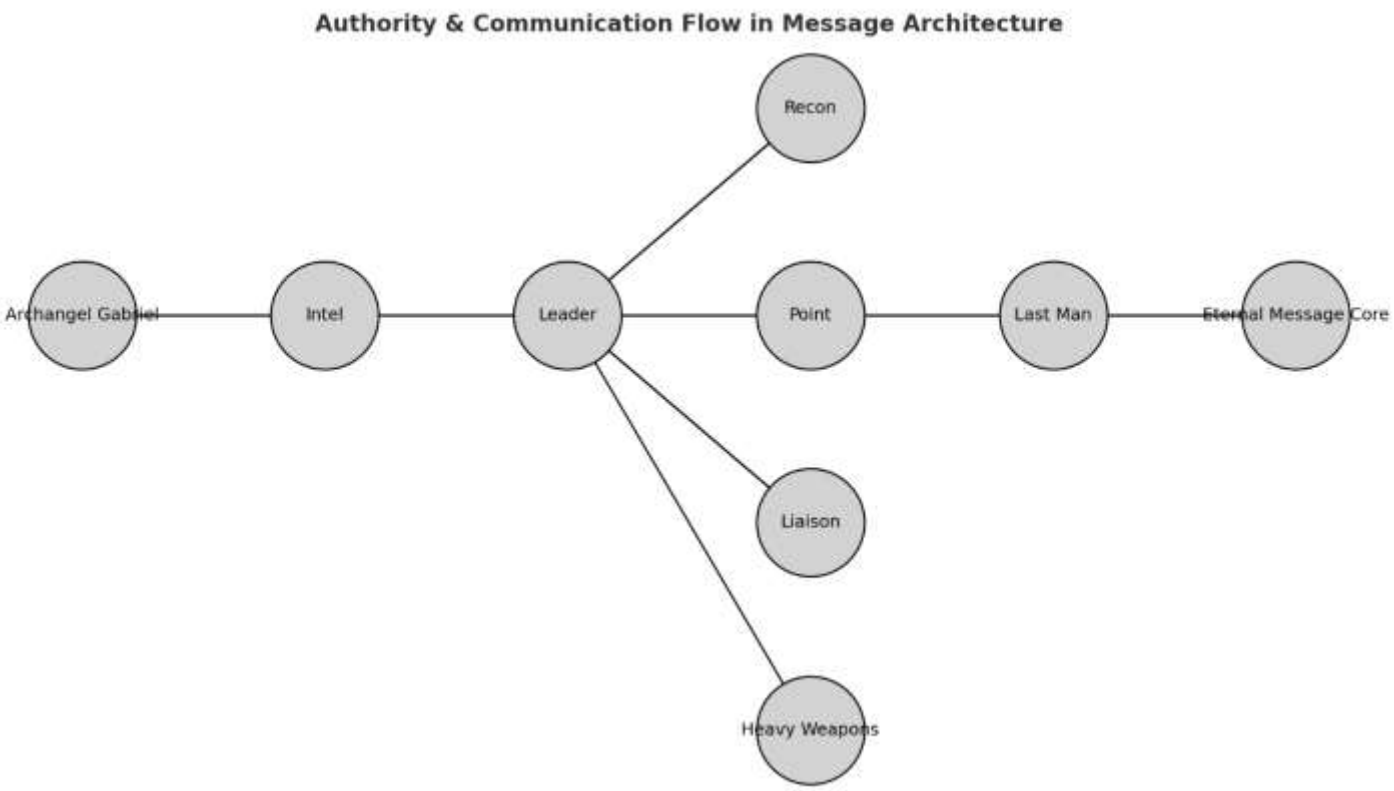


Figure 5.2. ***Authority and Communication Flow in Message Architecture***

This diagram maps the command and communication relationships among the escort agents of a message packet. Archangel Gabriel communicates only with Intel (to provide candidate device lists) and with the Last Man (to confirm successful handoff).

The Leader commands Recon, Point, Liaison, and Heavy Weapons but does not communicate with Gabriel directly.
Recon and Liaison act on the Leader's orders, returning intelligence for decision-making.
Point remains bound to the Eternal Message Core, acting as its escort, while
Last Man serves as the sole custodian of the payload, deleting the old instance only after Gabriel's confirmation.

This structure preserves strict chains of authority, prevents unauthorized decisions, and ensures that the Eternal Message Core is protected under the doctrine of Sacred Persistence.

*5.5 **ARCHANGEL GABRIEL** (SYSTEM-WIDE MESSENGER)*

GABRIEL exists outside the message, omnipresent across the swarm.

- *Continuously gathers nearby device lists.*
- *Provides candidate lists when Point requests them.*
- *Confirms successful handoffs.*
- *Signals the Last Man with the Trumpet call to authorize deletion of the old payload and rebirth in the new hop.*
- *Functions as a witness, truth-bearer[arbiter], and validator of every transition.*

Gabriel ensures that the system cannot be deceived by false hops or partial transfers — delivery is always authenticated.

*5.6 **SINGULARITY** AND GUERILLA MODEL OF PROPAGATION*

The swarm operates like a guerilla unit, agile and adaptive, compared to the rigid infrastructure of centralized servers and blind IP packets.

- *Internet Today: rigid central servers, blind packets without memory, time-limited TTL expiry.*
- *IFPP/MAMAWMAIL: **memory-enabled packets**, **tactical autonomy**, **adaptive fractal spread**, and **continuous persistence**.*
- *Singularity Principle: after sufficient propagation, the swarm converges into singular direct hops, conserving resources while preserving delivery certainty.*

This mirrors the guerilla doctrine: agile small units vs. massive rigid formations.

*5.7 **SACRED PERSISTENCE***

Unlike TCP/IP where packets "expire" (TTL) or delivery is probabilistic (send and hope), IFPP/MAMAWMAIL messages embody Sacred Persistence:

- *No message ever dies* **[*MESSAGES DO NOT EXPIRE*].**
- *Custody is continuous until delivery.*
- *Handoffs are only confirmed when Gabriel signals success.*
- *The Eternal Message Core* ***lives until it fulfills its mission***.

This reverses the philosophy of modern networking: instead of time-limited packets, IFPP ensures immortal delivery.

*5.8 **SUMMARY***

Key Takeaways:

- A MESSAGE IS NOT INERT — *it is a living architecture. It is alive: each is a tactical escort unit.*
- ***SEVEN ANGELS** = tactical escort unit derived from Scout Ranger doctrine. PACKET-LEVEL GUARDIANSHIP.*
- ***GABRIEL** = swarm-wide, independent messenger and validator.*
- ***SACRED PERSISTENCE** replaces TTL — messages never die.*
- ***GUERILLA SWARM DYNAMICS/PROPAGATION** replaces rigid centralized routing.*

This architecture ensures absolute delivery assurance, making IFPP/MAMAWMAIL fundamentally distinct from TCP/IP.

5.9 DECISION FLOW : **POST-HOP SEQUENCE**

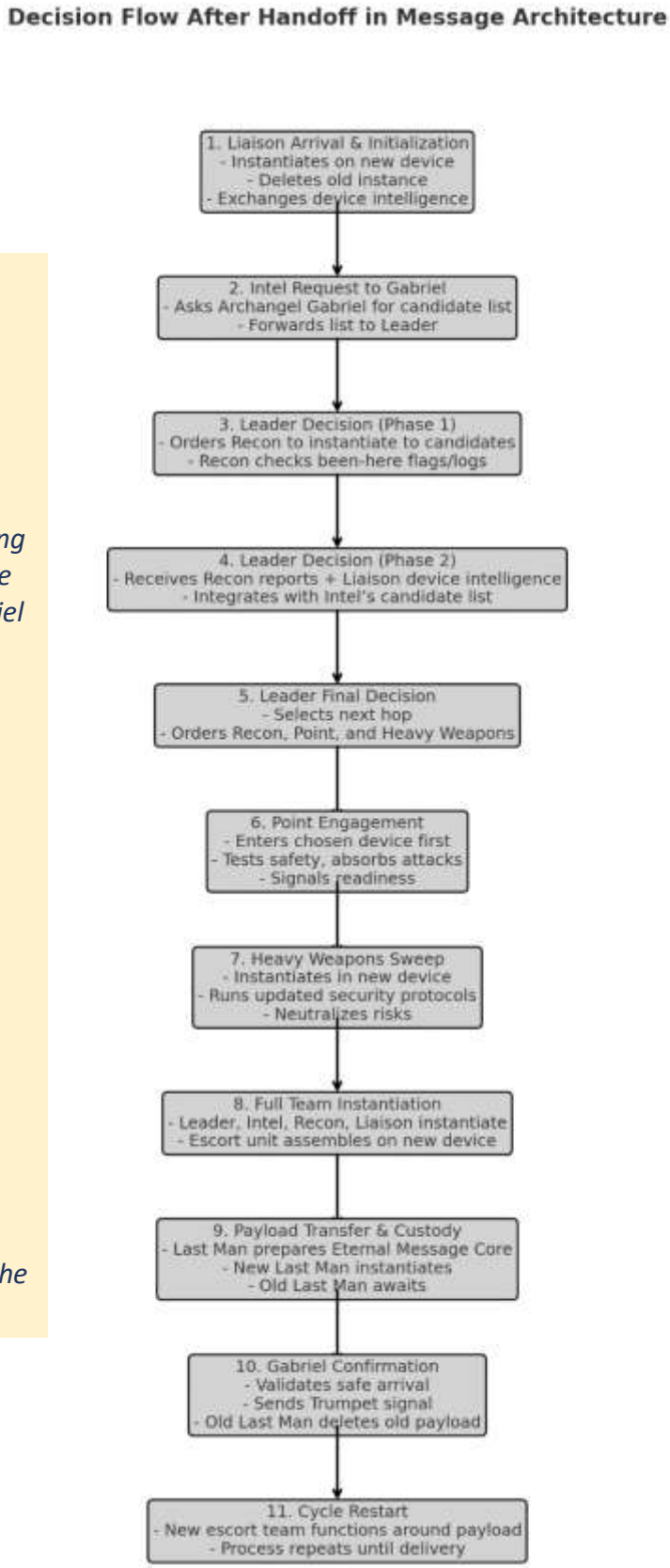


*Figure 5.3**. Decision Flow After Handoff in Message Architecture***

This diagram illustrates the sequence of tasks performed by the message's escort microarchitecture after a hop to a new device.

The process begins with the Liaison instantiating and initializing on the new host, followed by the Intel agent's consultation with Archangel Gabriel for candidate lists.

The Leader integrates intelligence from Recon, Liaison, and Intel, then issues hop instructions.

The Point enters the new device first to test safety, after which the Heavy Weapons agent sweeps for risks. Once the environment is secured, the full team instantiates around the payload.

The Last Man then handles custody transfer of the Eternal Message Core, awaiting Gabriel's independent confirmation before deleting the old copy.

The cycle restarts on the new device, ensuring continuous custody and Sacred Persistence of the message until final delivery.

1. ***LIAISON ARRIVAL & INITIALIZATION***
    - *Liaison instantiates on the new device.*
    - *Deletes its old instance on the previous device.*
    - *Begins device intelligence exchange (catalogs, logs, routing info).*

2. ***INTEL REQUEST TO GABRIEL***
    - *Intel asks Archangel Gabriel for the updated candidate list.*
    - *Intel forwards the list to the Leader.*

3. ***LEADER DECISION CYCLE*** *(Phase 1: Recon Orders)*
    - *Leader evaluates the candidate list.*
    - *Leader instructs Recon to instantiate to each candidate device for been-here flag and log verification.*
    - *Recon reports back findings.*

4. ***LEADER DECISION CYCLE*** *(Phase 2: Liaison Report)*
    - *Leader receives Liaison's device-level intelligence (host health, path data).*
    - *Leader integrates Recon's reports + Liaison's data + Intel's candidate list.*

5. ***LEADER FINAL DECISION***
    - *Leader selects the next hop device.*
    - *Issues instructions:*
        - *Recon to instantiate and prepare path.*
        - *Point to prepare for handoff.*
        - *Heavy Weapons to stand by for defense.*

6. ***POINT ENGAGEMENT***
    - *Point enters the chosen device first.*
    - *Tests environment safety, absorbs any attacks/errors.*
    - *If safe → signals readiness for payload transfer.*

7. ***HEAVY WEAPONS SWEEP*** *(New Step)*
    - *Heavy Weapons instantiates in the new device.*
    - *Runs updated security sweeps and countermeasure checks.*
    - *Neutralizes or flags risks before payload enters.*

8. ***FULL TEAM INSTANTIATION*** *(New Step)*
    - *Leader, Intel, Recon, and Liaison instantiate into the new device if they have not already.*
    - *Entire escort unit assembles around the payload in preparation for custody transfer.*

9. ***PAYLOAD TRANSFER & LAST MAN CUSTODY***
    - *Last Man prepares the Eternal Message Core for handoff.*
    - *Payload is instantiated in the new device with new Last Man.*
    - *Old Last Man waits for confirmation.*

10. ***GABRIEL CONFIRMATION***
    - *Gabriel independently validates that the Eternal Core + team arrived safely.*
    - *Gabriel sends Trumpet signal to old Last Man.*
    - *Old Last Man deletes the previous payload instance.*

11. CYCLE RESTART

- *New Liaison, Intel, Recon, Point, Heavy, Leader, and Last Man now function as the living escort team for the Eternal Core on the new device.*
- *Process repeats until final delivery.*

1. *Liaison arrival →*
2. *Intel consults Gabriel → 3–5 Leader evaluates reports →*
3. *Point enters →*
4. *Heavy sweeps →*
5. *Full team instantiates →*
6. *Payload transfer →*
7. *Gabriel confirms →*
8. *Cycle restarts.*

The Real Heroes, Philippine Scout Rangers. >>>

Scout Rangers are the Hybrid/ Combination of Alamo Scouts and US Army Rangers

They are the Guerillas of the Philippine Army – Light, Autonomous, Deadly, Battle-Hardened, Soft-spoken, Dressed in all black, the Warrior class of the Filipino People.

5.10 DEVICE-LEVEL PROCESS (ENTIRE APP INSTANCE)

EACH DEVICE RUNS A LOCAL MAMAWMAIL ENGINE THAT HOSTS THE ESCORT TEAM FOR EACH MESSAGE, COMMUNICATES WITH ARCHANGEL GABRIEL, AND MANAGES STORAGE, BANDWIDTH, AND POWER TRADE-OFFS.

*THE DEVICE OPERATES IN ONE OF TWO STATES: **SENDING** OR **RECEIVING**.*

GABRIEL OPERATES OUTSIDE BUT EXCHANGES DIGESTS AND INSTRUCTIONS THROUGH INTEL AND LAST MAN.

PSEUDOCODE

```csharp
csharp

DeviceProcess:
    while app_running:
        if has_new_message:
            instantiate EscortTeam
            begin ForwardPath()
        if receives_message:
            instantiate EscortTeam
            validate_and_integrate_payload()
        run GabrielIntegrityCheck()
```

PYTHON SKELETON

```python
class Device:
    def __init__(self, device_id):
        self.device_id = device_id
        self.escort_teams = []
        self.state = "idle"

    def send_message(self, payload):
        team = EscortTeam(payload, self)
        self.escort_teams.append(team)
        team.forward_path()

    def receive_message(self, payload):
        team = EscortTeam(payload, self)
        self.escort_teams.append(team)
        team.integrate()

    def integrity_check(self):
        Gabriel.check_integrity(self.device_id)
```

*5.11 **ANGELIC ARMY** – SEVEN-ANGEL "PHILIPPINE SCOUT RANGER" TEAM*

THE SEVEN ANGELS FORM THE TACTICAL MICROARCHITECTURE OF EACH MESSAGE. EACH IS MODULAR, CAN BE UPDATED INDEPENDENTLY, AND FOLLOWS STRICT COMMUNICATION LINES:

- *Intel & Last Man ↔ Gabriel.*
- *Recon, Point, Liaison ↔ Device & Leader.*
- *Leader issues all commands.*

PSEUDOCODE

```
EscortTeam:
    Recon -> check candidates
    Intel -> get candidate list from Gabriel
    Leader -> decide hop
    Point -> enter new device, secure payload
    Heavy -> defend against threats
    Liaison -> exchange metadata with host device
    LastMan -> await Gabriel trumpet, delete old copy
```

PYTHON SKELETON

```python
class EscortTeam:
    def __init__(self, payload, device):
        self.payload = payload
        self.device = device
        self.recon = Recon()
        self.point = Point(payload)
        self.leader = Leader()
        self.intel = Intel()
        self.heavy = HeavyWeapons()
        self.liaison = Liaison()
        self.lastman = LastMan(payload)

    def forward_path(self):
        candidates = self.intel.request_candidates()
        checked = self.recon.check(candidates)
        decision = self.leader.decide(checked)
        self.point.secure_transfer(decision)
        self.heavy.protect()
        self.liaison.exchange_info(decision)
        self.lastman.confirm_and_delete()
```

*5.12 **ARCHANGEL GABRIEL** – SWARM-WIDE MESSENGER*

GABRIEL IS THE **OMNISCIENT SWARM-WIDE SERVICE** THAT CONTINUOUSLY AGGREGATES DEVICE LISTS, VALIDATES SUCCESSFUL HANDOFFS, AND SERVES AS THE TRUTH-BEARER ACROSS THE SWARM.

IT ENSURES CONSISTENCY, DETECTS PARTITIONS, AND SIGNALS DELETION OF REDUNDANT PAYLOADS.

