WHITEPAPER · REVIEW DRAFT V0.30

PayTP: An Open HTTP Payment Protocol with Channels and a Capped Software Incentive

The full specification, rendered from the review-draft source. Read online, or download the PDF.

PayTP: An Open HTTP Payment Protocol with Channels and a Capped Software Incentive

Martin Walfisz

martin@paytp.org

paytp.org

VERSION 0.30 (REVIEW DRAFT)

JULY 2026

DRAFT SPECIFICATION FOR PUBLIC COMMENT

Abstract

Online payments today primarily run through gateways above the web — card networks, payment processors, platform checkouts — that add fees and delay to every transaction. That architecture is a historical default, not a necessity. We propose a different approach, running inside ordinary HTTPS, x402-compatible by design, and requiring no new ports or changes to routers or middleboxes. It rides existing payment rails and introduces no currency or token of its own. A payment settles as a single transfer or as a metered channel between two endpoints, and the software that carries it earns a capped, protocol-defined share. PayTP rests on three commitments:

(1) *Masterless, end-to-end*: the protocol's terms are set by an open, capped governance charter rather than an owner — no company can price, reprice, or revoke access. No

mandatory intermediary sits in the payment's path — only the payer, the merchant, and the software they chose.

(2) *Enabler incentives*: with no owner promoting the protocol for gain, someone still has to carry it. A governed, capped distribution incentive, carved from the merchant's side, remunerates the enabling software that makes payment possible — browsers, agent frameworks, wallets, operating systems — so the motivation to carry it is built into the protocol itself, not a subsidy that can be taken away.

(3) *Bounded, negotiated trust*: with no intermediary vouching between the parties, they still have to trust each other enough to transact, so trust is made explicit instead. A one-off purchase settles before delivery; an ongoing relationship runs as a tab with a spending limit both sides set, so neither is ever exposed beyond what it accepted.

> **What this paper claims — and what it does not.** PayTP is a review-draft protocol design, published for public comment and to establish prior art. It claims a novel *combination* — direct end-to-end settlement, bounded and evidenced trust, and a capped on-wire software incentive — specified in full, with the case for and against (Appendix D). It does **not** claim: a formal cryptographic proof (Chapter 7 gives a threat model and bounded-exposure analysis, not a security reduction); measured performance (Chapter 9's claims are structural, not benchmarked); production deployment or independent validation (a first reference implementation is under active development); a settled baseline settlement rail (the choice is open, Chapter 11); or a proven economic equilibrium (the incentive's economics are reasoned, not yet tested by deployment).

This document is published to establish prior art and is freely available for implementation. Comments and feedback are welcome at martin@paytp.org.

---

1 Introduction

The Internet has a native primitive for moving data and none for moving value. Commerce on the web runs through application-layer gateways — card networks, payment processors, proprietary wallet APIs — that sit above the connection. Most checkouts detour through them, and the detour adds costs that have nothing to do

with the payment itself: fees that make small payments uneconomic, settlement that can take days, and a handful of organizations with discretionary say over what may be sold to whom.

Three families of remedies have been tried:

- **Decentralized currencies.** Blockchains removed the gatekeepers and made cheap peer-to-peer transfer feasible. But they never became a payment layer *of the web*: each is a per-chain island with its own wallets, no way to negotiate payment inside a web connection, and no reason for the web's software to carry it.
- **Closed-platform wallets.** A single platform's balances and flows — an app store, a super-app, an in-app currency — are smooth inside the silo but fragment the web, demand per-platform integration, and reproduce the gatekeeping we want to escape.
- **Open payment standards for the web itself.** A lineage from Interledger[1] to Web Monetization[2] to x402[3] has tried to fill HTTP's reserved 402 status code. Right layer, right instinct — but each lacked at least one leg: relationship semantics beyond the single payment, settlement free of any one rail, or a durable reason for the software that ships it to carry it.

What value transfer has never had is what the web already gives data: universality, simplicity, and endpoint autonomy, all at once. Each attempt above supplied some and missed others — or supplied them only as a revocable subsidy — leaving a familiar set of failures: new gatekeepers, trust nobody extends, or a design carried only as long as a sponsor pays.

PayTP combines all three. It embeds payment in the ordinary HTTPS connection: a merchant advertises support in standard headers, the payer's wallet answers, and payment travels inside the same TLS-protected exchange as the interaction it pays for, then settles on the rail beneath. It is x402-compatible and introduces no currency or token of its own. It rests on three commitments:

1. **Masterless, end-to-end.** No company owns the protocol or can price, reprice, or revoke access to it. Its terms live in an open, capped charter that no single participant controls. No mandatory gateway sits in the payment's path: the decisions belong to the payer, the merchant, and the software each of them chose. The rail beneath — along with liquidity and compliance — is a service the two sides pick, not a gatekeeper they are forced through.

1. **Enabler incentives.** The enabling software that makes a payment possible — the browser or agent framework, the wallet, the operating system — together earn a fixed one percent of it, carved from the merchant's side so the payer still pays the listed price. The share moves as part of the payment itself, paid to the software that carried the flow, so no separate collector sits in the middle. It is earned only on the payments that software actually carried, never on traffic it did not bring. Because that software decides whether an open payment standard is carried at all, the reason to carry it lives in the protocol, not in a platform's goodwill. This governed share — a base of 100 basis points — is the *meed*.

1. **Bounded, negotiated trust.** With no network vouching between the parties, exposure is made explicit instead of assumed. A one-off purchase settles before delivery. An ongoing relationship runs as a tab — a running total with a spending limit both sides set and periodically co-sign — so the rail is touched only at thresholds, and a hundred thousand tiny payments cost a handful of transfers. Trust here is bounded, not abolished: each side's worst case is a figure it accepted in advance, and settling before delivery bounds the residual risk rather than making it vanish. Where a transaction needs more — dispute handling, a refund path, a guarantor — those are optional, paid roles a flow can add, not a premium every payment carries.

Distribution is not free the way a data format is. A browser ships HTML for nothing because it improves its own product. A payment standard does not: it carries risk and upkeep, and has never had a built-in reason to be carried. Putting that reason on the wire — rather than leaving it to a sponsor who can reprice or withdraw it — is the case for the meed, argued in full in Appendix D.

A cash payment happens directly between the two parties, with no third party deciding whether it may proceed. PayTP brings that directness online: authorization, evidence, and spending policy live in the payer's wallet and at the merchant, while rails and liquidity remain chosen, replaceable services beneath them. It is a glue layer that lets any rail flow through the connections that already carry the world's data. *The ambition is a web where paying is native to the connection itself: value moving directly between two endpoints, at any size, as secure and unremarkable as the data alongside it.*

1.1 PayTP in Plain Language

Payments built into the connection. When you pay online today, the checkout hands you off from the page to separate services — card networks, payment providers — and back before the purchase completes. PayTP moves the payment into the same encrypted connection that already carries the page or the API response, so it happens directly between you and the site — one tap, or none for things you pre-approved. If a site doesn't speak PayTP, nothing breaks: you get the normal checkout.

Running a tab where you pay often. For places you trust and pay all the time — a daily news site, or an agent calling a paid API thousands of times an hour — you approve a tab: a running total with a spending limit you set. Each payment is a tiny entry that adds to the tab. Now and then both sides sign the total together, so each holds the same provable record. Real money moves only when the tab settles, over whatever rails both sides accept — so a hundred thousand tiny payments cost a handful of transfers.

Paying the software that carries it. Earlier standards had no money in it for the software that would have to carry them. PayTP reserves a share of each payment — one percent, from the merchant's side — for the software that made it possible: the browser or agent framework, the wallet, the operating system. That share is fixed by the protocol and visible in the terms both sides accept: nobody can raise it alone, and no sponsor can quietly take it away.

1.2 How to Read This Document

Chapter 2 states the motivation and design goals, and **Chapter 3** surveys the payment landscape and what remains unsolved. **Chapter 4** is the map and the right starting point for most readers. **Chapters 5–7** are the technical core: the HTTP wire profile, the channel and settlement model, and the threat model and security properties, with explicit trust bounds. **Chapters 8–9** treat privacy and scalability, **Chapter 10** defines the meed schema and why collection holds, and **Chapters 11–14** cover implementation, compatibility, future work, and conclusions.

Four appendices support the main text: **Appendix A** walks a two-leg per-payment payment through in full. **Appendix B — Design Rationale** sets out which parts of the architecture the three commitments *force*, which are free calibration, and why the machinery resists simplification. **Appendix C** develops the reserved dispute-resolution role. **Appendix D — The Ecosystem Case** argues, outside the technical

specification, the case for and against the incentive and the principle beneath it. A glossary closes the document.

This whitepaper is self-contained at the level of what each mechanism does and why. The byte-exact wire formats, field layouts, state machines, and fee arithmetic it defers to live in a companion formal interoperability specification, published for reference at paytp.org, alongside a reference implementation under active development.

---

2 Motivation and Design Goals

2.1 Motivation

PayTP exists to make paying on the web *direct*: value moving between the two endpoints, over a rail they choose, carried by software that finally has a reason to ship it. That motivation serves three things at once:

Commerce that has no home today. Machine commerce — software that pays as it goes, an AI agent buying data one query at a time, or one service paying another by the request — has no good path. Card rails are built for people at a checkout, and their fixed per-charge fee makes anything under about fifty cents cost more to collect than it earns.[4] PayTP is built for payments across a wide range of sizes, from a fraction of a cent to thousands of dollars.

Direct end-to-end settlement for commerce that already exists. For payments that already work, PayTP changes who stands in the path. The payer and the merchant settle directly: they decide whether to proceed, how far to trust each other, and which rail carries the money — with no unchosen gatekeeper setting terms in between. Governance of the protocol sits with a planned PayTP Foundation built to limit its own power (§10.5).

A gap no existing system fills. No currently deployed payment system combines all three of Chapter 1's properties. PayTP proposes a mechanism that does — offered as a contribution worth exploring, not a claim that it is better, only that the combination is novel. Whether it is worth adopting, with the case for and against, is detailed in Appendix D.

2.2 What It Looks Like When It Works

Imagine a web where subscription paywalls are not the only way to access news and interesting articles. Instead, you can pay a small amount for each article you read. Over a month you read many articles across a dozen different sites — a news report on one, a how-to on another, an explainer on a third. You would never take out a monthly subscription to all of them, so today you just hit the paywall and move on.

In that web you tap once, pay a few cents, and read. The site gets paid automatically — and so does the software that made the payment work: the browser that offered the tap, the wallet that held the money, and the phone's operating system. The payment happens right there on the page — no checkout, no redirect, no separate step. Seamless.

Metered machine payments are already real: a company's AI agents spend a thousand dollars a month on APIs — a fraction of a cent per call, thousands of calls a day. The usage is priced by the request, but the money still moves the old way — through accounts, signed contracts, and monthly invoices. In the PayTP web, each call pays as it happens, and everyone involved is paid automatically: the API providers, plus the agent framework that ran the calls, the wallet that held the funds, and the protocol's upkeep. The framework earns its share by supporting the standard, not by striking a deal.

Both cases run into the same wall: today's rails can't move small amounts straight between two parties. A card's fixed fee makes a few-cent charge cost more to collect than it earns. So reading stays locked behind subscriptions, and machine commerce stays locked behind accounts and invoices. PayTP moves the value directly between the two ends, in amounts small enough to be worth it, with everyone who helped paid automatically. Chapter 10 defines who gets paid and how much.

2.3 Design Principles

Masterless, end-to-end: decisions at the endpoints.

Goal	Rationale	Consequence for the design
End-to-end simplicity	Preserve the Internet's minimal core: smart edges, dumb middle.	Payments ride inside TLS in standard HTTP headers and bodies; no new ports; routers and middleboxes need no awareness (Ch 5).
Native protocol integration	Eliminate the out-of-band detour.	Support advertised and negotiated in HTTP headers; payment objects share the connection with the commerce they belong to (Ch 5).
Currency and rail agnosticism	Markets choose rails; a payment layer should not — but interoperability needs at least one rail in common.	Denomination is a channel property; settlement is pluggable and chosen by the parties — stablecoins, Lightning, instant bank rails, whatever they both prefer. One common baseline rail every implementation also supports gives any two a rail in common (Ch 6). Which rail that is stays deliberately open (Ch 11); the reference implementation uses Solana today — a build choice, not the protocol's.
Privacy by default	A payment layer must not become a tracking layer.	Merchant-scoped payer keys — PayTP defines no cross-site payer identifier, though rail and transport metadata still need their own privacy controls; per-payment purchases require no PayTP payer identity at all (Ch 5, 8).
Regulatory compatibility	KYC/AML obligations attach to whoever holds and moves funds.	Regulated functions concentrate in the wallet role — a chosen, replaceable service under the partner-wallet model (§10.4).

Bounded, negotiated trust: explicit, capped, evidenced.

Goal	Rationale	Consequence for the design
One-shot and metered commerce, one protocol	One-shot purchases and metered relationships have opposite cost structures.	A stateless per-payment profile — x402-compatible, settles before delivery — and payment channels: a running tab that meters usage and settles in batches (Ch 5–6).
Bounded, explicit trust	Removing trust entirely is out of scope; bounded, negotiated risk beats hidden risk.	Every exposure is capped by a signed limit — credit window, deposit, evidence bound. Strangers settle before delivery or prepay; credit extends only where the relationship or a contract supports it (Ch 6–7).
Micropayment-first scalability	The underserved segment is high-frequency small payments.	Tiny per-use messages (~35 bytes); the rail is touched only at settlement, so the overhead per payment stays negligible. Large one-shot payments use the per-payment profile (Ch 5, 9).

Enabler incentives: distribution priced into the wire.

Goal	Rationale	Consequence for the design
Transparent distribution meed	Distribution software ships what pays it — and must not be able to gouge.	The meed vector (`MEED_VECTOR`) — the on-wire split of the meed — is schema-fixed and auditable: the base roles hold exactly 1% in every schema; everything above, to the hard 1.5% cap, pays only opt-in service roles. Exposure terms are negotiated per channel; meed terms are governed, never negotiated per transaction (Ch 10).
Incremental deployment	No flag day; the web upgrades edge-first.	A merchant adds a library; a payer runs a wallet; header-stripping intermediaries cause fallback to legacy checkout rather than breakage — and the software at the edge has a paid reason to ship the upgrade (Ch 5, 12).

---

3 Background: The Payment Landscape

This chapter surveys the systems that carry today's payments and the attempts to build a web-native alternative. Each shows why the gap persists; §3.7 scores them against PayTP's three commitments.

3.1 Card Networks and Their Gateways

Visa, Mastercard, UnionPay, and their acquiring processors dominate web checkout.[5]

- **Protocol position** — all signalling happens above the encrypted connection, through REST APIs, redirects, and iFrames after TLS terminates.
- **Cost profile** — a fixed fee around $0.30 plus 2–3%[4] — a floor that makes amounts below roughly fifty cents uneconomic.
- **Latency, revocability, and float** — authorization in milliseconds, but settlement in days, a rolling reserve that can hold part of the money for months, and chargebacks possible long after delivery — with the merchant bearing the fraud risk and financing the delay.
- **Integration and gatekeeping** — card-industry security compliance, extra fraud-check steps at checkout, and network rules; schemes and acquirers can and do decline entire merchant categories and regions.

The result: small payments are uneconomic, machine-to-machine flows are impractical, and a card merchant's revenue passes through a handful of parties with discretionary power over it.

3.2 Wallet APIs and Super-Apps

PayPal, Apple Pay, Google Pay, Alipay, and their successors fix checkout friction by storing credentials server-side and approving with a click or a biometric. The UX genuinely improves; the economics do not. Fees stay card-linked, the wallet often adds its own layer, and each provider is another integration on its own terms — another silo. Wallet APIs moved the pain, not the structure.

Platform storefronts sit at the far end of the same spectrum: app stores and marketplaces take 15–30%,[6] priced by a monopoly and bundled with the store itself. They are a walled payment method, not an open rail.

3.3 Blockchain Settlement: On-Chain and Layer 2

Direct on-chain payment settles peer-to-peer without intermediaries, but at a cost: on many major chains, fee volatility, confirmation lag of minutes, and key-management burdens make it impractical for interactive commerce and uneconomic for sub-cent transfers.

Layer-2 channel networks — Lightning above all — sharply reduce latency and cost and prove that streaming micropayment settlement is technically feasible. What they lack is everything above the rail: applications manage liquidity and routes themselves, every currency needs its own network of funded channels, no broadly adopted rail-neutral way exists to use them from the web, and nothing pays a browser or framework to ship support. Commercial platforms of the Lightspark class package the liquidity problem for enterprises[7] — under PayTP, they become candidates for the settlement role rather than competitors.

3.4 Instant Account-to-Account Rails

The strongest recent evidence that cheap, instant settlement works at national scale is public infrastructure: Brazil's Pix, India's UPI, FedNow in the United States, SEPA Instant in Europe. Transfers clear in seconds at fees between zero and a few cents.[8] Their limits mirror their strengths: they are domestic and currency-bound, connect bank accounts rather than web endpoints, and define no vocabulary for the web's needs — no per-request metering, no machine-negotiable relationship, cross-border reach only through a patchwork of links, no incentive for the software layer. Under PayTP they are settlement rails like any other — arguably the best ones where they reach.

3.5 Web-Native Payment Attempts

HTTP has reserved status code 402 (Payment Required) since the 1990s; filling it in has been attempted ever since.

- **Interledger (ILP/STREAM)**[1] — packetized, asset-neutral value routing; deployment stayed confined to niche wallets and grant-funded pilots.
- **Web Monetization / Coil**[2] — the most complete experiment to date: a funded wallet and a streaming-payment API in a browser extension, with early browser-standardization work and substantial capital behind it; shut down in 2023 with

negligible publisher revenue. It funded distribution off the wire — a sponsor subsidy, not a protocol-level revenue share for the software carrying it.

