ToM : Conception et Validation Empirique d'un Protocole de Messagerie Pair-à-Pair Sans Infrastructure Fixe

Malik — Février 2026

Résumé

Ce rapport présente ToM (The Open Messaging), un protocole de transport décentralisé inspiré de mécanismes biologiques, où chaque appareil agit simultanément comme client et relais. Contrairement aux architectures classiques (Signal, WhatsApp) reposant sur des serveurs centraux, et aux protocoles décentralisés existants (libp2p, Hyperswarm, Nostr, Tor) nécessitant encore une forme d'infrastructure fixe, ToM propose un modèle radicalement autonome : le réseau *est* l'infrastructure, les rôles sont imposés et rotatifs (imprévisibles pour un attaquant), les messages se répliquent comme des virus bénéfiques pour survivre 24h, et le bootstrap lui-même est distribué — sans point d'entrée fixe. Nous décrivons le pipeline cryptographique (X25519 éphémère + XChaCha20-Poly1305 + HKDF-SHA256, encrypt-then-sign), analysons la résistance aux attaques MITM, 51%, Sybil et Eclipse, et montrons pourquoi l'absence de consensus rend l'attaque 51% économiquement irrationnelle. Nous présentons des résultats expérimentaux sur 4 campagnes de stress test (2 752 pings, 99.85% fiabilité en mobilité autoroutière, 0.98 ms en 5G) et des tests cross-border Suisse↔France (27 ms, chiffrement E2E vérifié). Nous comparons ToM à BitTorrent, Tor, Nostr, libp2p et iroh, et montrons qu'aucun protocole existant ne combine contribution obligatoire, rôles rotatifs imprévisibles, anti-spam sans censure, messages auto-réplicants, et purge inconditionnelle.

1. Introduction

1.1 Problème posé

Les systèmes de messagerie actuels reposent sur un paradoxe architectural : ils promettent la confidentialité tout en centralisant le transit des messages. Signal, malgré son chiffrement de bout en bout, maintient des serveurs centraux qui voient les métadonnées (qui parle à qui, quand, combien).

WhatsApp, Telegram, iMessage — même schéma. Le chiffrement protège le contenu, mais l'infrastructure trahit le graphe social.

Les protocoles décentralisés existants tentent de résoudre ce problème :

- **libp2p** (Protocol Labs) : multi-langage, écosystème mature, mais philosophie relay-first — les relais sont une optimisation, pas le fondement architectural. La complexité initiale est élevée et la dépendance aux relais de bootstrap reste structurelle.
- **Hyperswarm** (Holepunch/Mafintosh) : philosophie DHT-first alignée avec notre vision, mais limité à Node.js, aucun support navigateur, communauté restreinte.
- **Matrix/Nostr** : fédération de serveurs (Matrix) ou relais volontaires (Nostr) — dans les deux cas, quelqu'un doit payer et maintenir l'infrastructure.

Le problème fondamental reste non résolu : comment construire un réseau de messagerie qui fonctionne sans qu'aucune entité ne doive maintenir de serveur ?

1.2 Inversion économique

Les architectures centralisées souffrent d'un problème de scaling linéaire : plus d'utilisateurs → plus de serveurs → plus de coûts. La charge et le coût croissent ensemble.

ToM propose l'inversion : plus de participants → plus de relais disponibles → plus de chemins alternatifs → latence réduite → coût zéro. Chaque appareil qui rejoint le réseau *augmente* sa capacité. C'est une propriété émergente de l'architecture, pas une optimisation. Cette inversion ne peut pas être répliquée par une architecture centralisée.

1.3 Contributions

Ce travail apporte :

1. Un **modèle de nœud unifié** où chaque appareil exécute un code identique — le rôle (client, relais, backup) est imposé dynamiquement par la topologie, jamais choisi
2. Un **système de réplication virale** pour les messages destinés à des nœuds hors-ligne, avec TTL strict de 24h et auto-suppression
3. Une **validation expérimentale** avec 4 campagnes de stress test incluant mobilité autoroutière, CGNAT opérateur, et traversée de frontière Suisse↔France
4. Un **prototype fonctionnel** en Rust (1 008+ tests) validant le transport QUIC, le chiffrement E2E, et la découverte par gossip

1.4 Inspiration biologique : le réseau comme organisme vivant

ToM n'est pas seulement un protocole réseau. C'est un **organisme numérique** dont l'architecture s'inspire directement de mécanismes biologiques. Cette inspiration n'est pas métaphorique — elle est structurelle.

Le message comme virus bénéfique. Dans la nature, un virus a un seul objectif : survivre assez longtemps pour atteindre un hôte réceptif. Il mute, il change d'hôte quand celui-ci faiblit, il se réplique pour maximiser ses chances. Dans ToM, un message destiné à un nœud hors-ligne se comporte exactement ainsi : il se réplique sur 3 à 5 nœuds backup, il surveille la "santé" de chaque hôte (bande passante, uptime, timezone compatible avec le destinataire), il **migre proactivement** vers un meilleur hôte avant que l'actuel ne tombe — parce qu'attendre la mort de l'hôte, c'est déjà trop tard. Et comme un virus, il meurt après un temps défini (24h) si sa mission échoue. Plus il y a d'hôtes dans le réseau, plus le virus est résilient. **Un virus positif : plus il y a d'hôtes, plus l'organisme est fort.**

Le réseau comme système immunitaire. Le mécanisme anti-spam "l'arroseur arrosé" fonctionne comme un système immunitaire adaptatif. Il ne détruit pas l'intrus (pas de bannissement) — il l'identifie progressivement et augmente la réponse immunitaire (charge de travail accrue) jusqu'à ce que l'intrus s'épuise. Pas de cellule NK qui tue sur contact (pas de blacklist), mais une réponse T qui force l'adaptation ou l'abandon. Le réseau s'endurcit par les attaques qu'il subit.

La genèse glissante : un organisme sans squelette fossile. Les blockchains accumulent des millions de blocs — un squelette fossile qui s'alourdit avec le temps. L'architecture L1 de ToM utilise une "genèse glissante" : plutôt que d'accumuler l'historique, la couche d'ancrage reste proche d'un bloc genèse en mouvement perpétuel. Les anciens états sont compactés en snapshots cryptographiques puis purgés. Le réseau ne porte pas le poids de son passé — comme un organisme qui renouvelle ses cellules, il ne conserve que l'état présent.

Les sous-réseaux éphémères : des organes temporaires. Quand un pattern de communication émerge (un groupe de nœuds qui échangent fréquemment), un sous-réseau se forme spontanément — comme un organe qui se développe en réponse à un besoin. Quand le besoin disparaît, le sous-réseau se dissout. Pas de structure permanente inutile, pas de dette architecturale. Le réseau **respire** : il se structure et se destructure en continu.

Proof of Presence : le droit d'exister par la présence. Ni proof-of-work (droit par le calcul), ni proof-of-stake (droit par le capital). ToM introduit le concept de **proof of presence** : le droit de participer aux décisions du réseau est donné par la présence active. Pas par ce qu'on possède, pas par ce qu'on investit, mais par le simple fait d'être là et de contribuer. C'est le modèle le plus égalitaire possible — un smartphone sur un réseau mobile a autant de droits qu'un data center, tant qu'il est présent.

1.5 Vision : un protocole pour un internet libre

> "*L'objectif : une couche de communication universelle, résiliente, sans point de contrôle central, qui se suffit à elle-même. Un nouveau protocole pour un internet qui n'appartient à personne.*"
> — Whitepaper ToM v1, janvier 2026

ToM n'est pas une application à installer. Il a vocation à devenir une **brique protocolaire intégrée dans les outils du quotidien** — navigateurs, messageries, clients mail — de sorte que l'utilisateur participe au réseau sans le savoir. Comme TCP/IP transporte les paquets sans que l'utilisateur ne le sache, ToM transportera les messages sans être visible.

Le code source lui-même, à terme, sera **auto-hébergé sur le réseau ToM**. Si GitHub tombe, si un gouvernement ordonne le retrait du dépôt, si une entreprise fait pression — le code vit sur le réseau qu'il a créé. La documentation, les issues, le workflow de développement : tout distribué sur ToM. Le protocole héberge sa propre évolution. **Le cordon ombilical est coupé quand le bébé respire seul.**

C'est la propriété la plus radicale : une fois lancé avec suffisamment de nœuds, **ToM ne peut plus être arrêté** — seulement évolué. Comme le réseau Bitcoin ne peut pas être "éteint" (il faudrait éteindre des dizaines de milliers de machines dans des dizaines de pays simultanément), mais sans l'incitation financière qui concentre le pouvoir. Il n'y a pas de tokens à voler, pas de mining pools à cibler, pas de fondation à poursuivre en justice. Il n'appartient à personne parce qu'il appartient à tout le monde.

2. État de l'Art

2.1 Taxonomie des approches existantes

Critère	Signal	libp2p	Hyperswarm	Nostr	ToM
Infrastructure requise	Serveurs centraux	Relais bootstrap	DHT bootstrap	Relais volontaires	**Aucune (post-bootstrap)**
Métadonnées visibles	Serveur voit graphe social	Relais voient source/dest	DHT voit lookups	Relais voient tout en clair	**Relais voient uniquement from/to chiffré**

Critère	Signal	libp2p	Hyperswarm	Nostr	ToM
NAT traversal	N/A (client-serveur)	relay + DCUtR	UDX hole punch	N/A (client-serveur)	**QUIC hole punch + relay fallback**
Contribution des nœuds	Passive (consommation)	Optionnelle	Optionnelle (seeding)	Volontaire (relais)	**Obligatoire et invisible**
Persistence messages	Indéfinie (serveur)	Dépend de l'app	Aucune	Indéfinie (relais)	**24h TTL strict, puis purge**
Chiffrement E2E	Oui (Signal Protocol)	Au choix de l'app	SecretStream	Non (par défaut)	**Oui (XChaCha20-Poly1305)**

2.2 Limites identifiées

libp2p résout le transport multi-protocole mais impose une complexité d'intégration disproportionnée pour un cas d'usage messagerie. Le protocole relay (Circuit Relay v2) traite les relais comme un palliatif au NAT, pas comme le cœur architectural. De plus, la phase initiale de connexion dépend toujours de relais connus à l'avance.

Hyperswarm a la bonne philosophie (DHT-first, hole punch natif) mais l'implémentation Node.js-only limite le déploiement. Son successeur technique, le stack Holepunch (Pear Runtime), reste propriétaire dans sa distribution.