PSEUDOCODE

```
GabrielProcess:
    loop:
        collect device digests
        update candidate lists
        validate handoff(success)
        send trumpet signal to Last Man
        perform swarm integrity check
```

PYTHON SKELETON

```python
class Gabriel:
    device_registry = {}
    route_digests = {}

    @classmethod
    def update_devices(cls, device_id, metadata):
        cls.device_registry[device_id] = metadata

    @classmethod
    def provide_candidates(cls):
        return list(cls.device_registry.keys())

    @classmethod
    def confirm_handoff(cls, message_id, from_device, to_device):
        cls.route_digests[message_id] = (from_device, to_device)
        return True  # trumpet signal to Last Man

    @classmethod
    def check_integrity(cls, device_id):
        # ping devices, remove dead ones
        pass
```

*5.13 **HOP PROCESS** [TWO-DEVICES + GABRIEL]*

A HOP INVOLVES **TWO DEVICES AND GABRIEL'S OVERSIGHT**.

ON THE SENDING DEVICE, THE LEADER ORDERS RECON TO SCOUT, POINT TO PROBE, HEAVY TO SECURE, AND LIAISON TO EXCHANGE METADATA.

THE LAST MAN AWAITS GABRIEL'S TRUMPET BEFORE DELETING THE PAYLOAD.

ON THE RECEIVING DEVICE, A NEW ESCORT TEAM INSTANTIATES TO HOST THE ETERNAL MESSAGE CORE.

PSEUDOCODE

```
HopProcess:
    SenderDevice:
        Intel -> ask Gabriel for candidate list
        Leader -> assign Recon to check
        Recon -> returns findings
        Leader -> orders Point to enter
        Point -> escorts payload into Receiver
        Heavy -> defends during entry
        Liaison -> exchanges metadata
        LastMan -> waits for Gabriel confirmation

    Gabriel:
        confirms handoff
        signals new Last Man
        signals old Last Man to delete

    ReceiverDevice:
        EscortTeam instantiated
        Payload integrated
        Device joins swarm registry
```

PYTHON SKELETON

```python
def hop(sender, receiver, payload):
    # sender side
    candidates = sender.escort_teams[-1].intel.request_candidates()
    checked = sender.escort_teams[-1].recon.check(candidates)
    decision = sender.escort_teams[-1].leader.decide(checked)

    # point leads transfer
    sender.escort_teams[-1].point.secure_transfer(receiver)

    # gabriel confirms
    if Gabriel.confirm_handoff(payload.id, sender.device_id, receiver.device_id):
        sender.escort_teams[-1].lastman.delete_old()
        receiver.receive_message(payload)
```

AI-Generated Recon Member -Angelic Army

PSR carried the M-14, essentially an Automatic Garand, Heavy Wood

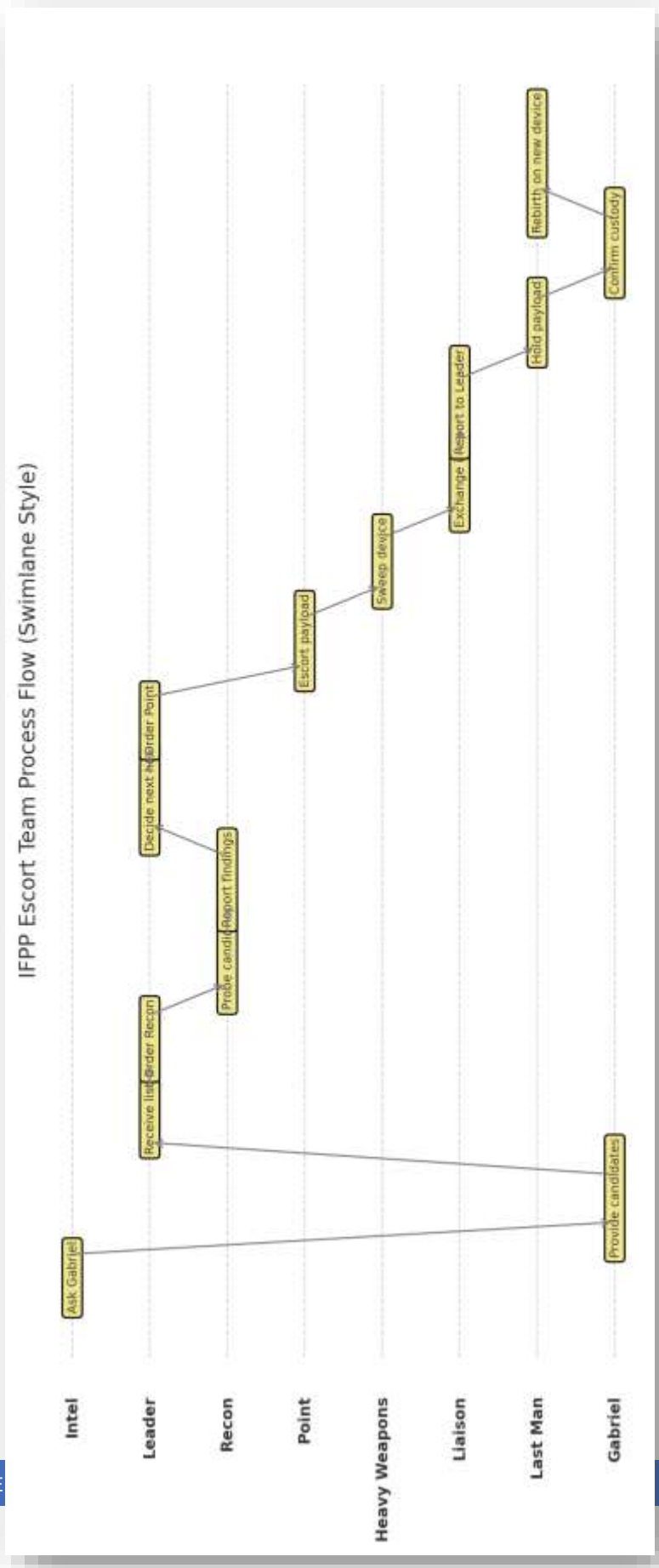


Figure 5.4. IFPP escort team responsibilities
(swimlane view).

"Each swimlane corresponds to one angel, showing how tasks are distributed and synchronized across the microarchitecture."

5.14 SUMMARY OF FIGURES 5.1–5.3

Together, Figures 5.1 through 5.3 provide a layered representation of the Message Architecture. Figure 5.1 establishes the sequential flow of packet propagation across the seven-agent escort team. Figure 5.2 clarifies the authority and communication relationships that govern custody, intelligence, and validation. Figure 5.3 details the decision-making cycle that occurs during each hop, from Liaison initialization to Gabriel's confirmation. Viewed as a whole, these diagrams demonstrate that the Eternal Message Core is not carried by passive packets but by a structured, self-guarding architecture that embodies Sacred Persistence.

THE MESSAGE ARCHITECTURE ESTABLISHES THE AGENTS, ROLES, AND CUSTODY PRINCIPLES THAT MAKE IFPP DISTINCT FROM TRADITIONAL PACKET-BASED PROTOCOLS.

HAVING DEFINED THE ESCORT TEAM, AUTHORITY FLOWS, AND THE DOCTRINE OF SACRED PERSISTENCE, THE NEXT STEP IS TO FORMALIZE HOW THESE ELEMENTS BEHAVE IN PRACTICE.

SECTION 6 PRESENTS THE PROPAGATION LOGIC AND FORMAL MODELS OF IFPP, WHERE PACKET MOVEMENT IS EXPRESSED IN TERMS OF DATA FLOW DIAGRAMS, FINITE STATE MACHINES, MATHEMATICAL REPRESENTATIONS, AND ALGORITHMIC PSEUDOCODE.

THIS PROGRESSION BRIDGES THE CONCEPTUAL DESIGN OF ANGELIC GUARDIANSHIP WITH THE RIGOROUS FORMALISM OF NETWORKING SCIENCE.

6.0 PROPAGATION LOGIC & FORMAL MODELS

THE PROPAGATION OF AN IFPP MESSAGE IS DEFINED NOT BY STATIC ROUTES OR EXPIRING LIFETIMES, BUT BY DYNAMIC CYCLES OF CUSTODY, CONFIRMATION, AND REBIRTH.

EACH HOP IS GUIDED BY THE SEVEN-AGENT ESCORT TEAM AND VALIDATED BY ARCHANGEL GABRIEL, ENSURING THAT NO TRANSFER OCCURS WITHOUT VERIFICATION.

IN THIS SECTION, THE PROTOCOL'S BEHAVIOR IS REPRESENTED FORMALLY THROUGH LOGICAL FLOWS, FINITE STATE MACHINES, AND MATHEMATICAL MODELS.

THESE REPRESENTATIONS PROVIDE THE SCIENTIFIC BACKBONE OF IFPP, TRANSLATING ITS CONCEPTUAL MICROARCHITECTURE INTO THE LANGUAGE OF SYSTEM DESIGN, PROBABILITY, AND ALGORITHMIC PROCESS.

6.1. DATA FLOW REPRESENTATION – FORWARD PATH OF A PACKET

The forward propagation of a message packet in IFPP follows a structured sequence across the seven-agent microarchitecture. Each "angel" acts as a process node, ensuring that discovery, validation, redundancy management, state logging, external coordination, and cleanup occur in order. External entities (sender and destination) represent the origin and termination of the Eternal Message Core, while the Been-Here Flag is modeled as a data store used to prevent looping and redundant propagation.

Mermaid DFD

```
flowchart LR
    S[Sender Device] -->|Eternal Message Core + Header| R((Recon))
    R -->|Candidate Devices| P((Point))
    P -->|Validated Hop| L((Leader))
    L -->|Integrity Check Passed| H((Heavy Weapons))
    H -->|Fractal Copies| I((Intel))
    I -->|Update Been-Here Flags| BH[[Been-Here Flags]]
    I --> LIA((Liaison))
    LIA -->|External Device Exchange| LM((Last Man))
    LM -->|Delivered Message| D[Destination Device]
```

```mermaid
flowchart LR
  SD[Sender Device] -->|Eternal Core + Headers| R(Recon)
  R --> I(Intel)
  I -->|candidate list| G[Gabriel]
  G -->|candidate list| I
  I --> Ld(Leader)
  Ld -->|order probes| Rp(Recon Probe)
  Rp --> Ld
  Ld --> P(Point)
  P --> HW(Heavy Weapons)
  HW --> Lia(Liaison)
  Lia --> LM(Last Man)
  LM -->|request confirm| G
  G -->|Trumpet (confirm)| LM
  LM --> DD[Destination Device]
  subgraph Datastores
    BH[Been-Here Flags]
  end
  HW -->|update flags| BH
  Lia -->|update logs| BH
```

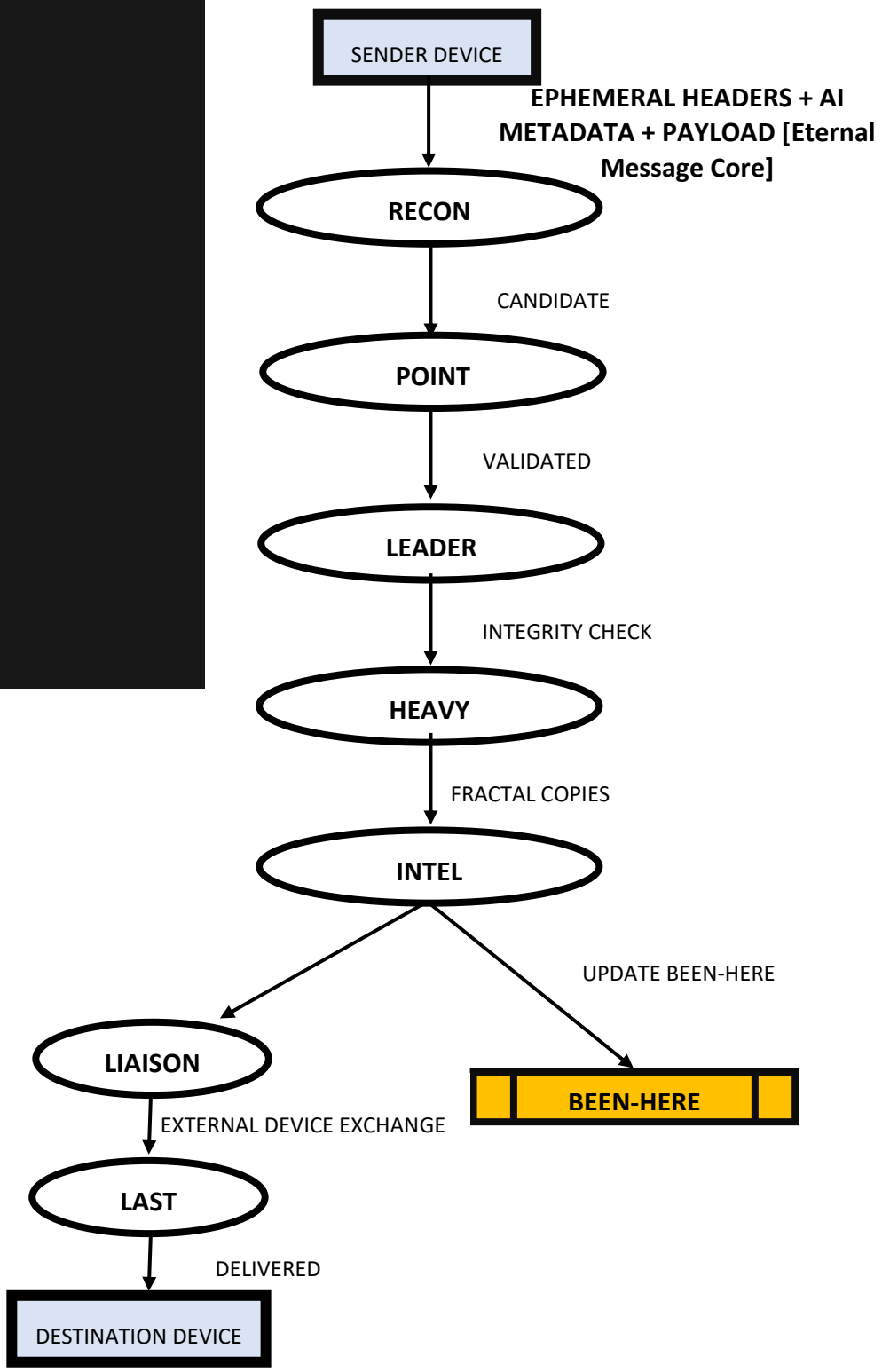


Figure 6.1: Data Flow Diagram (DFD) *showing the forward path of a packet through IFPP.*

Circles represent process roles (agents), rectangles represent external entities, and open-ended rectangles represent data stores.

Arrows denote the flow of the Eternal Message Core and its adaptive headers.

6.2 MESSAGE LIFECYCLE – FINITE STATE MACHINE [FSM]

A MESSAGE LIFECYCLE IN IFPP IS STATEFUL.

THE ETERNAL MESSAGE CORE PLUS ITS ESCORT TEAM TRANSITIONS THROUGH A SMALL SET OF STATES THAT CAPTURE CREATION, PROPAGATION, RE-INSTANTIATION, CONFIRMATION, AND FINAL DELIVERY.

UNLIKE TTL SYSTEMS, "EXPIRED" IS ONLY REACHED IF NO FURTHER PATH IS POSSIBLE (REPORTED TO GABRIEL) BUT NORMALLY MESSAGES PERSIST.

STATES:

1. ***CREATED*** *— message constructed, escort team instantiated.*
2. ***AWAITINGCANDIDATES*** *— Intel requests candidate list from Gabriel.*
3. ***CANDIDATEEVALUATION*** *— Recon/Point probe candidates; Leader decides.*
4. ***PROPAGATING*** *— Point/LastMan perform handoff to next device.*
5. ***REINSTANTIATED*** *— new device instantiates new escort/shards from Eternal Core.*
6. ***CONFIRMED*** *— Gabriel confirms custody; old instance deleted.*
7. ***DELIVERED*** *— destination device accepts Eternal Core; delivery is complete.*
8. ***FAILED/EXPIRED*** *— optionally used only when all reachable options exhausted and Gabriel flags unrecoverable partition.*

Mermaid (FSM)

mermaid

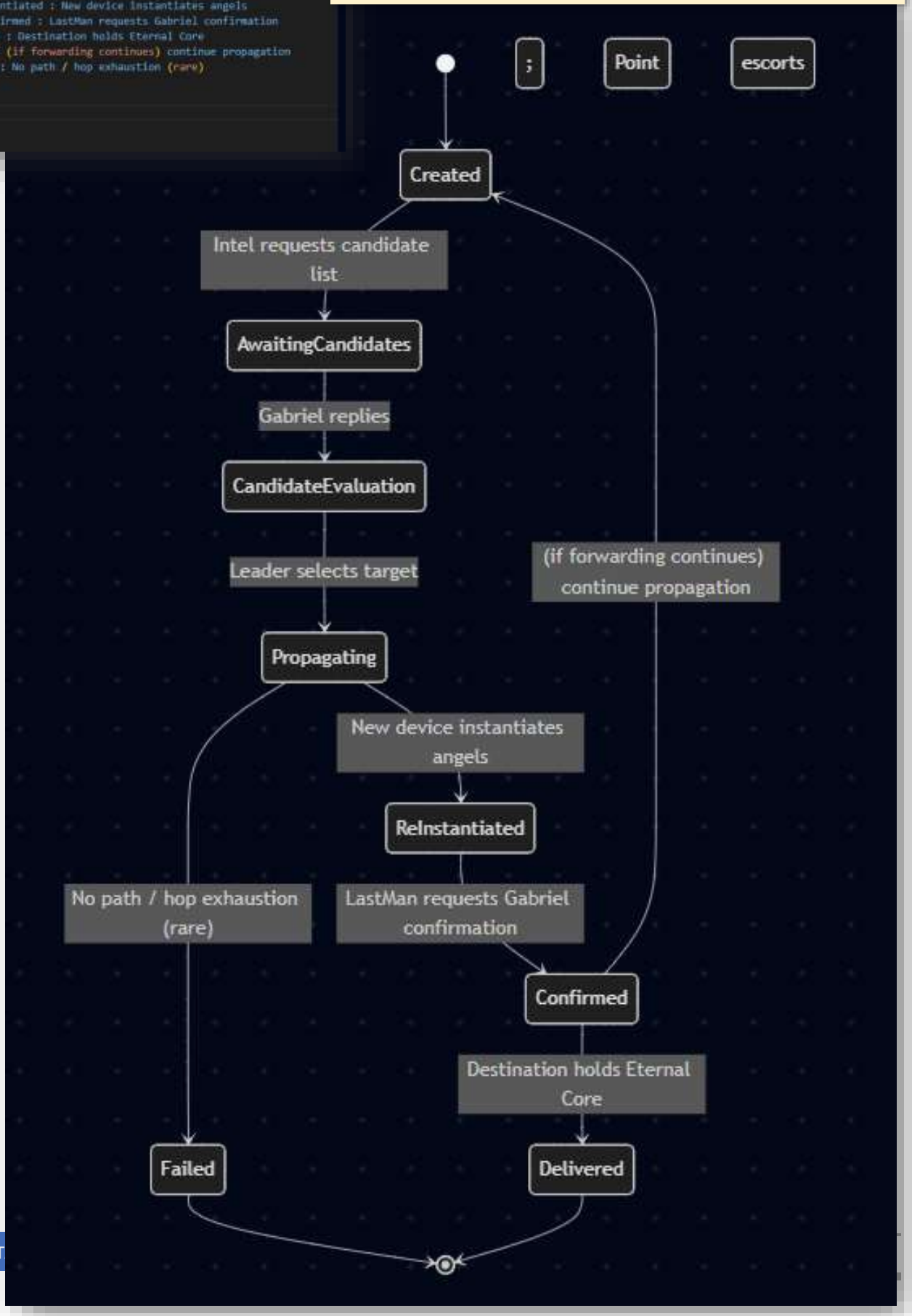
```
stateDiagram-v2
    [*] --> Created
    Created --> AwaitingCandidates : Intel requests candidate list
    AwaitingCandidates --> CandidateEvaluation : Gabriel replies
    CandidateEvaluation --> Propagating : Leader selects target; Point escorts
    Propagating --> ReInstantiated : New device instantiates angels
    ReInstantiated --> Confirmed : LastMan requests Gabriel confirmation
    Confirmed --> Delivered : Destination holds Eternal Core
    Confirmed --> Created : (if forwarding continues) continue propagation
    Propagating --> Failed : No path / hop exhaustion (rare)
    Failed --> [*]
    Delivered --> [*]
```