- **LSAT / L402**[9] — pairs Lightning invoices with HTTP 402, and is the closest ancestor of per-payment web protocols, including PayTP's own. What it lacks is the rest: it is bound to one rail, its credentials are bearer tokens rather than signed quotes and receipts, it defines no channel, credit, or checkpoint semantics above the rail, and it pays its enabling software nothing.
- **Brave / BAT**[10] — the one attempt that paid the browser: an attention token funding a browser-native flow of ad revenue and tips to users and publishers. But a proprietary token, bound to one browser and an ad model, does not generalize into open payment infrastructure.
- **Transport-level proposals** — academic work (e.g. "QUIC Bitcoin", 2022)[11] embeds payment exchange into the transport itself; rail-specific, reliant on non-standard transport features, and silent on distribution.

Every entry either left distribution unpaid or paid it through a silo that did not generalize. Its newest member, x402, finally achieved distribution by a third method — sponsors and service providers (§3.6).

3.6 The Agentic-Commerce Stack

The most consequential recent shift is that payment infrastructure for machine payers stopped being hypothetical. As of this writing (mid-2026):

- **x402**[3] — the HTTP-payments standard originated by Coinbase and hosted by the Linux Foundation since 2026. Executes per-request payments over crypto rails, carried in HTTP headers; version 2 defines an `extensions` mechanism. Its base flow is one payment per exchange, but batch settlement is now documented too — on Ethereum-style chains, many small payments build up off-chain and settle together as a prefunded, escrow-backed scheme, with reusable wallet sessions emerging alongside.
- **MPP (Machine Payments Protocol)**[12] — from Stripe and Tempo Labs; like x402 it carries per-request payment inside the HTTP exchange, but expressed through the HTTP authentication framework — a `402` with a `WWW-Authenticate: Payment` challenge, answered by an `Authorization: Payment` credential — and submitted to the IETF as a standards-track Internet-Draft. A payment-method-

agnostic core spans stablecoins, cards, and Lightning, with extensions for method-specific flows, discovery, and identity. It is the second major in-band-402 standard, and the one whose wire model sits closest to PayTP's — settlement execution, no protocol-level meed.

- **AP2 (Agent Payments Protocol)**[13] — contributed by Google to the FIDO Alliance; defines verifiable mandates for what an agent may buy. It governs authorization, not settlement execution.
- **UCP (Universal Commerce Protocol)**[14] — Google- and Shopify-led semantics for agent discovery, cart, checkout, and post-purchase interaction.
- **ACP (Agentic Commerce Protocol)**[15] — from Stripe, OpenAI, and Meta: checkout, cart, and delegated payment that bridge agents into a merchant's existing payment stack via a Shared Payment Token, not a rail of its own.

A division of labor is emerging: UCP and ACP shape the commerce surface, AP2 supplies spending authority, and x402, cards, and wallets execute the value movement. PayTP takes a position inside that execution layer rather than beside it.

x402 and PayTP. Of everything surveyed here, x402 comes closest to PayTP's goals, and the relationship is kinship, not rivalry. PayTP's per-payment tier **is** an x402 flow carrying a `paytp` extension (§5.6): the merchant keeps its x402 stack — challenge, payment, facilitator relationships where used, and settlement — and adds signed PayTP terms for the software that wants them. Plain x402 and PayTP offers coexist, and a payment is a PayTP payment only when PayTP-aware software selects one. That awareness is what earns PayTP's value: the hardened exchange (§7) and the channels need PayTP-aware software on both sides. A plain x402 client still pays a baseline quote — and the split divides the meed by construction — but it earns no share of its own and runs none of PayTP's wallet checks.

Where x402 is ahead, it is ahead decisively: standardization, production deployments, facilitator infrastructure, and an active scheme ecosystem, including batch settlement. It is the first of the 402 lineage (§3.5) to reach broad distribution, and how it got there is instructive — sponsor funding, paid facilitators, and member backing carried a zero-fee standard.

PayTP does not try to beat x402 at that. It commits, on the wire and under governance, to three things x402's core leaves to the market around it:

- **Distribution economics.** x402 charges no protocol fee, so the software that carries payments has to be paid off the wire — by sponsors and facilitators whose support can be repriced or withdrawn as business models change. The direction is already visible: in early 2026 the leading hosted facilitator began charging per settlement for what it had previously offered for free.[16] PayTP instead carries a small, capped, visible meed inside each payment, owed to the browsers, agent runtimes, wallets, and operating systems that enable it (Chapter 10).
- **Relationship semantics.** For recurring, high-frequency commerce, PayTP defines a single kind of running tab between two parties — a capped exposure window, signed checkpoints that stand as evidence, and thresholds that trigger settlement — described the same way whatever rail settles beneath it (Chapter 6). x402's batch schemes solve part of this but leave the backing and redemption rules to each network; PayTP standardizes the relationship itself.
- **A hardened exchange.** Two properties matter here: a payment should buy the exact resource it named, and one authorization should never be redeemed for more than one delivery. x402 can provide both — through optional extensions and scheme practice — but they are not universal requirements, so a deployment that omits them can pay for a resource that never arrives, or honor one authorization twice.[17] PayTP makes them mandatory, and adds an origin requirement of its own: it binds each payment to a signed quote tied to the merchant's authenticated origin, and holds delivery until the merchant itself has confirmed settlement, with no facilitator cast as the authority (Chapters 5 and 7). The reference implementation and conformance suite are built to enforce this (Chapter 11).

None of this frames x402 as lacking: it is deployed, extensible, governed, and could add several of these itself. PayTP's narrower claim is that fixing them on the wire and under a charter is what makes them dependable for the software deciding whether to ship a payment path at all.

This compatibility is a technical choice, not a governance one: the `paytp` extension rides on x402's infrastructure, while PayTP's meed schema, role registry, and Development Fund stay governed on their own. That separation matters most for the meed — a fee that a zero-fee standard like x402 could never carry in its own core.

The same relationship holds for MPP: a parallel in-band-402 execution standard, not a rival to PayTP's contribution. Its core, like x402's, is fee-neutral by design, so nei-

ther carries the meed. MPP hardens more of the exchange in its core than x402 does — single-use, a receipt header, and on-chain challenge-binding on its recommended EVM method — yet it still leaves endpoints to decide whether a challenge came from the merchant origin they intended. PayTP's origin-authenticated quote adds that check on either substrate. Because MPP likewise defines an extension mechanism, PayTP relates to both the same way — riding the standard the parties already use and adding, on the wire, what a zero-fee standard cannot (§13.2).

3.7 The Three-Commitment Test

Scoring the systems surveyed here against the three commitments:

System	Masterless, end-to-end	Bounded, negotiated trust	Enabler incentives
Card rails	No — gateway-mediated by design	No — vouched trust, insurance bundled into every payment	No — interchange pays issuers, not the web's software
Platform wallets	No — the platform is the path	No — the platform vouches, on its terms	No — card-linked fees to issuers and wallets, nothing for the web's software
On-chain / layer 2	Yes — direct, endpoint-keyed, though siloed by chain with no web negotiation	Partial — trustless or collateralized only; no negotiated spectrum	No — validators and miners, not browsers or frameworks
Instant A2A rails	No — bank-account endpoints, no web vocabulary	No — bank-mediated	No — public-rail operators, not the web's software
Interledger	Partial — endpoint conditions, but connector chains in the value path	Partial — connector risk, no specified relationship semantics	No
Web Monetization / Coil	No — a subsidizer in the middle	No — streams and contributions without negotiated bounds	No — a subsidy is not revenue
LSAT / L402	Yes — direct over Lightning, though rail-bound	Partial — per-request only	No

System	Masterless, end-to-end	Bounded, negotiated trust	Enabler incentives
x402	Yes — end-to-end capable and Foundation-governed	Partial — settles per request; batch models exist (escrowed deposits, off-chain vouchers), but each network defines its own trust rules, not one shared model between the two endpoints	No — distribution funded off the wire and repriceable, not a protocol-level meed
MPP	Yes — end-to-end capable, IETF Internet-Draft	Partial — settles per request; extensions define method-specific flows, not one shared relationship model between the endpoints	No — fee-neutral core, no protocol-level meed

No surveyed system provides all three.[18] Several are masterless and end-to-end, and some bound trust in part — but not one defines a durable, governed, protocol-level meed for the software that carries payments.

---

4 Protocol Overview

This chapter maps PayTP's three commitments onto ordinary HTTPS. Masterless, end-to-end becomes payment objects the wallet and merchant sign directly, on terms no single participant controls. Bounded, negotiated trust becomes a choice between settling each payment on the rail and metering many payments inside a channel window. Enabler incentives become a fixed meed, carried in the signed terms of the payment.

On the wire that is three additions to an ordinary HTTPS connection: a negotiation in standard headers, a stateless flow for one-shot purchases, and a channel for high-frequency relationships — all inside the TLS-protected connection, so to anything that does not speak PayTP the exchange is ordinary HTTPS. This chapter is the map; the normative detail lives in Chapters 5–7 and 10, and the design rationale in **Appendix B**.

4.1 Actors and Keys

- The **payer** is the side money comes from — but it is two components with deliberately different powers. The **wallet** holds all keys and enforces the spending policy; the **interaction layer** — the browser, agent framework, or app actually driving the requests — initiates and mediates payments but cannot mint slices, extract keys, or spend outside wallet policy (Chapter 7).
- The **merchant** is the value-receiving endpoint: an API provider, a publisher, a creator. It is identified by a stable merchant key, bound at channel establishment to the TLS certificate the client verified (Chapter 5).
- **Meed roles** — the interaction layer, the operating system, the wallet, and the protocol's Development Fund — earn a capped, schema-fixed share of every payment, carved from the merchant's side (Chapter 10).

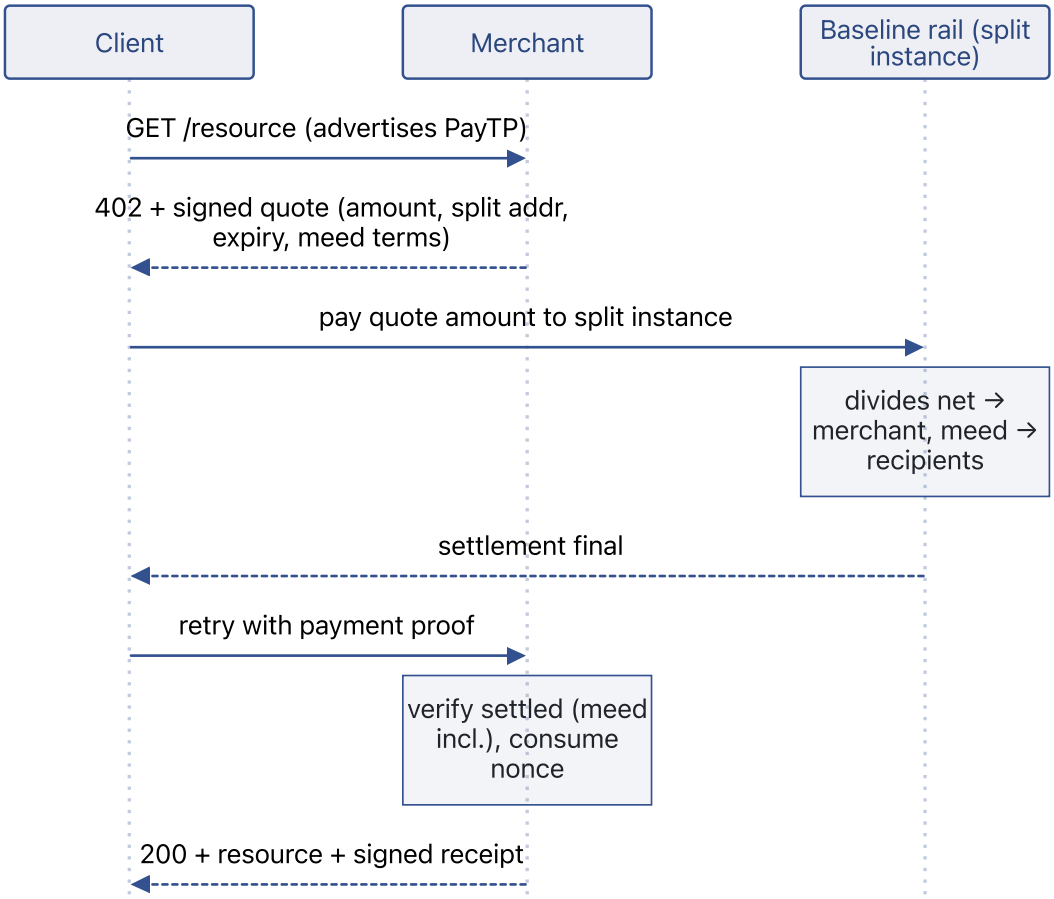
Payer identity is merchant-scoped: a distinct key per merchant, so no cross-site identifier exists to track (§5.5, §8.1). Tier 0 requires no PayTP payer identity at all.

4.2 Tier 0 — Paying Per Payment

For a first contact or a one-shot purchase, PayTP is stateless and settles on the rail before delivery:

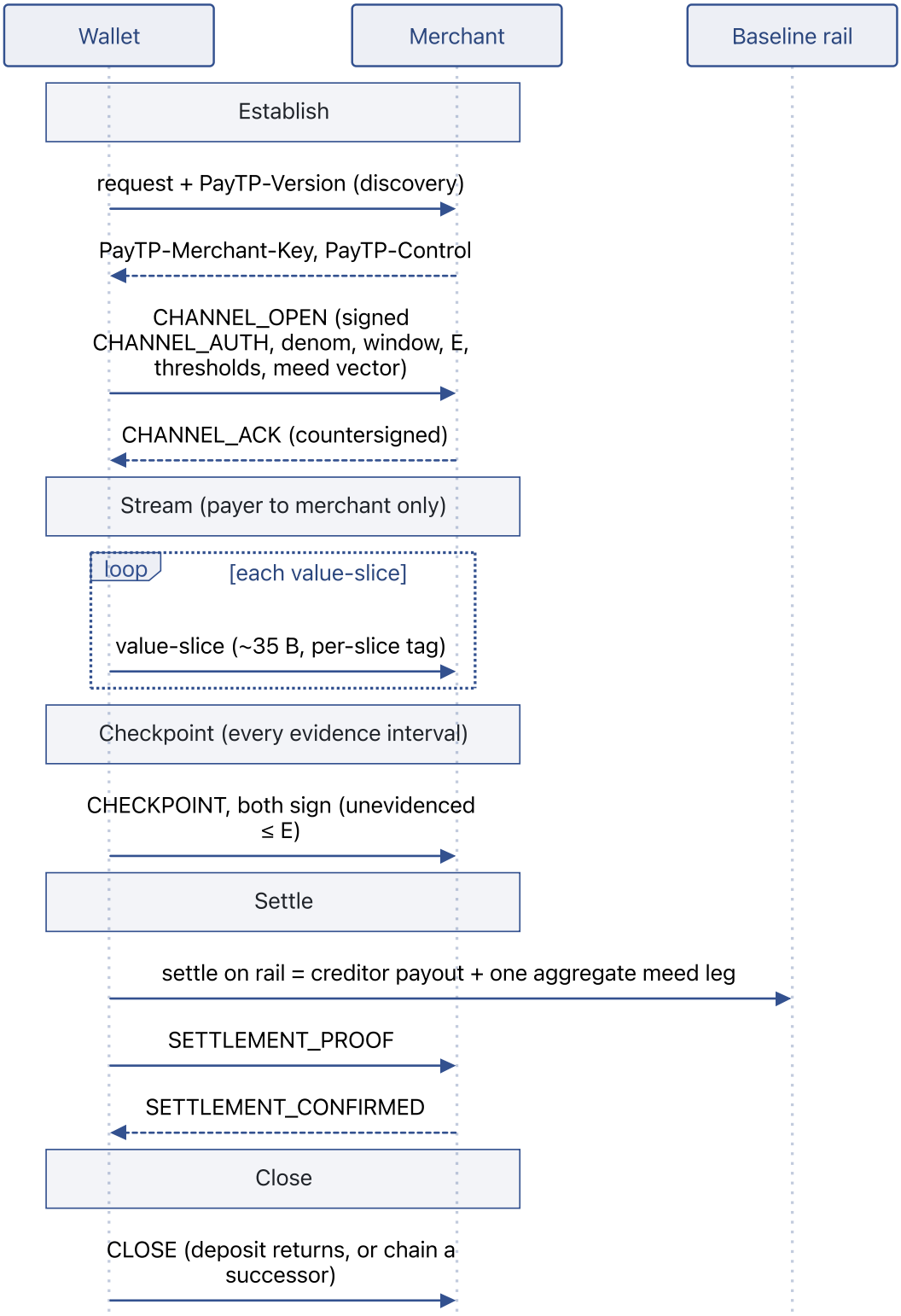
1. The client requests a paid resource, advertising PayTP support — and, if the origin has already advertised its own, the client's role information (`PayTP-Roles`; §8.4).
2. The merchant answers `402 Payment Required` with a **signed quote**: amount, asset, destination, expiry, and the meed terms — an x402-compatible challenge with PayTP's extension.
3. The client pays — either into a baseline-rail payment address that divides merchant and meed shares by construction, or, on any other rail, by paying the meed on the baseline and the merchant's net amount on the chosen rail (§5.6) — and retries with the payment proof.
4. The merchant verifies — settlement, meed included, precedes delivery — then serves the resource and returns a **signed receipt**.

Proofs are single-use, so a lost response cannot double-charge. The x402-compatibility and meed-execution details are in §5.6.



4.3 Tier 1 — Running a Channel

When a relationship has frequency — an agent metering an API, a reader's regular site — the parties open a **channel**: a metering ledger with a negotiated exposure window, in which payments become ~35-byte **value-slices** and the rail is touched only at settlement.



A channel moves through five stages:

- **Establishment.** The wallet signs the channel's complete terms — the denomination, the exposure window, the evidence bound, the settlement thresholds, and

the meed vector — and the merchant countersigns (§5.4).