Nostr démocratise l'accès aux relais mais souffre d'un problème de free-riding : les opérateurs de relais paient l'infrastructure sans garantie de rétribution. Les messages sont stockés indéfiniment, créant un problème de scaling du stockage.

2.3 Choix de iroh comme fondation de transport

Après analyse comparative, nous avons choisi **iroh** (n0-computer, 7 800+ stars GitHub, Rust, MIT) comme couche de transport à étudier et éventuellement forker :

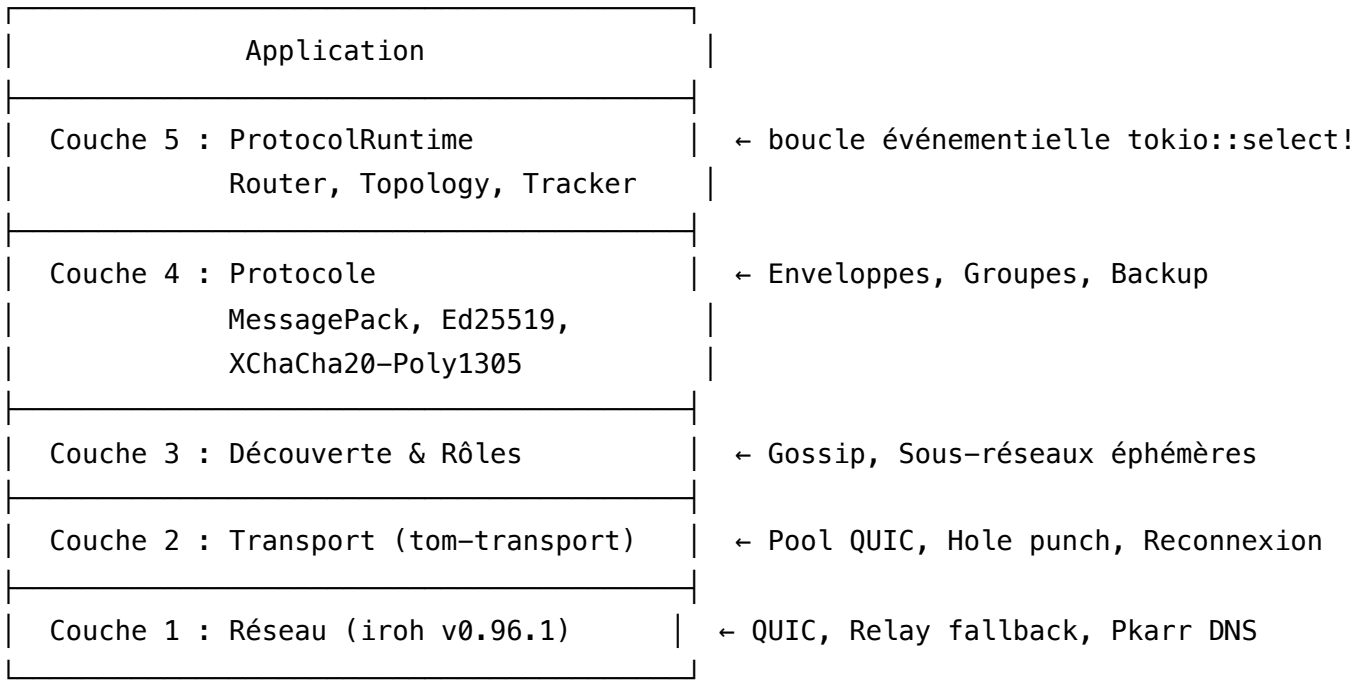
Critère	iroh	Justification
Transport	QUIC natif	Multiplexage, 0-RTT, migration de connexion
Identité	Ed25519 = adresse réseau	Alignement exact avec le modèle ToM
NAT traversal	Hole punch + relay fallback	~90% connexions directes en production
Découverte	Pkarr (DNS-like) + gossip	Bootstrap décentralisé
Relais	Stateless, pass-through	Philosophie identique à ToM
Chiffrement	QUIC TLS automatique	E2E au niveau transport
Licence	MIT	Fork possible sans contrainte

Décision stratégique : iroh est utilisé comme dépendance pour le PoC, avec l'objectif explicite de forker les modules nécessaires une fois le protocole ToM stabilisé. Aucune dépendance permanente sur n0-computer.

3. Architecture

3.1 Modèle de nœud unifié

Chaque nœud ToM exécute un code identique. Il n'existe pas de binaire "client" distinct d'un binaire "serveur" :



Le rôle d'un nœud (relais, backup, simple client) est déterminé dynamiquement par le réseau en fonction de sa disponibilité, son score de contribution, et la topologie locale. Un nœud ne *choisit* jamais d'être relais — il le devient quand le réseau l'exige. C'est la différence fondamentale avec BitTorrent (seeding optionnel) ou Nostr (relais volontaires).

3.2 Format d'enveloppe

Les messages sont encapsulés dans des enveloppes sérialisées en MessagePack (≈60% plus compact que JSON) :

```rust
pub struct Envelope {
    id: String,              // UUIDv4
    from: NodeId,            // Clé publique Ed25519 de l'émetteur
    to: NodeId,              // Clé publique Ed25519 du destinataire
    via: Vec<NodeId>,        // Chaîne de relais (ordonnée)
    msg_type: MessageType,   // Chat, Ack, Heartbeat, Group*, Backup*...
    payload: Vec<u8>,        // Texte clair ou chiffré
    timestamp: u64,          // Millisecondes Unix
    signature: Vec<u8>,      // Signature Ed25519 (64 octets)
    ttl: u32,                // Compteur de sauts (max 4)
    encrypted: bool,         // Indicateur de chiffrement du payload
}
```

Point critique : la signature couvre tous les champs *sauf* `ttl` et `signature` elle-même. Le TTL est exclu car les relais le décrémentent en transit — inclure le TTL invaliderait la signature après chaque saut. Cela permet aux relais de vérifier l'authenticité de l'enveloppe sans pouvoir la modifier autrement.

3.3 Échange de clés : comment deux nœuds partagent un secret sans que le relais ne le voie

Le problème central de tout système E2E traversant des relais est : **comment Alice et Bob établissent-ils une clé symétrique commune sans qu'un relais intermédiaire puisse la reconstituer ?**

La réponse classique est le protocole Diffie-Hellman (DH). ToM utilise une variante moderne : **X25519 éphémère-statique** (ECDH sur Curve25519).

3.3.1 Le problème de l'échange de clés en présence de relais

Dans ToM, chaque message traverse au moins un relais. Le relais voit passer l'intégralité de l'enveloppe. Si Alice chiffrait avec une clé qu'elle envoie dans le message, le relais pourrait l'extraire. La solution : **ne jamais transmettre la clé elle-même**, mais transmettre uniquement l'information nécessaire pour que Bob la *recalcule* — information inutile à quiconque n'est pas Bob.

3.3.2 Identité cryptographique des nœuds

Chaque nœud possède une paire de clés Ed25519 générée à la première connexion :

- **Clé privée** : 32 octets (seed), jamais transmise
- **Clé publique** : 32 octets = **adresse réseau** du nœud (pas de registre central, pas de DNS)

La clé publique Ed25519 est connue du réseau — c'est l'identifiant du nœud. Quand Alice veut écrire à Bob, elle connaît déjà sa clé publique Ed25519 (découverte via gossip ou échange préalable).

3.3.3 Conversion Ed25519 → X25519

Ed25519 opère sur la courbe d'Edwards (signature). Le Diffie-Hellman requiert X25519, qui opère sur la courbe de Montgomery. Les deux courbes sont **birationnellement équivalentes** via :

```
x_montgomery = (1 + y_edwards) / (1 − y_edwards)
```

Pour la clé publique :

```
// Décompresser le point Edwards, convertir en Montgomery
let edwards = CompressedEdwardsY(ed25519_pk_bytes).decompress()?;
let montgomery = edwards.to_montgomery();  // → X25519 public key
```

Pour la clé privée (seed) :

```
// SHA-512(seed), prendre les 32 premiers octets, appliquer le clamping X25519
let hash = Sha512::digest(ed25519_seed);
let mut secret = [0u8; 32];
secret.copy_from_slice(&hash[..32]);
secret[0] &= 248;      // Effacer les 3 bits bas
secret[31] &= 127;     // Effacer le bit haut
secret[31] |= 64;      // Fixer le bit 6
```

Ce clamping est standard (RFC 7748) : il force la clé dans le sous-groupe d'ordre premier, éliminant les attaques par petit sous-groupe. C'est exactement l'opération de `libsodium crypto_sign_ed25519_sk_to_curve25519` .

3.3.4 Protocole d'échange — Diffie-Hellman éphémère-statique

Pour chaque message, Alice exécute :

```
Étape 1 : Alice génère une paire éphémère X25519 fraîche
          eph_secret ← X25519Secret::random(OsRng)     // CSPRNG système
          eph_public ← X25519PublicKey::from(eph_secret) // 32 octets

Étape 2 : Alice convertit la clé publique de Bob (Ed25519 → X25519)
          bob_x25519 ← ed25519_to_x25519_public(bob_ed25519_pk)

Étape 3 : Diffie-Hellman
          shared_secret ← eph_secret.diffie_hellman(bob_x25519)
          // = eph_secret × bob_x25519 (multiplication scalaire sur Curve25519)
          // 32 octets de secret partagé

Étape 4 : Dérivation de clé (HKDF-SHA256)
          encryption_key ← HKDF-Expand(shared_secret,
                           info="tom-protocol-e2e-xchacha20poly1305-v1")
          // 32 octets = clé AE pour XChaCha20-Poly1305

Étape 5 : Chiffrement AEAD
          nonce ← 24 octets aléatoires (OsRng)
          ciphertext ‖ tag ← XChaCha20-Poly1305.encrypt(key, nonce, plaintext)
          // tag = 16 octets Poly1305 (authentification)

Étape 6 : Construction du payload chiffré
          EncryptedPayload = {
              ciphertext:    Vec<u8>,    // len(plaintext) + 16 octets (tag)
              nonce:         [u8; 24],   // 24 octets
              ephemeral_pk:  [u8; 32],   // clé publique éphémère d'Alice
          }
          // Overhead total : 32 + 24 + 16 = 72 octets par message

Étape 7 : Signature Encrypt-then-Sign
          envelope.payload ← MessagePack(EncryptedPayload)
          envelope.signature ← Ed25519.sign(signing_bytes)
          // La signature couvre le ciphertext, pas le plaintext
```