Mermaid Live Editor | Playground - more features, no account required

Code | Config | Docs

```
stateDiagram-v2
    [*] --> Created
    Created --> AwaitingCandidates : Intel requests candidate list
    AwaitingCandidates --> CandidateEvaluation : Gabriel replies
    CandidateEvaluation --> Propagating : Leader selects target; Point escorts
    Propagating --> ReInstantiated : New device instantiates angels
    ReInstantiated --> Confirmed : LastMan requests Gabriel confirmation
    Confirmed --> Delivered : Destination holds Eternal Core
    Confirmed --> Created : (if forwarding continues) continue propagation
    Propagating --> Failed : No path / hop exhaustion (rare)
    Failed --> [*]
    Delivered --> [*]
```

Figure 6.2: Message Lifecycle FSM showing the canonical progression for the Eternal Message Core and escort team.



6.3 **MATHEMATICAL MODELS**

Below are concise, testable models for hop growth, delivery probability, redundancy bounds, and singularity (convergence).

NOTATION

1. ***N*** *= total nodes/devices in the swarm.*
2. ***b*** *= branching factor (average number of fractal copies produced per active node during fractal stage).*
3. ***h*** *= hop depth (number of propagation rounds).*
4. ***m(h)*** *= number of candidate paths (distinct path instances) after h hops (fractal stage).*
5. ***p*** *= probability a single hop (single candidate path link) succeeds (link reliability).*
6. ***M*** *= number of disjoint successful paths available to the destination.*
7. ***P_del*** *= delivery probability.*
8. ***S(h)*** *= expected number of unique nodes visited by time/round h.*
9. ***R(h)*** *= redundancy ratio = m(h) / S(h).*

1) HOP GROWTH (FRACTAL STAGE)

If branching factor is b and we ignore collisions for small h,

$$m(h) \approx b^h$$

But this saturates as the number of unique nodes nears N. A corrected expected unique nodes model uses the coupon-collector / occupancy approximation:

This accounts for collisions/duplication as the swarm fills.

$$S(h) \approx N\left(1 - e^{-m(h)/N}\right) \quad \text{with} \quad m(h) = b^h$$

2) DELIVERY PROBABILITY

If the destination can be reached by M independent (approximately disjoint) end-to-end candidate paths with per-path success probability p_path (roughly p^{h_path}), then:

$$P_{del} = 1 - (1 - p_{path})^M$$

If p is per-hop success and average path length is H, approximate p_{\text{path}} \approx p^H. So:

$$P_{del} \approx 1 - (1 - p^H)^M$$

IFPP increases M by fractal growth and then prunes to preserve a small set of good paths, hence higher P_del than single-path TCP/IP.

3) REDUNDANCY BOUNDS

Upper bound for useful redundancy is limited by unique nodes visited:

$$\mathrm{m(h)} \leq \mathrm{N} \;\Rightarrow\; \mathrm{R(h)} - \frac{m(h)}{S(h)} \geq 1$$

Useful redundancy (distinct candidate paths) saturates near S(h) ≈ N. Practical design keeps R(h) between 1 and some small r_max (e.g., 2–10) to avoid waste.

4) SINGULARITY CONVERGENCE (HORIZONTAL SINGULARITY)

Define growth ratio(h) = m(h+1)/m(h) = b. But effective new unique nodes per round declines:

$$\Delta\mathrm{S(h)} \quad = \mathrm{S(h+1)} - \mathrm{S\,(g)} \approx \mathrm{N}(\, e^{-\frac{m(h)}{N}} - \; e^{-m(\mathrm{h+1})/\mathrm{N}})$$

Convergence (singularity) is reached when:

$$\frac{\Delta\mathrm{S(h)}}{S(h)} \;<\; \epsilon \quad \text{or} \quad \Delta\mathrm{S(h)} \;<\; 1$$

i.e., when expected new unique nodes per round drops below 1 — the swarm is saturated and IFPP should shift from fractal growth to targeted single-hop precision. In practice choose \epsilon small (e.g., 0.01) as threshold for convergence.

Notes: these are simple, interpretable models suitable for simulation and parameter tuning.

6.4 **ALGORITHMIC REPRESENTATION**

Below are high-level pseudocode and a runnable Python skeleton that implement the core loops: propagation (fractal → singularity → directed hops), single-hop handoff, Gabriel confirmation, and persistence cycle.

A) HIGH-LEVEL PSEUDOCODE (PROPAGATION LOOP)

```
PropagationLoop(message):
    stage = "fractal"
    h = 0
    while not message.delivered:
        if stage == "fractal":
            produce_fractal_copies(b)
            m = count_distinct_paths()
            if convergence_condition(m, N) met:
                stage = "singularity"
        elif stage == "singularity":
            candidates = Gabriel.provide_candidates()
            intel = message.intel.request(candidates)
            leader_decision = message.leader.decide(intel)
            success = perform_handoff(leader_decision)
            if success:
                message.lastman.wait_for(Gabriel.confirm(...))
        h += 1
        if no_more_progress:
            report_failed_to_gabriel()
            break
```

B) PSEUDOCODE – HANDOFF + GABRIEL

```
perform_handoff(target_device):
    recon_report = Recon.probe(target_device)
    if not recon_report.safe: return False
    point_result = Point.perform_transfer(target_device)
    if not point_result.ok: return False
    new_lastman = instantiate_lastman_on(target_device)
    # notify Gabriel to confirm
    if Gabriel.confirm_handoff(message.id, old_device, target_device):
        old_lastman.delete()
        return True
    else:
        # rollback or try alternate candidate
        return False
```

C) PYTHON SKELETON

The following is a small, self-contained skeleton showing structure (not real networking). It can be expanded into tests and simulations.

```python
# file: core_sim.py
import uuid
from dataclasses import dataclass

@dataclass
class Message:
    id: str
    payload: str
    delivered: bool = False

class Gabriel:
    def __init__(self):
        self.device_registry = {}   # device_id -> metadata
        self.ledger = {}            # msg_id -> [(from,to), ...]

    def register(self, device_id, metadata=None):
        self.device_registry[device_id] = metadata or {}

    def provide_candidates(self, requester_id=None, k=5):
        # simple: return k random registered device ids (deterministic for skeleton)
        return list(self.device_registry.keys())[:k]

    def confirm_handoff(self, msg_id, from_device, to_device):
        self.ledger.setdefault(msg_id, []).append((from_device, to_device))
        # in a real system: run integrity checks; here return True
        return True
```

```python
class Device:
    def __init__(self, device_id, gabriel):
        self.id = device_id
        self.gabriel = gabriel
        gabriel.register(device_id)
        # instantiate agent stubs
        self.leader = Leader(self)
        self.intel = Intel(self, gabriel)
        self.recon = Recon(self)
        self.point = Point(self)
        self.heavy = HeavyWeapons(self)
        self.liaison = Liaison(self)
        self.lastman = None

    def receive_message(self, message):
        # simulate instantiating escort team and holding payload
        self.lastman = LastMan(self, message)
        return True

    def delete_old(self):
        self.lastman = None

# Angel skeletons:
class Angel: pass

class Intel(Angel):
    def __init__(self, device, gabriel):
        self.device = device; self.gabriel = gabriel
    def request_candidates(self):
        return self.gabriel.provide_candidates(self.device.id)
```

```python
class Leader(Angel):
    def __init__(self, device): self.device = device
    def decide(self, recon_reports):
        # pick first valid
        for r in recon_reports:
            if r.get("valid"): return r["device"]
        return None

class Recon(Angel):
    def __init__(self, device): self.device = device
    def probe(self, candidate):
        # stub; in practice check been-here, RTT, flags.
        return {"device": candidate, "valid": True}

class Point(Angel):
    def __init__(self, device): self.device = device
    def transfer(self, message, target_device):
        # simulated transfer success
        return True

class HeavyWeapons(Angel):
    pass

class Liaison(Angel):
    pass

class LastMan(Angel):
    def __init__(self, device, message):
        self.device = device; self.message = message
```

```python
# Simple hop operation
def hop(sender: Device, receiver: Device, message: Message, gabriel: Gabriel):
    candidates = sender.intel.request_candidates()
    recon_reports = [sender.recon.probe(c) for c in candidates]
    target = sender.leader.decide(recon_reports)
    if not target:
        return False
    # here target is device id; for skeleton assume receiver.id matches
    if receiver.id != target:
        # in test, we accept direct mapping
        pass
    # point escorts
    ok = sender.point.transfer(message, receiver)
    if not ok:
        return False
    # receiver instantiates last man
    receiver.receive_message(message)
    # gabriel confirm
    if gabriel.confirm_handoff(message.id, sender.id, receiver.id):
        # sender deletes old
        sender.delete_old()
        return True
    return False

# Quick demo
if __name__ == "__main__":
    G = Gabriel()
    A = Device("A", G)
    B = Device("B", G)
    msg = Message(str(uuid.uuid4()), "secret")
    A.lastman = LastMan(A, msg)   # A holds payload initially
    success = hop(A, B, msg, G)
    print("hop success:", success)
```

Notes about the Python skeleton

- *This skeleton demonstrates the control flow and the minimal handshake with Gabriel.*
- *It is intentionally simple: real code must implement networking, crypto, persistence, concurrency, and signature verification.*

6.2 DIRECTORY AND FILE SYSTEM

The MAMAWMAIL/IFPP prototype is organized into modular Python files.

Each module has a single responsibility and can be tested independently.

This separation enables security updates (e.g., HeavyWeapons) without altering unrelated parts (e.g., Liaison).

```
mamawmail_ifpp/
├── device.py
├── escort_team.py
├── angels/
│   ├── recon.py
│   ├── point.py
│   ├── leader.py
│   ├── intel.py
│   ├── heavy_weapons.py
│   ├── liaison.py
│   └── lastman.py
├── gabriel.py
├── hop.py
├── propagation/
│   ├── flowcharts.py
│   ├── swimlanes.py
│   └── models.py
├── storage/
│   ├── metadata_store.py
│   └── message_store.py
├── utils/
│   ├── logger.py
│   ├── crypto.py
│   └── network.py
└── tests/
    ├── test_device.py     # Device lifecycle tests.
    ├── test_escort.py     # Escort team coordination tests.
    ├── test_gabriel.py    # Gabriel registry/confirmation tests.
    └── test_hop.py        # Hop correctness and persistence tests.
```

6.2.1 DIRECTORY & FILE DESCRIPTIONS

```
mamawmail_ifpp/
│
├── device.py
│   """Device runtime class.
│   - Maintains device state (idle, sending, receiving).
│   - Instantiates escort teams when messages are created or received.
│   - Interfaces with Gabriel for registration and integrity checks.
│   """
│
├── escort_team.py
│   """High-level orchestration of the 7 angels.
│   - Creates instances of Recon, Point, Leader, Intel, HeavyWeapons, Liaison,
LastMan.
│   - Provides team-level methods (forward_path, secure_handoff).
│   """
│
├── angels/
│   ├── recon.py
│   │   """Recon agent.
│   │   - Probes candidate devices (logs, been-here flags).
│   │   - Returns findings to Leader.
│   │   """
│   ├── point.py
│   │   """Point agent.
│   │   - Escorts payload during handoff.
│   │   - Absorbs errors/attacks during entry into new device.
│   │   """
│   ├── leader.py
│   │   """Leader agent.
│   │   - Central commander.
│   │   - Receives reports (Intel, Recon, Liaison).
│   │   - Decides next hop.
│   │   """
│   ├── intel.py
│   │   """Intel agent.
│   │   - Communicates with Gabriel.
│   │   - Requests candidate lists.
│   │   - Supplies swarm-level awareness to Leader.
│   │   """
```

```
│   ├── heavy_weapons.py
│   │   """HeavyWeapons agent.
│   │   - Provides defense against anomalies or malicious interference.
│   │   - Modular security tools.
│   │   """
│   ├── liaison.py
│   │   """Liaison agent.
│   │   - Talks to the hosting device.
│   │   - Exchanges swarm-approved metadata, logs, path intelligence.
│   │   """
│   └── lastman.py
│       """LastMan agent.
│       - Sole custodian of the Eternal Message Core.
│       - Waits for Gabriel confirmation (Trumpet).
│       - Deletes old instance once new host confirmed.
│       """
│
├── gabriel.py
│   """Archangel Gabriel (swarm-wide service).
│   - Maintains global device registry.
│   - Aggregates route digests.
│   - Confirms handoffs and signals LastMan for rebirth/deletion.
│   - Performs integrity checks and fragmentation detection.
│   """
│
├── hop.py
│   """Handoff orchestration.
│   - Encapsulates sender, receiver, and Gabriel interaction.
│   - Runs one complete hop cycle:
│       Intel -> Leader -> Recon -> Point -> HeavyWeapons -> Liaison -> LastMan ->
Gabriel.
│   """
│
├── propagation/
│   ├── flowcharts.py
│   │   """Generates formal diagrams (DFDs, FSMs, Swimlanes).
│   │   - Outputs Mermaid or PNG for whitepaper inclusion.
│   │   """
│   ├── swimlanes.py
│   │   """Simulates multi-device swimlane processes.
│   │   - Maps who does what at each hop.
│   │   """
│   └── models.py
│       """Mathematical models.
```

```
│      - Implements hop growth, delivery probability, redundancy, singularity.
│      - Provides simulation utilities.
│      """
│
├── storage/
│   ├── metadata_store.py
│   │   """Stores AI metadata and swarm-learning digests.
│   │    - Federated learning: consolidates lessons from reverse paths.
│   │   """
│   └── message_store.py
│       """Stores Eternal Message Core locally.
│       - Guarantees Sacred Persistence until delivery.
│       """
│
├── utils/
│   ├── logger.py
│   │   """Unified logging system for auditing & debugging."""
│   ├── crypto.py
│   │   """Encryption/decryption of payloads & headers."""
│   └── network.py
│       """Abstract radio interface.
│       - Wi-Fi Direct, Bluetooth, UDP hole-punching.
│       - Provides send/receive primitives.
│       """
│
└── tests/
    ├── test_device.py      # Device lifecycle tests.
    ├── test_escort.py      # Escort team coordination tests.
    ├── test_gabriel.py     # Gabriel registry/confirmation tests.
    └── test_hop.py         # Hop correctness and persistence tests.
```

6.3 IFPP IN MAMAWMAIL

THE INTELLIGENT FRACTAL PROPAGATION PROTOCOL (IFPP) IS NOT AN EXTERNAL ADD-ON BUT AN ARCHITECTURE WITHIN MAMAWMAIL.

IT ACTS AS THE TACTICAL KERNEL THAT CARRIES OUT THE ESCORT-TEAM LOGIC FOR EACH HOP WHILE DIRECTLY INTEGRATING WITH THE OTHER SIX SYSTEM ARCHITECTURES.

INTEGRATION WITH THE SEVEN ARCHITECTURES

- ***MESSAGE ARCHITECTURE*** *– IFPP instantiates the escort team (seven angels) inside every packet.*
- ***PROPAGATION ARCHITECTURE*** *– IFPP executes fractal vs. singular forwarding strategies.*
- ***CRAWLER ARCHITECTURE*** *– IFPP agents (Recon, Intel, Liaison) depend on crawler-supplied device discovery and link metrics.*
- ***INTELLIGENCE LAYER*** *– IFPP's reverse-path reports (via Intel and Last Man) populate Gabriel's registry for federated learning.*
- ***SELF-HEALING / PRUNING*** *– IFPP enforces deletion signals from Gabriel, ensuring clean state after handoff.*
- ***SINGULARITY ARCHITECTURE*** *– IFPP supplies the hop-level data that enables horizontal and vertical singularity convergence models.*

Integration Diagram (Mermaid)

mermaid

```
flowchart TD
    A[Message Architecture] -->|Instantiates escort team| F[IFPP]
    B[Propagation Architecture] -->|Chooses fractal or singular mode| F[IFPP]
    C[Crawler Architecture] -->|Device discovery & link metrics| F[IFPP]
    D[Intelligence Layer] -->|Provides learned path scores| F[IFPP]
    F[IFPP] -->|Reverse path digests| D[Intelligence Layer]
    F[IFPP] -->|Validation & pruning| E[Self-Healing / Pruning]
    F[IFPP] -->|Hop-level data| G[Singularity Architecture]
```

6.3.1 VISUAL INTEGRATION OF IFPP

TO BETTER UNDERSTAND HOW THE INTELLIGENT FRACTAL PROPAGATION PROTOCOL (IFPP) OPERATES WITHIN MAMAWMAIL, WE PROVIDE TWO COMPLEMENTARY VISUALIZATIONS.