- **Streaming.** Slices flow payer → merchant only, each authenticated by a per-slice tag. The negotiated window caps each side's exposure and halts payment once it is exhausted, like a full TCP receive window (§6.1).
- **Checkpoint.** At each evidence interval the two sides exchange a **bilateral checkpoint** — one statement, signed by both, that is at once the payer's receipt and the merchant's claim. The merchant never lets unevidenced value exceed the bound `E` (§6.3).
- **Settlement.** The debtor settles the accumulated balance on the rail, with proofs covering the creditor's payout and the round's meed. Every round's meed settles on the baseline rail as one aggregate leg to the channel's meed instance, whatever rail carries the rest. A conforming counterparty treats a settlement that omits the meed as incomplete: the channel closes and the shortfall stands in signed evidence (§6.4, §7.6).
- **Close.** Any unconsumed deposit returns — unless a successor channel chains from the final checkpoint and imports it, which is how a prepay float or a running tab outlives any single connection (§6.2).

4.4 What Is Guaranteed — and What Is Not

PayTP v0.1 is **bounded-trust, evidence-backed streaming settlement**. Each party's maximum loss is a number it chose — a credit limit, a deposit, or one payment in Tier 0 — and everything a dispute could turn on is signed, with unsigned value capped by the evidence bound.

It does **not** claim trustlessness: unsettled balances are real, bounded exposure, and no rail enforces PayTP receipts; settled value has exactly the finality the chosen rail provides. Chapter 7 develops the guarantees, the adversary model, and the bounds in full.

4.5 Where the Money Goes

Under the initial schema the merchant keeps 99% of gross. The remaining 1% is the meed — split across the roles that made the payment possible (interaction layer 50 bp, wallet 30, OS 10, Development Fund 10), fixed in every schema; opt-in service roles a merchant may add draw only from the band above that base. This is the pro-

tocol split, not the all-in cost — rail fees, gas, and conversion sit outside it. The full schema is §10.1.

Whether an on-wire meed is the right choice at all is argued, with its objections, outside the specification in Appendix D.

---

5 Protocol Embedding: The HTTP Profile

PayTP v0.1 rides entirely on ordinary HTTPS — no new ports, no transport changes. A merchant adds a library, a payer runs a wallet, and payments travel in standard HTTP headers and request bodies over TLS. To anything that does not speak PayTP, the exchange is ordinary HTTPS. This chapter gives the design of the two payment tiers and the properties that make them safe.

The exact wire format — encodings, field layouts, canonical forms, and state machines — lives in the companion formal specification (published at paytp.org). The body states what each mechanism does and why.

5.1 Negotiation and Discovery

A server advertises PayTP support in standard response headers — its protocol version, its stable merchant key, and the path of its control endpoint:

```
PayTP-Version:      1
PayTP-Merchant-Key: <base64url Ed25519 public key>
PayTP-Control:      /.well-known/paytp
```

A client advertises support with `PayTP-Version: 1` and sends no wallet identity at discovery time. Tier 0 purchases need no PayTP payer identity at all. PayTP requires TLS 1.3 (or 1.2) and MUST NOT run over plaintext HTTP. An intermediary that strips PayTP headers can only deny the capability — both sides fall back to legacy checkout — never forge or alter a payment, because every payment object is authenticated end-to-end at the PayTP layer.

5.2 Wire Encoding

PayTP's binary objects — slices and channel messages — are TLV sequences. Its Tier 0 objects are JSON, for x402 compatibility. Both have a canonical form over which every signature and tag is computed. Every signed object type also carries a distinct domain-separation label, so no object or signature can be replayed as a different type — this is what keeps, for instance, a redemption attestation and a cancellation (§5.6) from ever passing for one another. PayTP's header fields are sensitive by rule — never-indexed encoding where the HTTP version supports it, and `Cache-Control: no-store` on PayTP-authenticated and quote-bearing responses.

5.3 The Value-Slice

A channel payment is a **value-slice**: a minimal ~35-byte object carrying a per-channel sequence number, an amount in the channel's denomination, and a Poly1305 tag for integrity — never a signature (§6.3). There is no per-slice currency or fee field. The denomination and meed terms are fixed once in the channel's signed terms and accrue as terms × amount. Slices flow payer → merchant only, either on the request being paid for or batched to the control endpoint, and are accounted by uniqueness rather than arrival order (so reordering and retries never double-count).

5.4 Channel Messages

A channel's life is carried by a handful of signed control objects: `CHANNEL_OPEN` / `CHANNEL_ACK` establish it, `FUNDING_PROOF` funds a prepay deposit, `CHECKPOINT` records bilateral evidence (§6.3), a settlement round moves value to the rail (§6.4), and `CLOSE` ends it. Two properties of establishment are load-bearing enough to state here.

Establishment pins what settlement cannot reinterpret. `CHANNEL_OPEN` carries `CHANNEL_AUTH`, the wallet's signature over the channel's complete terms, which the merchant countersigns in `CHANNEL_ACK`. Those terms fix, for the channel's life, everything a later settlement round might otherwise redirect: the baseline rail and the inputs that determine the meed instance's address, the conversion-rate source, and the finality each rail's transfers require. So no round — original or retry — can name a different instance, choose its own exchange rate, or weaken what counts as final. The channel's confidentiality rides a fresh secret sealed to the merchant alone,

carried in `CHANNEL_OPEN` but never in the clear, so relaying software can move the message without reading it.

The meed vector is policed by the software that would be shorted. The signed terms carry the `MEED_VECTOR` — the share and destination for each enabler role (interaction layer, OS, wallet, Development Fund). Both sides validate it before committing: the wallet checks schema conformance and destinations before signing, and the interaction layer verifies its own entry before relaying — a `CHANNEL_OPEN` it refuses to relay is a channel that never exists. The software that constructs the payment is the software that would lose its share to a forged vector, so it polices its own cut.

Continuity across connections is **chaining**: when a connection ends, the wallet opens a successor channel that imports the predecessor's final checkpoint — its balance and any unsettled obligations — and carries on (§6.2). It is never session resumption. Every channel has fresh keys, so nothing replays across the boundary.

5.5 Keys and Identity

The merchant key is stable and independent of TLS certificates. A merchant proves control of it with a signed **binding artifact**, served over the very connection being bound, that ties the key to the current TLS certificate. The client accepts the merchant key only if the certificate it verified appears there. Certificate rotation republishes the artifact. The merchant key — and with it chaining, receipts, and reputation — persists. A CDN that holds the certificate is, for the channels it terminates, the merchant for PayTP purposes. Alternatively, a merchant may authorize a terminator to front an origin that meters behind it (Chapter 12).

The payer key is merchant-scoped. The wallet holds a master key that never appears on the wire and derives a distinct key per merchant — precisely, per registrable domain — stable for that relationship but unlinkable across merchants. So no cross-site identifier exists to track, and an operator fronting many domains still meets a different payer key at each. Tier 0 involves no payer key at all.

A channel binds to the payer↔merchant relationship, not to a TLS connection. Its session keys derive from the sealed secret and the two parties' public keys, not from any transport interface, so a slice or checkpoint authenticates independently of the connection that carries it.

That binding governs *carriage*, not *lifetime*. Closing the connection that carried `CHANNEL_OPEN` stops the channel from accepting new slices — a v0.1 lifecycle rule, not a limit the keys impose. After that, its control plane (checkpoints, settlement, chaining) continues over any connection, which is what connection-independent carriage is for. Carrying the relationship on past that close is a chained successor, never the same channel resumed (§5.4, §6.2). Endpoints erase the spent session secret only once the channel has closed (Chapter 7).

5.6 The Per-Payment Profile (Tier 0)

The channel-free tier for one-shot purchases — no establishment, no channel state, no PayTP payer identity, and the meed settled in the same act as the payment. Tier 0 **is an x402 flow** (x402 is the Linux Foundation HTTP-payments standard) carrying a `paytp` entry in x402's `extensions` mechanism.

Whether a meed divides is a property of the **address** a payment reaches, not of the paying software's awareness. So a PayTP payment and a plain x402 payment can sit side by side at one merchant: the merchant reuses its whole x402 stack and adds signed PayTP terms for the software that wants them, an unaware x402 client that pays a baseline PayTP quote still divides the meed by construction, and plain traffic the merchant serves at its own address carries none.

5.6.1 The Baseline Flow#

The baseline flow is four steps — an x402 exchange with PayTP terms layered on:

1. **Challenge.** The merchant answers a request for a paid resource with `402 Payment Required` and a **signed quote**: the ordinary x402 payment options plus a `paytp` extension carrying the challenge nonce, the schema, and the meed vector. The signature turns an unsigned x402 challenge into a non-repudiable quote.
2. **Payment.** The quote's `payTo` *is the split* — a small, unchangeable contract whose address is derived from the quote, the merchant's own net destination included, so the money can only ever arrive already divided among the merchant and the meed recipients. No trusted party chooses or controls the distribution. Funds leave only along the fixed split, and even an unaware x402 client that pays it completes the purchase.
3. **Redemption.** The client retries with its payment proof. The merchant verifies — **settlement precedes delivery** — that the payment reached the quoted split for

at least the quoted amount. It then credits that settlement in its durable record to the first accepted matching quote and no other, marks the nonce consumed, and delivers — so one payment redeems one purchase. It confirms rail arrival itself, so an x402 facilitator the payer's side may use holds no trust authority in the flow. A retry with the same proof returns the stored result without a second charge. The quote is self-verifying, and the merchant stores nothing for it until a proof arrives.

4. **Receipt.** The response carries a **merchant-signed receipt** over the challenge and payment references — the purchaser's durable evidence for exactly this purchase, which base x402 does not provide.

A baseline payment is an ordinary transfer to the quoted split; what ties that settlement to a specific quote is the merchant's durable record — each settlement credits a single quote — not a marker carried in the rail transfer. (A two-leg quote additionally binds both of its legs to the quote on the rail; see §5.6.2.)

5.6.2 The Two-Leg Variant#

The two-leg variant applies when the merchant settles its net separately from the meed — typically to take it on another rail, avoiding a conversion on the whole amount. The payment splits into two transfers, **meed first — and first means final**: the wallet funds the meed as a reclaimable entry at a baseline-rail meed instance, waits for the finality the quote names, then pays the merchant's net leg. Redemption (step 3) verifies that both legs reached the quoted finality before the merchant delivers.

Because the meed funds first, an interruption between the legs strands at most the meed, never the merchant's net (Chapter 7). The two forms are distinguished by the quote's terms, not by which rail is named — a two-leg net leg may itself settle on the baseline. Two-leg offers are PayTP-only, served only to clients that advertised support, so an unaware client can never pay one leg and lose the other.

5.6.3 Closing a Two-Leg Entry#

Closing the entry is the merchant's duty. On delivery the merchant MUST post a signed attestation that releases the recipients' shares and blocks any refund. If the purchase fails for good, the payer may reclaim the entry after a contest delay. The full entry lifecycle — reclaim, contest, attestation, and cancellation ordering, and the

address derivation — is a state machine walked in **Appendix A** and held in the formal spec.

Tier 0 is the on-ramp: the control endpoint advertises the upgrade to Tier 1 when a relationship recurs.

5.7 Errors and Lifecycle

A channel moves through `NEGOTIATING → OPEN ⇄ PAUSED → SETTLING → CLOSED`. Pauses are entered and released as the window or evidence bounds are hit, and `SETTLING` / `CLOSED` are one-way. Errors travel in a `PayTP-Error` header on `402` / `409` responses.

5.8 Versioning and Extension

The protocol version travels in `PayTP-Version`. Endpoints use the highest both list and MUST fall back gracefully — to a lower version, to Tier 0, or to no PayTP — without disturbing the HTTP session. Future versions can add must-understand fields deliberately — the covered-content model defined in the formal spec marks which fields a peer must understand or reject — and new signature suites (e.g. post-quantum) arrive as new signed-object types negotiated at establishment, without breaking v0.1 endpoints.

---

6 Payment Channels and Settlement

A PayTP **channel** is an accounting relationship between two endpoints — a **payer** and a **merchant** — established over a secure web connection (the HTTP profile in v0.1). It lets the payer stream thousands of authenticated value-slices to the merchant while the settlement rail is touched only a handful of times, keeping rail cost and latency out of the payment path.

PayTP v0.1 makes a deliberate trust choice: **a channel is a metering ledger with a negotiated exposure window, not a trustless escrow.** The word *metering* matters. This is not the collateralized, on-chain-enforced payment channel of a system like

Lightning, and a reader who imports those assumptions will misjudge what it guarantees.

Exposure is capped by negotiated limits, evidenced by signed checkpoints, and extinguished by periodic hard settlement on the chosen rail. Section 6.5 states precisely what is and is not guaranteed. A fully collateralized profile is out of scope for v0.1 (Chapter 13).

6.1 The Channel Model

Value-slices flow **payer → merchant only**. A relationship where both sides earn uses a pair of channels, and refunds happen at settlement, on the rail, or through a reverse channel. Every channel has a single negotiated **denomination**. Multi-currency relationships use multiple channels, and conversion is a wallet or hub concern outside the protocol.

Both parties maintain the same **balance** `B`: an accepted slice adds its amount, a confirmed funding transfer subtracts, and a settlement round decreases `B` by the gross it settles — the net plus the meed (§6.4); a prepay interim meed round is the exception, moving the meed to the instance while `B` stays unchanged. The channel invariant is

```
−L_prepay  ≤  B  ≤  +L_credit
```

and v0.1 defines two modes:

- **Prepay** `[−L_prepay, 0]` — the payer deposits first; slices consume the deposit; the merchant bears no credit risk. For *recurring* relationships where frequency justifies a float.
- **Postpay** `[0, +L_credit]` — the merchant extends bounded credit, settled in arrears. For contracted counterparties: API providers, agent operators.

Enforcement is single-ended and simple. The merchant MUST reject any slice that would push `B` above its mode's upper bound (`PAYTP_WINDOW_EXCEEDED`) — in prepay that bound is zero, so a deposit can never be overspent. The payer alone decides how much to fund, never placing more than `L_prepay` at risk. When a slice cannot be accepted, payment halts until funding or settlement re-opens the window: the window is flow control for value, the monetary analogue of a **TCP receive window**.

Exposure is explicit. The merchant's maximum loss if the payer defaults is `max(B, 0) ≤ L_credit`. The payer's maximum loss if the merchant absconds is the unconsumed deposit, `max(−B, 0) ≤ L_prepay`. Limits are unilateral risk decisions, and SHOULD scale with relationship history, as commercial credit does. Per-channel limits do not bound a counterparty that opens many channels, so creditors SHOULD track aggregate exposure per wallet key — an endpoint risk policy, not a wire rule. Postpay credit is an identity-and-relationship decision, never extended to a fresh key on trust.

Channel money is not wallet money. A user's general-purpose balance lives in their *wallet* — funded once, spendable everywhere, outside the channel model. A channel prepay deposit is a *per-counterparty float*, appropriate only where the relationship recurs. Casual one-shot purchases at unfamiliar merchants use the **per-payment profile** (channel-free, delivery gated on settlement proof, Chapter 5); they never require depositing with a stranger. Credit belongs to established relationships, not first visits.

6.2 Channel Establishment

The payer side is identified by its merchant-scoped **payer key**, held by the wallet; the merchant side by a stable **merchant key** bound to the connection's TLS certificate (Chapter 5).

A channel exists once two signed statements are exchanged. **`CHANNEL_AUTH`** is the wallet's signature over the channel's complete terms — including the mode and its limits, the meed schema and `MEED_VECTOR`, and the baseline rail and finality rules that no settlement round may later reinterpret. The merchant countersigns with **`CHANNEL_ACK`**, adding its settlement pointer. Those terms — destinations included — are immutable for the channel's life; changing one means closing and reopening, which is cheap. A channel MUST NOT carry value before both statements complete. (Full field list and wire formats: the formal spec; §5.4 gives the establishment concept.)

Channel chaining. A `CHANNEL_AUTH` MAY reference the final bilateral checkpoint of a **closed** predecessor with the same parties and denomination, importing as the opening state its balance, its unsettled obligations, and the settlement clock standing at that checkpoint. A successor chains under the predecessor's schema and `MEED_VECTOR`, with the same resolved destinations. A relationship changing any des-

tination settles the predecessor's accrued meeds first, so imported accruals are never re-divided under new destinations. Each closing checkpoint may be imported by **at most one successor**: of two racing, the first accepted wins and the other starts fresh. Chaining is how state outlives a connection — a prepay float persists across visits, a postpay tab continues across reconnects. Every channel nevertheless has a fresh identifier and session key, so nothing replays across the boundary; there is no resumption machinery to secure. (The exact chaining-validation and import rules are in the formal specification.)

Funding. On a prepay channel the payer funds by rail transfer to the merchant's settlement pointer and presents a `FUNDING_PROOF`. On confirmation the window opens. A confirmed deposit is evidenced by the rail record and the proof themselves, so it is credited even if the connection died before any checkpoint recorded it — a crash never erases rail-confirmed funding. Where both parties share a custodial ledger, the proof is the custodian's transfer reference, and no public chain is involved.

6.3 Evidence: Slices, Transcripts, and Checkpoints

PayTP separates two jobs with very different cost profiles: **live integrity**, continuous and cheap, and **durable evidence**, episodic and signed.

The slice tag. Every slice carries a Poly1305 tag under the channel's session integrity key. Because the verifier holds the same key, the tag is **not a signature and is never treated as one**: it provides integrity, continuity, and channel binding — slices cannot be altered, injected by code outside the wallet's trust boundary, or replayed across channels — but it proves nothing to a third party. The merchant's assurance that the *wallet* stands behind the stream comes from `CHANNEL_AUTH` and the checkpoints. Between checkpoints, slice accounting is bilateral bookkeeping whose dispute exposure is capped by `E`.

Checkpoints. A checkpoint is the unit of durable evidence for **metering** — what the channel has accepted. It is a canonical statement of the balance, the cumulative slice total and meed accruals, the accepted sequence ranges, a transcript head over the itemized history, and references to funding and settlement events — **signed by both parties**. One checkpoint is simultaneously the payer's receipt and the merchant's claim (a receipt for accepted payment, not for delivery — service disputes ride the application layer and the future dispute machinery, Chapter 13).