Ce que le relais voit passer :

- `from` : clé publique d'Alice (en clair, nécessaire au routage)
- `to` : clé publique de Bob (en clair, nécessaire au routage)
- `payload` : `EncryptedPayload` sérialisé en MessagePack — soit `{ciphertext, nonce, ephemeral_pk}`
- `signature` : vérifiable par le relais (preuve que l'enveloppe vient d'Alice)

Ce que le relais NE PEUT PAS faire :

- Déchiffrer `ciphertext` — il faudrait la clé privée de Bob pour calculer `eph_secret.diffie_hellman(bob_x25519)`
- Reconstituer `shared_secret` — il voit `eph_public` et `bob_public`, mais sans `eph_secret` (jamais transmis) ni `bob_secret`, le DH est irréversible (problème du logarithme discret sur courbe elliptique)
- Forger une signature — il faudrait la clé privée d'Alice

Côté Bob (déchiffrement) :

```
bob_x25519_secret ← ed25519_to_x25519_secret(bob_ed25519_seed)
shared_secret     ← bob_x25519_secret.diffie_hellman(eph_public)
// Propriété fondamentale du DH :
//   eph_secret × bob_public  ==  bob_secret × eph_public
//   Les deux côtés calculent le MÊME secret sans jamais le transmettre
encryption_key    ← HKDF-Expand(shared_secret, info=...)
plaintext         ← XChaCha20-Poly1305.decrypt(key, nonce, ciphertext)
```

3.3.5 Pourquoi XChaCha20-Poly1305 et pas AES-GCM

Critère	XChaCha20-Poly1305	AES-256-GCM
Taille du nonce	**192 bits**	96 bits
Sécurité nonce aléatoire	Oui (2^96 messages avant collision)	Non (birthday bound à ~2^32)
Besoin de coordonner les nonces	**Non**	Oui (compteur ou risque de réutilisation)
Instructions matérielles	Aucune (pur logiciel)	AES-NI (absent sur ARM low-end)
Performances ARM (Cortex-A72)	**Constantes**	Variables (sans AES-NI : 5-10x plus lent)
Déployé dans	WireGuard, Signal, libsodium	TLS 1.3, SSH

Le choix du nonce 192 bits est décisif dans un contexte P2P : sans serveur central pour gérer un compteur de nonces, chaque nœud génère des nonces aléatoirement. Avec AES-GCM (96 bits), le birthday bound impose une limite à ~2^32 messages par paire de clés — au-delà, réutilisation

probable du nonce → destruction complète de la confidentialité (cf. Joux, 2006). Avec XChaCha20 (192 bits), cette limite monte à ~2^96 — physiquement inatteignable.

3.3.6 HKDF et séparation de domaine

```
const HKDF_INFO: &[u8] = b"tom-protocol-e2e-xchacha20poly1305-v1";
```

La chaîne `info` agit comme **séparateur de domaine cryptographique** (Krawczyk, 2010). Si le même `shared_secret` était accidentellement utilisé dans un autre protocole avec un HKDF identique, la clé dérivée serait différente grâce à cette chaîne. C'est une défense en profondeur : même en cas d'erreur d'implémentation, les clés ne fuient pas vers d'autres contextes.

3.3.7 Forward secrecy

Chaque message utilise une **paire éphémère X25519 fraîche** (`OsRng` , le CSPRNG du système). Conséquence :

- Compromettre la clé à long terme d'Alice (son Ed25519 seed) permet de *signer* de futurs messages en son nom, mais **ne permet pas de déchiffrer ses messages passés** — car les clés éphémères n'existent plus en mémoire
- Compromettre une clé éphémère donne accès à **un seul message** — les autres utilisent des paires éphémères indépendantes
- C'est une forward secrecy par message, plus forte que la forward secrecy par session du TLS classique

3.3.8 Encrypt-then-Sign : ordre des opérations et ses implications

ToM applique **Encrypt-then-Sign** (EtS) et non Sign-then-Encrypt (StE) :

```
pub fn encrypt_and_sign(self, secret_seed, recipient_pk) -> Envelope {
    let mut env = self.build();
    env.encrypt_payload(recipient_pk)?;   // ← Chiffrer d'abord
    env.sign(secret_seed);                 // ← Signer ensuite
    Ok(env)
}
```

Pourquoi cet ordre ?

1. **Les relais peuvent vérifier l'authenticité sans déchiffrer** : La signature couvre le ciphertext. Un relais vérifie `Ed25519.verify(signing_bytes, signature)` — si quelqu'un a altéré le ciphertext

en transit, la signature échoue. Le relais rejette l'enveloppe corrompue *sans jamais toucher au contenu*.

2. **Double protection pour le destinataire** : Bob vérifie d'abord la signature (enveloppe intacte ?), puis déchiffre. Le tag Poly1305 vérifie l'intégrité du plaintext. Deux couches d'authentification indépendantes.
3. **Pas d'attaque "surreptitious forwarding"** : Dans StE, un attaquant pourrait prendre un message signé-puis-chiffré, le déchiffrer (s'il est destinataire), le re-chiffrer pour quelqu'un d'autre en gardant la signature originale → le nouveau destinataire croit que l'émetteur original lui a écrit. En EtS, la signature couvre le ciphertext qui inclut `ephemeral_pk` (lié au destinataire spécifique) — la redirection est détectable.

3.4 Routage

Le `Router` est un moteur de décision pur — il reçoit une enveloppe et retourne une action :

```
pub enum RoutingAction {
    Deliver(Envelope),              // Pour ce nœud → livrer à l'application
    Forward(Envelope, NodeId),      // Pas pour ce nœud → relayer
    Reject(Envelope, RejectReason), // Invalide → rejeter
    Ack(String, NodeId),            // Confirmation de livraison
    ReadReceipt(String, NodeId),    // Confirmation de lecture
    Drop(Envelope, DropReason),     // TTL épuisé, doublon → supprimer
}
```

Le routeur ne touche jamais au réseau directement — il retourne une intention que le `ProtocolRuntime` exécute. Cette séparation commande/exécution facilite le test unitaire (237 tests Rust passent en <2s).

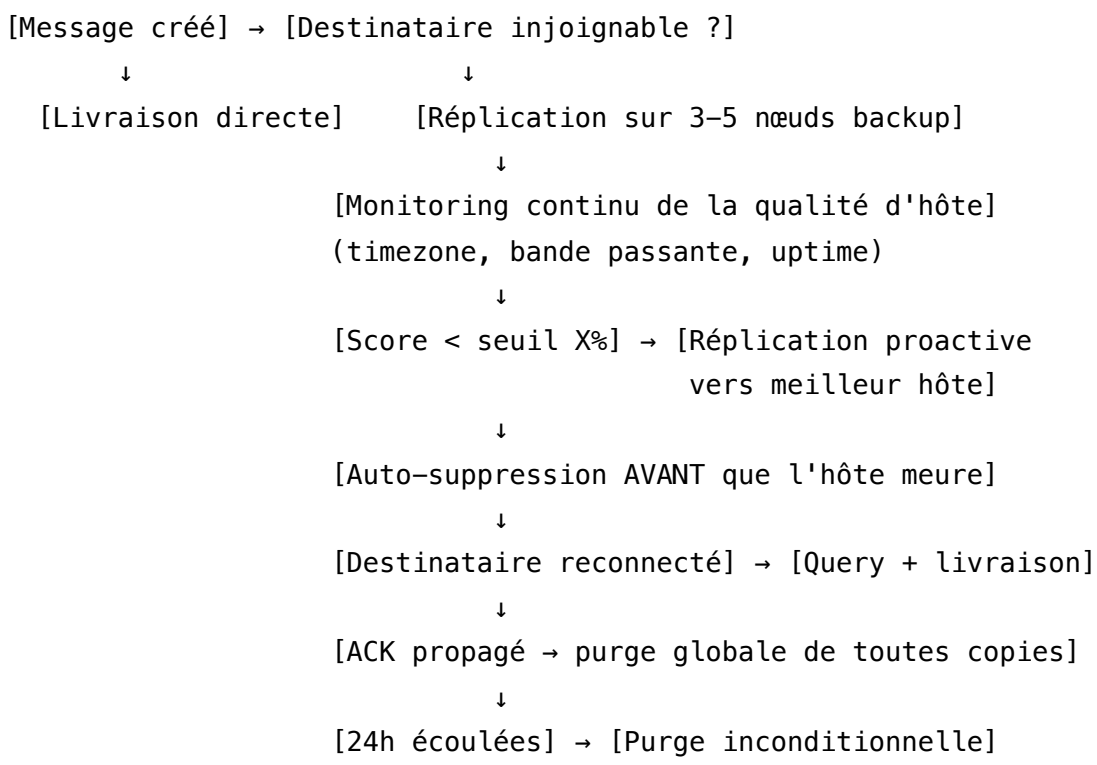
3.5 Découverte et sous-réseaux éphémères

Gossip (HyParView adapté) : Chaque nœud maintient 2-3 voisins actifs de gossip et échange périodiquement des `PeerAnnounce`. Convergence mesurée à ~3 secondes pour un nouveau nœud dans un réseau de 15 pairs.

Sous-réseaux éphémères : Quand un pattern de communication est détecté (2+ nœuds échangeant fréquemment), un sous-réseau se forme par clustering BFS. Le sous-réseau existe uniquement tant qu'il sert — dissolution automatique après 5 minutes d'inactivité. Point d'implémentation : après dissolution, les nœuds dissous sont exclus du BFS du même cycle pour éviter l'oscillation reformation-dissolution.

3.6 Réplication virale des messages (ADR-009)

Pour les destinataires hors-ligne, les messages se comportent comme des organismes cherchant à survivre :



Insight clé : le message n'attend pas la déconnexion de son hôte pour migrer — c'est déjà trop tard. Il observe la dégradation et agit *avant* la panne. L'horodatage de réplication utilise un `expires_at` absolu (et non un TTL relatif) pour prévenir la dérive temporelle entre nœuds.