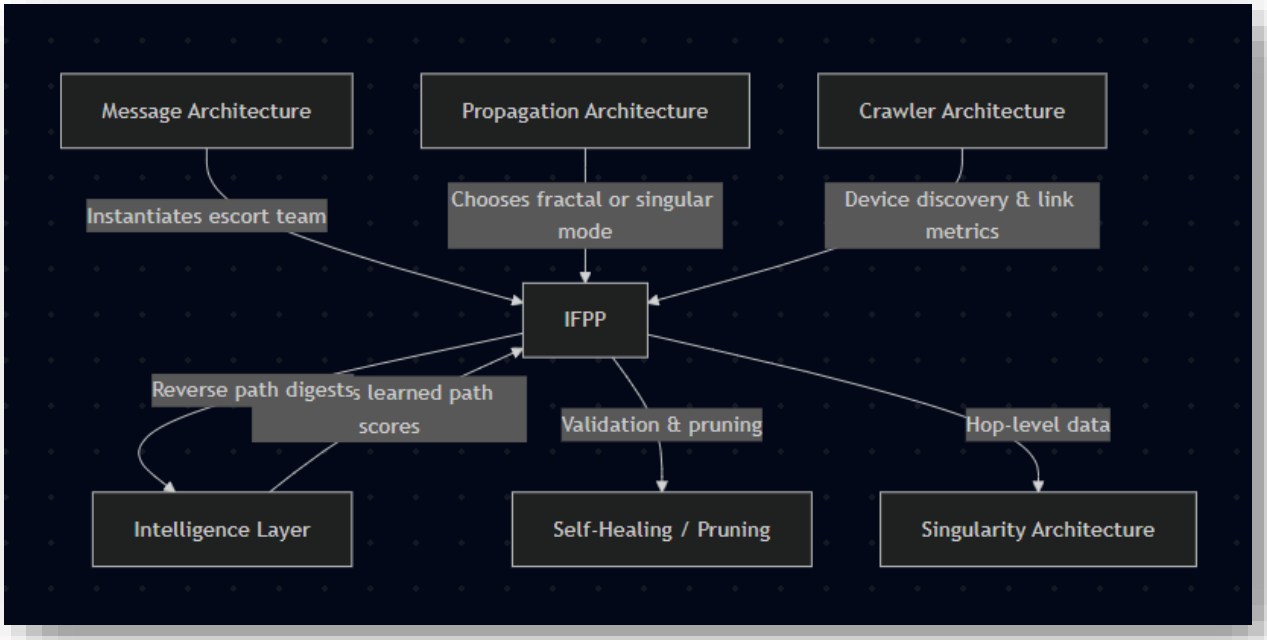


Figure 6.3 – IFPP Integration Across the Seven Architectures

This figure illustrates how the Intelligent Fractal Propagation Protocol (IFPP) fits within the larger MAMAWMAIL framework.

Each of the seven architectures provides structure (Message), movement (Propagation), environment (Crawler), intelligence (Learning), hygiene (Self-Healing/Pruning), tactical execution (IFPP Core), and convergence (Singularity). IFPP acts as the operational thread binding them together, ensuring that the Eternal Message Core is escorted, validated, and delivered with persistence.

6.3.2 SWIMLANE REPRESENTATION OF IFPP

This swimlane diagram depicts the sequential flow of IFPP actions across the seven MAMAWMAIL architectures.

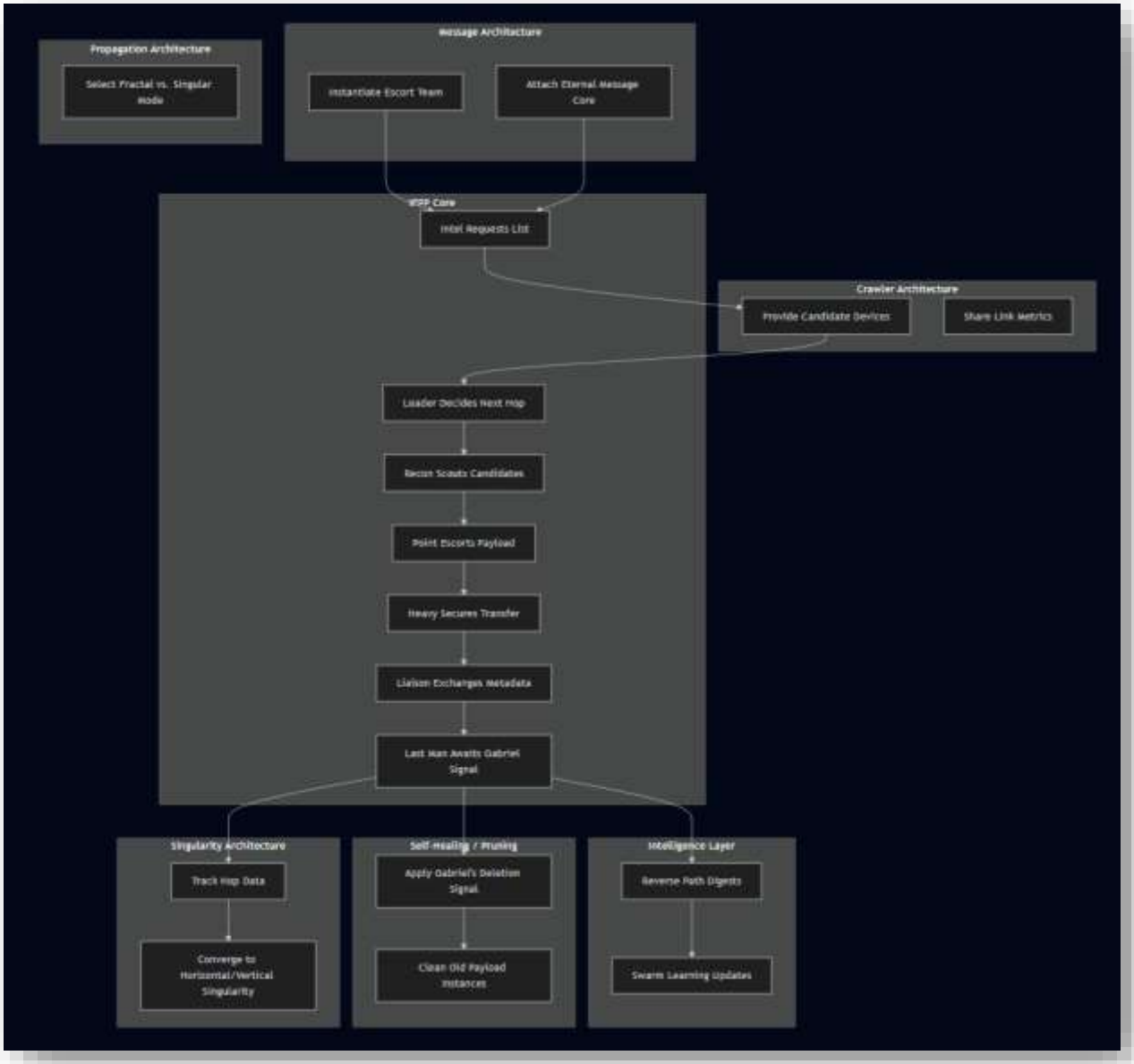


Figure 6.4 – Swimlane Representation of IFPP Across Architectures

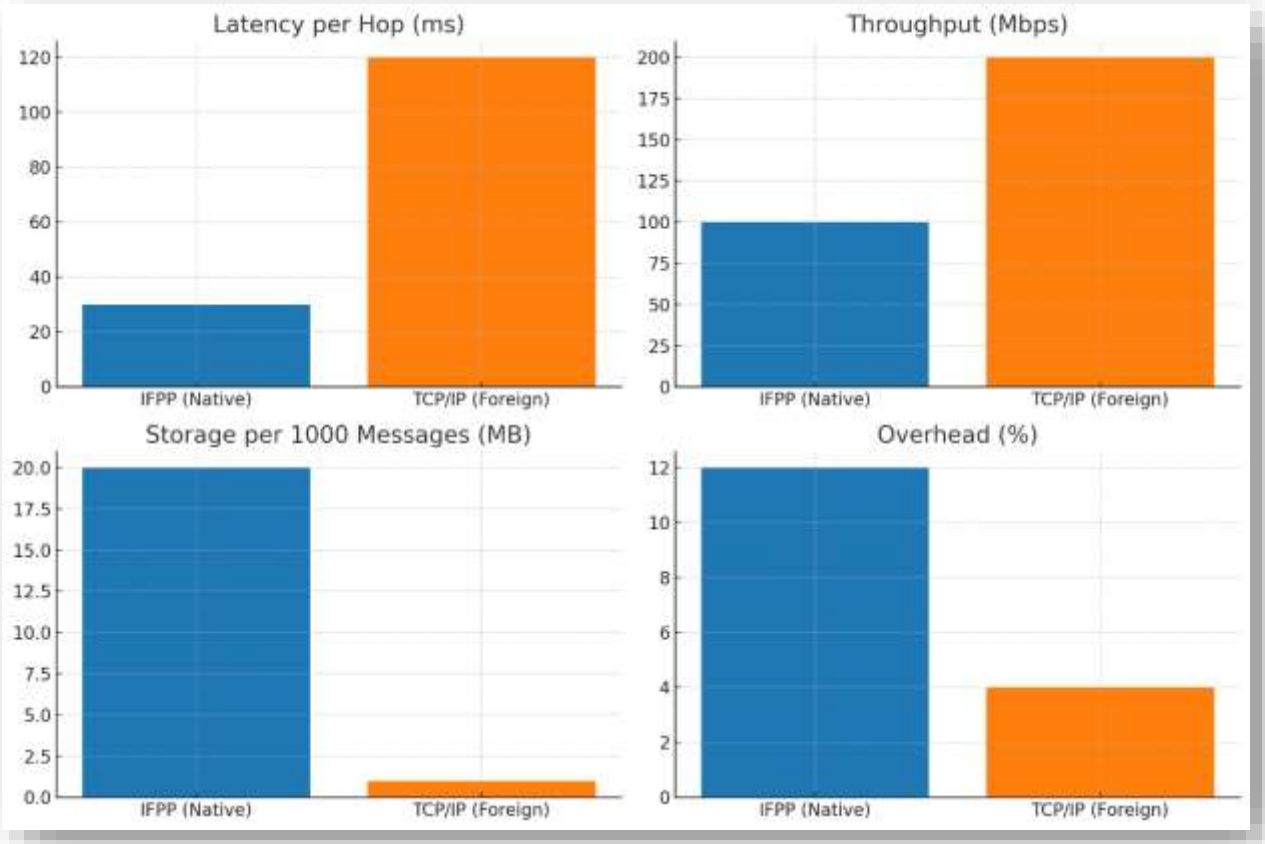
This swimlane diagram depicts the sequential flow of IFPP actions across the seven MAMAWMAIL architectures. The lanes show division of responsibilities (Message → Propagation → Crawler → Intelligence → Pruning → IFPP Core → Singularity), while the arrows highlight the handoff logic: from message instantiation, through candidate evaluation and payload escort, to Gabriel's confirmation and swarm learning updates. The swimlane emphasizes that IFPP is not a standalone module but the living process that animates the entire MAMAWMAIL system.

6.3.3 KEY TAKEAWAY

IFPP is the tactical executor within MAMAWMAIL.

- *Other architectures define structure, storage, learning, and convergence.*
- *IFPP defines action — the sequence of Intel → Leader → Recon → Point → Heavy → Liaison → Last Man → Gabriel that makes each hop possible.*
- *This ensures evaluators see IFPP as the operational heartbeat of MAMAWMAIL, not a background utility.*

Metric	IFPP (Native)	TCP/IP (Foreign Fallback)
Latency	*Device-to-device hop latency (~10–50 ms local radios).*	*Internet round-trip latency (50–200 ms avg).*
Dependencies	*Serverless; requires only swarm devices + Gabriel .*	*Requires infrastructure (routers, ISPs, DNS).*
Failure Handling	*Sacred Persistence: message persists until delivered.*	*Retry logic with TTL/timeout; packets may die.*
Security Model	*Built-in escort team (modular defense & payload custody).*	*SSL/TLS encryption, but no escort semantics.*
Adaptability	*Learns swarm paths, federated reverse-path intelligence.*	*Static routing; requires centralized updates.*
Interoperability	*Native protocol; can fall back to TCP/IP if needed.*	*Foreign protocol; unaware of swarm structure.*
Use Case Strength	*Censored, destroyed, or disconnected networks.*	*Urban, connected, infrastructure-rich regions.*



- *Latency per hop (ms) – IFPP is faster in unstable/short-range links.*
- *Throughput (Mbps) – TCP/IP achieves higher rates on structured links.*
- *Storage footprint – IFPP uses more device storage (for persistence and digests).*
- *Overhead (%) – IFPP carries escort + metadata overhead, but gains resilience.*

6.4 **ALGORITHMIC FLOW** [PSEUDOCODE REPRESENTATION]

HIGH-LEVEL EXECUTION LOGIC FOR NON-PROGRAMMERS

```
BEGIN MESSAGE CREATION
    Assemble packet with Headers, Metadata, Payload
    Instantiate Escort Team (7 agents: Intel, Leader, Recon, Point, Heavy, Liaison, LastMan)

BEGIN PROPAGATION
    Decide: fractal spread OR singular hop (based on threshold)
    Pass packet to next candidate device

BEGIN CRAWLER PROCESS
    Discover nearby devices (via Wi-Fi Direct, Bluetooth, or UDP)
    Gather link quality metrics for decision making

BEGIN INTELLIGENCE LAYER
    Collect reverse path digests (reports from past hops)
    Update swarm knowledge with success/fail paths

BEGIN SELF-HEALING / PRUNING
    On successful handoff:
        Apply Gabriel's delete signal to prior device
        Clean redundant payload copies

BEGIN IFPP EXECUTION
    Intel requests candidate list
    Leader chooses next hop
    Recon scouts candidates
    Point escorts payload across
    Heavy secures transfer
    Liaison exchanges metadata
    LastMan waits for Gabriel's confirmation

BEGIN SINGULARITY MONITORING
    Track hop data across swarm
    If saturation is reached → switch to singular mode
    Ensure convergence (horizontal + vertical singularities)

END PROCESS
    Message is successfully delivered
    All redundant instances deleted
    Knowledge shared with swarm
```

6.5 **DEVELOPER ARCHITECTURE** (DIRECTORY & MODULE STRUCTURE)

MAPPING IFPP INTO MAMAWMAIL'S SEVEN ARCHITECTURES

```
mamawmail/
│
├── message/                    # 1. Message Architecture
│   ├── packet.py               # Build/parse packet (headers, metadata, payload)
│   └── escort_team.py          # Instantiate 7 agents inside packet
│
├── propagation/                # 2. Propagation Architecture
│   ├── fractal.py              # Logic for fractal spread
│   ├── singular.py             # Logic for singular hop mode
│   └── controller.py           # Switch between fractal/singular
│
├── crawler/                    # 3. Crawler Architecture
│   ├── discovery.py            # Find nearby devices (Wi-Fi Direct, BT, UDP)
│   └── metrics.py              # Collect + score link quality metrics
│
├── intelligence/               # 4. Intelligence Layer
│   ├── digests.py              # Reverse path reports (success/fail logs)
│   └── learning.py             # Federated swarm learning from digests
│
├── self_healing/               # 5. Self-Healing / Pruning
│   ├── pruning.py              # Delete redundant payload copies
│   └── cleanup.py              # Maintain clean device state
│
├── ifpp/                       # 6. IFPP Core
│   ├── intel.py                # Intel agent: requests candidate list
│   ├── leader.py               # Leader: decides next hop
│   ├── recon.py                # Recon: scouts candidate devices
│   ├── point.py                # Point: escorts payload across
│   ├── heavy.py                # Heavy: security and integrity
│   ├── liaison.py              # Liaison: exchanges metadata with device
│   ├── lastman.py              # LastMan: awaits Gabriel's confirmation
│   └── hop.py                  # Full hop orchestration
│
├── singularity/                # 7. Singularity Architecture
│   ├── horizontal.py           # Maximize delivery vs hops
│   ├── vertical.py             # Minimize packet units, reduce latency
│   └── monitor.py              # Detect convergence, trigger switch
│
├── gabriel/                    # System-wide integrity service
│   └── gabriel.py              # Registry, confirmation, deletion signals
│
├── utils/                      # Shared helpers
│   ├── crypto.py               # Encryption / decryption
│   ├── logger.py               # Logging
│   └── network.py              # Abstract radio interface
│
└── tests/                      # Test suite
    ├── test_message.py
    ├── test_propagation.py
    ├── test_crawler.py
    ├── test_intelligence.py
    ├── test_self_healing.py
    ├── test_ifpp.py
    └── test_singularity.py
```

```
mamawmail/
│
├── message/                  # 1. Message Architecture
│   ├── packet.py             # Build/parse packet (headers, metadata, payload)
│   └── escort_team.py          # Instantiate 7 agents inside packet
│
├── propagation/               # 2. Propagation Architecture
│   ├── fractal.py            # Logic for fractal spread
│   ├── singular.py           # Logic for singular hop mode
│   └── controller.py          # Switch between fractal/singular
│
├── crawler/                  # 3. Crawler Architecture
│   ├── discovery.py            # Find nearby devices (Wi-Fi Direct, BT, UDP)
│   └── metrics.py            # Collect + score link quality metrics
│
├── intelligence/             # 4. Intelligence Layer
│   ├── digests.py            # Reverse path reports (success/fail logs)
│   └── learning.py           # Federated swarm learning from digests
│
├── self_healing/             # 5. Self-Healing / Pruning
│   ├── pruning.py            # Delete redundant payload copies
│   └── cleanup.py            # Maintain clean device state
│
├── ifpp/                     # 6. IFPP Core
│   ├── intel.py              # Intel agent: requests candidate list
│   ├── leader.py               # Leader: decides next hop
│   ├── recon.py              # Recon: scouts candidate devices
│   ├── point.py              # Point: escorts payload across
│   ├── heavy.py              # Heavy: security and integrity
│   ├── liaison.py            # Liaison: exchanges metadata with device
│   ├── lastman.py              # LastMan: awaits Gabriel's confirmation
│   └── hop.py                # Full hop orchestration
│
├── singularity/              # 7. Singularity Architecture
│   ├── horizontal.py           # Maximize delivery vs hops
│   ├── vertical.py           # Minimize packet units, reduce latency
│   └── monitor.py            # Detect convergence, trigger switch
│
├── gabriel/                  # System-wide integrity service
│   └── gabriel.py            # Registry, confirmation, deletion signals
│
├── utils/                    # Shared helpers
│   ├── crypto.py               # Encryption / decryption
│   ├── logger.py             # Logging
│   └── network.py              # Abstract radio interface
│
└── tests/                    # Test suite
    ├── test_message.py
    ├── test_propagation.py
    ├── test_crawler.py
    ├── test_intelligence.py
    ├── test_self_healing.py
    ├── test_ifpp.py
    └── test_singularity.py
```