What has been *paid* — funding and settlement legs — lives on the **rail**, credited by its own on-chain proof. So the durable record is the pair **checkpoint (metered) + rail (paid)**, and what remains owed is their reconciliation: the latest checkpoint's accepted totals, less what the rail shows transferred. No bilateral counter can drift from the rail, and no lost re-anchor can strand a payment the rail already carried.

Cadence is governed by the single negotiated bound `E`: **the merchant MUST NOT let unevidenced accepted slice value since the last checkpoint exceed `E`.** Approaching `E`, it initiates a checkpoint and pauses the channel (`PAYTP_EVIDENCE_REQUIRED`). It MAY also initiate early and keep accepting within the remaining headroom, so a healthy stream checkpoints without pausing (`E` is sized above one checkpoint round-trip).

The exchange cannot deadlock: a request carries the initiator's already-signed proposed state, and the responder countersigns or returns `PAYTP_STATE_MISMATCH` within a timeout — a mismatch fails the one proposal, never the channel. Concurrent initiations are benign: checkpoints form a chain, and the one with the higher cumulative total supersedes (with a deterministic tiebreak), so the final checkpoint is always well-defined. An event present only in a superseded twin is never lost, because it is on the rail. Setting `E` to one payment yields a signed receipt per purchase — the consumer configuration. (The exact checkpoint fields, supersession, and tiebreak are in the formal specification.)

Recovery. After a crash, disconnection, or mismatch, the last bilateral checkpoint is the anchor: the parties settle from it, or chain a new channel from it (§6.2) and continue.

6.4 Settlement

Settlement extinguishes accumulated obligations on the rail and re-anchors the window. Each side tracks the accepted value not yet covered by a completed round. A round MUST begin when **settleable** unsettled value reaches `TH_value`, or on time (`TH_time`), or at window exhaustion, or at close — unless a successor chains and imports the unsettled state instead (§6.2). *Settleable* means a round can actually move it: a net payment due, or accrued meed that extinguishes at least one unit. A sub-unit remainder carries and never forces a zero-progress round (the dust rule of §10.2), which keeps a low threshold from stranding a prepay stream.

Thresholds are ceilings, not schedules — the debtor MAY settle earlier — and the round is the debtor's to start. Against a debtor that will not, the creditor's remedy is to pause at the window or evidence bound and close from the last checkpoint.

The exchange is a short signed handshake — the debtor proposes against a named checkpoint, proves the rail transfers, and the creditor confirms — with one property load-bearing enough to state here and the exact message shapes in the formal specification. **Meed first, and first means final, as one aggregate leg to the channel's instance.** Every round that settles meed pays it as a single transfer to the channel's meed instance on the baseline rail — never as loose outputs, and never handed to a channel party for onward distribution — converted (where the denomination differs from the baseline asset) at the rate and source the channel's establishment terms fixed, so no debtor picks its oracle at settlement. The instance address and the required finality were fixed at establishment, so no round can redirect the leg or weaken what counts as final; the creditor confirms only after the meed leg finalizes. Because the instance divides among the destinations fixed at establishment, no conforming counterparty acknowledges a settlement complete without that leg.

Settlement is **all-or-close**: if the proofs and confirmation do not complete within `SETTLE_TIMEOUT`, no re-anchor checkpoint is signed, the channel closes, and whatever remains unpaid stands as an obligation in the signed checkpoints — a partial round is never recorded as settlement. Any leg that did finalize is credited by its own rail proof, never paid twice, in any later round or close.

Direction follows the mode: in postpay the payer settles the net owed; in prepay, consumption was prepaid, so the merchant is the debtor and interim rounds execute only the meed leg (`B` does not move), returning any unconsumed deposit to the refund pointer at close — unless a successor chains and imports it instead. Slice amounts are **gross**: meed is carved out at settlement and the merchant receives the remainder (arithmetic in Chapter 10).

Unsettled meed is a bounded exposure. In prepay the merchant is the party that executes the meed round, so a merchant that kept accepting slices while never settling would draw down a deposit whose meed share never reached the recipients. The payer's wallet bounds this with the leverage it already holds: it alone decides how much to stream. A conformant wallet stops sending slices once a due meed round has not completed, and resumes on the merchant's signed notice of an interim

meed draw — though it confirms on the rail, not the signature, that the meed reached the recipients.

So a withholding merchant simply stops earning further consumption, and the meed it can strip is bounded to the one round already accrued — as large as the thresholds the parties signed, and no larger. The channel bounds the loss; it does not promise zero. This rests on wallet conformance and the certification regime, not cryptography (§6.5, §10.3): a wallet aligned with its user against a third-party recipient could decline to halt. The deposit itself is the separate abscond risk §6.1 already bounds to `L_prepay` and sends only to recurring relationships. In postpay the payer settles and its own wallet executes the meed leg, so the exposure does not arise.

Failure and uncooperative close. If a peer becomes unreachable or refuses to checkpoint or settle, the promises are exactly these: service already stopped at the window or evidence bound, and the survivor holds bilateral checkpoints and `CHANNEL_AUTH` — cryptographic evidence of what was authorized — as the basis for recovery through contract, guarantor, or legal process. **No settlement rail enforces PayTP receipts on-chain, and this specification claims none does.** A merchant that cannot tolerate credit runs prepay. A party that will carry no counterparty exposure at all uses per-payment settlement, the only truly settle-first mode. Either can await the collateralized profile.

6.5 Trust Model Summary

Mechanism	Guarantees	Does NOT guarantee
Secure connection (TLS)	Wire confidentiality and integrity	Anything about payment authorization
Slice tag	Stream integrity, continuity, channel binding	Wallet origin to the counterparty; third-party evidence
`CHANNEL_AUTH` / `CHANNEL_ACK`	Mutually signed terms: mode, limits, thresholds, denomination, meed vector, destinations, session-key commitment	That any particular slice was sent
Transcript + bilateral checkpoints	Non-repudiable cumulative state and exact history, signed by both parties	Payment — evidence is not money
Settlement + rail	Actual value transfer with the rail's finality	Recovery of unsettled deltas
Window + `E`	Loss bounded: `≤ L_credit` (merchant), `≤ L_prepay` (payer), unevidenced `≤ E`	Zero loss; trustlessness
Settlement cadence + wallet halt on overdue meed	Prepay meed leakage bounded to one settlement round	Zero meed leakage; trustless collection

PayTP v0.1 is **bounded-trust, evidence-backed streaming settlement**. Implementations MUST NOT describe slice tags as signatures or receipts as rail-enforceable claims.

6.6 Worked Examples

6.6.1 An agent metering a paid API (postpay)#

An agent operator's wallet and an API provider negotiate: USDC on the baseline rail, postpay window `+$25`, `E = $1`, settle at `$10` or daily. The agent calls the API at ~10 requests/second, each request carrying a $0.0001 slice — verified with a MAC check, no rail or facilitator in the path. Roughly every $1 the two sides exchange a

bilateral checkpoint, so both books agree provably as they go. At $10, a settlement round comes due and the operator's wallet (the debtor) proposes and executes it: the merchant's net in one baseline transfer, alongside one aggregate meed leg to the channel's instance — divided among the agent framework, the wallet provider, the Development Fund, and (since a headless server uses no OS secure facilities) the independent open-source fund that receives the OS share where no approved OS applies. A hundred thousand metered calls touch the rail a handful of times a day.

6.6.2 A 10-cent article (consumer)#

A reader's wallet already holds funds — topped up once, at the wallet level, spendable anywhere; *that* is the transit-card analogy, and it involves no relationship with any particular merchant. Visiting a news site for the first time, the reader taps "Unlock for €0.10": the wallet uses the **per-payment profile** — one payment, settled immediately on the baseline rail with merchant and meed shares divided at the payment address (§5.6), delivery gated on the proof, a signed receipt for exactly this purchase. No deposit and no credit; the reader's exposure is bounded to one payment and evidenced by the quote and rail record if delivery fails. If the reader becomes a regular, the wallet MAY offer to open a small prepay float (say €5, mode `[-5, 0]`) so subsequent unlocks skip the rail entirely — a float justified by frequency, never demanded by a first visit.

---

7 Threat Model and Security Properties

7.1 Objectives — and Non-Objectives

Security is where Chapter 1's commitments become mechanism: endpoint-held spending authority, bounded negotiated exposure, tamper-evident meed accounting. PayTP v0.1 provides **bounded-trust, evidence-backed payment security** — precisely, per party:

- **A merchant** never carries more exposure than the limits it set: credit at most `L_credit` per channel, and accepted-but-unevidenced slice value at most `E`, each enforced by the merchant itself in both modes.

- **A payer** never has more at risk than it chose: `≤ L_prepay` per channel, or one payment in Tier 0, and it holds a signed receipt for every delivered purchase (a paid-but-undelivered Tier 0 attempt leaves the signed quote and rail record instead).
- **Both parties** hold cryptographic evidence — mutually signed terms, bilateral checkpoints, receipts — sufficient to pursue recovery through contract, guarantor, or legal process.
- **Meed recipients** get tamper-evident accounting of everything that accrues to them. The roles present in the flow read it directly. For the remote ones it is held in endpoint evidence and discoverable under certification and audit — not automatically visible (§8.2). And no conforming counterparty acknowledges a channel settlement complete without the meed leg that pays them (§7.6). Against a *colluding* counterparty their protection is evidence and the selection argument (§10.3), not the settlement math — a gap baseline-denominated settlement narrows and a Tier 0 baseline payment closes.
- **Third parties** (network attackers, other sites, other channels) can neither forge, replay, nor read in-band payment objects. What traffic shape and rail settlement still expose is Chapter 8's subject.

The **non-objectives** matter equally. PayTP v0.1 is not trustless. Unsettled balances are genuine, bounded counterparty exposure. No settlement rail enforces PayTP receipts on-chain. Payment evidence is not payment. A party that will carry no counterparty exposure runs prepay (the payer carries the bounded float) or Tier 0's settle-before-deliver flow — or awaits the future collateralized profile (Chapter 13).

7.2 Adversary Model

1. **On-path network attacker** — observes, injects, reorders, or blocks traffic. Contained by TLS plus PayTP-layer authentication of every payment object. Stripping PayTP headers can deny payment capability but cannot forge or alter a payment.
2. **Compromised application layer** — malicious page scripts, a buggy agent framework, injected code *driving* the connection but outside the wallet. This is the boundary that matters most in an agent-driven web: the entity that speaks HTTP is routinely less trusted than the one that holds funds. All spending authority lives with the wallet, so compromised application code **cannot mint slices or extract keys** — off-wallet forgery is impossible. What it can still attempt is *misuse*

within policy: requesting unwanted payments, or steering an authorized slice onto another request of the same channel. The perimeter against that is the wallet's spending policy (budgets, thresholds, consent). That is why authorization semantics live in the wallet and never in the page or framework — and why wallet pluggability is a conformance requirement where one vendor would otherwise ship both (§10.4).

3. **Malicious counterparty** — a payer who consumes and never settles, a merchant who takes funds and does not deliver, either forging or repudiating history. Contained by the window, the evidence bound, bilateral signing, and Tier 0's settle-before-deliver flow.
4. **Cross-context replay** — reusing objects across channels, connections, or roles. Contained by binding and domain separation (§7.4).
5. **Colluding endpoints** — payer and merchant cooperating to bypass meeds. Explicitly *not* cryptographically preventable; addressed economically (§7.6, §10.3).
6. **Rail-level adversaries** — double-spends, reorgs, censorship on the settlement rail. Out of scope by construction: settled value has exactly the finality the chosen rail provides, which is why settlement completion is defined by proofs and confirmation, never by trust.

7.3 Turning Trust into Arithmetic

The mechanisms and what each guarantees are the trust-model table of §6.5. This section states the one property that is the security core: **every exposure is a number the exposed party chose.** At every moment in a channel:

```
merchant credit exposure   =  max(B, 0)    ≤  L_credit
payer deposit exposure     =  max(−B, 0)   ≤  L_prepay
unevidenced accepted value (since last checkpoint)  ≤  E
```

These bounds hold against a fully malicious counterparty because each is enforced *unilaterally*: the credit and deposit bounds by the party each protects (the merchant stops serving; the payer never over-funds), and the evidence bound by the merchant in both modes — accepted slices beyond the last checkpoint are service delivered without durable evidence (§6.3). Every attack on the accounting reduces to "the loss is capped at a number you chose."

Settlement then converts evidence into finality with destinations fixed at establishment and all-or-close semantics: a settlement that does not complete leaves the obligations standing in signed checkpoints, never half-applied.

Tier 0 has a simpler invariant: **settlement — the meed included — precedes delivery.** The merchant risks nothing. The payer risks one payment against a merchant-signed quote, with a single-use proof and idempotent redemption (a lost response cannot double-charge). A Tier 0 payment proves the merchant quoted and received funds. It does not prove delivery or create an in-protocol refund right for the purchase amount — purchase-price protection is bought explicitly (Chapter 13).

7.4 Replay and Binding

Every object is valid in exactly one channel, sequence, and role — **not** one HTTP request. A slice binds to its channel and sequence, not to the resource it rides, so steering an authorized slice onto another request of the same channel is possible — the within-policy misuse §7.2 assigns to wallet spending policy, not a forgery. Beyond that, replay is closed on every axis:

- unique, never-reused sequence numbers;
- a distinct domain-separation label per object type, so a checkpoint cannot pass as a settlement message, a quote as a receipt, or an attestation as a cancellation;
- fresh per-channel keys, so a slice verifies only in the channel it was minted for;
- no channel resumption — continuity is chaining, which consumes a closed predecessor's final checkpoint exactly once;
- a merchant identifier-retention rule that bars replaying a captured `CHANNEL_OPEN` onto a new connection;
- slices barred from TLS 0-RTT early data;
- and, in Tier 0, single-use expiring nonces with idempotent redemption against a consumed-nonce record.

The mechanisms themselves are Chapter 5; this is why they suffice.

Origin-authenticated quotes. PayTP's per-payment tier rides on x402, and — once the embedding lands — on MPP (§13.2). Those substrates bind a payment to the challenge it answers, but leave authenticating that challenge's origin to the endpoints. Left there, an agent router, an in-path proxy, or the interaction layer itself

can present its own challenge, naming its own recipient, and the payer pays the substitute.

PayTP closes this by making the quote a merchant-signed object bound to the merchant's authenticated origin, with delivery held until the merchant itself confirms settlement. How far it reaches depends on the wallet: a wallet that authenticates the intended merchant origin refuses a substituted quote before any funds move; a wallet fully delegated to an in-path party it cannot authenticate instead pays the merchant directly, or uses PayTP's own two-leg flow — which x402 and MPP do not define — whose settlement-rail binding closes the redirect. The single-leg baseline keeps a narrow residual of its own (§7.8). A party that terminates TLS with the merchant's own certificate is inside the origin, not an intermediary to it (§12.2).

7.5 Key Compromise and Blast Radius

- **Wallet master key** — never on the wire; per-merchant payer keys derive from it. One compromised *derived* key affects one relationship and enables no cross-merchant correlation or forgery. Compromise of the master key is wallet-wide. That is why it never leaves the wallet's custody core (§10.4).
- **Session secret / session key** — lets an attacker forge slices on that one channel, and only until the checkpoint machinery refuses to co-sign: bounded by the same window and evidence arithmetic as any dishonest counterparty.
- **Merchant key** — lets an attacker sign fraudulent quotes and receipts, and sign false attestations or cancellations against the open meed entries of two-leg purchases, until the key is rotated. Damage is bounded to the meed amounts in open entries. Channel funds are never exposed, because destinations were fixed at establishment under both signatures. Rotation starts a *fresh* merchant identity — new payer-key derivation, new receipts, no imported reputation or chaining — so it is a break, not a handover. Honest holders of quotes and channels under the retired key recover exactly as against any merchant that stops honoring (§7.8). The ceremony and any successor attestation are governance, not wire protocol (Chapter 11).
- **Channel-encryption key** — lets an attacker who also recorded a channel's establishment recover that channel's session secret, and with it that channel's slice keys. The forged slices stay inside the same window-and-evidence bounds. Rotation is an artifact republish. Compromised-key channels are closed and re-established.

- **Past-session protection** — compromise of long-term signing keys does not reveal past session keys (they derive from the fresh per-channel secret), the channel-encryption-key case above excepted. The guarantee assumes endpoints erase session secrets at close.
- **Algorithm agility** — signature suites are negotiated TLV types, so post-quantum suites arrive without breaking v0.1 endpoints (§5.8).

7.6 Meed Integrity

Within the protocol, meed accounting is tamper-evident end to end: the `MEED_VECTOR` is signed into `CHANNEL_AUTH`, its cumulative accruals into every bilateral checkpoint, and a settlement's meed leg pays only the channel's instance, whose code pays the destinations fixed at establishment. No single party can silently **alter or redirect** the meed.

Understatement is the weaker case. The recipients do not sign the settlement, so two colluding channel parties — or a conversion rate no third party sourced — can settle a smaller aggregate leg than accrued. It takes *both* parties: the conforming counterparty verifies the leg against the checkpoint before it confirms, so a unilateral shortfall only halts the channel. What protects the recipients there is evidence and conformance, not cryptography. The accounting is tamper-*evident*, not tamper-*proof* against a conforming-looking collusion: the signed vector and checkpoints prove what was owed, and the pinned rate source makes a shortfall provable wherever volume evidence is disclosed.

Baseline-denominated settlement narrows the gap — no rate to shade, and the leg's amount is deterministic from the signed totals. Only a Tier 0 baseline payment removes even the misstatement, because the gross lands in code that divides it. On every rail the recipients hold the same bounded-trust position as the payments themselves, and collection ultimately rests on §10.3's selection argument, not the settlement math.

What cryptography does **not** do is force endpoints to use PayTP at all: plain x402, cards, and direct transfers remain open beside it, exactly as cash avoids card interchange. Collection therefore rests on distribution power — the software that implements PayTP inserts the vector as a condition of its participation — plus auditability. Why that equilibrium holds for mainstream traffic is Chapter 10's economic argu-

ment. This chapter claims only that the *accounting* is tamper-evident wherever PayTP is actually spoken.