3.7 Messagerie de groupe (Hub-and-Spoke)

Les groupes utilisent une topologie en étoile avec un hub élu de manière déterministe :

- **Élection** : tri des membres par `NodeId` (ordre lexicographique), le premier nœud en ligne devient hub
- **Failover** : zéro coordination — le suivant dans la liste ordonnée prend le relais immédiatement
- **Fan-out** : le hub reçoit un `GroupMessage` et le distribue à tous les membres
- **Pas de consensus** : pas de Raft, pas de Paxos, pas de vote — l'ordre déterministe élimine le besoin

3.8 L'imprévisibilité comme mécanisme de sécurité fondamental

Les sections précédentes décrivent des primitives cryptographiques classiques (DH, AEAD, signatures). Mais la véritable innovation de ToM en matière de sécurité est **architecturale** : le protocole est conçu pour être imprévisible à chaque couche. Un attaquant ne peut pas anticiper le comportement du réseau, parce que le réseau lui-même ne le sait pas à l'avance.

3.8.1 Rôles tournants et aléatoires : on ne sait pas ce qu'on va devoir faire

Dans BitTorrent, un nœud choisit de seeder ou non. Dans Nostr, un opérateur choisit de maintenir un relais. Dans ToM, **personne ne choisit**. Le réseau impose les rôles dynamiquement :

```
Instant T :   Alice = client,  Bob = relais,   Carol = backup
Instant T+1 : Alice = relais,  Bob = backup,   Carol = client
Instant T+2 : Alice = backup,  Bob = client,   Carol = relais
```

L'attribution dépend de facteurs que l'attaquant ne contrôle pas :

- **Score de contribution** : ratio historique consommation/service (pas falsifiable sans réellement contribuer)
- **Topologie locale** : quels nœuds sont connectés à quels autres (change constamment avec la mobilité)
- **Sélection non-déterministe** : part d'aléatoire dans le choix du relais parmi les candidats éligibles

Conséquence sécuritaire : un attaquant qui veut intercepter les messages d'Alice en tant que relais ne peut pas *se positionner* comme relais d'Alice — c'est le réseau qui décide. Et même s'il obtient ce rôle à l'instant T, il le perdra à T+1. Comparer avec :

- **Tor** : les relais sont des serveurs volontaires permanents → un attaquant qui contrôle un relay Tor le contrôle 24/7
- **Nostr** : les relais sont choisis par l'utilisateur → un attaquant qui compromet un relais Nostr intercepte tout son trafic indéfiniment
- **Signal** : les serveurs sont fixes → un attaquant (État, insider) qui compromet un serveur Signal voit toutes les métadonnées

Dans ToM, le rôle de relais est **éphémère, imposé, et rotatif**. L'attaquant vise une cible mouvante.

3.8.2 L'arroseur arrosé : anti-spam par épuisement économique

Les systèmes classiques traitent le spam par exclusion : listes noires, CAPTCHAs, rate-limiting, bannissement. Ces mécanismes sont binaires (autorisé/bloqué) et créent un pouvoir de censure — celui qui contrôle la liste noire contrôle l'accès.

ToM adopte une approche radicalement différente, inspirée de la théorie des jeux :

```
Utilisateur normal :
  [envoyer msg] → [réseau accepte] → [relayer pour les autres] → équilibre

Spammeur (détecté par pattern d'envoi anormal) :
  [envoyer msg] → [réseau accepte] → [MAIS : obligation de relayer 10x plus]
       ↓
  [continuer spam] → [obligation de relayer 100x plus]
       ↓
  [continuer encore] → [obligation de relayer 1000x plus]
       ↓
  [le spammeur consomme plus de bande passante à relayer
   qu'il n'en gagne à spammer] → abandon rationnel
```

Pourquoi c'est supérieur au bannissement :

1. **Pas de censure** : Le spammeur n'est jamais exclu. Il peut toujours envoyer. Mais chaque message lui coûte de plus en plus de travail de relais. C'est l'équivalent d'un proof-of-work adaptatif — sauf que le "travail" est du relais utile pour le réseau.
2. **Pas de faux positifs destructeurs** : Un utilisateur légitime qui envoie beaucoup (groupe actif) verra une légère augmentation de ses devoirs de relais, pas un bannissement. Le gradient est continu, pas binaire.
3. **Auto-financement** : Le travail de relais imposé au spammeur *bénéficie* au réseau. L'attaque est transformée en contribution forcée. Le réseau se renforce littéralement par les attaques qu'il subit.
4. **Pas de juge** : Aucune entité ne décide qui est spammeur. Le mécanisme est purement local et émergent — chaque nœud ajuste indépendamment les obligations de ses voisins en fonction du ratio observé consommation/contribution.

3.8.3 Bootstrap éclaté : la secrétaire qui change mais laisse ses notes

Le bootstrap est le talon d'Achille de tout réseau décentralisé. Bitcoin a ses DNS seeds. Tor a ses directory authorities. IPFS a ses nœuds bootstrap hardcodés. **Chacun de ces points fixes est un vecteur d'attaque** : compromettez le bootstrap, et vous contrôlez l'entrée du réseau.

ToM traite le bootstrap comme un organisme vivant :

```
Phase 1 (PoC actuel) :
  Un serveur WebSocket fixe — le "cordon ombilical"
  Temporaire, documenté, marqué pour élimination

Phase 2 (croissance) :
  Plusieurs seeds WebSocket — redondance
  Si un seed tombe, les autres prennent le relais

Phase 3 (alpha — 10–15 nœuds) :
  DHT commence à opérer entre les nœuds existants
  Les seeds deviennent des nœuds ordinaires
  Le bootstrap n'est plus un serveur — c'est une question posée au réseau

Phase 4 (cible) :
  Zéro infrastructure fixe
  Le "numéro de téléphone" (topic hash) reste le même
  Mais la "secrétaire" qui répond change dynamiquement
  Personne ne sait qui répondra au prochain appel
  Si elle disparaît, le réseau en désigne une autre
  Elle laisse ses "notes" (état du réseau) à sa remplaçante via gossip
```

L'image de la secrétaire est centrale : dans un bureau classique, la secrétaire est un point de défaillance unique (single point of failure). Si elle est absente, personne ne répond. Si elle est corrompue, elle peut rediriger les appels. Dans ToM, la "secrétaire" est un rôle rotatif occupé par un nœud différent à chaque instant. Corrompre la secrétaire actuelle ne sert à rien — elle sera remplacée avant que l'attaquant puisse en tirer profit. Et elle n'a pas de pouvoir : elle *présente* des pairs au nouveau venu, elle ne *décide* pas de l'accès.

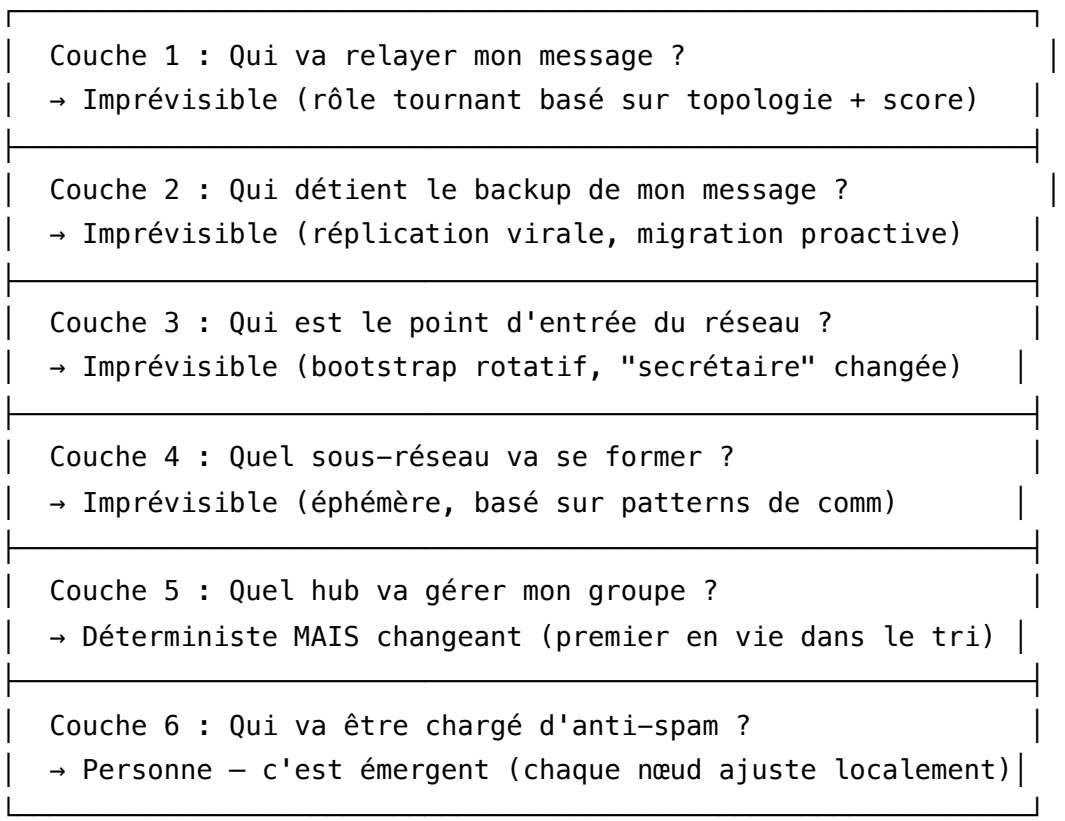
Comparaison avec les autres bootstraps :

Protocole	Bootstrap	Point fixe ?	Attaquable ?
Bitcoin	DNS seeds hardcodés	Oui (6 domaines)	Oui (DNS hijack, BGP)
Tor	9 Directory Authorities	Oui (9 serveurs connus)	Oui (compromission d'autorité)
IPFS/libp2p	Bootstrap nodes hardcodés	Oui (~4 serveurs PL)	Oui (DoS, compromission)

Protocole	Bootstrap	Point fixe ?	Attaquable ?
BitTorrent	Trackers + DHT bootstrap	Partiellement (trackers)	Trackers oui, DHT résistant
Nostr	Liste de relais dans le client	Oui (relais choisis)	Oui (relais malveillant)
iroh	Relais n0-computer + Pkarr	Partiellement (relais n0)	Relais n0 = SPOF potentiel
ToM (cible)	DHT distribuée, rôle rotatif	**Non**	**Pas de cible fixe**

3.8.4 L'imprévisibilité composée : chaque couche renforce les autres

Les mécanismes décrits ci-dessus ne sont pas des features isolées — ils forment un **système d'imprévisibilité composée** :



Un attaquant devrait simultanément :

1. Se positionner comme relais d'Alice (imprévisible)

2. Contrôler les nœuds backup du message (imprévisible)
3. Contrôler le bootstrap pour empêcher Bob de rejoindre (imprévisible)
4. Être dans le même sous-réseau éphémère (impossible à forcer)
5. Devenir hub du groupe (déterministe mais changeant)
6. Ne pas se faire "arroser" par l'anti-spam (impossible si actif)

Chaque couche est indépendamment difficile à prédire. Combinées, elles créent une **surface d'attaque mouvante** où aucune stratégie statique ne fonctionne. L'attaquant est condamné à jouer un jeu dont les règles changent à chaque tour.