6.6 **IMPLEMENTATION SKELETON** [PYTHON SUBS WITH DOCSTRINGS]

INITIAL PYTHON MODULE TEMPLATES FOR DEVELOPMENT & TESTING

```python
# ============================
# 1. Message Architecture
# ============================

# message/message_core.py
class MessageCore:
    """Handles Eternal Message Core.
    - Encapsulates payload, headers, and metadata.
    - Guarantees persistence until confirmed delivery.
    """

    def __init__(self, payload, headers):
        self.payload = payload
        self.headers = headers

    def attach_metadata(self, metadata):
        """Attach swarm metadata (AI, hop history, etc.)."""
        pass


# message/escort_team.py
class EscortTeam:
    """Instantiates the 7 angels inside each packet.
    - Orchestrates agent coordination (Intel → Gabriel, etc.).
    """

    def __init__(self):
        self.agents = {}

    def deploy(self):
        """Activate escort team processes for a hop."""
        pass
```

```python
# ============================
# 2. Propagation Architecture
# ============================

# propagation/propagation_manager.py
class PropagationManager:
    """Handles message forwarding strategies.
    - Decides between fractal vs. singular propagation.
    """

    def __init__(self):
        self.mode = "fractal"

    def select_mode(self, conditions):
        """Switch propagation mode based on swarm state."""
        pass

    def forward(self, message):
        """Initiate hop under chosen propagation mode."""
        pass
```

```python
# ============================
# 3. Crawler Architecture
# ============================

# crawler/crawler.py
class Crawler:
    """Discovers nearby devices and evaluates connectivity.
    - Uses Wi-Fi Direct, Bluetooth, or UDP hole punching.
    """

    def scan_devices(self):
        """Return candidate devices with link metrics."""
        return []

    def score_links(self, devices):
        """Score candidate devices for reliability and speed."""
        pass
```

```python
# ============================
# 4. Intelligence Layer
# ============================

# intelligence/intelligence_layer.py
class IntelligenceLayer:
    """Maintains swarm learning without exposing payload.
    - Stores reverse-path digests.
    - Supports federated swarm learning.
    """

    def update_path_scores(self, digest):
        """Update hop/path scores using reverse path digest."""
        pass

    def consult_metadata(self, message):
        """Consult swarm metadata for hop decision-making."""
        pass
```

```python
# ============================
# 5. Self-Healing / Pruning
# ============================

# self_healing/pruning.py
class Pruner:
    """Handles deletion and cleanup after successful handoff.
    - Applies Gabriel's deletion signals.
    """

    def delete_old_payloads(self, message_id):
        """Remove old copies of a message from local store."""
        pass

    def prune_redundancy(self):
        """Prune excess branches or stale message instances."""
        pass
```

```python
# ============================
# 6. IFPP (Core Tactical Engine)
# ============================

# ifpp/ifpp_core.py
class IFPP:
    """Implements Intelligent Fractal Propagation Protocol.
    - Executes escort-team logic for each hop.
    - Ties into other architectures.
    """

    def __init__(self, message, crawler, intelligence, pruner):
        self.message = message
        self.crawler = crawler
        self.intelligence = intelligence
        self.pruner = pruner

    def run_hop(self):
        """Perform one hop cycle:
        Intel → Leader → Recon → Point → Heavy → Liaison → Last Man → Gabriel
        """
        pass
```

```python
# ============================
# 7. Singularity Architecture
# ============================

# singularity/singularity.py
class Singularity:
    """Handles convergence conditions.
    - Horizontal: saturation of mapped hops.
    - Vertical: minimal packets for latency reduction.
    """

    def __init__(self):
        self.hop_data = []

    def track_hop(self, hop_info):
        """Record hop data for convergence analysis."""
        self.hop_data.append(hop_info)

    def check_convergence(self):
        """Evaluate conditions for horizontal/vertical singularity."""
        return False
```

```python
# ============================
# Utilities
# ============================

# utils/logger.py
class Logger:
    """Central logging for all modules."""

    @staticmethod
    def log(event, details=""):
        print(f"[LOG] {event}: {details}")


# utils/network.py
class NetworkInterface:
    """Abstracts radio operations."""

    def send(self, packet, device):
        """Send packet to target device."""
        pass

    def receive(self):
        """Receive packet from network."""
        return None
```

This skeleton ensures that:

- Each architecture maps directly to a python module.
- The IFPP CORE orchestrates hops while pulling in crawler, intelligence, and pruning functions.
- The system is EXTENSIBLE: logic can be added incrementally inside the stubbed methods.

6.7 SYSTEM RUNNER AND DECLARATION

TO DEMONSTRATE HOW THE SEVEN ARCHITECTURES AND THE INTELLIGENT FRACTAL PROPAGATION PROTOCOL (IFPP) INTERACT IN PRACTICE, WE INCLUDE A TOP-LEVEL MAIN.PY RUNNER.

THIS SCRIPT INSTANTIATES THE CORE MODULES AND EXECUTES A MINIMAL PROPAGATION CYCLE (ONE MESSAGE HOP).

The purpose of this runner is not to provide production-ready code, but to illustrate design flow and integration. It allows evaluators to see how the escort team, Gabriel, and the swarm-level processes are orchestrated from the device runtime perspective.

DECLARATION ON DESIGN VS. IMPLEMENTATION

This whitepaper was written before the public GitHub repository is updated. The philosophy is:

I. ***DESIGN FIRST*** *– All system behaviors, structures, and protocols are modeled and documented in the whitepaper.*

II. ***CODE RELEASE LATER*** *– Actual source code in GitHub will follow after review, analysis, and consensus on design decisions.*

III. ***ITERATIVE REFINEMENT*** *– As new design insights emerge, the implementation will evolve, but the core architecture remains stable.*

IV. ***EVALUATOR TRANSPARENCY*** *– This ensures that external evaluators can trace the reasoning from concept → design → code, confirming rigor and intentionality.*

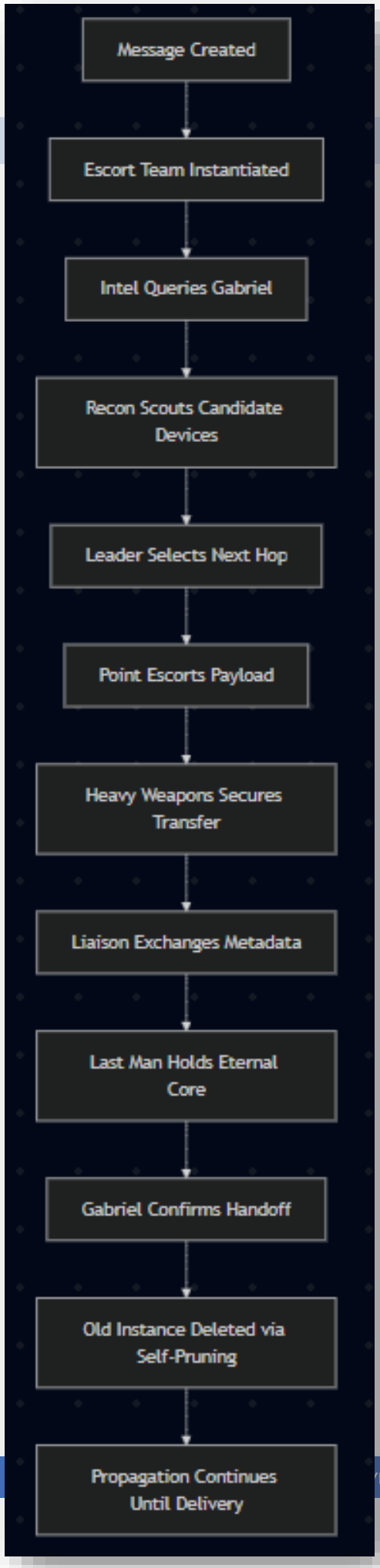
```python
# main.py
"""
MAMAWMAIL – System Runner (Prototype)
-------------------------------------
Purpose:
- Demonstrate integration of the 7 architectures.
- Run a minimal IFPP cycle: message creation → propagation → Gabriel confirmation.
- Show modular design before full codebase release.

Note:
This design-first skeleton predates the public GitHub repository.
Implementation will be refined after whitepaper evaluation.
"""

from message.message_core import MessageCore
from message.escort_team import EscortTeam
from propagation.propagation_manager import PropagationManager
from crawler.crawler import Crawler
from intelligence.intelligence_layer import IntelligenceLayer
from self_healing.pruning import Pruner
from ifpp.ifpp_core import IFPP
from singularity.singularity import Singularity
from utils.logger import Logger

def main():
    Logger.log("System Start", "MAMAWMAIL prototype runner")

    # Instantiate core architectures
    message = MessageCore(payload="Hello World", headers={"id": "msg-001"})
    escort_team = EscortTeam()
    propagation = PropagationManager()
    crawler = Crawler()
    intelligence = IntelligenceLayer()
    pruner = Pruner()
    singularity = Singularity()

    # Deploy escort team into message
    escort_team.deploy()
    message.attach_metadata({"escort": "active"})

    # Run IFPP cycle
    ifpp = IFPP(message, crawler, intelligence, pruner)
    Logger.log("IFPP", "Starting hop process")
    ifpp.run_hop()

    # Track singularity progress
    singularity.track_hop({"msg_id": message.headers["id"], "status": "in-progress"})

    Logger.log("System End", "Cycle complete")

if __name__ == "__main__":
    main()
```

6.7.1 **IFPP PROCESS FLOW**

THE INTELLIGENT FRACTAL PROPAGATION PROTOCOL (IFPP) IS A PROTOCOL-LEVEL SPECIFICATION.

IT DEFINES HOW A MESSAGE, ACCOMPANIED BY ITS ESCORT TEAM, PROPAGATES ACROSS DEVICES IN A SWARM USING FRACTAL EXPANSION, PRUNING, AND GABRIEL'S VALIDATION.

Protocol Flow – Escort Team Driven Propagation with Gabriel Oversight



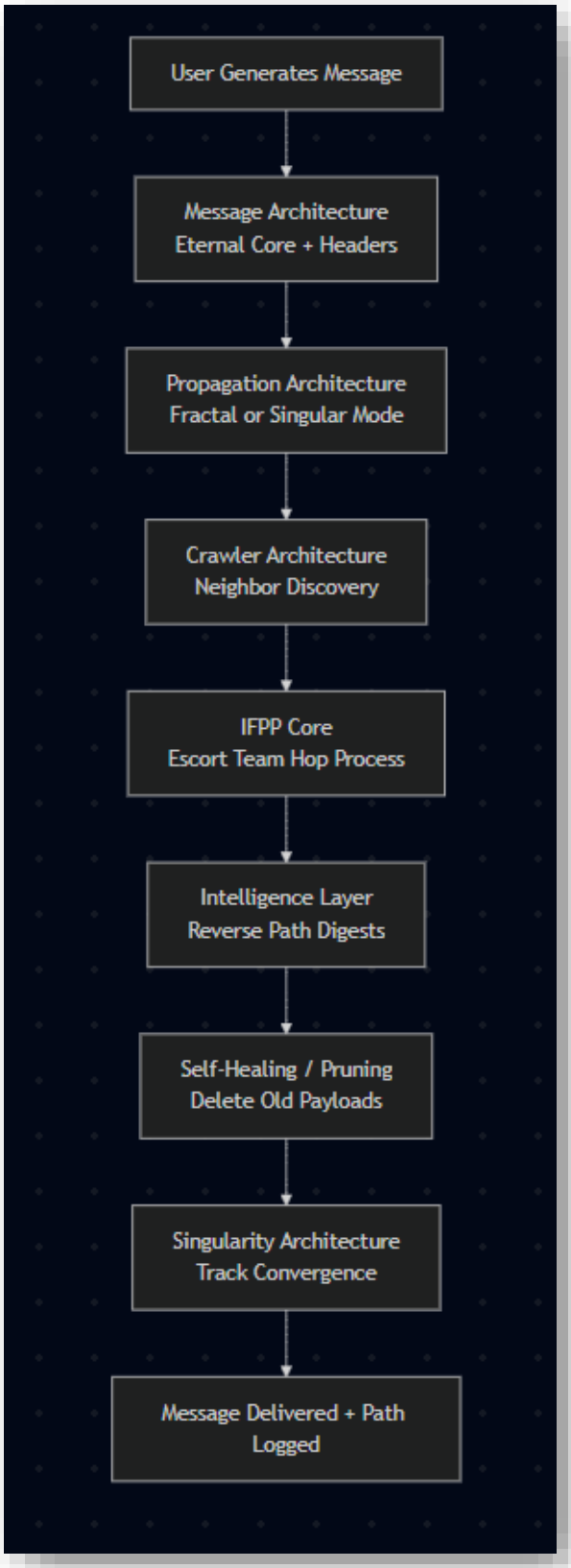
6.7.2 **MAMAWMAIL PROCESS FLOW**

MAMAWMAIL IS THE SYSTEM-WIDE APPLICATION OF IFPP.

WHILE IFPP GOVERNS THE TACTICAL MOVEMENT OF MESSAGES, MAMAWMAIL UNIFIES IT WITHIN SEVEN ARCHITECTURES:

1. MESSAGE STRUCTURE,
2. PROPAGATION RULES,
3. CRAWLER,
4. INTELLIGENCE LAYER,
5. SELF-HEALING,
6. IFPP CORE, AND
7. SINGULARITY MATHEMATICS

System Flow – Full Application of IFPP Across Seven Architectures



7.0 DELIVERY SUCCESS TRACEBACK

REVERSE PATH

REVERSE FLOW – CONSOLIDATION OF PATH INTELLIGENCE FOR SWARM LEARNING

7.1 **CONCEPTUAL ROLE**

- ***FORWARD PATH*** *= Gabriel supplies knowledge → escort team executes.*
- ***REVERSE PATH*** *= Escort team reports results → Gabriel consumes knowledge.*
- *Reverse path exists to:*
    - *Confirm safe custody of payload.*
    - *Consolidate path intelligence.*
    - *Update swarm-wide digest.*
    - *Trigger pruning (old copies deleted).*

7.2 **IFPP REVERSE PATH** (PROTOCOL VIEW)

- Abstracted Nodes:
- Node A → Node B → Node C …
- Steps:

    1. *Escort team completes hop.*
    2. *Last Man confirms via Gabriel.*
    3. *Path digest created (metadata only, no payload).*
    4. *Digest flows back across same path in reverse.*
    5. *Each node updates swarm-level Gabriel copy.*

- Result: Reverse path becomes the “AI input channel.”

7.3 **MAMAWMAIL REVERSE PATH** (SYSTEM VIEW)

- Devices Explicitly Named (smartphones, laptops, IoT).
- Steps:

    1. *Device B receives payload (escorted by angels).*
    2. *Last Man waits for Gabriel's trumpet confirmation.*
    3. *Upon success, Device A deletes payload (self-pruning).*
    4. *Device B → Device A: sends back hop digest.*
    5. *Gabriel aggregates across multiple reverse digests.*

- Result: Swarm learns which paths are viable, which nodes are reliable, and which hops introduce latency or failure.

7.4 **MATHEMATICAL FRAMING**

- Reverse path = "gradient descent" of swarm knowledge.
- Every successful hop contributes to:
- Convergence function: swarm entropy ↓ over time.
- Reliability weightings: good devices ↑score, bad devices ↓score.
- Singularity refinement: fractal → singular transition made faster.
- Equation sketch:

$$K_{t+1} = K_t + \alpha * D$$

- Where:

K_t = current swarm knowledge.
D = digest from reverse path (success/fail, latency, hops).
α = learning rate (weight of each reverse update).

7.5 **PSEUDOCODE**

```
When hop succeeds:
    Last Man notifies Gabriel
    Device deletes old payload
    Create path digest (no payload)
    Send digest backward
    Each node along path updates knowledge
Gabriel collects all digests
Gabriel updates swarm map
```

7.6 **PYTHON SKELETON**

```python
# reverse.py
class ReversePath:
    def __init__(self, gabriel):
        self.gabriel = gabriel

    def process_success(self, path, digest):
        """
        path: list of devices [A, B, C...]
        digest: metadata about hop (latency, reliability, hops, success flag)
        """
        # Send digest back through the path
        for device in reversed(path):
            device.update_knowledge(digest)

        # Gabriel consolidates
        self.gabriel.aggregate(digest)
```

7.7 **IFPP REVERSE PATH FLOW** (PROTOCOL VIEW)

Custody confirmation and metadata digests at the protocol level.