7.7 Denial of Service

- **Invalid traffic** — slice tags are verified in constant time before any state mutation. Invalid or malformed objects are dropped with no accounting effect.
- **Paying to occupy** — the window makes resource occupation self-limiting: an attacker holds a channel open only by actually paying or funding, and a channel at its bound pauses at zero marginal cost to the merchant.
- **Channel proliferation** — many channels multiply negotiated state, not wire-enforced exposure. Per-channel bounds are enforced on the wire, aggregate exposure is the creditor's own risk policy, and establishment is cheap to refuse.
- **Tier 0 challenge storage** — unpaid quotes are self-verifying and cost no storage. State exists only for consumed nonces, so memory is bounded by paid redemptions × retention, not request rate. Issuance costs one signature per request, bounded by ordinary rate limiting.
- **Control endpoint** — accepted control messages are authenticated by channel-party signatures. An unauthenticated flood costs cheap parsing plus at most one signature verification per message, and legitimate verification load is amortized at checkpoint cadence, not slice cadence.

7.8 Residual Risks

Each risk, its bound and mitigation:

Risk	Bound / mitigation
Payer defaults on unsettled credit	`≤ L_credit`; limits scale with relationship history; signed evidence supports recovery
Merchant absconds with a deposit	`≤ L_prepay`; floats are for recurring relationships only; strangers use Tier 0
Paid but undelivered (Tier 0)	One payment; signed quote + rail record as evidence; idempotent retry excludes double-charge
Single-leg baseline redirected to another of the merchant's quotes for ≤ the payment (any resource)	Same split, so the victim's funds still reach the merchant, not an attacker; but the victim is left paid-but-undelivered while the redirecting party takes its own — possibly cheaper, different — delivery. A visible redirect, not silent fund theft; a two-leg quote closes it
Meed entry on a genuinely failed two-leg purchase	The payer reclaims the funded baseline-asset entry after the honor and contest windows pass with no attestation (refund is the baseline-asset amount, not original-currency value); wallets SHOULD reclaim automatically. A baseline-settled purchase has no such entry
Delivered two-leg entry reclaimed, or falsely cancelled, instead of attested	The merchant MUST post the delivery attestation for every delivered two-leg purchase, which closes the entry before any reclaim executes — so an honest posting defeats an automated post-delivery reclaim. A delivered entry reclaimed or cancelled is merchant non-conformance, not a payer capability; bounded to the meed amount on delivered non-baseline volume, detected by the on-chain funded-to-reclaimed ratio plus the payer's signed receipt, answered under certification. Baseline Tier 0 has no such entry
False merchant attestation on an undelivered purchase	Steals the payer's reclaim, bounded to the meed amount; the attestation is the merchant's own signed claim — evidence for recourse and certification, not a cryptographic bar
Colluding channel parties understate the meed	Not cryptographically bounded; the signed vector and checkpoints make the shortfall provable where volume evidence is disclosed (§7.6); rests on conformance, certification, and selection economics; Tier 0 baseline payments have no such gap

Risk	Bound / mitigation
Residual meed-entry dust	Permissionless batch reclaim to fixed pointers makes recovery rational at any size; what remains is entries nothing indexes, entries below batched execution cost, stale refund pointers, and FX drift
Statutory obligations survive	PayTP removes card-network chargebacks from the protocol path; it does not remove statutory refund, warranty, consumer-protection, tax, or reporting obligations
Split-contract defect	Reference contracts are auditable, versioned, and immutable per instance; blast radius is flows quoting that version; fixes ship as new versions, never upgrades
Evidence requires off-protocol recovery	True by design — contracts, guarantors, courts, insurers. PayTP caps exposure; it does not absorb loss. Strongest in contracted B2B, weakest for anonymous cross-border micro-amounts (which should run prepay or Tier 0)
Rail finality reversal	Settled value inherits the rail's guarantees, no more; all-or-close keeps failed settlements from corrupting channel state
FX exposure across assets	Out of protocol scope: channels are single-denomination; conversion risk lives with the wallet or hub performing it
Implementation defects	The protocol surface is deliberately small; the reference implementation and conformance suite (Chapter 11) are the working mitigation

PayTP's security reduces to TLS for the wire, Ed25519 signatures over mutually held state for evidence[19], arithmetic bounds each party chooses for exposure, and the chosen rail's guarantees for settled value. No counterparty exposure is uncapped or unevidenced — and what remains assumed rather than proven, the protocol states plainly: registry and schema governance, wallet policy, off-protocol recovery.

---

8 Privacy Considerations

A payment layer must answer for its metadata, not only its message contents. PayTP's privacy properties follow from the same architecture as its security: the three commitments decide not only who executes a payment and who carries its risk, but who is in a position to observe it. This chapter states what each party learns, what reaches the settlement rail, and what remains exposed. PayTP is not an anonymity system. Its aim is narrower: no observer learns more than its seat at the table requires, and the protocol mandates no observer beyond the transacting parties, the software each runs, the rail both accepted, and the meed recipients its schema names — none of whom sits in the payment's path.

8.1 Payment Identity

PayTP defines no global payment identity. The wallet's master key never appears on the wire. What a merchant sees is a **merchant-scoped payer key**, stable across that relationship but unlinkable across merchants and scoped per registrable domain (§5.5). The stability is deliberate: receipts, credit history, and chaining need a durable counterparty, so each relationship gets a lasting pseudonym while the protocol refuses to create a universal one. The card model is the counterexample: one account number presented to every merchant is a cross-merchant tracking key by construction. Tier 0 involves no PayTP payer identity at all — no payer key, no channel, no registration (§5.6). What identifies the payer there is whatever the chosen settlement path exposes (§8.3).

The asymmetry is deliberate. Merchant identity is global by design — one stable key, publicly bound to its origin — because merchants trade in public and their signed history is an asset: the receipts a merchant issues are portable proofs any third party can verify, the raw material of market reputation (Chapter 13). Payer identity is scoped precisely so that no protocol-level counterpart record can exist: the protocol's identifiers cannot assemble a cross-merchant history of a payer, so there is nothing for a payer rating to attach to. These properties cover the protocol's own identifiers. They hold only if the meed metadata stays brand-level (§8.4) and the settlement-facing values follow §8.3's hygiene — discipline the payer's software applies, not a wire rule.

8.2 Who Sees What

Every observer of a PayTP payment is an endpoint, software an endpoint chose, the rail both sides accepted, or a meed recipient the schema names — and that last one observes only its own accruals, from outside the payment's path:

Seat	What it sees
Merchant	Its own relationship, in full: the per-relationship payer key, the signed terms, every slice, checkpoint, and receipt — plus the payer-side software brands carried in the meed metadata (§8.4).
Interaction layer	The flows it drives. A browser or agent framework already mediates the traffic it now meters; PayTP adds payment objects to what it necessarily sees.
Wallet	The payer's complete PayTP history across all merchants — custody and policy enforcement require nothing less. It is the one full-history seat, which is why substitution is a conformance requirement (§10.4): switching wallets partitions history.
Meed recipients	Their own accruals. Local roles read the signed checkpoint totals from the position they already occupy in the flow; remote recipients see the rounds' aggregate legs arrive at the instance they claim from — amounts and timing, never per-slice consumption, though a Tier 0 purchase is one rail event a transparent rail can expose (§8.3).
Settlement rail	Funding and settlement events only (§8.3).
On-path observer	TLS ciphertext. Every PayTP object travels inside the connection's encryption; what remains observable is traffic shape (§8.5).
TLS-terminating intermediary	A CDN that holds the certificate is the merchant for PayTP purposes (§5.5): it occupies the merchant's seat, sees what the merchant sees, and owes what the merchant owes (§7.6). For pass-through per-payment flows (§12.2) it is ordinary HTTP infrastructure instead — which still reads what it forwards: quotes, role headers, payment proofs, receipts.
x402 facilitator	Where a Tier 0 flow uses one to execute the rail transfer, it sees those purchases — an observer the payer's side chose, and can omit.[20]

No seat is given the graph of payments: the protocol mandates no party that observes payments across payers *and* merchants. The wallet's view is one payer wide, the merchant's one shop wide, the rail's as coarse as settlement makes it, and what a universal recipient like the Development Fund assembles from watching its own accruals arrive is that same settlement-level rail picture (§8.3), never the payments inside. Card networks and platform wallets hold the full-graph view by construction. In PayTP, breadth can only be earned: a widely used wallet or facilitator sees far more than one payer, but that position is granted per user and lost per user, not a structural feature of the wire.

8.3 What Reaches the Rail

A channel touches the rail at two kinds of moment (Chapter 6): **funding**, when a prepay deposit or top-up moves, and **settlement rounds**, at the negotiated thresholds, at window exhaustion, and at close. Everything between is invisible to the rail: ten thousand slices metering an API stream become one settlement transfer, and what was bought, when, and at what cadence stays between the endpoints. The thresholds are thus privacy parameters as well as exposure parameters: coarser settlement, coarser rail picture — a trade owned by the parties who set them.

What aggregation does not hide is participation. On a transparent rail, a settlement round has a recognizable shape: the merchant's net transfer beside an aggregate meed leg to the channel's instance. The meed outputs add PayTP-identifying metadata the underlying transfer would not otherwise carry, and an observer can classify it as a PayTP settlement and read aggregate payout sizes and timing. Settled off the baseline, the meed leg still lands on the baseline instance, a cross-rail pair matchable by timing. Rail choice is the lever — rails differ widely in what they publish, and rail-agnosticism means rails with stronger confidentiality qualify as they mature.

Tier 0 touches the rail once per purchase on the baseline — into the quoted split address — and twice on every other rail: a small meed leg first on the baseline, then the merchant's net leg, a pair an observer of both rails can match by timing. Concentrating every meed execution on one rail also concentrates its metadata: one chain carries the protocol's meed picture, which sharpens the recipients' view and an observer's alike. The no-payer-identity property is exactly as strong as the rail addresses behind it: pay ten merchants from one address and the rail links what the protocol did not.

Payers who want §8.1's unlinkability to extend to settlement SHOULD use fresh or per-relationship rail addresses where the rail supports them — at Tier 0 purchases and at channel establishment or chaining. Fresh addresses help only as far as their funding does. Gas or balance drawn from one source re-links them, so the stronger levers are rails with native confidentiality or fee abstraction as they qualify. Refund pointers follow the same hygiene — fresh or purchase-scoped where the rail permits, since a reused pointer links refunds and a stale one strands them. Destinations are immutable while a channel lives (§6.2), so rotation happens between channels, never within one.

8.4 What the Meed Reveals

Enabler incentives put metadata on the wire that would not otherwise be there. The `MEED_VECTOR` and the `PayTP-Roles` header name the payer-side software by its meed destinations: the merchant learns the wallet's brand and, where an agent is paying, the framework's. (The browser's brand it already had from `User-Agent`. The OS entry names an approved OS recipient or the fallback — platform facts it could usually infer, though the fallback signals that no approved OS applies.) This is the metadata cost of auditable distribution: meed recipients that can be verified on the wire (§7.6) are recipients that can be read on the wire.

The disclosure stays brand-level by stated discipline. Destination pointers identify meed-collecting organizations, and payer-side pointers SHOULD NOT encode per-user, per-device, or otherwise cross-merchant-stable values. §8.1's unlinkability is conditioned on exactly this holding. The exception that tests the rule is self-custody, where the wallet role's destination is the user's own: such wallets SHOULD derive per-merchant destinations where the rail permits, scoping the destination exactly as payer keys are scoped.

Timing matters as much as content. Role pointers are payer-side facts, and a client SHOULD send `PayTP-Roles` only to origins that have advertised PayTP support. Discovery is server-first, and the client's own advertisement (`PayTP-Version`) carries capability, not identity (§5.1). A PayTP-aware client that has not yet sent roles still gets a signed quote: the unasserted payer-side shares route to the Development Fund (§5.6), and retrying with roles redirects them to the client's actual interaction layer and wallet. So no payer-side brand or wallet identity reaches an origin that never asked.

Meed validation is designed to add no lookups. The role registry is versioned (§10.5), and implementations SHOULD validate against a cached or bundled snapshot rather than fetch per payment — so the registry operator sees snapshot fetches, not the per-payment timing picture the snapshot discipline exists to withhold.

In Tier 0 the vector is visible at the settlement layer too. The payment address is derived from the merchant, schema, and destination vector (§5.6), so on transparent rails an observer can classify PayTP purchases and often read the payer-side software brands from the address — including that a client supplied no roles at all. Two-leg meed entries add per-purchase records on the baseline rail — nonce, amount, timing, refund pointer — with the same classification value. And the merchant's delivery attestation for a delivered two-leg purchase (§5.6) makes the outcome and rough timing visible on the baseline rail (batching blurs that timing; the merchant's control endpoint can reveal it sooner).

That is the price of a meed collection that needs no one's cooperation. The mitigations are the rail's own confidentiality and the same pointer discipline — brand-level destinations, never per-user ones.

8.5 Traffic Analysis and Wire Hygiene

Inside the connection, hygiene is already normative. PayTP header fields are sensitive by rule — never-indexed encoding where the HTTP version supports it, `Cache-Control: no-store` on PayTP-authenticated responses (§5.2). And implementations SHOULD exclude payment headers from request logs and analytics, so payment metadata is not retained in routine logging by default.

What TLS does not hide is shape. Slice-bearing requests are slightly larger. Metering batches have a cadence. A paid unlock changes response patterns. An on-path observer can infer probable PayTP use from such signals, and no padding or timing scheme in v0.1 defeats traffic analysis. Coalescing slices into batches — already the metering path (§5.3) — blunts per-event resolution, and network-layer anonymity systems compose beneath TLS for flows that need them. Both are mitigations of degree, not guarantees.

Payment outcomes are visible to the application context that requested them, which makes wallet policy the defense against probing: consent prompts, per-merchant allowances, and budgets (§7.2) bound and record what a malicious script can learn by

attempting payments — though spending inside a standing allowance still reveals its success or failure to the context that requested it.

8.6 Residual Exposure

Exposure	Posture
The wallet sees the payer's whole history	Inherent to custody and policy enforcement. The seat is chosen, and substitution is a conformance requirement (§10.4); switching wallets partitions history.
Rail-level linkage and settlement fingerprinting	Inherent to transparent rails; bounded by aggregation, address hygiene, and rail choice (§8.3).
Settlement timing correlates with activity	A rail observer or counterparty can align settlement rounds with relationship activity; coarser thresholds blur the correlation, at the price of exposure (Ch 6).
Brand disclosure to merchants	Bounded to brand-level pointers by §8.4's discipline; the advertise-first rule keeps it from non-PayTP origins.
Traffic analysis	Not defeated. Batching reduces resolution; network-layer anonymity is out of scope and composable.
Durable purchase records	Receipts and checkpoints are non-repudiable by design, and portable: either endpoint can prove its side of the history to third parties (the raw material of merchant reputation, Chapter 13). Retention and disclosure are endpoint choices; the protocol mandates no retention beyond open obligations and no disclosure at all.
Merchant support is public	Discovery headers and the control endpoint make a site's PayTP support externally observable — site metadata, like advertising any checkout method.

The pattern is the architecture's: nothing in the payment's path is protocol-mandated beyond the parties, the software each chose, and the rail both accepted. Where PayTP cannot make an exposure smaller, it fixes who bears it and names the lever — thresholds, addresses, rail, wallet — in the hands of the party exposed.

---

9 Scalability

PayTP adds almost nothing to the web's compute and wire costs, and the reason is structural. The reference implementation has not yet benchmarked the design, so this chapter claims only what follows from the structure and leaves the numbers to that measurement (Chapter 11).

9.1 Nothing Global Grows

PayTP operates no shared component in any payment's path: no PayTP ledger, no PayTP consensus, no central switch to transit. The channel hot path touches no shared component at all. Tier 0's split contracts are immutable instances — one per merchant, schema, and destination set — code at addresses rather than a shared service, and outside the channel hot path entirely (§5.6). All hot-path state is bilateral, per channel, at the two endpoints that opened it, so capacity is a sum over independent relationships: adding one adds state at its two endpoints and nowhere else. Card networks scale through central switch capacity and blockchains through global consensus; PayTP's *hot path* scales like the web itself — the same absence of a mandatory intermediary that gives Chapter 1 its autonomy and Chapter 8 its privacy leaves nothing of PayTP's own to congest.

The one honest exception is the baseline rail, which carries every meed execution — inside the payment itself on baseline purchases, one small leg per purchase elsewhere. Its availability and throughput therefore bound every Tier 0 purchase and every meed-bearing settlement round: PayTP's one-shot and settlement capacity is the baseline rail's, not the web's, and a baseline-rail outage stalls all new one-shot and channel-settlement commerce until it clears, degrading affected merchants to legacy checkout (§11.2). Channels sharply reduce that load — one leg per settlement round, not per payment — but do not remove it (§6.4).

9.2 The Work per Payment Is Small — and Chosen

A slice costs one constant-time tag check, a sequence check, and counter updates — ~35 bytes riding connections that already exist, with no signature, no rail interaction, and no required disk write (Chapters 5 §5.3, 7 §7.7). Signatures appear only at establishment, checkpoints, and settlement, so asymmetric crypto per value moved

scales with the evidence bound `E` and the settlement thresholds — dials the parties set (Chapter 6).

The rail is touched only at funding and at settlement rounds (`TH_value`, `TH_time`, window exhaustion, close): hundreds of thousands of metered calls become single-digit daily transfers (§6.6), and coarser thresholds mean fewer still — each round is one net transfer plus one baseline meed leg, never a transfer per recipient. Tier 0 inherits its rails instead — one settlement into the split address per purchase, or a meed leg paid first on the baseline rail, at the rate fixed in the signed quote, plus the merchant's net leg — which is exactly why recurring relationships upgrade to channels.