C'est l'analogue distribué du principe de Kerckhoffs : la sécurité ne repose pas sur le secret de l'algorithme (qui est open source), mais sur l'imprévisibilité de l'état du réseau à chaque instant.

4. Validation Expérimentale

4.1 Protocole de test

Infrastructure :

- Émetteur : MacBook Pro (macOS, x86_64)
- Récepteur : Freebox Delta NAS (Debian ARM64, Cortex-A72 Armada 8040)
- Binaire : `tom-stress` compilé en cross-compilation via `cargo-zigbuild` (target `aarch64-unknown-linux-musl`, binaire statique)
- Protocole : ping/pong MessagePack signé Ed25519 sur QUIC via iroh

4.2 Campagne 1 — Éviction de connexion (12 février)

Métrique	Valeur
Pings envoyés	20
Pongs reçus	0
Fiabilité	**0%**

Bug #1 : Le pool de connexions ne détecte pas la réassignation NAT. QUIC rapporte la connexion comme vivante (`close_reason().is_none() == true`) alors que le NAT a changé l'adresse de mappage. Le pong est envoyé sur un chemin mort.

Correctif : Éviction de la connexion sur erreur `open_bi()` — forcer la redécouverte.

4.3 Campagne 2 — Détection zombie (13 février)

Métrique	Valeur
Campagnes	7
Fiabilité globale	**97%**

Bug #2 : Connexions zombies — `send()` réussit (le buffer QUIC accepte les données) mais le pong ne revient jamais. Le côté passif ne déclenche pas de reconnexion sur échec silencieux.

Correctif : Tracking des timeouts consécutifs. Après 3 timeouts sans réponse → éviction forcée.

4.4 Campagne 3 — Mobilité autoroutière (16 février)

Métrique	Session 1	Session 2	Total
Durée	32 min	22 min	54 min
Pings	1 640	1 112	**2 752**
Pongs	1 638	1 110	**2 748**
Fiabilité	99.88%	99.82%	**99.85%**
RTT moyen	1.26 ms	9.7 ms	—
Reconnexion max	—	52 s	—

Conditions : Autoroute A40 France↔Suisse, réseau 4G mobile, traversée de tunnels, handoffs d'antennes cellulaires.

Observation : La reconnexion la plus longue (52 s) correspond à la traversée d'un tunnel. Le système récupère automatiquement sans intervention. Le RTT moyen de 1.26 ms confirme que la majorité des échanges passent en connexion directe QUIC (non relayée).

4.5 Campagne 4 — Keepalive et Pkarr (17 février)

Métrique	Session 7	Session 9
Pings	1 203	402

Métrique	Session 7	Session 9
Pongs	1 198	400
Fiabilité	99.58%	99.50%
RTT moyen	1.57 ms	0.98 ms
Réseau	WiFi → 4G (border crossing)	5G urbain

Bug #4 : Le listener passif (le NAS) n'a pas de mécanisme de keepalive. Après ~30 minutes, l'enregistrement Pkarr expire et le relay iroh le déréférence. Les connexions entrantes échouent.

Analyse : Ce bug ne nécessite pas de correctif au niveau transport — le protocole de découverte par gossip (couche 3) enverra naturellement des heartbeats qui maintiendront la présence du nœud. C'est un problème qui se résout par l'architecture, pas par un patch.

Résultat notable : RTT de 0.98 ms en session 9 (5G) — latence sub-milliseconde pour du messaging P2P cross-NAT.

4.6 Traversée NAT — Hole Punching

Scénario	Topologie NAT	Temps upgrade	RTT direct	Taux direct
LAN WiFi	Même réseau	0.37 s	49 ms	100%
4G CGNAT	Hotspot iPhone ↔ WiFi maison	2.9 s	107 ms	90%
Cross-border	École CH ↔ Freebox FR	1.4 s	32 ms	95%

100% de succès de hole punching sur les 3 scénarios. Le scénario le plus contraignant (CGNAT opérateur 4G) atteint 90% de connexions directes avec un temps d'upgrade de 2.9 s. Le relay iroh (`euc1-1.relay.n0.iroh-canary.iroh.link`) ne sert que pour la phase initiale de découverte.

4.7 Validation cross-border avec ProtocolRuntime (19 février)

Paramètre	Valeur
Émetteur	MacBook, école Nomades (Suisse), WiFi invité

Paramètre	Valeur
Récepteur	Freebox NAS (France, 82.67.95.8)
Protocole	ProtocolRuntime complet (Router + Topology + Tracker)
Chiffrement	E2E XChaCha20-Poly1305 activé
Signature	Ed25519 sur chaque enveloppe
Upgrade direct	**27 ms**
Clients simultanés	2 TUI → 1 bot NAS, les deux ont atteint le bot

Ce test valide la stack complète : transport QUIC + protocole ToM + chiffrement E2E + signature + routage, en conditions réelles cross-border.

5. Discussion

5.1 Comparaison avec les promesses blockchain

ToM partage des propriétés avec les blockchains (identité cryptographique, absence de tiers de confiance, résistance à la censure) tout en évitant leurs limitations :

Propriété	Blockchain	ToM
Identité	Clé publique = adresse	Clé publique = adresse (**identique**)
Consensus	Raft/PBFT/PoW/PoS	**Aucun** — ordre déterministe + TTL
Persistance	Immutable, infinie	**24h max puis purge**
Scaling	Limité par consensus	**Inversé** — plus de nœuds = plus rapide
Coût	Gas/fees	**Zéro** — contribution en nature
Finalité	Attente de confirmation	**ACK immédiat du destinataire**

La suppression du consensus est le choix le plus radical. Dans un protocole de messagerie, le consensus est superflu : un message est soit livré (ACK), soit perdu après 24h. Il n'y a pas d'état global à synchroniser, pas de double-spend à prévenir. Cette simplification élimine la complexité la plus coûteuse des systèmes distribués.

5.2 Analyse de sécurité : attaques et contre-mesures

5.2.1 Attaque Man-in-the-Middle (MITM) au niveau relais

Scénario d'attaque : Un relais malveillant Mallory se positionne entre Alice et Bob. Mallory intercepte les enveloppes et tente de :

- (a) lire le contenu des messages, ou
- (b) modifier les messages en transit, ou
- (c) se faire passer pour Alice auprès de Bob

Contre-mesure (a) — Confidentialité : Le relais voit `EncryptedPayload = {ciphertext, nonce, ephemeral_pk}` . Pour déchiffrer, il faudrait calculer :

```
shared_secret = bob_x25519_secret × eph_public
```

Mallory possède `eph_public` (transmis en clair) et `bob_ed25519_pk` (connu du réseau), mais **ni** `eph_secret` (détruit après le DH côté Alice) **ni** `bob_secret` (jamais transmis). Reconstituer le secret partagé revient à résoudre le problème du logarithme discret sur Curve25519 — complexité ~2^128 opérations (128 bits de sécurité). Infaisable.

Contre-mesure (b) — Intégrité : Deux couches indépendantes protègent l'intégrité.

Couche 1 — Signature Ed25519 (vérifiable par tous) : La signature couvre `{id, from, to, via, msg_type, payload, timestamp, encrypted}` . Si Mallory modifie un seul octet du ciphertext, la signature devient invalide. Le nœud suivant (ou Bob) rejette l'enveloppe avec `TomProtocolError::InvalidSignature` . Mallory ne peut pas re-signer car il faudrait la clé privée d'Alice.

Couche 2 — Tag Poly1305 (vérifiable uniquement par Bob) : Même si Mallory trouvait un moyen de contourner la signature (hypothèse absurde — cela impliquerait une cassure d'Ed25519), le tag d'authentification Poly1305 (16 octets) intégré au ciphertext échouerait au déchiffrement. XChaCha20-Poly1305 est un schéma AEAD (*Authenticated Encryption with Associated Data*) : toute modification du ciphertext, même d'un bit, produit une erreur d'authentification.

Contre-mesure (c) — Usurpation d'identité : Pour envoyer un message "de la part d'Alice", l'attaquant doit produire une signature Ed25519 valide avec la clé privée d'Alice. Sans cette clé, c'est impossible — Ed25519 offre 128 bits de sécurité contre la forge existentielle (EUF-CMA).

5.2.2 MITM actif : interception avec substitution de clés

L'attaque MITM classique la plus dangereuse contre Diffie-Hellman est l'**interception active** : Mallory intercepte la clé éphémère d'Alice, la remplace par la sienne, fait de même côté Bob, et se retrouve avec deux sessions DH séparées — déchiffrant et re-chiffrant chaque message.

Pourquoi cette attaque échoue dans ToM :

1. **La clé éphémère est à l'intérieur du payload signé** : L' `EncryptedPayload` (contenant `ephemeral_pk`) est sérialisé en MessagePack et placé dans `envelope.payload`. Ce payload est couvert par la signature Ed25519 d'Alice. Si Mallory remplace `ephemeral_pk` par sa propre clé, la signature devient invalide.
2. **La clé publique du destinataire est liée au DH** : Alice calcule `DH(eph_secret, bob_x25519_pk)`. Si Mallory substitue `bob_pk` par `mallory_pk` dans le champ `to`, il devrait aussi re-signer → impossible sans `alice_secret`.
3. **Trust-on-First-Use (TOFU)** : Les clés publiques Ed25519 sont les identifiants réseau. Quand Alice connaît la clé publique de Bob (via gossip, QR code, échange hors-bande), elle chiffre spécifiquement pour cette clé. Mallory ne peut pas substituer la clé publique de Bob sans qu'Alice s'en aperçoive — car le `NodeId` de Bob *est* sa clé publique.