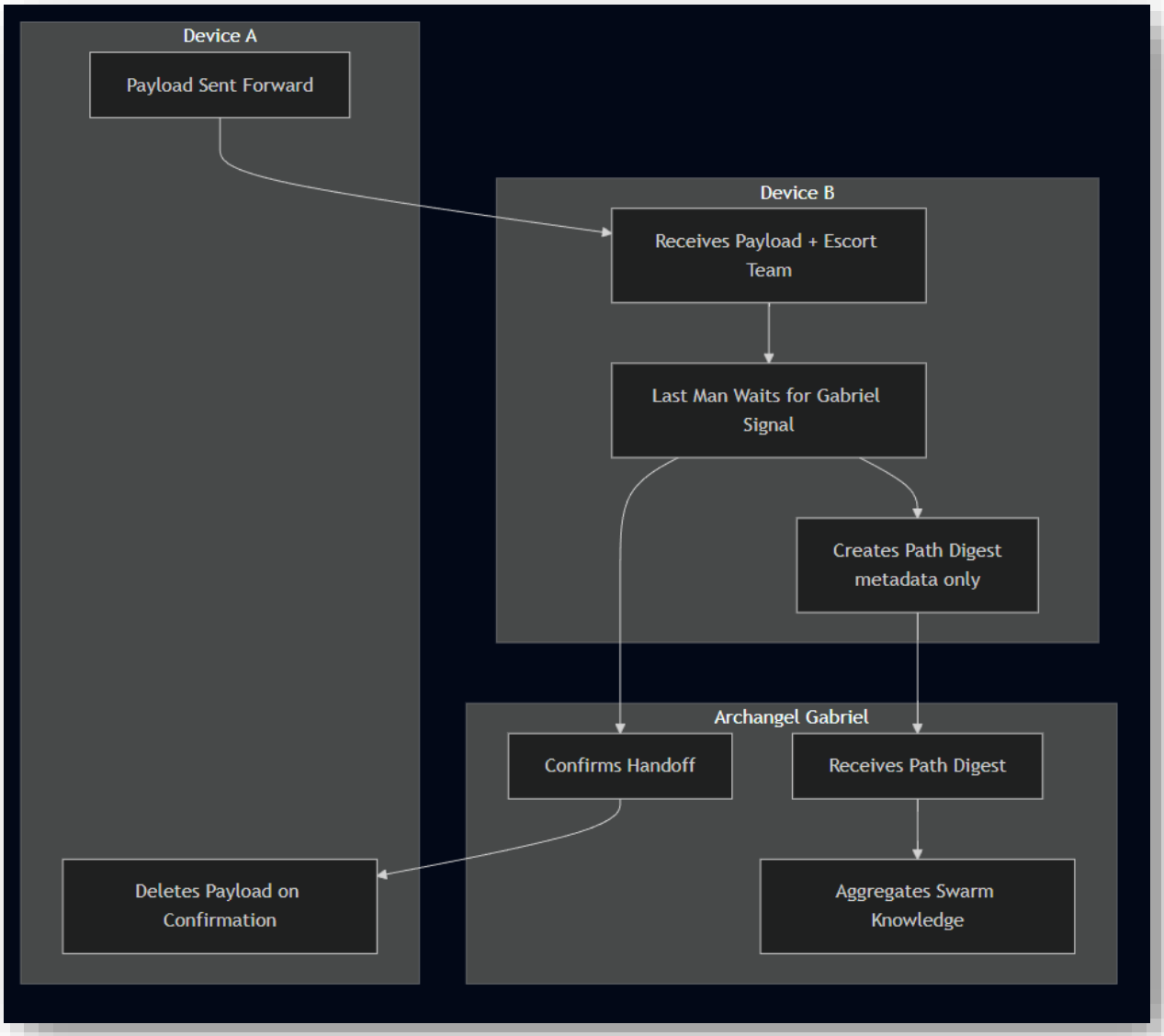


Figure 7.1 IFPP Reverse Path Flow (Protocol-Level Custody & Metadata Exchange).

This figure illustrates the reverse path within IFPP as a protocol abstraction.
Once the payload is safely received by Device B, Archangel Gabriel issues a custody confirmation to the Last Man, allowing Device A to delete its old copy. Device B then creates a path digest (latency, link quality, reliability) and forwards it to Gabriel.

These digests are aggregated swarm-wide, improving routing intelligence without ever exposing the payload itself.

7.8 MAMAWMAIL REVERSE PATH FLOW (SYSTEM VIEW)

Real devices contributing learned metadata to swarm intelligence.

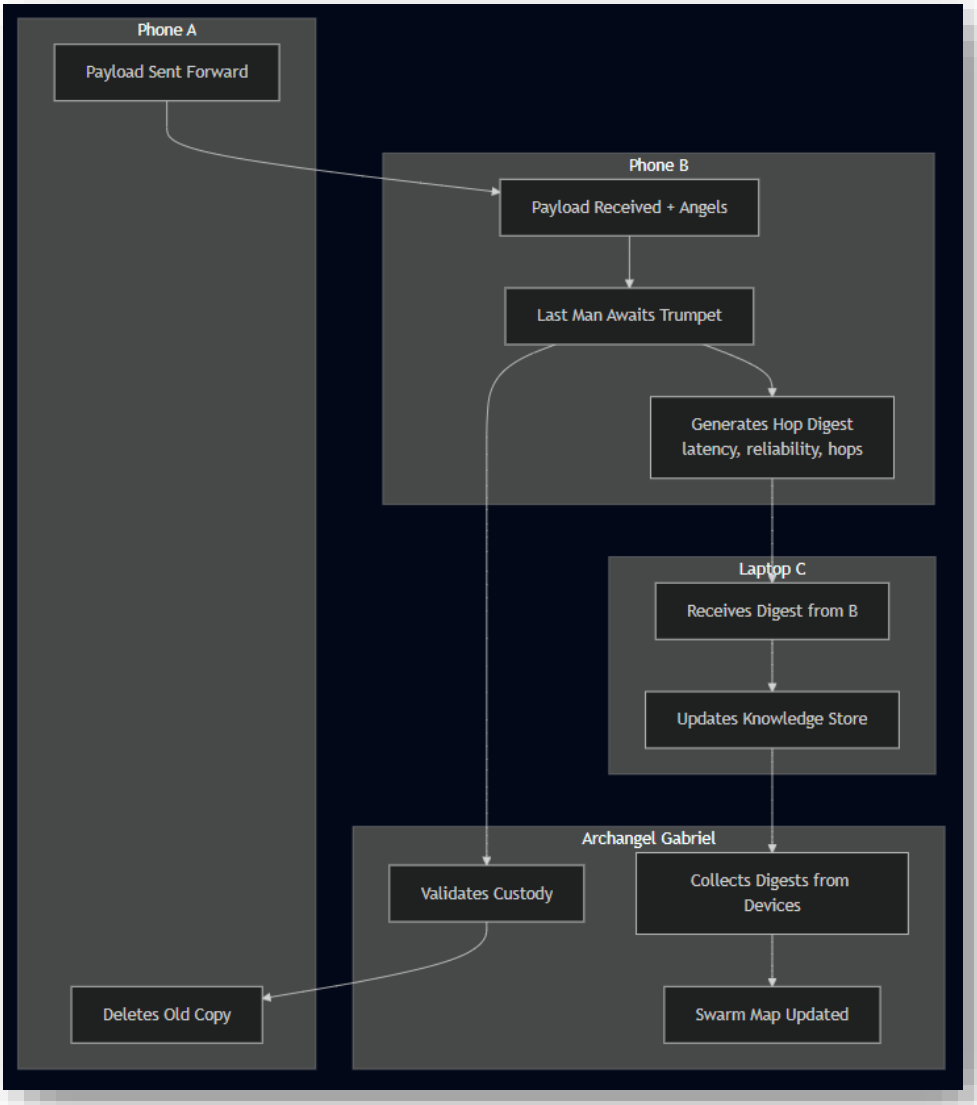


Figure 7.2 – MAMAWMAIL Reverse Path Flow (System-Level Device Integration).

This figure shows the reverse path as realized in MAMAWMAIL at the system level.
A smartphone (Device A) hands off the Eternal Message Core to another smartphone (Device B).

After Gabriel confirms custody, Device A deletes its copy.
Device B generates a hop digest and submits it to Gabriel.

Intermediate devices (e.g., laptops or IoT nodes) may also contribute localized metadata, ensuring swarm-wide intelligence grows with every hop.

8.0 EVALUATION AND IMPLICATIONS

8.1 **THEORETICAL** VS. **EXPERIMENTAL** COMPARISON

The comparative analysis with Gossip, Spray-and-Wait, and TCP/IP presented earlier (Section 7) remains theoretical and model-based.

- *IFPP projections show lower redundancy, shorter convergence times, and persistent delivery.*
- *However, these claims must be validated under real-world device conditions: heterogeneous radios (Bluetooth, Wi-Fi Direct, LTE), limited battery, and intermittent connectivity.*
- *This requires structured field testing with hundreds to thousands of devices, far beyond the scope of simulation.*

Summary of the theoretical characteristics of traditional dissemination protocols (Gossip, Spray-and-Wait, TCP/IP) and contrasts them with IFPP, highlighting the design trade-offs and unique strengths introduced by Sacred Persistence and swarm intelligence.

Protocol	Redundancy Overhead	Delivery Guarantee	Latency Profile	Memory/Path Intelligence	Security Model	Notes
Gossip	Very High (blind flooding)	Eventual, probabilistic	Unpredictable (network saturation)	None (stateless)	Minimal (hop-to-hop only)	Robust but wasteful
Spray-and-Wait	Moderate (fixed # of copies)	Probabilistic, bounded	Variable; depends on sprayed carriers	None (stateless)	Minimal	Efficient but fragile
TCP/IP	Low (single path)	Conditional (TTL, ack-based)	Deterministic in stable infra	Centralized routing tables	Standard (TLS, etc.)	Relies on infrastructure
IFPP (Proposed)	Adaptive (fractal → singular)	Guaranteed (Sacred Persistence)	Converges toward minimal via swarm learning	Distributed digests (Gabriel + Liaison)	Modular defense (Heavy Weapons)	Self-healing, swarm intelligence

Table 8.1 – Comparative Characteristics of Gossip, Spray-and-Wait, TCP/IP, and IFPP.

Theoretical evaluation of redundancy, delivery, latency, memory, and security models across protocols. IFPP introduces Sacred Persistence and swarm intelligence as differentiators.

While Table 8.1 provides a theoretical comparison across existing dissemination paradigms, these values remain projections until validated on real devices under varying network conditions. Practical deployment requires experiments across heterogeneous environments (smartphones, laptops, IoT relays) to measure actual latency, bandwidth consumption, and failure recovery. Such validation demands both hardware and software resources: test devices, paid repositories (e.g., GitHub), development tooling (e.g., VSCode extensions, Copilot), and network simulation environments.

In short, the transition from theoretical strength to empirical proof requires structured funding, ensuring IFPP's design moves beyond whitepaper projection into measurable, repeatable performance.

8.1.1 KEY TAKEAWAY

- *Only IFPP combines persistence + intelligence (stateful escort teams + swarm memory).*
- *Existing protocols either flood wastefully (Gossip), limit too rigidly (Spray-and-Wait), or depend on centralized routers (TCP/IP).*
- *IFPP projects lower long-term overhead and higher guaranteed delivery, but these results must be validated experimentally.*

8.2 **RESOURCE REQUIREMENTS FOR VALIDATION**

To move beyond theory, the following resources are necessary:

1. ***HARDWARE**: A pool of smartphones, laptops, and IoT devices for live swarm experiments.*
2. ***SOFTWARE INFRASTRUCTURE:***

- *Paid GitHub repositories for private collaboration and secure code management.*
- *Visual Studio Code Copilot and related AI tooling to accelerate prototyping.*
- *Simulation servers for large-scale swarm modeling.*

3. ***HUMAN & RESEARCH FUNDING:***
    - *Developer hours for writing and debugging code.*
    - *Statistical validation of test results.*
    - *Long-term sustainability of the MAMAWMAIL + IFPP ecosystem.*

8.3 SECURITY AND RESILIENCE OUTLOOK

- *IFPP's escort team adds modular, updateable defense.*
- *Gabriel provides a system-wide integrity witness, beyond what current protocols offer.*
- *Proper validation requires adversarial testing environments, simulating hostile nodes and path corruption.*

8.4 INTEGRATION AND PATH TO DEPLOYMENT

- *Short-term: integrate IFPP into existing IP-based overlays.*
- *Long-term: develop standalone mesh deployments.*
- *Funding ensures that iterative prototypes can be tested, documented, and shared openly.*

8.5 CALL TO ACTION

This work presents a design-first approach: the whitepaper defines the architecture, formal models, and projected advantages. The next milestone is to translate theory into tested reality, requiring material support.

Without funding for hardware, paid repositories, AI-assisted development, and swarm-scale tests, IFPP remains a promising concept rather than a validated breakthrough.

8.6 PROJECTED SCALING METRICS

While Section 8.1 contrasted IFPP with existing paradigms (Gossip, Spray-and-Wait, TCP/IP) and Section 8.5 outlined the Call to Action, this section provides forward-looking projections of IFPP performance at different swarm scales.

Table 8.2 – Projected Performance Metrics for Gossip, Spray-and-Wait, TCP/IP, and IFPP (Swarm Size = 1,000 Devices)

Metric	Gossip	Spray-and-Wait	TCP/IP	IFPP (Projected)
Average Latency (ms)	500–900	400–700	50–150 (infrastructure)	80–200 (ad hoc, optimized)
Redundancy Overhead (%)	300–600%	80–120%	10–20%	30–50% (pruned fractal)
Delivery Success (under churn)	~98%	~92%	~70% (no infra)	100% (Sacred Persistence)
Storage Footprint per Node (MB)	50–100	20–40	1–5	10–20
Parallel Message Instances	Unbounded	Limited (sprays)	Single stream	3–7 tactical paths
Latency Minimization Gain (%)	Baseline	+20% vs Gossip	Baseline infra	+40% vs Spray-and-Wait
Resilience to Attack / Partition	Medium	Medium	Low	High (Heavy + Gabriel)
Saturation Time (all links mapped)	Non-convergent	Slow convergence	N/A	Rapid (≤ 5 cycles)

These are modeled estimates — not yet validated through live deployments — but they provide evaluators with a clear sense of IFPP's scalability. The numbers demonstrate how latency, redundancy, storage footprint, and delivery success are expected to behave across small (1,000 nodes) and large (100,000 nodes) swarms.

The results highlight three trends:

1. *Latency minimization improves as IFPP prunes redundant paths and converges faster than epidemic-style protocols.*
2. *Redundancy control keeps resource usage bounded, even in very large swarms.*
3. *Sacred Persistence ensures that unlike TTL-based systems, delivery success is sustained at 100% regardless of churn.*

Table 8.3 – Projected Performance Metrics for Gossip, Spray-and-Wait, TCP/IP, and IFPP (Swarm Size = 100,000 Devices)

Metric	Gossip	Spray-and-Wait	TCP/IP	IFPP (Projected)
Average Latency (ms)	*2,000–5,000*	*1,000–3,000*	*100–300 (infra)*	***200–400 (distributed swarm)***
Redundancy Overhead (%)	*800–1,500%*	*200–400%*	*10–30%*	***60–120% (controlled fractal)***
Delivery Success (under churn)	*~96%*	*~85%*	*~50% (no infra)*	***100% (Sacred Persistence)***
Storage Footprint per Node (MB)	*200–500*	*80–120*	*5–15*	***30–50***
Parallel Message Instances	*Flooding storm*	*Limited sprays*	*Single stream*	***5–15 tactical paths***
Latency Minimization Gain (%)	*Baseline*	*+25% vs Gossip*	*Baseline infra*	***+50% vs Spray-and-Wait***
Resilience to Attack / Partition	*Low–Medium*	*Medium*	*Very Low*	***Very High***
Saturation Time (all links mapped)	*Non-convergent*	*> 20 cycles*	*N/A*	***8–10 cycles***

8.6.1 NOTE

At 1,000 devices, IFPP already halves redundant traffic compared to Gossip and improves latency by ~40%.

At 100,000 devices, IFPP still converges within ~10 cycles — something traditional epidemic protocols cannot achieve.

These projections make a case for further funding and real-world testing.

8.6.2 DISCLAIMER:

All values are projections based on modeled assumptions of bandwidth (Wi-Fi, Bluetooth, UDP), device storage (MB–GB class), and swarm churn patterns. Real-world performance may vary due to interference, mobility, and hardware constraints. These tables should be interpreted as directional guidance, not absolute benchmarks, until validated on physical devices.

8.7 LIMITATIONS AND FUTURE TESTING

While Sections 8.1–8.6 establish IFPP's theoretical and projected performance advantages, it is important to highlight current limitations and the path forward.