9.3 Endpoints, Latency, Limits

Per-channel state is constant whatever the slice count: the signed terms, session keys, counters, the transcript's chain head, the last checkpoint. The hot path shards by channel with no coordination between machines, while the bookkeeping that must span channels — aggregate exposure per counterparty, consumed chaining references — grows with relationships, never with traffic.

In steady state payments add no round trips: slices ride the requests they pay for, and checkpoints and settlement run off the request path. The interruptions are negotiated, not accidental: a pause at the evidence bound `E` until the bilateral checkpoint completes, a wait at window exhaustion, and Tier 0's rail latency — settlement precedes delivery by design. A one-shot two-leg payment is the slowest case: it waits *serially* for baseline finality on the meed leg, then the net leg (§5.6, Appendix A) — the price of converting only the meed rather than the whole amount, and a reason recurring relationships graduate to channels, where the rail leaves the request path entirely.

What bounds the system at scale: aggregate rail capacity at the thresholds the parties chose; checkpoint signing at extreme evidence cadence, amortized by `E`; control-endpoint signature verification, bounded by rate limiting and amortized at checkpoint cadence (§7.7); establishment cost for one-shot flows, which is what Tier 0 is for; split-contract deployment and recipient sweep tooling, amortized across purchases; and the cross-channel indexes above. Every number beyond these — slices per second, settlement throughput, end-to-end latency — is a measurement for Chapter 11. The design's claim is narrower: work per payment is constant, state

per relationship is constant, nothing the protocol itself operates grows with payment count — baseline meed load grows with one-shot volume and settlement rounds, the §9.1 dependency channels exist to compress — and every knob that turns volume into cost is held by the parties who bear it.

---

10 Incentive Structure

This chapter defines the meed as a technical object: the schema, the `MEED_VECTOR`, the governance rules, and the mechanism by which collection holds. The meed is a small, capped, published share of every payment, paid to the software roles that make payment possible — so the software carrying payments has a protocol-native reason to do so. Whether that incentive is good for the ecosystem, and the objections to it, are argued outside this specification, in Appendix D.

What the meed guarantees — the plain promises the design is built to keep:

- The software that makes a payment possible is paid on the wire, and no single party can switch that off or quietly raise the meed — collection rests on conformance and evidence, not a central enforcer (§7.6).
- No PayTP meed is added to the buyer's price — the incentive share comes from the merchant's side (the buyer's own rail and conversion costs are separate, as on any rail, §10.1).
- The share is a fixed, published number, identical on every payment, so every party can forecast its income and none can gouge.
- It works the same for a one-off purchase and a high-frequency relationship, and whatever currency or rail the two sides use.
- The merchant cannot keep anyone else's share: the split and meed instances divide it by code. The one mode where the merchant itself executes the meed — a prepay channel — is bounded to at most one settlement round's worth of accrued meed (Chapter 6).
- Every recipient, including those far from the sale, can find and audit against the signed record what it is owed — subject to the limits §7.6 sets out for remote recipients and collusive channels.

How the design keeps these true against collusion and tampering — and the bounds where collection rests on conformance rather than cryptography — is Chapter 7's subject (§7.6, §7.8).

10.1 The Meed Schema

The meed is fixed per schema, not negotiated per payment. Every payment under a schema carries the same shares, so a merchant cannot vary the meed by user, and every role can forecast its income. That predictability is deliberate — payment standards succeed through it. A flow picks among the schemas on offer (§10.5). It never alters one.

The meed has two tiers. Every schema allocates **exactly 100 basis points (1.0%)** to the base roles below. Where a role does not apply, its share redirects (as the OS fallback does); no schema reduces or exceeds that base. Above it, the protocol's hard cap of **150 basis points (1.5%)** reserves the remaining range exclusively for **opt-in service roles** — things like escrow or dispute resolution — seated per flow (§10.5). Services add to a vector, never displace the base. The initial schema:

```
MEED_SCHEMA_ID = 0x01:
  Interaction Layer (0x10):  50 bp
  OS                (0x11):   10 bp  (→ an independent open-source
                                      fund where no approved OS applies)
  Wallet            (0x12):   30 bp
  Development Fund (0x13):   10 bp
```

The schema is named in the mutually signed `CHANNEL_AUTH` (validated once at establishment) or the merchant-signed Tier 0 quote (validated by the client before paying). Schema changes — including any rebalancing within the 100 bp base — require broad ecosystem consensus under the Foundation charter (§10.5). No single stakeholder can alter a schema unilaterally, and the 150 bp cap binds every future schema.

The distance between base and cap is a worst-case guarantee, not a growth allowance: a payment carries basis points above 100 only for a service role its parties explicitly agreed on.

The roles:

- **Interaction Layer (`0x10`)** — the software that initiates and mediates the payment flow: a browser, an agent runtime or framework, a native app, a game engine, a POS system. The role is defined by function, not category — whatever software drives the paying side's requests earns it. Where layers nest, it belongs to the one that initiates and mediates and asserts it in `PayTP-Roles` (a contested assertion is a certification matter, §10.5). The layer that decides whether payments are possible is the layer the meed pays first.
- **Operating System (`0x11`) — the 10 bp is always allocated: to the OS whose secure facilities the wallet uses, drawn from a Foundation-published registry of approved recipients — or, where no approved OS applies, to an independent open-source fund the PayTP Foundation does not control** — its steward set by governance, changeable only to another independent one. Fixing the total removes the gain from gaming the entry — the merchant pays 100 bp regardless, all destinations are pinned, so a false assertion can only deny an OS its share (routing it to the fallback), never move value to the asserter. And because that fallback leaves the Foundation entirely, no one who decides OS eligibility gains from an absent OS. An OS earns registry entry by actively supporting PayTP (secure key custody, background settlement, platform APIs), through the open process of §10.5.
- **Wallet (`0x12`)** — key custody, channel liquidity, settlement execution, compliance. Wallets additionally earn FX spreads and liquidity fees off-protocol. The on-wire meed covers the protocol-mandated work, not the balance-sheet business.
- **Development Fund (`0x13`)** — specification maintenance, security audits, conformance testing, bug bounties. Ten basis points is the insurance premium against critical open infrastructure becoming the next under-maintained dependency the internet quietly relies on.

The redirect pattern generalizes: an absent role's share never reduces the total — it routes to that role's pinned fallback. For payer-side roles a PayTP-aware flow leaves unasserted, that destination is the Development Fund. For the OS role it is the independent open-source fund above.

The meed pays payer-side software by design. Merchant-side software — a storefront plugin, a gateway — already bills the merchant it serves. Payer-side software has no such customer to pay it.

The **merchant** receives gross minus the meed — exactly 99.0% under schema `0x01` (the scheduled split, up to the sub-unit dust §10.2 bounds), with no card-style chargeback at the PayTP layer. The figure is the protocol split, not the all-in cost: rail fees, gas, conversion, and any opt-in service shares sit outside it (and rail costs are what channels amortize, Chapter 9).

How the meed schema is priced. The split is deliberately not cost-based — on costs alone the wallet, which carries custody and capital, would out-earn the interaction layer, which ships software. The split prices **scarcity**: distribution is the bottleneck, so the largest share goes to the role whose absence kills the network — the interaction layer, while the wallet's protocol share is complemented by its off-protocol liquidity and FX business. Each role clears its own participation bar. None is priced to gouge.

The schema numbers are a versioned hypothesis: governance can rebalance shares against real adoption data, within the 100 bp total, under the same hard process as any schema change.

In agentic deployments the mapping is direct: the agent runtime is the Interaction Layer, earning `0x10` without negotiating with anyone; the wallet is the operator's treasury (a custody provider, self-custody, or the framework's partner wallet — operator-selected, §10.4); "user consent" is the operator's policy in the wallet; and a headless server uses no OS secure facilities, so the OS share routes to the independent open-source fund — the fallback working as designed. The payer pays the listed price. Meeds are carved from the merchant side, never added on top.

10.2 Meed Arithmetic

The exact accrual, conversion, division, and rounding rules are the formal specification's (F7). Three invariants hold at design level:

- the meed accrues per recipient as an exact integer — amount × basis points — with no division or rounding until settlement;
- division happens only at settlement, floored to the rail's unit: a channel round pays one aggregate leg to the meed instance, which divides it among the pinned destinations, while a Tier 0 payment settles or funds its meed in the same act as the payment;

- rounding never mints or owes: sub-unit dust is absorbed at settlement (for a channel, out of the merchant's net) and is never a debt on anyone's books.

In both tiers a payment is complete only when its meed execution is (§7.6).

10.3 Why Collection Holds

The security chapter (§7.6) establishes what cryptography does and does not do. Meed *accounting* is tamper-evident, and settlement is all-or-close within the protocol. But nothing forces endpoints to speak PayTP at all. Collection therefore rests on selection, not capture, and the mechanism has three parts:

1. **The software that selects the payment path earns on exactly one of the options.** The interaction layer and wallet earn their shares only on PayTP payments. So where a merchant offers plain x402 and PayTP at comparable price and reliability, PayTP-aware software has no reason to complete the plain offer: it earns nothing there, and its user gives up the PayTP bundle (a signed quote and receipt bound to the meed vector, plus the channel-upgrade path and optional dispute resolution). Mainstream payers run distribution-built, auto-updated software, and the selection runs on rails that software controls. Self-supply is no leak: an operator whose own software drives its requests is the interaction layer for those flows and earns that share.
2. **The meed is additive.** It falls only on payments PayTP-aware software brings, prefers, or meters through channels — never on the plain traffic a merchant already serves, which is untouched. A merchant offers PayTP's terms to reach the buyers PayTP software brings, and stays free to offer plain paths beside it. Why merchants accept the meed is argued in Appendix D.
3. **Each role's protection matches its position in the flow.** The interaction layer operates the very connection each channel binds to, so it can verify its own entry — and the OS entry, against the platform it runs on — before relaying `CHANNEL_AUTH`. The wallet signs everything it accrues. The OS and Development Fund, with no position in the flow, have registry- and schema-pinned destinations that cannot be silently redirected. Local roles read the signed checkpoint totals. Remote recipients claim rail-visible payouts from the channel's instance. And Tier 0 meed execution is verified by the merchant before delivery and by PayTP-aware clients before payment — there is no merchant-onward remittance step left to under-perform (understatement of a channel's aggregate leg remains

the §7.6 evidence-and-conformance case). Pointer freedom exactly where a role can defend itself in-line; pinned destinations exactly where it cannot.

Selection must serve the payer. Consider a merchant that prices its PayTP path above a plain one. Conformant PayTP software MUST resolve that to the best benefit for the payer, not for its own meed: comparing paths on the payer's total cost, never hiding a cheaper one, honoring user or operator policy over its own incentive, and disclosing the share it earns. Preferring PayTP is legitimate only where it costs the payer no more. No master enforces this. No gate shuts non-conformant software out, so the protocol makes such steering visible and costly to a vendor's standing — the marks, the registry, the auditable on-wire meed — rather than impossible.

10.4 Wallet Authority and the Partner-Wallet Model

The wallet holds all spending authority. Channels exist only under a wallet-signed `CHANNEL_AUTH`. Slices and checkpoints require wallet-held keys and signatures. The interaction layer initiates and mediates but cannot mint slices, extract keys, or spend outside wallet policy. What a compromised browser or framework retains is misuse *within* that policy — which the wallet's budgets, thresholds, and consent rules bound (§7.2).

This separation is a regulatory feature, not a technicality. Money-transmission licensing, KYC/AML, and custody regulation attach to the party holding and moving funds — the wallet role. An interaction layer that integrated custody would inherit those obligations in every jurisdiction it ships in. Under PayTP it instead *partners* with licensed wallet providers and never touches customer funds. The protocol's role separation is designed to support the regulatory separation the industry already requires, though jurisdiction-specific analysis remains every implementer's own.

PayTP does not insure custodial funds, recover lost keys, or override legal freezes: the wallet is a chosen, regulated counterparty. Destination pointers are accepted, never imposed — a merchant MAY decline to offer terms naming a destination it cannot lawfully pay (the flow then ends, or the payer retries without the role and the fallback applies) — but a merchant never rewrites a present role's destination, and nothing reduces the 100 bp base.

Regulated functions stay in the wallet. No other per-payment actor holds or redirects customer funds: the merchant receives payment for its own goods, distribution software receives its own revenue share, and infrastructure operating a merchant's

endpoint carries messages, not money. In the agent model, funds sit with the operator's wallet, whose customer is the operator, and the agent spends only under that operator's wallet policy. Duties that do reach a merchant come from what it sells, not the PayTP role itself. The credential lane (Chapter 13) exists for those.

When one vendor holds both roles. Nothing on the wire stops a single company shipping both the interaction layer and the wallet — and 80 bp combined is a real inducement. The protections independent of their separation hold regardless (OS and Development Fund destinations pinned, schema totals invariant). What a bundled stack erodes is the trust boundary of §7.2: one vendor's bug now reaches both roles, and if bundling becomes lock-in, the contestability of the wallet market suffers too. PayTP therefore makes pluggability normative: **an interaction layer MUST allow the user or operator to select an external wallet.** Bundling a default is legitimate competition. Preventing substitution is non-conformance.

10.5 Governance and Future Roles

Meed schemas, role eligibility, and certification are governance, not wire protocol. They belong to the planned PayTP Foundation charter — including the Development Fund's administration, a multisig under elected maintainers in the working design. And every governed quantity, from schema to shares to destinations, is signed and auditable on the wire, so governance runs on facts.

The Foundation's neutrality is structural, not asserted. Because an absent OS's share routes to an independent fund it does not control, approving or denying an OS changes its income by zero — exclusion earns it nothing. And the charter restricts the Development Fund to specification work, audits, conformance, and grants. Denials and revocations are appealable to a body outside its own appointment. Changing the meed split is meant to be hard and rare, under a process the charter will settle before deployment. The 100 bp base and 150 bp cap bind every version regardless.

The trademark carries a standing covenant: any implementation that correctly speaks the protocol may use the PayTP marks, and may organise its own governance under them, provided it claims no Foundation endorsement. So the Foundation cannot lock the ecosystem to itself — an abusive one can be forked away from. Its limits are in Appendix D.5.

Future and service roles. Currently only one future role is reserved: **dispute resolution** (`0x14` proposed) — escrow/refund agents earning a dedicated share where buyers, verticals or jurisdictions require them (Chapter 13). A new *base* role (one every payment pays) can be created only by redistributing the existing 100 bp under the full process. A *service* role is optional: only the payments that use it pay for it. The merchant decides whether to offer and fund it, a wallet can refuse to buy without one, and since every payment runs under exactly one schema, no payment can ever carry more than 50 bp of opt-in services.

Three rules keep the tier honest: a service must be a real service actually used in that payment (not a base role renamed); it applies only where the merchant offers it and the buyer accepts it; and it stays capped.

The reason to write a service into the vector at all — rather than buy it privately — is that the vector is what the payer can check. A dispute-resolution entry in the signed terms is verified by the wallet before it pays, and collected by the same machinery as every meed. The on-wire fee is only the minimum share: a dispute agent may charge for premium terms off-protocol. And competition keeps service fees near cost — which requires that providers stay interchangeable, so policies SHOULD demand "a registry-listed dispute agent," never one named company.

---

11 Implementation Considerations

A first reference implementation is under active development; nothing here is yet production-proven or independently validated. This chapter describes what a conforming implementation consists of, which integration paths are open today, and what only running code can decide. The protocol surface is deliberately small; most of the work is wallet plumbing, settlement adapters, and the operational tooling around the split contracts.

11.1 What a Conforming Implementation Consists Of

- **A core library** — the parts that must be byte-identical everywhere, built once and bound into every role's software: the encodings and canonical forms, the signature and MAC operations, the channel state machine, the fee arithmetic

(§10.2), and role-registry snapshot handling (cached, versioned, validated against the acceptance window).

- **A wallet** — key custody and per-merchant key derivation, the spending-policy engine that holds all authorization (Chapter 7), rail execution (including two-leg Tier 0 flows and every channel settlement it owes as debtor), and channel-lifecycle management. Wallets SHOULD track the two-leg entries they fund and, for any lacking a receipt or delivered response, open reclaim automatically and refund unattended after the contest delay (refunds can only reach the recorded pointer). Wallet substitution is a conformance requirement (§10.4).
- **A merchant library** — discovery, quote construction, redemption verification, the control endpoint, and channel settlement. It runs wherever TLS terminates, since that party holds the merchant's certificate. Because channels bind to the payer↔merchant relationship rather than the transport, an ordinary load balancer does not blind it, and an origin-authorized terminator MAY even forward establishment for the origin to meter (Chapter 12).
- **Settlement adapters**, one per rail family (stablecoin transfers, Lightning, instant bank rails). Rail-agnosticism lives here: rail-specific execution and finality sit behind the adapter, while quote and redemption logic consume only the adapter's declared capabilities — contract support, finality levels, assets, and how a transfer binds to a quote. A rail that cannot bind a transfer to a quote — via an immutable sender-chosen memo or a unique per-channel pointer — cannot carry a Tier 0 net leg, nor a channel's funding, net, or meed leg. And where a rail permits later recall, a proof is final only to the level the adapter names — the merchant bearing that rail-level reversal risk exactly as it would outside the protocol.
- **The baseline contract kit** — the Foundation-published factory and reference contracts, deployed only on the baseline rail, in the two forms of §5.6 (the split and the meed instance), plus the tooling those contracts assume: derivation, deployment, claim/sweep for recipients, reclaim for payers, batch operations, and indexing that discovers instances and entries by watching the baseline rail (§11.5). Recipients watch one rail for all PayTP income.
- **A conformance suite** — wire-format vectors, state-machine transitions, replay and tampering corpora, fee-arithmetic cases, and the Tier 0 redemption matrix. Certification under the Foundation charter rides on it (§10.5).

11.2 The Baseline Rail

Conformance includes one hard interoperability rule: **every implementation must be able to settle over the common baseline rail**, so any two conformant endpoints always share a settlement path. The baseline carries every meed execution PayTP performs — the split on baseline purchases, the meed leg of every other purchase, and every channel round's meed leg.