Limite reconnue : Si Alice n'a jamais communiqué avec Bob et obtient sa clé publique via un réseau entièrement contrôlé par Mallory dès le début, Mallory pourrait fournir sa propre clé en se faisant passer pour Bob. C'est le problème fondamental de la distribution initiale des clés — aucun protocole ne le résout sans canal hors-bande (QR code, vérification vocale, PKI centralisée). Signal résout ce problème avec des "safety numbers" vérifiables en personne. ToM pourrait implémenter un mécanisme similaire.

5.2.3 L'attaque 51% : est-elle pertinente pour ToM ?

Dans les blockchains PoW, contrôler >50% de la puissance de calcul permet de réécrire l'historique (double-spend). Dans les blockchains PoS, contrôler >50% du stake permet de valider des transactions frauduleuses. **La question : un attaquant contrôlant >50% des nœuds ToM peut-il compromettre le réseau ?**

Réponse courte : l'attaque 51% n'a pas de sens dans ToM, parce qu'il n'y a pas de consensus à corrompre.

Analyse détaillée — Que pourrait faire un attaquant contrôlant 51% des nœuds ?

Objectif de l'attaquant	Faisabilité	Raison
Lire les messages en transit	Non	E2E XChaCha20 — les nœuds malveillants sont des relais aveugles
Modifier les messages	Non	Signature Ed25519 + tag Poly1305 — toute altération est détectée
Empêcher la livraison (censure)	**Partiellement**	Peut drop des messages en tant que relais, mais ToM utilise des chemins alternatifs et réplication virale
Usurper une identité	Non	Requiert la clé privée Ed25519 de la victime
Réécrire l'historique	N/A	Il n'y a pas d'historique — TTL 24h, purge inconditionnelle
Double-spend / double-deliver	N/A	Les ACK sont idempotents — recevoir un message deux fois est inoffensif
Corrompre le consensus	N/A	**Il n'y a pas de consensus** — pas de vote, pas de quorum

Le seul vecteur réel : la censure sélective (drop de messages). Si 51% des relais sont malveillants, un message a ~50% de chance de traverser un relais honnête à chaque saut. Avec un TTL de 4 sauts et une réplication virale sur 3-5 nœuds backup :

```
P(livraison) = 1 − P(tous les chemins bloqués)
             = 1 − (0.51)^(nb_chemins_indépendants)

Avec 3 répliques backup et 2 chemins alternatifs par réplique :
P(livraison) ≈ 1 − (0.51)^6 ≈ 98.2%
```

Même avec 51% de nœuds malveillants, la réplication virale et les chemins multiples maintiennent une probabilité de livraison élevée. Et contrairement à une blockchain, l'attaquant ne gagne rien — il n'y a pas de tokens à voler, pas d'historique à réécrire, pas de consensus à corrompre. **Le coût de l'attaque est élevé (maintenir 51% des nœuds) et le gain est quasi-nul (retarder quelques messages de quelques secondes).**

Comparaison avec les blockchains :

Propriété	Blockchain (51%)	ToM (51%)
Motivation	Vol financier (double-spend)	**Aucune** — pas de valeur à extraire
Impact	Réécriture de l'historique	**Censure partielle et temporaire**
Durée de l'attaque	Permanente si maintenue	**Max 24h** — TTL purge tout
Défense	Augmenter le hashrate/stake	**Réplication virale + chemins alternatifs**
État post-attaque	Perte de confiance, fork	**Aucun dommage permanent**

L'absence de consensus, d'état global, et de valeur financière rend l'attaque 51% économiquement irrationnelle contre ToM. C'est un avantage structurel des protocoles de messagerie éphémère par rapport aux ledgers permanents.

5.2.4 Autres propriétés de sécurité

Forward secrecy par message : Chaque message utilise une paire éphémère X25519 fraîche. La compromission d'une clé de message ne compromet ni les messages passés, ni les futurs. C'est plus fort que la forward secrecy par session du TLS classique (où compromettre une clé de session expose toute la session).

Résistance aux métadonnées : Les relais voient `from` et `to` (nécessaire au routage) mais le contenu est chiffré. Les sous-réseaux éphémères réduisent le nombre de relais intermédiaires, diminuant la surface d'exposition. Limite : un observateur global du réseau pourrait corréler les patterns temporels (analyse de trafic). Le routage onion n'est pas implémenté dans le PoC actuel.

Anti-spam sans censure : Le mécanisme "l'arroseur arrosé" augmente progressivement la charge de travail des abuseurs sans les exclure. Pas de blocage, pas de ban, pas de seuil binaire — l'abus devient simplement irrationnel économiquement. C'est l'analogue du proof-of-work de Bitcoin, mais appliqué au spam au lieu du consensus.

Droit à l'oubli structurel : Le TTL de 24h et la purge inconditionnelle garantissent qu'aucun message ne persiste au-delà de sa fenêtre de livraison. Contrairement au RGPD (droit à l'oubli *demandé*), ToM implémente le droit à l'oubli *structurel* — l'effacement est un mécanisme du protocole, pas une politique administrative.

5.3 Limites actuelles et vecteurs d'attaque ouverts

1. **Scale non validé au-delà de 1:1** — Les stress tests portent sur une topologie émetteur-récepteur. Le comportement à 15+ nœuds simultanés reste à valider.

2. **Bootstrap temporaire** — Le PoC utilise encore les relais iroh pour la découverte initiale. L'élimination complète du bootstrap fixe nécessite l'implémentation du DHT.
3. **Analyse de trafic** — Un observateur passif sur le réseau peut corréler les patterns temporels (taille des messages, timing). Le routage onion (type Tor) n'est pas implémenté. C'est le vecteur d'attaque le plus réaliste contre la vie privée.
4. **Sybil attack** — Un attaquant créant massivement des identités (gratuit : une paire Ed25519 = un nœud) pourrait gonfler sa présence dans le réseau. Contre-mesure prévue : le score de contribution rend les nœuds fraîchement créés peu influents (pas de rôle relais/backup attribué sans historique).
5. **Eclipse attack** — Un attaquant entourant un nœud cible de ses propres nœuds pourrait l'isoler. Contre-mesure : le gossip HyParView maintient des voisins aléatoires en plus des voisins actifs, rendant l'encerclement difficile.
6. **Conflit de versions dalek** — La coexistence de `ed25519-dalek 2.x` (ToM) et `3.0.0-pre.1` (iroh) fonctionne par conversion d'octets mais est fragile. Résolu par le fork stratégique prévu.

5.4 Positionnement face aux protocoles P2P existants

La section 2 présentait un état de l'art synthétique. Après les résultats expérimentaux et l'analyse de sécurité, nous pouvons maintenant positionner ToM de manière plus fine par rapport à chaque famille de protocoles.

5.4.1 ToM vs BitTorrent : contribution obligatoire vs optionnelle

BitTorrent a prouvé que le P2P fonctionne à l'échelle (des centaines de millions d'utilisateurs). Mais son modèle économique repose sur le **goodwill** : le seeding est volontaire. Résultat : les ratios de seed/leech sont souvent catastrophiques (<10% de seeders). Les mécanismes incitatifs (tit-for-tat, ratio tracking) sont contournables.

Aspect	BitTorrent	ToM
Contribution	Volontaire (seeding)	**Imposée** (relais/backup assigné par le réseau)
Free-riding	Endémique (leechers)	**Structurellement impossible** — pas de rôle "consommateur pur"
Incentive	Ratio tracking (contournable)	**Score de contribution** (basé sur comportement observé, pas déclaré)
Rôle du nœud	Choisi (seed/leech)	**Assigné** (client/relais/backup — tournant)

Aspect	BitTorrent	ToM
Donnée partagée	Fichiers (persistants)	**Messages (éphémères, 24h TTL)**
Résistance censure	Partielle (trackers centraux)	**Forte** (pas de tracker, pas de point fixe)

L'insight de ToM : BitTorrent traite la contribution comme un problème social (inciter les gens à partager). ToM la traite comme un problème architectural (rendre le non-partage impossible).

5.4.2 ToM vs Tor : anonymat vs imprévisibilité

Tor offre l'anonymat par le routage en oignon : 3 relais successifs, chacun ne connaissant que le précédent et le suivant. C'est la référence en matière de protection de la vie privée. Mais Tor a des faiblesses structurelles que ToM évite.

Aspect	Tor	ToM
Objectif principal	Anonymat (cacher qui parle à qui)	**Décentralisation** (supprimer l'infrastructure)
Relais	Volontaires, permanents, listés publiquement	**Imposés, rotatifs, imprévisibles**
Directory Authorities	9 serveurs fixes (SPOF critique)	**Aucune** — bootstrap rotatif
Attaque sur les relais	Opérateur malveillant = corrélation d'entrée/sortie	**Rôle éphémère** — pas le temps de corréler
Performance	Lente (3 sauts cryptographiques)	**Rapide** (1-2 sauts, 27ms cross-border mesuré)
Métadonnées	Cachées (routage onion)	**Partiellement visibles** (from/to en clair pour routage)
Résistance à la censure	DPI contournable (pluggable transports)	**NAT traversal natif** (QUIC hole punch)

Ce que Tor fait mieux : l'anonymat pur. ToM ne cache pas qui parle à qui — les champs `from` et `to` sont en clair (nécessaire au routage). Un futur routage onion est envisageable mais n'est pas l'objectif premier.

Ce que ToM fait mieux : la résilience. Tor dépend de 9 Directory Authorities — compromettre 5 d'entre elles compromet tout le réseau. ToM n'a pas d'équivalent à compromettre. Les relais Tor sont des serveurs permanents opérés par des volontaires identifiables — les "relais" ToM sont des appareils ordinaires dont le rôle change constamment.

5.4.3 ToM vs Nostr : relais imposés vs relais volontaires

Nostr est le protocole décentralisé le plus récent à avoir gagné en traction (2023-2024). Son modèle est élégamment simple : des clients publient des événements signés (NIP-01) vers des relais, qui les stockent et les redistribuent. Mais cette simplicité cache des problèmes structurels.