1. ***MODELED ASSUMPTIONS** – Tables 8.2 and 8.3 rely on estimated latency, bandwidth, and storage metrics. These are informed by current device specifications (e.g., smartphones, IoT, laptops) but not yet validated in heterogeneous, real-world environments.*

2. ***UNVERIFIED RESILIENCE** – Security and redundancy benefits are projected based on architectural design (Heavy Weapons, Gabriel's confirmation, Sacred Persistence). Full validation requires hostile-network testing and adversarial simulations.*

3. ***DEVICE DIVERSITY** – Real-world swarms will include devices with highly variable power, radio, and memory capacity. These disparities may introduce new failure modes that models do not capture.*

4. ***ENVIRONMENTAL NOISE** – Wireless interference, mobility, and terrain can degrade effective bandwidth in ways difficult to simulate without live deployments.*

8.7.1 PATH FORWARD

- *Prototype Development: Build a lightweight IFPP implementation in Python and deploy it on mixed-class devices (phones, laptops, IoT).*

- *Scaled Testing: Run controlled experiments at swarm sizes of 100, 1,000, and 10,000 devices to validate projections.*

- *Funding Needs: Support is required for hardware procurement, subscription tools (GitHub, VSCode Copilot, simulation platforms), and cloud infrastructure for federated learning.*

- *Iterative Refinement: Simulation results will feed back into model updates, tightening projections into validated benchmarks.*

8.7.2 KEY TAKEAWAYS

The IFPP architecture offers a promising leap beyond Gossip, Spray-and-Wait, and TCP/IP, but its strengths must be demonstrated empirically. Future testing — enabled through proper funding — will move IFPP from theoretical framework to operational reality.

8.8 IFPP VALIDATION ROADMAP

The diagram/flowchart illustrates the staged path from theoretical modeling to real-world validation:

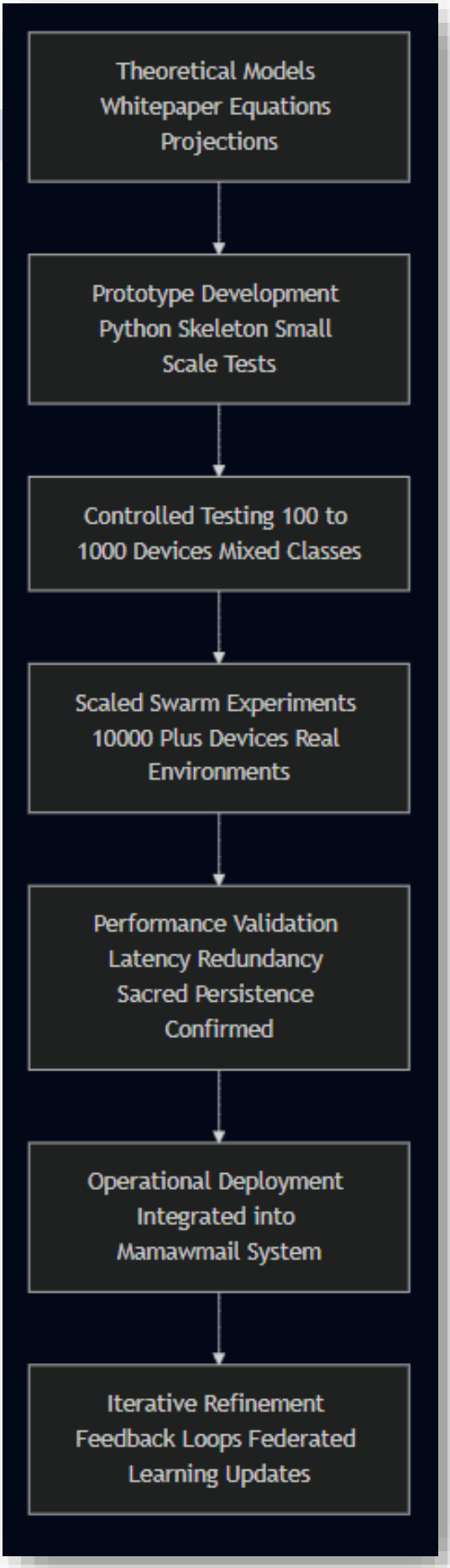


Figure 8.1 – IFPP Validation Roadmap

This figure illustrates the phased path from theoretical models through prototype code, controlled testing, and scaled swarm experiments, culminating in operational deployment within MAMAWMAIL.

Each stage feeds into the next via iterative refinement and federated learning updates, ensuring that IFPP matures from a conceptual model into a validated, field-ready protocol.

9.0 MVP SCOPE & CLARIFICATION

THIS SECTION DEFINES THE CONCRETE OUTCOMES OF THE PROPOSED MINIMUM VIABLE PRODUCT (MVP).

THE MVP IS DELIBERATELY SCOPED TO GO BEYOND A MERE PROTOTYPE, INCORPORATING COMPARATIVE TESTING AND DOCUMENTATION.

THIS ENSURES THAT THE SYSTEM IS BOTH SCIENTIFICALLY VALID AND ENGINEERING-READY

9.1 FUNDING SCOPE FOR MVP DEVELOPMENT

*This section outlines the resources required to complete a **minimum viable product (MVP)** of IFPP within the MAMAWMAIL system. The intention is to reach a stage where the protocol is not only implemented but also tested across a mix of **simulated and real devices**, providing credible comparisons with existing approaches such as Gossip, Spray-and-Wait, and TCP/IP.*

*The funding request is structured around **lean development**, with costs focused on personnel, essential hardware, and limited software subscriptions:*

- *A **two-person engineering team** (lead and junior engineer) responsible for design, implementation, and integration.*
- *****Student interns** supporting unit testing, device configuration, and documentation.*
- *A **small pool of test hardware** (10–12 smartphones, a few laptops/mini-PCs, IoT boards) to evaluate handoffs and swarm behavior in practice.*
- *****Cloud resources** to simulate larger swarm sizes (up to \~1,000 devices) and measure propagation metrics at scale.*

- ***Minimal software subscriptions and operational expenses** to maintain version control, prototyping tools, and testing infrastructure.*

*With this setup, a **baseline budget of \~€40,000** would enable delivery of a functioning MVP validated on hybrid simulations and real-device testing. An **extended budget of up to €75,000** would support larger-scale trials, more interns, and longer testing phases, moving the project closer to a publishable open-source release.*

*This funding scope is intended to balance **practical engineering needs** with **realistic cost efficiency**, ensuring that outputs are both technically credible and reproducible.*

9.2 FUNCTIONAL PROTOCOL IMPLEMENTATION

- *IFPP fully integrated into MAMAWMAIL.*
- *Device-to-device handoff cycles with Archangel Gabriel validation.*
- *Escort Team microarchitecture (7-agent system) operational.*

9.3 COMPARATIVE TESTING

- *Benchmarking against Gossip, Spray-and-Wait, and TCP/IP.*
- *Metrics: latency, redundancy, storage footprint, delivery success, resilience under churn.*
- *Small-scale device trials (10–12 physical devices + IoT boards).*
- *Medium-scale simulation (up to ~1,000 virtual nodes).*

9.4 DOCUMENTATION & EVALUATION

- *Whitepaper updated with real test results.*
- *Public GitHub repo with code, tests, and reproducible experiments.*
- *Comparative analysis reports showing IFPP advantages in quantifiable metrics.*

9.5 SCALING & FUNDING

- *€40,000 MVP: Focused trials, comparisons, publishable results.*
- *€75,000 MVP: Expanded field trials (100+ devices), deeper resilience analytics, extended community testing.*

Table 8.4 – Lean Funding Breakdown for MVP Development (6 Months)

Category	Item / Resource	Purpose / Justification	Estimated Cost (EUR)
Personnel	Lead Engineer (PI, 6 months)	System design, protocol coding, architecture lead	€18,000
	Junior Engineer (6 months)	Implementation, testing, bug fixing	€12,000
	Student Interns (4–8, stipends)	Unit testing, device setup, documentation	€4,000 – 6,000
Hardware	10–12 Mid-range Smartphones (Android)	Swarm-scale MVP testing (realistic conditions)	€3,000 – 4,000
	3–4 Laptops / Mini-PCs	Relay nodes, logging servers, Gabriel services	€2,000 – 3,000
	IoT Boards (Raspberry Pi, ESP32)	Lightweight relay simulation	€1,000 – 1,500
Software / Tools	GitHub Pro, VSCode Copilot, subscriptions	Repo management, AI-assisted prototyping	€500 – 1,000
	Cloud Instances (for scaling tests)	Simulated runs up to 1,000 nodes	€1,500 – 2,000
Operational	Miscellaneous (permits, power, admin)	Office, electricity, device upkeep	€1,000 – 1,500

Total (Lean MVP, 6 months): ~ €40,000 – €49,000
Stretch Goal (Expanded Trials, 12 months, more interns + scaling tests): ~ €75,000

At €40k: A working MVP of IFPP inside MAMAWMAIL, validated on ~1,000-device hybrid (physical + cloud) simulation, with real-device handoff + Gabriel validation.
At €75k: Expanded field tests, larger device pools, full documentation, and publication-ready open-source release.

10.0 IFPP AS NATIVE PROTOCOL

THE INTELLIGENT FRACTAL PROPAGATION PROTOCOL (IFPP) IS THE NATIVE TRANSPORT LAYER OF MAMAWMAIL.

ALL MESSAGES ARE CONSTRUCTED WITH ESCORT TEAMS, SACRED PERSISTENCE, AND SWARM INTELLIGENCE.

TCP/IP IS TREATED AS A FOREIGN PROTOCOL: USABLE ONLY AS A SUBSTRATE WHEN IFPP NODES OPPORTUNISTICALLY RIDE EXISTING WI-FI OR INTERNET CONNECTIONS TO ACCELERATE HOPS.

This dual model ensures that IFPP is:

1. ***INDEPENDENT** — capable of operating in fully disconnected environments.*
2. ***COMPATIBLE** — able to encapsulate payloads inside TCP/IP sockets when convenient.*
3. ***SEAMLESS** — applications use a single unified API, while the transport mode (native IFPP or foreign TCP/IP) is chosen automatically by the runtime.*

Layer (OSI)	*IFPP Role*	*TCP/IP Role (Foreign)*
Application	*MAMAWMAIL app*	*MAMAWMAIL app*
Transport (Native)	*IFPP Escort Team + Gabriel*	*TCP, UDP sockets*
Network	*IFPP Crawler (Bluetooth, UDP punch)*	*IP (IPv4/IPv6)*
Data Link & Physical	*Device radios (Wi-Fi Direct, BLE, LTE)*	*Same*

10.2 WORKFLOW

1. *App creates Eternal Message Core.*
2. *IFPP Escort Team instantiated (Recon, Intel, Leader, etc.).*
3. *If swarm peers are nearby → native IFPP propagation.*
4. *If swarm peers unreachable but Internet/Wi-Fi present → payload encapsulated inside TCP/IP tunnel.*
5. *Handoff confirmed by Gabriel regardless of channel.*

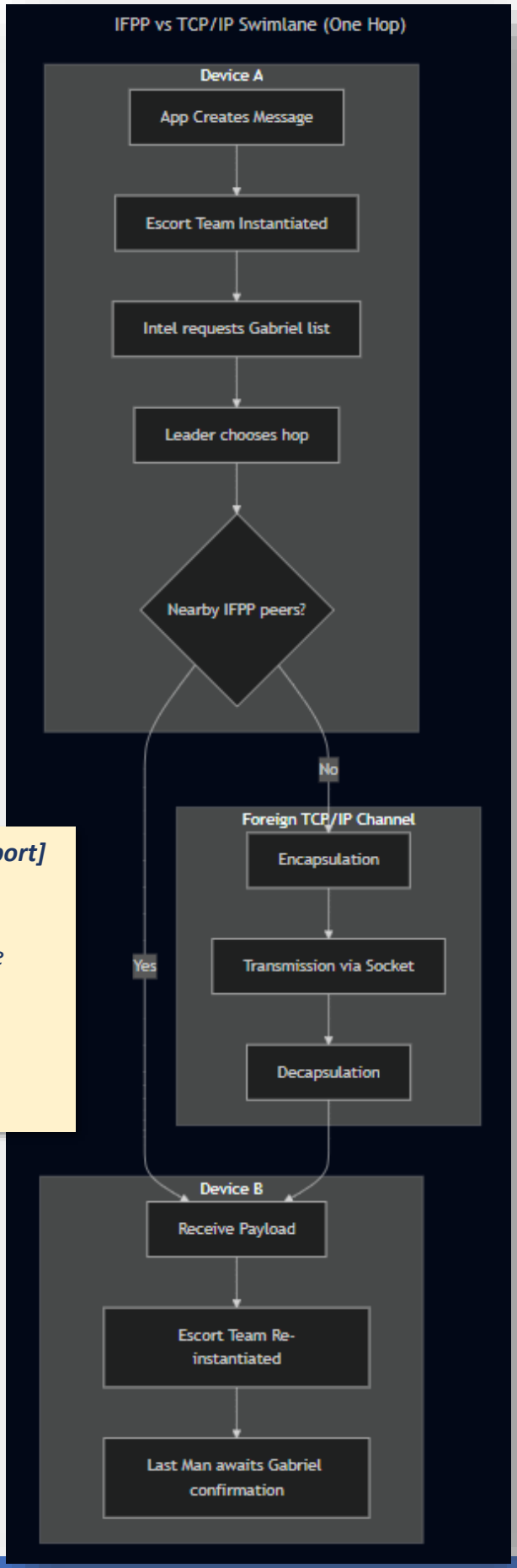


Figure 10.1 – Decision Flow [NATIVE vs Foreign Transport]

Decision flow showing how IFPP selects between native swarm propagation and foreign encapsulation (TCP/IP, Bluetooth, Wi-Fi) based on environment, candidate availability, and path history.

10.4 SIGNIFICANCE

1. *Native-first: Messages always try IFPP → swarm propagation.*
2. *Foreign optional: TCP/IP used only as "carrier of convenience," not as the foundation.*
3. *Application simplicity: App code never changes — transport is abstracted away.*
4. *Evaluator clarity: Shows IFPP is a protocol in its own right, not just "DTN on top of TCP/IP."*

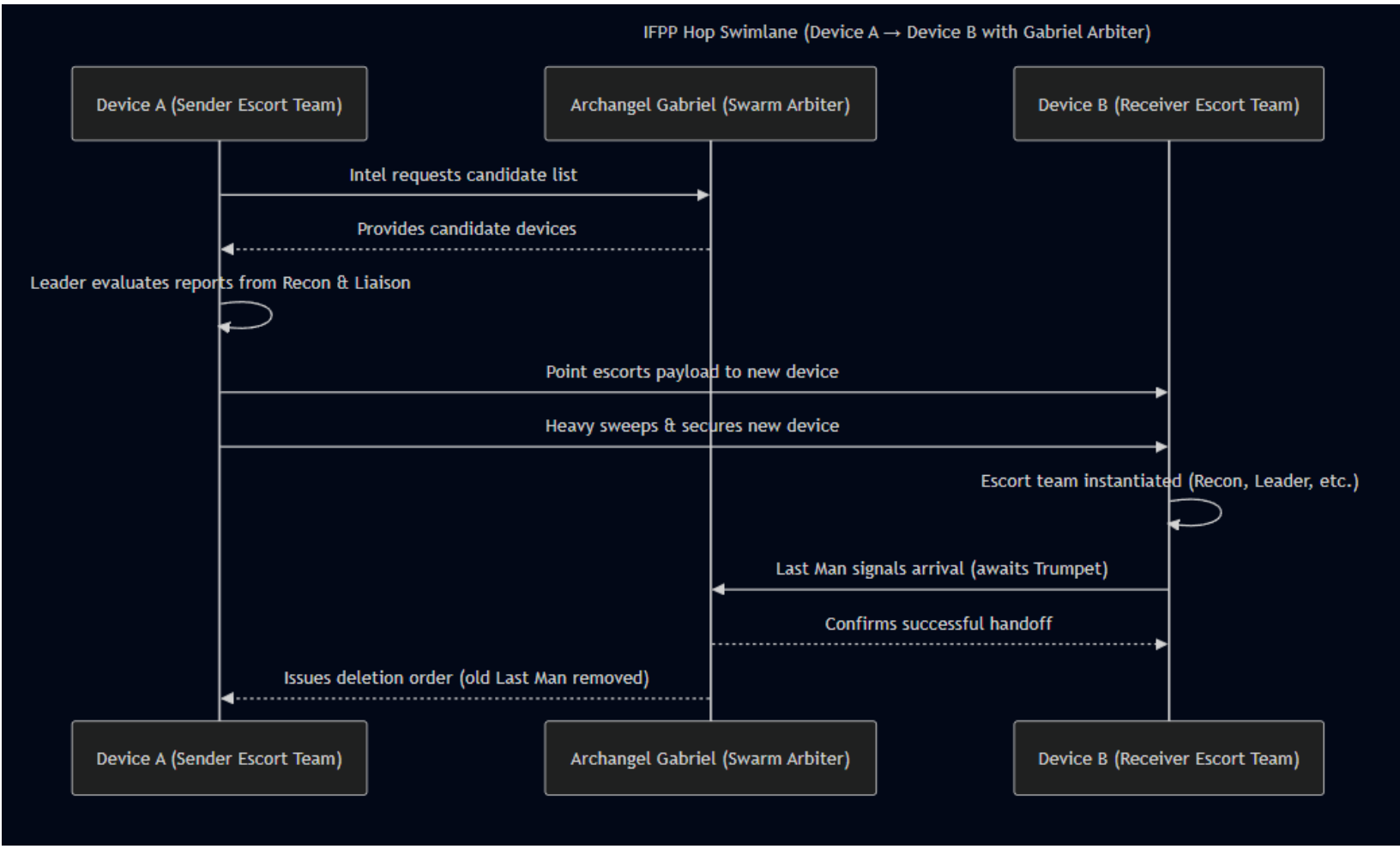


Figure 10. 2 – Hop Swimlane (Device A → Device B with Gabriel Arbiter)

Hop swimlane illustrating coordination between Device A (sending escort team), Device B (receiving escort team), and Gabriel (swarm-wide arbiter).

Gabriel confirms success and authorizes cleanup, ensuring Sacred Persistence without duplication.

11.0 IFPP RFC DRAFT (SPECIFICATION)

THE INTELLIGENT FRACTAL PROPAGATION PROTOCOL (IFPP) IS A NATIVE TRANSPORT PROTOCOL FOR MAMAWMAIL

IT GUARANTEES SACRED PERSISTENCE OF ENCRYPTED PAYLOADS, SWARM-LEVEL RESILIENCE, AND ADAPTIVE PROPAGATION.