That load fixes the selection criteria. The rail must be:

- **contract-capable** — immutable code at addresses derived from the code itself, no upgrade authority, and no way for other code to occupy a derived address (the invariants the address-is-the-contract verification of §5.6 rests on);
- **economic for meed execution along every path the design uses** — no marginal transfer inside a baseline payment, one aggregated leg per channel round, a per-payment leg on other rails above a documented minimum size, under ordinary and congested conditions;
- **open and nondiscriminatory** for deploying and calling instances, with no discretionary operator approval or withdrawal;
- **liquid in the assets implementations need**, judging asset and rail together — an asset that cannot lawfully carry a market's expected volume fails in that market whatever the rail's merits;
- **final quickly and predictably enough** for interactive redemption.

Which rail meets these best is left open here on purpose — a decision for the implementation phase and wider-community review, made on measured fees and finality, with a migration path if the first choice proves wrong. Until it is fixed, the baseline requirement is a design constraint conformance binds to, not yet an interoperability guarantee between independent implementations. Strong current candidates are the established smart-contract settlement networks with deep regulated-stablecoin liquidity — among them Solana and the major Ethereum layer-2s (Arbitrum, Base) — with purpose-built payment chains (Stripe's Tempo, Circle's Arc) as emerging entrants to watch.[21] The first reference implementation builds against Solana as its working target — a build choice, not the protocol's.

The baseline is an interoperability requirement, not an availability guarantee. Since it is the one place every meed executes, its availability is priced honestly. While it is unavailable or uneconomic, per-payment quoting waits with it, and channel rounds

defer within their negotiated thresholds. Deferral loses nothing: meeds already accrued in a channel's countersigned state settle in full at the eventual round, and no PayTP payment ever completes without its meed execution. Unavailability narrows *which* payments can be PayTP payments, never *what* a PayTP payment pays.

Past the thresholds, a round that cannot execute its baseline leg runs into the ordinary all-or-close machinery: the channel pauses or closes with obligations standing in signed evidence. Degradation never extends credit. Implementations SHOULD treat brief unavailability as latency — meed-leg funding retries idempotently within the quote's validity, and merchants MAY lengthen quote validity while weighing the longer rate exposure.

This concentration is the deliberate price of giving recipients one rail to watch. It is an implementation obligation, not a user-facing one: wallets source the baseline asset as part of executing the meed leg, at the quote's pinned rate — payers never hold it.

11.3 Integration Paths

Agent frameworks first. An agent runtime owns its HTTP client outright, so it faces none of the constraints below — it links the core library, delegates custody to the operator's wallet (§10.4), enforces the operator's policy, and earns the interaction-layer share by shipping the integration. A first deployment needs exactly two adopting parties — one framework, one API provider; everything else (the wallet, the schema, the contract kit) is protocol infrastructure defined once for everyone, not negotiated per deal. That is why this path leads.

Browsers. Because a channel's keys bind to the relationship and the sealed secret rather than a TLS-exporter interface (§5.5), a browser extension does not need transport-level access it cannot reach. It can carry slices over ordinary `fetch`, so browser channels are technically reachable — whether an extension is the right place to hold channel keys is a wallet-trust question left open. Either way an extension can implement **all of Tier 0 today**: discovery, quotes, split-address validation, payment via a companion wallet, receipts. Native surfaces (a `navigator`-level payment API, toolbar spend controls) are future work for adopting browsers, not defined here. One cost is worth naming: a channel ends with its establishing connection, so a dropped transport needs a fresh chaining exchange — one signed round trip that never touches the rail. A future revision could let a channel survive a reconnect.

Merchant side. A library or reverse-proxy module on whatever terminates TLS; commerce-platform plugins are the long-tail path. A merchant's integration is deliberately smaller than a client's — advertise, quote, verify, deliver; no custody, no policy engine, no onward remittance. The one addition: a merchant offering two-leg quotes MUST post a signed delivery attestation for each completed two-leg purchase (a message that moves no funds).

Mobile and OS. Platform wallets with hardware-backed custody and an OS-level payment intent are the eventual shape. An OS vendor that ships these earns registry entry (§10.1). Nothing earlier waits on them.

The sequence: agent-and-API first, extension-plus-wallet Tier 0 for early human commerce, then native channel support where volume justifies it. The economics and timing live in the business planning around this document, not the specification.

The reference implementation is being built with Tier 0 as an x402 embedding, with an MPP profile as planned future work pending the §13.2 feasibility review; either way, the hardened exchange of §7 holds only where PayTP-aware software runs on both ends. A plain x402 client can interoperate with baseline quotes, but it does not perform the PayTP checks.

11.4 What Only Implementation Can Decide

Deferred to running code, deliberately:

- every performance number — Chapter 9's structural claims are the whole quantitative story until the reference implementation measures more;
- the baseline-rail selection under §11.2's criteria;
- split-contract gas economics and sweep cadences;
- wallet UX for budgets and consent;
- multipath settlement across rails (liquidity routing between hubs is future design, not v0.1).

The reference implementation, its conformance suite, and an adversarial review of both precede any production claim.

11.5 Operational Obligations Beyond the Wire

Several duties this specification relies on are procedures the reference implementation and the Foundation charter own. The specification's part is to fix the invariant each MUST preserve.

- **Exactly-once state is durable-or-fail.** The consumed-nonce, funding-reference, and chaining-reference records are a merchant's own liability: the rail proves money moved, not which decision consumed it. An operator that loses this state and cannot rebuild it MUST refuse the proofs and references it can no longer adjudicate, never guess — the cost is denied service, never a double credit or double delivery.
- **Instances a quote names must become spendable.** A merchant SHOULD deploy the split instances its quotes name (its own 99% is trapped until it does); the funding wallet ensures a meed instance is live before it funds an entry (§5.6). A counterfactually funded but never-deployed instance strands its counterparties' money and is non-conformance under certification.
- **Delivered two-leg entries are closed by the merchant.** A merchant that delivers a two-leg purchase MUST cause the delivery attestation to be posted before its reclaim can execute (batched, by itself or a poster service). The invariant: no delivered meed entry is left open for the payer to reclaim; a submitter's failure is the merchant's failure. A merchant that prefers no baseline-rail posting quotes on the baseline, where the split needs none.
- **Recipients discover by watching the rail, not by deriving.** A remote recipient cannot compute the addresses that owe it (the derivation includes payer-side pointers it never sees); it finds them by watching the baseline rail and the factory's deployments — which is why instances are deployed and factory-enumerable. The completeness of that index is an ecosystem deliverable, not a protocol guarantee.
- **Baseline migration preserves obligations.** The migration path is governed, not automatic: because a quote and a `CHANNEL_AUTH` name the baseline rail's network, endpoints can quote a new rail while honoring entries and instances already funded on the old, running an acceptance window across both until the old rail's obligations drain. No funded entry, split balance, or channel obligation is abandoned by the switch.
- **Registry and schema lifecycle is governance.** Revocation distribution, the version-acceptance window, and schema activation and sunset belong to the charter

(§10.5). The wire's part is only that a signed vector names its version and a validator rejects a version it knows revoked. A stale validator is a conformance failure, not a protocol ambiguity.

---

12 Compatibility with Existing Internet Infrastructure

PayTP requires no change to routers, NATs, firewalls, load balancers, or DNS: every PayTP object travels inside ordinary TLS on ordinary ports, so to anything that is not an endpoint a PayTP session looks like the HTTPS the network already carries (Chapter 4) — though traffic shape can still hint at probable use (§8.5). This chapter covers the three places where compatibility is a real question rather than a free property: TLS terminators, enterprise inspection, and the deployed x402 ecosystem.

12.1 To the Network, Nothing Changed

Core and edge routers see standard TCP on port 443. NATs, stateful firewalls, and DPI equipment see TLS application data with no new ports, options, flags, or plaintext bytes, so classifiers that match SNI or ALPN keep matching them. Slices are small additions to requests that already exist — request-carried slices respect ordinary header limits, and metered volume rides the control endpoint's body path (§5.3) — so MTU, QoS, and congestion behavior stay the underlying connection's own. Payment objects are encrypted in-stream data, not a reflection or amplification vector, so they add no volumetric-attack surface (protocol-level denial-of-service is §7.7's subject).

12.2 TLS Termination and CDNs

The one infrastructure element PayTP genuinely concerns is whatever terminates TLS, because the merchant key is bound to its certificate through the binding artifact (§5.5): a channel forms only where the artifact the origin publishes names the certificate that terminated the connection. A party terminating with the merchant's own certificate **is the merchant for PayTP purposes**; a party the origin authorizes may instead forward establishment for the origin to meter; Tier 0, having no channel ses-

sion key, is simpler still. For the large share of the web fronted by CDNs, that yields four configurations:

1. **The CDN implements the merchant side** — terminates TLS, runs the merchant library at the edge, and settles with its origin, exactly as it already provides TLS, caching, and WAF as services. This is the natural product path for edge networks, and the only configuration in which a CDN-fronted origin gets full PayTP without touching its own stack. The CDN takes the merchant's seat and its obligations — in per-payment flows that seat never holds meed money (§5.6), and no conforming settlement of its channels completes without the meed execution (§7.6).
2. **The merchant carves payment traffic out of the CDN** — a payment hostname or API path terminating at the origin, with static content staying on the edge. Common already for API traffic, the likely first place PayTP deploys.
3. **The CDN passes traffic through to the origin** — it forwards discovery headers, payment headers, redemption bodies, and channel establishment unmodified; the origin's merchant key signs quotes and receipts, the origin unseals session secrets and meters channels, and the edge is ordinary HTTP infrastructure. Tier 0 passes through on the merchant signature over the quote, so the edge rotating its TLS certificate does not disturb it. Channels pass through where the origin authorizes this edge by publishing a binding artifact naming the edge's certificate together with the origin's channel-encryption key: the client seals the session secret to the origin, the edge forwards ciphertext it cannot open, and the origin meters. The edge sees plaintext slices and can deny or drop, but cannot unseal the secret or forge a slice, so it steals no value. The trust is the web's own — an edge that can rewrite a page can substitute payment discovery too; an origin that does not accept that authority uses configuration 2.
4. **The CDN does nothing** — unknown headers are dropped, discovery never completes, and both sides fall back to legacy checkout. Nothing breaks; payment capability is silently absent. Whether that state persists is the merchant's choice, and the remedies are configurations 1–3 — though a header setting alone delivers only the per-payment tier: channels additionally need the edge to run the merchant library (1), an origin-terminated payment path (2), or an origin-authorized pass-through (3).

In configuration 3, one seam remains: the origin does not observe the wallet's own connection, so the rule that a channel ends with that connection (§5.5) needs the

terminator to propagate the wallet-facing close, or the origin to enforce an equivalent idle timeout.

12.3 Enterprise TLS Inspection

An enterprise proxy that intercepts TLS presents its own certificate, which the merchant never signed into a binding artifact — so the artifact check refuses to open a channel through it (§5.5), and PayTP falls back exactly as for any unauthorized terminator, the same position online banking and certificate-pinned applications already occupy. The established remedy applies unchanged: inspection policies exempt payment and financial origins by allowlist. Organizations that instead want visibility into what their own systems spend have a better tool than interception: the wallet is theirs, its policy is theirs, and every payment it makes is signed, budgeted, and logged at the endpoint (§7.2) — richer records than any middlebox could reconstruct.

12.4 The x402 Ecosystem

Compatibility extends to deployed payment infrastructure, not just plumbing. A PayTP merchant's per-payment endpoint is an x402 endpoint[3] (§5.6): the merchant reuses its challenge flow, facilitator relationships, and settlement tooling, adding signed PayTP terms for PayTP-aware clients.

Plain x402 offers coexist beside PayTP ones — in the same menu or on separate resources — and remain plain: a plain offer names the merchant's own address, not a split, so a payment on it is not a PayTP payment and carries no meed. Baseline PayTP quotes coexist safely too, because their `payTo` is the split — an unaware x402 client that pays one still completes it. Two-leg quotes are the exception: they need a second leg an unaware client would not execute, so a merchant serves them only to clients that advertised PayTP, never as a plain-selectable entry (§5.6). That is where the x402 reuse stops — everything up to it carries baseline PayTP as ordinary x402.

In the other direction, a PayTP-aware wallet completes plain x402 purchases anywhere as a matter of course — earning nothing on them, which is exactly why it prefers the PayTP offer wherever one exists (§10.3).

The hardened exchange (§7) is likewise available only when both ends speak PayTP: the wallet verifies the origin-bound quote, and the merchant gates delivery on its own settlement record. A plain x402 wallet still pays a baseline PayTP quote — the split divides the meed by construction — but it earns no share of its own and runs no origin check. Nothing in x402 or MPP prevents these protections; they ride in the PayTP terms aware endpoints exchange. Two-leg quotes remain PayTP-only.

12.5 Fallback

If either side omits the negotiation, the connection proceeds as ordinary HTTPS and the application falls back to whatever checkout it had — no reset, no retry, no penalty (§5.1). Every adoption state is therefore stable: merchants can run "PayTP preferred, legacy accepted" indefinitely, and clients gain PayTP capability without losing access to anything that lacks it. Incremental deployment is not a transition plan bolted on; it is what the wire behavior produces by itself.

---

13 Future Work

PayTP v0.1 deliberately limits its scope: two payment tiers over existing rails, bounded and evidenced trust, and the governed meed. The directions below are under investigation for future versions. None is normative here, and none carries a date.

13.1 Dispute Resolution and Buyer Protection

v0.1's posture is deliberate: settlement precedes delivery, every exposure is bounded, and purchase-price protection is bought explicitly rather than bundled into every payment (§7.3, §10.5). A future version turns that explicit purchase into protocol support: a reserved dispute-resolution role (`0x14`, §10.5) that holds a protected purchase's net at an immutable instance, released by evidence within a window and allocated by a registry-listed dispute agent under rules published with the schema. It builds on the receipt, checkpoint, and held-instance machinery the protocol already defines — though implementing it extends the spec rather than merely configuring it.

The full design is developed in **Appendix C**.

13.2 Other Directions

- **A collateralized channel profile** — channels whose exposure window is escrowed on a contract rail rather than bounded and evidenced, for counterparties that want trustlessness and will pay for it in locked capital.[22]
- **A credential lane** — merchant-required payer attributes as selective-disclosure attestations signed by the regulated wallet, bound to the merchant-scoped payer key, predicate-only — preserving Chapter 8's no-identity-on-wire design. The attributes a merchant needs — because law requires them or because screening risk makes them prudent — are narrow and nameable: proof of age for age-restricted goods and content, jurisdiction or residency where sales are restricted, sanctions-screening status, and identity above the reporting thresholds that attach to a few high-value goods categories. The checks payment regulation itself requires sit with the wallet, the regulated funds-holder (§10.4). What reaches a merchant comes from sector, sanctions, or tax rules, and arrives as a narrow question about the payer, not a duty to know who they are.[23] What remains for a future version is the attestation vocabulary and its wire formats.
- **Post-quantum signature suites** — negotiated TLV types per §5.8; no wire redesign required.
- **Transport-native embeddings** — the TLV wire format is transport-agnostic, so two deeper embeddings are anticipated once the HTTP profile has implementation experience behind it: a **QUIC transport profile** negotiating support via a `PAYTP_TP` transport parameter and carrying slices in a dedicated frame type (`0xC0`) interleaved with stream data, collapsing negotiation into the transport handshake for latency-critical streaming; and a **WebTransport profile** carrying slices in datagrams alongside media and game traffic without modifying any QUIC stack. QUIC additionally needs a standardization path for the frame type and transport parameter.[24]
- **An MPP embedding profile** — the TLV terms PayTP carries (the signed quote, meed vector, and receipt) are not bound to x402's wire encoding. MPP (§3.6), the IETF in-band-402 scheme, defines its own extension mechanism, so a PayTP-over-MPP profile — the same terms carried through MPP's `WWW-Authenticate: Payment` exchange — is a natural second embedding, with no change to the meed

schema, role registry, or Development Fund. It awaits a feasibility review confirming MPP's extension slot can carry the PayTP terms.

- **Multipath settlement** — liquidity routing across rails and hubs, deliberately excluded from v0.1 (§11.4).
- **A fallback meed rail** — a second, pre-declared instance family on another rail, quotable only while the baseline is provably unavailable, so one-shot PayTP quoting stays available through baseline outages. Set aside for v0.1 deliberately: recipients would watch two rails instead of one, the baseline contract kit would deploy twice, and quotes would need auditable outage-declaration rules — a permanent price for insurance against rare, bounded windows (§11.2). It becomes worth designing if pilot data shows one-shot availability gaps that matter.
- **Additional rail adapters** — instant bank rails and central-bank digital currencies as their interfaces open; each is an adapter, never a wire change.
- **Standardization** — the HTTP profile's header fields and the x402 extension registration are candidates for formal standardization once the reference implementation stabilizes.

Considered and set aside:

- **Per-hop incentives.** Paying forwarding infrastructure a micro-toll was studied and rejected: path position is not a role the commitments compensate — payment for carriage the endpoints did not choose is the unearned seat the end-to-end commitment exists to remove; in the HTTP profile PayTP is invisible inside TLS, so hop awareness is impossible anyway; and settlement-layer intermediaries are counterparties who already earn bilateral fees. No role or code point is reserved.

13.3 Reputation Ecosystems

The protocol already defines reputation's raw material without defining a reputation system. A merchant's identity is one stable key, publicly bound to its origin (§5.5), and the evidence it signs — quotes, receipts, bilateral checkpoints — is portable: any third party can verify any PayTP receipt against the merchant key, with no session, no permission, and no per-merchant integration, because the formats are standardized.

What can be built on that, entirely outside the protocol: wallets maintaining merchant scores and allowlists; agent frameworks consuming machine-readable trust feeds (machine payers need codified trust more than human ones); insurers pricing protection from verifiable transaction history (the recourse layer of §7.8); and reviews anchored to signed receipts, giving "verified purchase" a cryptographic meaning.