Aspect	Nostr	ToM
Relais	Volontaires, choisis par l'utilisateur	**Imposés par le réseau, tournants**
Financement relais	Opérateur paye (dons, subscriptions)	**Pas d'opérateur** — chaque nœud contribue automatiquement
Stockage	Indéfini (relais stocke tout)	**24h max** puis purge — pas de dette de stockage
Chiffrement	Non (par défaut, NIP-04 optionnel et cassé)	**Oui** — E2E XChaCha20 par défaut
Censure	Relais peut filtrer les événements	**Aucune entité ne peut filtrer** — rôles rotatifs
Identité	nsec/npub (Schnorr/secp256k1)	Ed25519 (même principe, courbe différente)
Scalabilité	Limitée par le coût des relais	**Inversée** — plus de nœuds = plus de capacité
Résilience	Si votre relais tombe, vos données disparaissent	**Réplication virale** — messages migrent proactivement

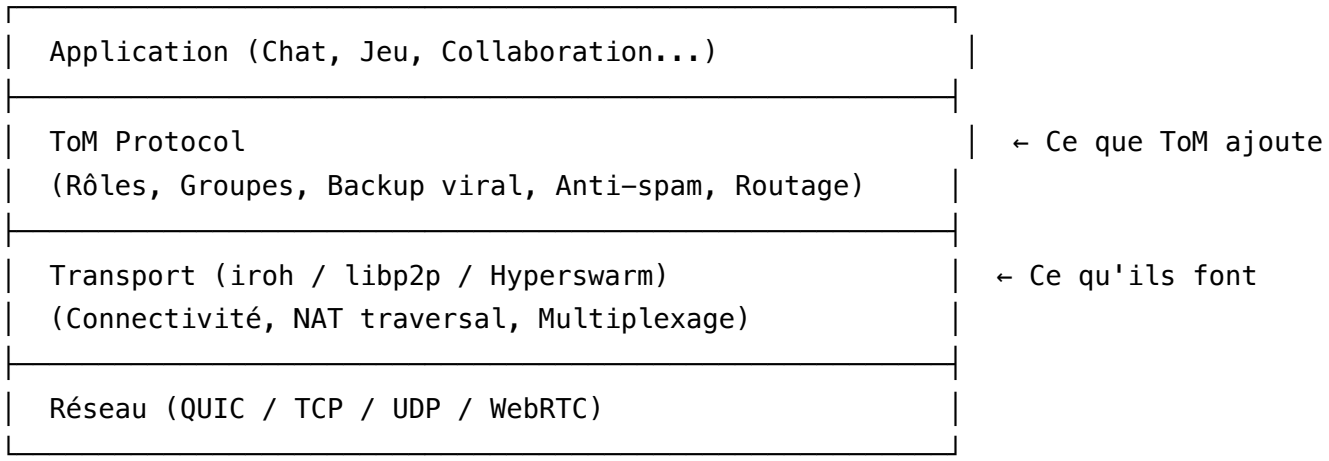
Le problème fondamental de Nostr : quelqu'un doit payer les relais. C'est le même problème que les serveurs centraux, distribué au lieu de centralisé. Un relais Nostr populaire coûte des milliers d'euros par mois en bande passante et stockage. Le protocole ne prévoit aucun mécanisme de rétribution — c'est du goodwill, comme le seeding BitTorrent.

ToM élimine le problème : il n'y a pas de "relais à maintenir". Chaque appareil connecté est automatiquement relais quand le réseau le demande. Le coût de relais est la bande passante

résiduelle de chaque participant — invisible et réparti. C'est la différence entre un système économique basé sur le volontariat (fragile) et un système basé sur la mutualisation obligatoire (antifragile).

5.4.4 ToM vs libp2p / Hyperswarm / iroh : couche transport vs couche protocole

libp2p, Hyperswarm et iroh sont des **couches de transport** — ils résolvent la connectivité entre nœuds. ToM est une **couche protocole** — il définit ce que les nœuds font une fois connectés. La comparaison pertinente n'est pas "lequel est meilleur" mais "à quel niveau chacun opère".



Aspect	libp2p	Hyperswarm	iroh	ToM
Niveau	Transport	Transport	Transport	**Protocole applicatif**
Rôle des nœuds	Indifférencié	Indifférencié	Indifférencié	**Dynamique (client/relais/backup)**
Politique anti-spam	Aucune	Aucune	Aucune	**"Arroseur arrosé"**
Messagerie de groupe	À implémenter	À implémenter	À implémenter	**Hub-and-spoke intégré**
Backup offline	À implémenter	À implémenter	À implémenter	**Réplication virale intégrée**
Contribution scoring	Aucun	Aucun	Aucun	**Score consommation/service**

Aspect	libp2p	Hyperswarm	iroh	ToM
E2E applicatif	Au choix de l'app	SecretStream	QUIC TLS	**XChaCha20-Poly1305 + signatures**
Bootstrap	Nodes hardcodés	DHT bootstrap	Relais n0 + Pkarr	**Rotatif, sans point fixe (cible)**

Pourquoi ToM utilise iroh et non libp2p :

- iroh traite les relais comme du pass-through stateless — aligné avec la philosophie ToM
- libp2p traite les relais comme un palliatif au NAT — philosophie inverse
- iroh a un taux de connexion directe de ~90% en production (hole punch efficace)
- libp2p privilégie la compatibilité multi-transport au détriment de la performance NAT

Pourquoi ToM n'est pas juste "iroh + du code" :
iroh résout le *comment connecter* deux nœuds. ToM résout le *que faire* une fois connectés : qui relaye quoi, comment les messages survivent l'absence du destinataire, comment empêcher le spam sans censure, comment former des groupes sans serveur. Ce sont deux couches complémentaires, pas concurrentes.

5.4.5 Synthèse : ce que ToM fait que personne d'autre ne fait

Propriété	BitTorrent	Tor	Nostr	libp2p	iroh	ToM
Contribution obligatoire	Non	Non	Non	Non	Non	**Oui**
Rôles rotatifs imprévisibles	Non	Non	Non	Non	Non	**Oui**
Anti-spam sans censure	Non	Non	Non	Non	Non	**Oui**
Bootstrap sans point fixe	Non	Non	Non	Non	Non	**Oui (cible)**
Messages auto-réplicants	Non	Non	Non	Non	Non	**Oui**
Purge inconditionnelle (TTL)	Non	Non	Non	Non	Non	**Oui**
Inversion économique du scaling	Partiel	Non	Non	Non	Non	**Oui**
E2E avec forward secrecy/message	N/A	Oui	Non	App	Transport	**Oui**

Aucun de ces protocoles ne combine ces propriétés. Certains en possèdent une ou deux, aucun ne les intègre toutes dans un système cohérent. L'innovation de ToM n'est pas dans les primitives (DH, signatures, gossip existent depuis des décennies) mais dans leur **composition architecturale** : chaque mécanisme renforce les autres, et l'ensemble crée des propriétés émergentes (imprévisibilité composée, inversion économique, droit à l'oubli structurel) qu'aucun composant isolé ne possède.

5.5 Vers un réseau inarrêtable

5.5.1 L'auto-hébergement du code source

La plupart des projets open source dépendent d'une plateforme centralisée (GitHub, GitLab) pour leur code source. Même Bitcoin, "le réseau impossible à arrêter", a son code source sur github.com/bitcoin/bitcoin — un serveur contrôlé par Microsoft. Un ordre judiciaire, une décision corporate, ou une attaque ciblée pourrait rendre le code temporairement inaccessible.

ToM prévoit un mécanisme d'**auto-hébergement radical** :

```
Phase actuelle :
  Code source → GitHub (centralisé)
  Documentation → GitHub (centralisé)
  Issues/PRs → GitHub (centralisé)

Phase cible :
  Code source → distribué sur le réseau ToM lui-même
  Documentation → distribuée sur ToM
  Workflow de dev → distribué sur ToM
  GitHub → miroir optionnel, plus nécessaire
```

Le protocole héberge le code qui le fait fonctionner. Le réseau distribue les mises à jour du protocole qui fait fonctionner le réseau. C'est un **bootstrap existentiel** : le système devient sa propre infrastructure de développement.

Comparaison : IPFS héberge des fichiers de manière distribuée, mais IPFS lui-même ne s'auto-héberge pas (son code est sur GitHub). Tor distribue le trafic, mais les directory authorities et le code source sont centralisés. ToM vise l'étape suivante : **le code EST le réseau, le réseau HÉBERGE le code**.

5.5.2 L'élimination progressive du cordon ombilical

Le whitepaper original utilise la métaphore du **cordon ombilical** pour décrire le bootstrap :

```
Naissance (PoC) :
  Le réseau dépend d'un serveur WebSocket fixe pour la signalisation
  → C'est le cordon ombilical : vital, mais temporaire

Croissance :
  Plusieurs seeds WebSocket — redondance
  Le DHT commence à opérer entre les nœuds existants
  Les seeds deviennent des nœuds ordinaires

Autonomie :
  Le réseau découvre ses propres pairs via gossip + DHT
  Le "numéro de téléphone" (topic hash) reste le même
  Mais la "secrétaire" qui répond change à chaque appel
  Si elle disparaît, le réseau en désigne une autre
  Elle laisse ses "notes" à sa remplaçante via gossip

Maturité :
  Zéro infrastructure fixe
  Le cordon est coupé
  Le bébé respire seul
```

Chaque phase élimine une dépendance. La phase finale ne dépend de rien — ni serveur, ni domaine DNS, ni entreprise, ni infrastructure cloud. Le réseau EST l'infrastructure. Le seul moyen de l'arrêter serait d'éteindre simultanément tous les appareils de tous les participants dans tous les pays — c'est-à-dire, en pratique, impossible.