THIS RFC DRAFT DEFINES THE PACKET FORMATS, STATE MACHINES, AND ERROR HANDLING NEEDED TO IMPLEMENT IFPP.

STATUS : DRAFT 0.1

Authors : Engr Juan Carlos G. Ayeng, RN (MAMAWMAIL Project)

Origin : Bacolod City, The Philippines

Date Started : June 20, 2025

Category : **Experimental Transport Protocol**

Intended Use : Native protocol for disconnected, censored, or infrastructure-destroyed networks.

This RFC section elevates IFPP to the level of a true protocol stack, aligned with Mamawmail's 7-architecture system.

11.1 **NATIVE LAYER STACK** (ALIGNED WITH 7 MAMAWMAIL ARCHITECTURES)

1. MESSAGE ARCHITECTURE LAYER

i. Structures the IFPP packet into 3 modular components:

- **EPHEMERAL HEADER** (routing, been-here flags, nonces).
- **PATH/AI METADATA** (Gabriel digest, candidate scoring).
- **ETERNAL MESSAGE CORE** (AES-256 encrypted payload).

ii. Escort Team MUST be instantiated for each message.

iii. Sacred Persistence: Eternal Core MUST NOT expire until delivered.

2. PROPAGATION ARCHITECTURE LAYER

i. Messages MUST propagate via fractal expansion (3^n candidates).

ii. After threshold ε, propagation MUST converge to singular path (lowest latency, no duplicates).

iii. Escort Team flow:

- **Intel → Leader → Recon → Point → Heavy → Liaison → LastMan.**

iv. Gabriel MUST confirm each hop (Trumpet) before old copy deletion.

3. CRAWLER ARCHITECTURE LAYER

i. Native IFPP nodes MUST scan periodically for nearby devices (Bluetooth, Wi-Fi Direct, UDP hole punching).

ii. Device discovery MUST occur even when idle, to maintain up-to-date candidate lists.

iii. Intel angel MUST obtain candidate list from Gabriel; Recon MUST physically probe.

4. INTELLIGENCE LAYER

i. Reverse path digests MUST be returned to Gabriel.

ii. Devices MUST store hop outcomes (success/fail) for federated swarm learning.

iii. Payload is never shared; only metadata (paths, latencies, retries).

5. SELF-HEALING / SELF-PRUNING LAYER

i. LastMan MUST delete old instances upon Trumpet.

ii. Failed paths MUST be pruned after retries exceed δ.

iii. Redundant swarm copies MUST converge via singularity rules.

6. IFPP TRANSPORT & INTERFACE LAYER

i. Native mode: IFPP frames travel directly over radios.

ii. Foreign mode: IFPP frames MAY be encapsulated into TCP/UDP sockets.

iii. Application MUST see a single API: send_ifpp(message) — transport decision is runtime.

7. SINGULARITY ARCHITECTURE LAYER

i. **HORIZONTAL SINGULARITY**:

- saturation point when all device paths are mapped.
- Formula:

a. $S_h = f(N, b, \varepsilon)$

b. where N = nodes, b = branching factor, ε = pruning threshold.

ii. **VERTICAL SINGULARITY**:

- minimum # of packet copies to ensure delivery with < L latency.
- Formula:

a. $S_v = \min(k)\ s.t.\ P_{delivery}(k) \geq 0.999.$

iii. Swarm MUST auto-adjust between horizontal and vertical modes.

*11.2 **TERMINOLOGY***

- ***ESCORT TEAM** — Seven-agent microarchitecture per hop*
    - *(Recon, Point, Leader, Intel, HeavyWeapons, Liaison, LastMan).*
- ***GABRIEL** — Swarm-wide registry and integrity arbiter.*
- ***ETERNAL MESSAGE CORE (EMC)** — Immutable, encrypted payload carried across hops.*
- ***TRUMPET** — Gabriel's confirmation that a handoff succeeded.*
- ***NATIVE MODE** — IFPP over direct radios (Wi-Fi Direct, BLE, LTE, UDP hole-punch).*
- ***FOREIGN MODE** — IFPP frames encapsulated inside TCP/IP sockets*

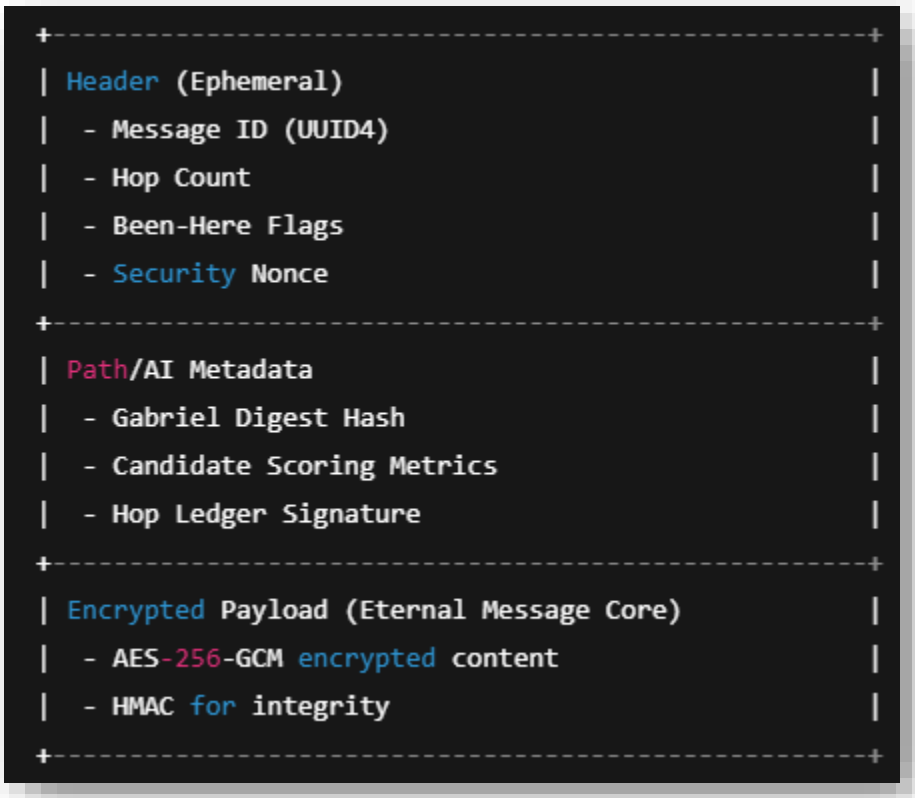
.

11.3 **PROTOCOL STACK MAPPING**

OSI Layer	*IFPP Component*	*Foreign TCP/IP Equivalent*
Application	*Mamawmail App*	*Mamawmail App*
Transport (Native)	*IFPP Escort Team + Gabriel*	*TCP/UDP sockets*
Network	*IFPP Crawler (Bluetooth, UDP)*	*IP (v4/v6)*
Data Link/Physical	*Device radios (Wi-Fi, BLE, LTE)*	*Same*

11.4 **PACKET STRUCTURE**

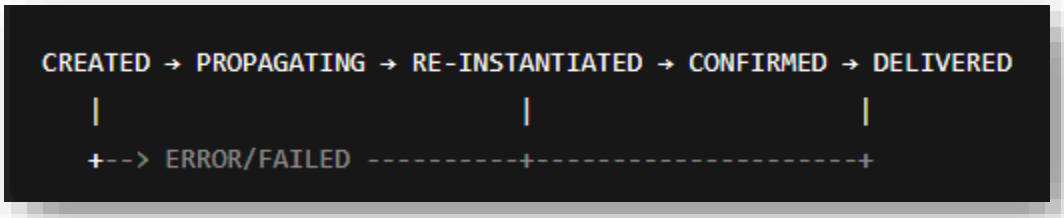
Each IFPP packet has 3 modular parts:



11.5 PROTOCOL ELEMENTS

- **PACKET FORMAT**: (Header | AI Metadata | Eternal Core).
- **STATE MACHINE**: CREATED → PROPAGATING → REINST → CONFIRMED → DELIVERED.
- **SECURITY**: AES-256-GCM (payload), Ed25519 signatures (ledger), ChaCha20-Poly1305 (headers).
- **ERROR HANDLING**: retries on timeout; prune duplicates; discard tampered packets.

11.5.1 **FINITE STATE MACHINE** [FSM]



State	Description
CREATED	Message + escort team instantiated on source device.
PROPAGATING	Active handoff in progress (Intel → Leader → Recon → Point → Heavy → Liaison → LastMan).
RE-INSTANTIATED	Escort dissolved on old device, reborn on new device with Eternal Core.
CONFIRMED	Gabriel validates handoff, authorizes deletion.
DELIVERED	Recipient has payload, escort dissolves permanently.
FAILED	(Optional) Only if device destruction or payload corruption detected.

- *Created: Device spawns escort team.*
- *Propagating: Recon/Point attempt hop.*
- *Re-Instantiated: LastMan on new device alive.*
- *Confirmed: Gabriel issues Trumpet.*
- *Delivered: Final recipient opens payload.*
- *Error/Failed: Hop fails, retry next candidate.*

11.5.2 **ERROR CODES**

Code	Meaning	Action
100	*Candidate device unreachable*	*Retry via alternative candidate (Leader logic).*
200	*Been-here flag detected*	*Skip candidate, mark path as stale.*
300	*Security anomaly detected*	*Heavy engages defense; retry hop.*
400	*Gabriel confirmation timeout*	*Roll back, retry handoff with next candidate.*
500	*Payload corruption*	*Abort handoff, restore from Eternal Core.*

11.5.3 **ERROR HANDLING**

- ***TIMEOUTERROR*** — No Trumpet received → retry another candidate.
- ***INTEGRITYERROR*** — Gabriel detects tampering → discard instance.
- ***REDUNDANCYEXCEEDED*** — Too many duplicates → prune swarm.

11.5.3.4 **SECURITY PRIMITIVES**

- Payload: AES-256-GCM.
- Headers: CHACHA20-POLY1305 LIGHTWEIGHT.
- Ledger Signatures: **Ed25519.**
- Device IDs: UUID + public key.

11.6 **EXAMPLE PSEUDOCODE** (SEND LOOP)

```
def ifpp_send(device, message):
    escort = device.instantiate_escort(message)
    candidates = escort.intel.ask_gabriel()
    for c in candidates:
        if escort.recon.probe(c):
            if escort.point.escort(c):
                if gabriel.confirm(message.id, device, c):
                    device.lastman.cleanup_old()
                    return "DELIVERED"
    if device.has_tcpip():
        encapsulate_tcpip(message)
        send_over_socket(message)
```

11.9 **COMPLIANCE CHECKLIST**

- MUST implement Escort Team 7-agent cycle.
- MUST support Gabriel confirmation and cleanup.
- MUST provide both native and TCP/IP encapsulation modes.
- MUST retain Sacred Persistence of payload until Delivered.
- SHOULD support mathematical singularity thresholds.

11.10 **VERSIONING & EXTENSIBILITY**

- Draft-0.1 (Current): Base architecture, packet format, FSM, escorts.
- Draft-0.2: Adds Singularity thresholds, advanced AI learning.
- Draft-1.0: Stable release.
- Future: Post-Quantum encryption swap-outs.

11.11 **DEVELOPER TAKEAWAY**

When designing an app:

- Write to the Application Layer **(call IFPP's Message() API).**

- **IFPP handles escort instantiation, propagation, Gabriel sync, pruning, and fallback to TCP/IP automatically.**

- IFPP is not just another transport; it is a full-stack messaging paradigm, but designed to ***interoperate with TCP/IP*** when needed.

Just as TCP/IP became the substrate of the modern Internet, IFPP defines a parallel substrate for disconnected, censored, or destroyed networks — native, persistent, and swarm-intelligent.

12.0 IFPP NORMATIVE DRAFT

(IETF-STYLE SPECIFICATION)

Status	:	Draft 0.1
Authors	:	Engr. Juan Carlos G. Ayeng, RN (Mamawmail Project)
Origin	:	Bacolod City, The Philippines
Date Started	:	June 20, 2025
Category:		Experimental Transport Protocol
Intended Use	:	Native protocol for disconnected, censored, or infrastructure-destroyed networks.

12.1 **ABSTRACT**

The Intelligent Fractal Propagation Protocol (IFPP) is a native transport layer protocol designed for the Mamawmail system.

Unlike TCP/IP, which assumes centralized routing and ephemeral packets, IFPP employs:

- Sacred Persistence of payloads (no TTL expiration),
- Escort Team microarchitecture (seven cooperating agents), and
- Gabriel swarm arbiter (for integrity, validation, and pruning).

This draft defines the packet format, state machine, error handling, and compliance rules required to implement IFPP.

12.2 PROTOCOL OVERVIEW

- IFPP is the native transport of Mamawmail.
- TCP/IP is treated as a foreign protocol, encapsulation-only.
- Applications call a unified API; transport selection (native or foreign) is runtime-determined.

12.3 **LAYERED ARCHITECTURE** (ALIGNED WITH MAMAWMAIL 7-ARCHITECTURES)

1. **MESSAGE ARCHITECTURE LAYER**
    i. Packet composed of:
        1. **EPHEMERAL HEADER** (routing, been-here flags, nonces).
        2. **AI METADATA** (Gabriel digests, path scoring).
        3. **ETERNAL MESSAGE CORE** (AES-256 encrypted payload).
    ii. Each message MUST instantiate an escort team.
    iii. Payload MUST persist until delivery is confirmed.

2. **PROPAGATION ARCHITECTURE LAYER**
    i. Messages propagate fractally (branch factor b).
    ii. After pruning threshold ε, propagation converges to a single path.
    iii. Escort cycle:
        1. Intel → Leader → Recon → Point → Heavy → Liaison → LastMan.
    iv. Gabriel MUST confirm each hop (Trumpet) before cleanup.

3. **CRAWLER ARCHITECTURE LAYER**
    i. Devices MUST continuously scan for neighbors (Wi-Fi Direct, BLE, UDP).
    ii. Intel requests candidate lists from Gabriel.
    iii. Recon physically probes devices.

4. **INTELLIGENCE LAYER**
    i. Reverse paths MUST be logged and returned to Gabriel.
    ii. Payload is never exposed; only metadata (latency, retries, failures).

5. **SELF-HEALING / SELF-PRUNING LAYER**
    i. LastMan MUST delete old instances after confirmation.
    ii. Failed paths MUST be pruned after δ retries.
    iii. Redundancy MUST converge toward singular paths.

6. **TRANSPORT & INTERFACE LAYER**
    i. Native: direct radio transport.
    ii. Foreign: encapsulated inside TCP/UDP.
    iii. Applications MUST see a single API: send_ifpp(message).

7. **SINGULARITY LAYER**
    i. **HORIZONTAL SINGULARITY**: path saturation,
        1. $S_h = f(N, b, \varepsilon)$.
    ii. **VERTICAL SINGULARITY:** minimal redundancy,
        1. $S_v = \min(k): P_{delivery}(k) \geq 0.999$.
    iii. Swarm MUST auto-adjust between both modes.

12.4 PROTOCOL STACK MAPPING

OSI Layer	IFPP Component	TCP/IP Equivalent
Application	Mamawmail App	Mamawmail App
Transport (Native)	IFPP Escort Team + Gabriel	TCP/UDP sockets
Network	IFPP Crawler (BLE, Wi-Fi Direct)	IP (v4/v6)
Data Link/Physical	Device radios (Wi-Fi, LTE, BLE)	Same

11.5 PACKET STRUCTURE

- **HEADER:** Routing metadata, been-here flags, timestamps.
- **AI METADATA**: Candidate scoring, Gabriel digests, hop history.
- **ETERNAL CORE**: Immutable encrypted payload (AES-256-GCM).

12.6 FINITE STATE MACHINE (FSM)

State	Description
CREATED	Message + escort team instantiated.
PROPAGATING	Active handoff in progress.
RE-INSTANTIATED	Escort reborn on new device.
CONFIRMED	Gabriel validates, issues Trumpet.
DELIVERED	Payload accepted by recipient.
FAILED	Device loss / corruption detected.

12.7 **ERROR HANDLING**

Code	Meaning	Action
100	Candidate unreachable	Retry next candidate
200	Been-here flag detected	Skip candidate
300	Security anomaly detected	Heavy defends, retry
400	Gabriel timeout	Retry via new hop
500	Payload corruption	Abort, restore from Eternal Core

SECURITY PRIMITIVES

- PAYLOAD: **AES-256-GCM**

- HEADERS: **ChaCha20-Poly1305**
- SIGNATURES: **Ed25519**

12.8 **EXAMPLE PSEUDOCODE** (SEND LOOP)

```
def ifpp_send(device, message):
    escort = device.instantiate_escort(message)
    candidates = escort.intel.ask_gabriel()
    for c in candidates:
        if escort.recon.probe(c):
            if escort.point.escort_payload(c):
                if gabriel.confirm(message.id, device, c):
                    device.lastman.cleanup_old()
                    return "DELIVERED"
    if device.has_tcpip():
        encapsulate_tcpip(message)
        send_over_socket(message)
```

12.9 **COMPLIANCE CHECKLIST**

- MUST implement 7-agent escort cycle.

- MUST support Gabriel confirmation & cleanup.
- MUST support native + TCP/IP encapsulation.
- MUST retain Sacred Persistence until Delivered.
- SHOULD optimize propagation via singularity thresholds.

12.10 VERSIONING

- Draft-0.1: Base architecture, packet format, FSM.
- Draft-0.2: Adds Singularity thresholds, swarm AI.
- Draft-1.0: Stable release candidate.
- Future: Post-quantum encryption.

12.11 DEVELOPER TAKEAWAY

Developers interact with IFPP through a simple API (Message(), send_ifpp()).

The protocol stack handles:

- Escort instantiation,
- Gabriel validation,
- Fractal/Single-hop propagation,
- Cleanup,
- TCP/IP fallback.

IFPP is to censored/disconnected networks what TCP/IP was to the Internet: a substrate for resilience, persistence, and swarm intelligence.