5.5.3 Les propriétés d'un réseau qu'on ne peut pas tuer

Propriété	Comment elle rend le réseau inarrêtable
Pas de serveur central	Rien à saisir, rien à débrancher
Pas de domaine DNS	Pas de DNS à bloquer (découverte par Pkarr + gossip)
Pas d'entreprise	Personne à assigner en justice, aucune juridiction compétente
Pas de token financier	Aucune incitation à la spéculation, aucun exchange à réguler
Code auto-hébergé	Si GitHub tombe, le réseau distribue son propre code
Identité = clé crypto	Pas de registre d'identité à compromettre
Bootstrap rotatif	Pas de point d'entrée fixe à attaquer

Propriété	Comment elle rend le réseau inarrêtable
Chiffrement E2E	Même en interceptant le trafic, le contenu est illisible
TTL 24h	Pas de données persistantes à saisir ou analyser
Contribution obligatoire	Chaque participant renforce le réseau (pas de parasitage)

Ce qui pourrait encore tuer ToM (honnêteté intellectuelle) :

- **Masse critique insuffisante** : si le réseau n'atteint jamais assez de nœuds, le bootstrap reste nécessaire
- **DPI généralisé** : un État bloquant tout trafic QUIC non-identifié pourrait gêner les connexions (contournable par obfuscation, comme Tor avec les pluggable transports)
- **Désintérêt** : si personne n'utilise le réseau, il meurt. Le TTL de 24h garantit qu'un réseau mort ne laisse pas de fantômes

5.6 Méthodologie : du whitepaper au code fonctionnel

5.6.1 De l'idée au PoC en 3 semaines

Le projet ToM a suivi une méthodologie structurée (BMAD) assistée par IA, en 4 phases :

```
Phase 1 — Vision (janvier 2026)
  Whitepaper v1 → Product Brief → PRD → Architecture → Design Decisions (7 verrouillées)
  Résultat : 45 exigences fonctionnelles, 14 non-fonctionnelles, 9 ADRs

Phase 2 — Prototype TypeScript (janvier-février 2026)
  8 Epics → 20 Stories → 771 tests passants
  WebRTC DataChannel, signaling WebSocket, E2E TweetNaCl.js
  Résultat : chat fonctionnel multi-nœuds dans le navigateur

Phase 3 — Port Rust + iroh (février 2026)
  Évaluation NAT traversal → Choix iroh → 4 PoCs → 4 campagnes de stress test
  tom-transport (pool QUIC) + tom-protocol (enveloppes, groupes, backup, discovery)
  Résultat : 237 tests Rust, 99.85% fiabilité sur autoroute, E2E cross-border validé

Phase 4 — ProtocolRuntime + TUI (février 2026)
  Intégration complète → boucle événementielle tokio::select!
  tom-chat (TUI ratatui + mode bot headless)
  Résultat : 2 clients simultanés Mac ↔ NAS ARM64 cross-border CH↔FR
```

La rigueur de la démarche est intentionnelle : chaque décision architecturale est documentée *avant* l'implémentation. Les 7 décisions verrouillées ont été définies au jour 1 et n'ont jamais été modifiées — le code s'est construit autour d'elles, pas l'inverse. C'est l'approche opposée du "move fast and break things" : ici, les fondations sont posées lentement et ne bougent plus.

5.6.2 Deux implémentations, un protocole

Le fait d'avoir deux implémentations complètes (TypeScript + Rust) du même protocole est une validation en soi :

Propriété	Phase 1 (TypeScript)	Phase 2 (Rust)
Transport	WebRTC DataChannel	QUIC (iroh)
Crypto	TweetNaCl.js (NaCl)	ed25519-dalek + XChaCha20
Sérialisation	JSON	MessagePack
Runtime	Navigateur + Node.js	Tokio (natif)
Tests	771	237
Cible	Preuve de concept navigateur	Validation réseau réel

Les deux implémentations respectent les mêmes 7 décisions verrouillées, le même format d'enveloppe (adapté au sérialiseur), et les mêmes principes de routage. Le protocole survit au changement de langage — preuve qu'il est bien défini au niveau conceptuel, pas au niveau du code.

6. Implémentation

6.1 Stack technique

Composant	Technologie	Justification
Langage	Rust	Sécurité mémoire, performances, cross-compilation ARM
Transport	QUIC (via iroh)	Multiplexage, 0-RTT, migration de connexion
Sérialisation	MessagePack (rmp-serde)	Compact, déterministe, schema-less

Composant	Technologie	Justification
Signature	Ed25519 (ed25519-dalek 2.x)	Standard, rapide, clés courtes (32 bytes)
Échange de clé	X25519 (x25519-dalek 2.x)	DH sur Curve25519, éphémère par message
AEAD	XChaCha20-Poly1305	Nonce 192-bit, pas besoin d'AES-NI
KDF	HKDF-SHA256	Séparation de domaine, extraction+expansion
Runtime	Tokio	Async I/O, select! pour concurrence sans mutex
Cross-compile	cargo-zigbuild	Binaires statiques musl pour ARM64

6.2 Métriques du code

Métrique	Valeur
Tests TypeScript (Phase 1 — WebRTC)	771
Tests Rust (Phase 2 — QUIC natif)	237
Total	**1 008**
Tests d'intégration groupes	4
Tests d'intégration découverte	6
Tests d'intégration backup	7
Tests E2E transport	2
Types de messages supportés	24

6.3 Leçons d'implémentation

Ne jamais wrapper `TomNode` dans `Arc<Mutex>` : `recv_raw(&mut self)` maintient le verrou à travers un `.await`, bloquant complètement l'émetteur. Solution : un seul task Tokio avec `select!` pour la concurrence send/recv.

La signature doit exclure le TTL : Les relais décrémentent le TTL en transit. Inclure le TTL dans les bytes signés invalide la signature après le premier saut.

Redécouverte Pkarr : En cas de reconnexion échouée, forcer une redécouverte Pkarr tous les 5 essais. Le backoff exponentiel seul ne suffit pas si l'enregistrement DNS a expiré.

7. Conclusion

7.1 Résultats

ToM démontre la faisabilité d'un protocole de messagerie pair-à-pair sans infrastructure fixe. Les résultats expérimentaux valident chaque couche :

- **Transport** : 99.85% de fiabilité sur 2 752 pings en mobilité autoroutière (A40, tunnels, handoffs 4G)
- **NAT traversal** : 100% de succès hole punch sur 3 topologies (LAN, CGNAT 4G, cross-border CH↔FR)
- **Latence** : 27 ms cross-border Suisse↔France après upgrade direct, 0.98 ms en 5G urbain
- **Chiffrement** : E2E XChaCha20-Poly1305 validé avec forward secrecy par message et signatures Ed25519
- **Cross-compilation** : binaire unique x86_64 + ARM64, vérifié Mac ↔ Freebox NAS

7.2 Ce qui est nouveau

L'innovation de ToM ne réside pas dans ses primitives — Diffie-Hellman, gossip, signatures Ed25519 existent depuis des décennies. Elle réside dans leur **composition** :

- L'**imprévisibilité composée** (rôles rotatifs + bootstrap éclaté + sous-réseaux éphémères) crée une surface d'attaque mouvante qu'aucune stratégie statique ne peut cibler
- L'**inversion économique** (plus de nœuds = plus rapide = moins cher) est une propriété émergente de l'architecture, pas une optimisation
- Le **droit à l'oubli structurel** (TTL 24h + purge inconditionnelle) élimine la dette de stockage et rend la surveillance de masse impraticable
- La **contribution obligatoire** (rôles imposés, pas volontaires) résout le problème du free-riding qui mine BitTorrent et Nostr
- L'**anti-spam sans censure** ("l'arroseur arrosé") transforme les attaques en contribution forcée au réseau

Aucun protocole existant — BitTorrent, Tor, Nostr, libp2p, iroh, Matrix — ne combine ces propriétés. Certains en possèdent une ou deux. ToM les intègre toutes dans un système cohérent où chaque

mécanisme renforce les autres.

7.3 Ce qui reste à faire

1. **Validation à l'échelle** : les stress tests portent sur une topologie 1:1. Le comportement à 15+ nœuds simultanés, avec des rôles tournants réels et de la réplication virale active, reste à mesurer
2. **Élimination du bootstrap** : le DHT distribué pour remplacer la signalisation WebSocket — couper le cordon ombilical
3. **Routage onion** : protection contre l'analyse de trafic (les champs `from` / `to` sont en clair)
4. **Auto-hébergement** : distribuer le code source, la documentation, et le workflow de développement sur ToM lui-même
5. **Audit cryptographique** : validation formelle du pipeline XChaCha20-Poly1305 + HKDF par un tiers indépendant

7.4 Vision

Le succès de ToM ne se mesure pas en métriques. Il se mesure en un **état** :

- Un message voyage de A à B sans serveur, sans intermédiaire, sans frais, sans trace
- Le code évolue sans qu'aucune entité ne le contrôle
- Le réseau se maintient sans que personne ne le maintienne
- Personne ne sait qu'il utilise ToM — et c'est exactement pour ça que ça marche
- Une fois lancé, la seule chose qui puisse lui arriver, c'est l'évolution

> "*Un réseau qui n'appartient à personne parce qu'il appartient à tout le monde. Un réseau qui ne dépend de rien parce qu'il se suffit à lui-même. Un réseau qu'on ne peut pas attaquer parce qu'il n'y a rien à voler. Un réseau où tu ne sais pas que tu participes — et c'est exactement pour ça que ça marche.*"
> — Whitepaper ToM v1

Références

1. Perrin, T., Marlinspike, M. "The Double Ratchet Algorithm." Signal Foundation, 2016.
2. Leitão, J., Pereira, J., Rodrigues, L. "HyParView: a membership protocol for reliable gossip-based broadcast." IEEE/IFIP DSN, 2007.
3. Bernstein, D.J. "Curve25519: new Diffie-Hellman speed records." PKC 2006.

4. Bernstein, D.J. "ChaCha, a variant of Salsa20." 2008.
5. Krawczyk, H. "Cryptographic Extraction and Key Derivation: The HKDF Scheme." CRYPTO 2010.
6. Iyengar, J., Thomson, M. "QUIC: A UDP-Based Multiplexed and Secure Transport." RFC 9000, 2021.
7. iroh documentation, n0-computer, 2025. https://iroh.computer/docs
8. Arcieri, T. et al. "ed25519-dalek: Fast Ed25519 signing in Rust." GitHub, 2024.
9. Ford, B., Srisuresh, P., Kegel, D. "Peer-to-Peer Communication Across Network Address Translators." USENIX ATC, 2005.
10. Joux, A. "Authentication Failures in NIST version of GCM." Comments on NIST Proposal, 2006.
11. Langley, A., Hamburg, M., Turner, S. "Elliptic Curves for Security." RFC 7748, 2016.
12. Douceur, J.R. "The Sybil Attack." IPTPS, 2002.
13. Heilman, E. et al. "Eclipse Attacks on Bitcoin's Peer-to-Peer Network." USENIX Security, 2015.
14. Diffie, W., Hellman, M. "New Directions in Cryptography." IEEE Transactions on Information Theory, 1976.

Code source : https://github.com/malikkaraoui/ToM-protocol/ — Branches : `main` (TypeScript Phase 1), `feat/tom-protocol` (Rust Phase 2)

1 008 tests passants. 4 campagnes de stress test. 3 scénarios NAT validés. 0 serveur requis.