One boundary is inherited by every such system: PayTP evidence proves transaction facts — a quote was signed, a payment was made, a receipt was issued — never fulfillment quality. Non-delivery is not provable on the wire (the verifiable-delivery direction of §13.1 would narrow this for digital goods), so quality signal remains ordinary feedback. The evidence only anchors it to purchases that demonstrably happened.

The protocol itself will not rate anyone, and the restraint is structural rather than modest. A protocol-governed merchant score would put a discretionary judgment seat back into the PayTP payment path — the seat the end-to-end commitment removes — and protocol governance operates on auditable facts (§10.5), while a rating is a judgment. PayTP's native reputation mechanism is local and bilateral instead: negotiated limits that scale with a relationship's own signed history (Chapters 6–7); the merchant-ratable / payer-not asymmetry is §8.1's, by the same construction.

---

14 Conclusion

The web gave everyone the same way to move information: a request to any endpoint, over an open protocol no one owns. It never got the same for money. Value still moves through detours layered above the web — routed by intermediaries who set the terms, and priced so that the smallest and most frequent payments are uneconomical.

PayTP makes a payment part of the connection itself. The two endpoints settle directly, over a rail they choose. The software that carries the payment — browser, wallet, framework — is paid on the wire for doing so, by a share that governance caps and no gatekeeper sets. No deployed payment system we are aware of combines these three properties: *direct end-to-end settlement*, *bounded and evidenced trust*, and a *distribution incentive* priced into the protocol. That combination is PayTP's contribution

to the design of web payments — a novel mechanism we put forward for the community to examine and build on.

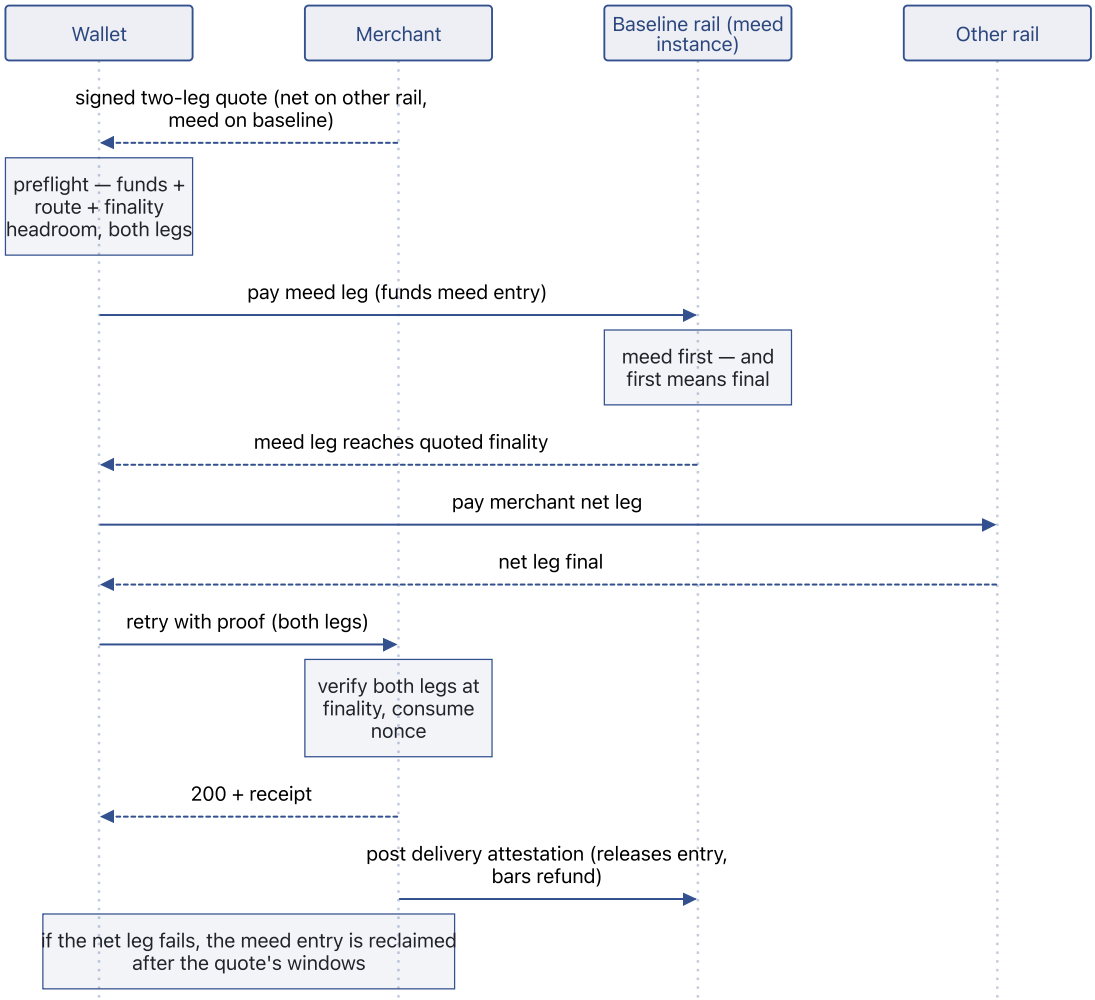
The protocol is specified here, and a first reference implementation is under active development; real-world validation remains before production use. We invite contributors to comment on the proposal, build against the specification, and report where it falls short. PayTP is designed to evolve: a minimal core now, with clearly marked extension points for the directions in Chapter 13.

PayTP is a payment layer for the web itself: open to any implementer, owned by no one, and built to pay the software that carries it — by an incentive the protocol itself sets and caps, not a gatekeeper.

Appendix A — A Worked Two-Leg Tier 0 Payment

This walks the off-baseline (two-leg) Tier 0 flow of §5.6 in full. The normative rules are in the formal specification. This is the plain-language version, for a reader who wants the whole mechanism. (A baseline offer is simpler — the payer pays one address that divides by construction — and needs no walk-through.)

A payer's agent wants a resource priced in USD, and its wallet will pay on an instant bank rail; the merchant's meed settles in the baseline asset on the baseline rail. The merchant's quote is therefore a **two-leg offer**: a net leg (the merchant's ~99%, on the bank rail) and a meed leg (the ~1%, on the baseline rail).



1. **The wallet checks it can finish before it starts.** It confirms it has funds and a route for both legs and enough time left on the quote for both to reach the finality the quote names. It sources the baseline asset itself at the quote's pinned rate — the payer never holds it. Both amounts are fixed by the signed quote — the net in the merchant's own asset, the meed at that pinned rate — so the finality wait between the legs exposes the payer to no exchange-rate drift while the quote is valid. The quote's honor window (its expiry plus finality grace), not market movement, is what bounds the wait.
2. **Meed first, and first means final.** The wallet pays the meed leg to the quote's **meed instance** — a small baseline-rail contract, its address derived from the quote (the merchant, asset, schema, meed vector, and contract version) so the wallet can re-derive and verify it. The transfer funds a **meed entry** recording the schema, vector, destinations, amount, the payer's refund pointer, and the entry's reclaim and contest windows. The entry's division among the recipients is fixed the moment it lands. If no one has deployed the instance yet, the wallet — hold-

ing the same derivation inputs — deploys it, because a reclaim window against an address with no contract would be meaningless. The wallet does **not** start the net leg until this meed leg has reached the quoted finality.

3. **Then the net leg.** The wallet pays the merchant's net amount on the agreed rail.
4. **Redemption.** The client retries with its proof. The merchant verifies both legs landed at the quoted finality — its own net on the agreed rail, and the meed entry at the instance it re-derives from its own quote — then marks the quote's nonce consumed and delivers.
5. **Closing the entry.** On delivery the merchant MUST post a signed **attestation** to the instance. That releases the recipients' shares and bars any refund forever. Posting moves no funds and names no destination. The merchant may delegate the submission, but the obligation to see a delivered entry closed rests with it — because it is the only party that both knows delivery happened and gains nothing from a refund. A delivered entry left un-attested — so that the payer could reclaim the meed after receiving the good — is merchant non-conformance, visible on the baseline rail and answerable under the certification regime (Chapter 10).

When it fails. Because the meed funds first, an interruption between the legs strands at most the meed amount, never the merchant's net (Chapter 7). A transient net-leg failure is retried under the same idempotency key. A permanent one ends in **reclaim** — the payer recovers the entry (to the refund pointer recorded at funding, and only there) after the quote's expiry, finality grace, and contest delay pass with neither attestation nor cancellation.

The windows are ordered so the merchant always has the full contest delay to attest after the last payment it must honor, so a delivered entry is never reclaimable before the merchant has had time to close it. An entry that passes every window with neither attestation nor reclaim becomes claimable by the recipients.

A merchant may also **cancel** an expired, unredeemed entry — a faster courtesy that skips the contest delay and refunds the payer's pointer at once. What bars a cancellation is a redeemable payment (a net leg that reached finality inside the honor window, which the merchant must still honor). Attestation, reclaim, and cancellation are one durable order per entry: whichever completes first settles the entry's fate.

Where reclaim applies, the wallet performs it automatically as background housekeeping (§11.1): the purchase itself already failed and fell back at redemption, so recovering the stranded meed is not a second user-facing event.

Two-leg offers are served only to clients that advertised PayTP support, and the `Vary` on the capability header keeps them out of caches shared with unaware clients. So an unaware x402 client can never be shown a two-leg quote — and never pay one leg while losing the other.

Appendix B — Design Rationale: What the Commitments Constrain

PayTP's shape is not a matter of taste. Take the three commitments as fixed — masterless and end-to-end, bounded and negotiated trust, a durable enabler incentive. Much of the architecture that follows is then not one option among many, but the point the commitments push toward — once a few hard facts about the world are admitted alongside them: round-trip latency, what middleboxes pass, the regulatory weight of issuing money, the cost of currency conversion.

It sorts the design along a gradient: the few things the commitments **hard-force**, the larger set they **strongly select** (forced given a named real-world constraint, with the condition that would relax each one stated), the parameters they leave **free but constrained**, and — granting the architecture — the machinery that proved **irreducible**. The point is not to relitigate each choice; the body does that inline. It is to show which parts of PayTP are load-bearing structure and which are calibration — where the design would bend, and where it would break.

B.1 What the Commitments Hard-Force

A **capped, governed meed carried on the wire, carved from the merchant's side** is what "a durable incentive, owned by no one" means once made concrete — and every part of that phrase is load-bearing. An incentive that lived off the wire could be repriced or withdrawn, so it would not be durable. One an owner could raise at will would be a toll, so it would not be masterless. Its incidence is fixed by the same logic: the payer has no revenue event to draw on — it earns nothing from paying — and funding the enablers by off-protocol licensing instead would make the incentive a revocable contract, reintroducing the gatekeeper masterless forbids. Merchant-side, capped, on the wire, under a charter is the one form that keeps every half — which is why the commitment states it, not merely implies it.

Bounded, evidenced exposure is likewise just what "bounded, negotiated trust" reduces to in mechanism: every exposure capped by a number a party chose, every enforceable claim resting on a checkpoint both parties signed, and the live delta since the last checkpoint itself bounded. Remove any of these and the commitment is gone — unbounded exposure is not bounded trust, and an unsigned claim is not evidence.

B.2 What the Commitments Strongly Select

Three architectural choices are not forced by the commitments alone, but become forced once a specific, hard constraint is admitted. Each is stated with that constraint and the condition under which it would flip.

Payment rides inside the connection. The alternative — a separate, out-of-band payment session — can itself be end-to-end and non-intermediated, so end-to-end alone does not decide this. What decides it is the cost of the separate session: extra round trips to resolve and open it, a merchant backend that parks the web request and waits on a second channel, the work of correlating the two, and the pull toward shared payment relays that would become chokepoints. Inside the connection, one library call does what a stateful two-channel dance would otherwise require. *Would change if* payment payloads grew too large for an ordinary connection — heavy zero-knowledge proofs, say — at which point a separate session earns its cost.

Settlement glues over existing rails, with no asset of PayTP's own. A native protocol asset would make meed collection simpler, and once held it can move end-to-end — so end-to-end alone does not rule it out. What rules it out is what a native asset drags in: users arrive holding ordinary money, so entering and leaving the asset needs liquidity gateways that sit back in the payment path and charge and screen there. And a standard that *issues* a transferable asset is a far heavier regulatory object than a messaging layer riding already-regulated rails — its validators at risk of being treated as money transmitters. Gluing over existing rails outsources money's two hardest problems — liquidity and compliance — to parties already equipped for them. *Would change if* an openly interoperable, instantly-settling digital fiat became broadly held in self-custody, dissolving the on-ramp problem a native asset otherwise creates.

The merchant may take its share on its own rail. Requiring every payment to settle wholly on the one baseline rail is genuinely simpler — it removes the entire two-

leg apparatus. But it forces every merchant to receive the baseline asset, and it imposes a currency conversion on the *whole* amount of every same-currency payment: a euro buyer paying a euro merchant would cross into the baseline asset and back, a spread paid twice on the full sum, which for ordinary domestic commerce dwarfs the roughly one-percent meed. Splitting the payment so only the small meed leg touches the baseline, while the merchant's 99% moves natively, is the price of not taxing every sale with an avoidable double conversion. *Would change if* the baseline asset became a broadly held, natively-settling money merchants were content to receive directly — the same condition under which a native asset would also become viable.

B.3 What the Commitments Leave Free

The architecture above is load-bearing. Several quantities inside it are not — they are calibration, not structure. Naming them is the honest part: advocacy hides its free parameters, analysis states them, each with the constraint it still answers to.

Which rail is the baseline. The commitments force *that* there is one mandatory, contract-capable, micro-fee rail — the guaranteed intersection any two implementations share — but not *which* one. The admissible set is narrow, not open (the selection criteria are §11.2's), but within it the choice is a decision for the implementation phase and community review, not a protocol constant. The reference implementation builds on one such rail today; that is a build choice, and the protocol is written to outlive it.

The exact fee schedule. The *structure* is forced — base roles fixed per schema, service roles only above them, the whole thing capped and merchant-side. What is free calibration is the split *among* the base roles: the 50/30/10/10 is a versioned hypothesis, rebalanced against real adoption data under governance. What is **not** a free knob is either bound. Every schema allocates exactly 100 basis points to the base roles, no more and no less, and the 150-basis-point hard cap above them is the anti-gouging promise the whole incentive rests on. Moving either bound is a charter-level act; only the division inside the 100 is tuning.

Where the merchant takes its share. The merchant's own settlement rail is a free choice among the rails a relationship can bind and finalize — its share can settle to its bank, another chain, or Lightning. Only the meed leg is pinned to the baseline.

The exposure and settlement windows. *That* exposure is bounded and evidenced is forced. The actual numbers are not. A channel's settlement thresholds, evidence

bound, and credit and deposit limits are risk parameters the two parties negotiate, and Tier 0's reclaim, contest, and finality windows are quote parameters — all constrained at the edges by liveness, dust, and rail latency, but otherwise set by the parties, and, for channels, expected to scale with a relationship's history as commercial credit does.

B.4 Why the Machinery Is Irreducible

Granting the architecture, is the machinery more intricate than it needs to be? Its two most involved parts — meed collection, and the channel-and-settlement machinery — were pressure-tested for a simpler equivalent, each candidate checked against the failure it must not reopen. Nearly every simplification that looked cleaner dropped a requirement. The ones worth recording:

- **Two settlement forms, not one.** An ordinary purchase settles as an unconditional split — merchant net and meed divided at once — while an off-baseline meed rides a conditional, meed-only record that can be attested, reclaimed, or claimed. These are two different settlement *semantics*, not one contract discriminating intent: a single program could host both, but only with distinct modes, state machines, and address derivations. And the plain-purchase path must be payable by an unaware client that supplies no PayTP-specific data at all, so its destination has to be self-describing — the division a property of where the money lands, not of a flag the payer would have to know to set.
- **Distinct prepay and postpay modes.** They look like halves of one number line, but they encode which party is the debtor, who returns a deposit at close, and which way funding flows — distinctions a bare signed range cannot recover. The unified form is either ambiguous at zero or the same two modes under one name.
- **Both parties authoring the signed record.** Letting one party alone author checkpoints would shed the tie-break for concurrent updates, but it strands the other's evidence and its ability to force a settlement it is owed, and lets a dishonest author equivocate. Both must be able to propose state.
- **A tag, not a signature, on every slice.** Replacing the cheap per-slice integrity tag plus periodic signed checkpoints with a signature on every slice is tidier on paper and puts a signature and a verification on the hottest object in the system, where a microsecond tag belongs.
- **Fresh channels, not resumption.** "Resume the same channel" stops being simpler once replay protection, key lifetime, and one-time continuity are all account-

ed for — and opening a fresh-keyed successor avoids a class of resumption bugs outright.

Two smaller simplifications *did* survive the check, and are noted for the reference implementation to adopt if it confirms them: carrying one running meed total rather than one per recipient, and folding a checkpoint-request wrapper into the checkpoint it carries. Neither changes what the protocol guarantees. Both are held so the first implementation proves the mechanism before refining its representation. That two passed and the rest did not is the point — the machinery is lean because each part was tested against the failure it exists to prevent, not because complexity was admired.

B.5 Contingent, but Constrained

Every choice here is contingent: each of the strongly-selected three names the condition that would change it, and the free parameters are labeled as such. But under today's conditions the design is not floating. The alternatives are each cleaner on one axis and worse on the ones that decide whether a payment protocol is adopted at all: latency, the cost of integrating it, who bears the fee, and whether it stays a neutral standard rather than a new gatekeeper or a regulated financial institution.

Appendix C — Dispute Resolution and Protected Checkout

This appendix elaborates the dispute-resolution direction of §13.1 into a fuller technical account of the proposed role `0x14`. It is non-normative: no part of v0.1 depends on it, and the role remains reserved for a future version.

C.1 What the Base Protocol Settles, and What It Leaves Open

PayTP settles before delivery, bounds every exposure, and makes purchase protection an explicit, opt-in purchase rather than a premium bundled into every payment (§7.3, §10.5). The evidence substrate is already present in an ordinary payment: the signed quote records what was promised and at what price, the rail record what was

paid, the receipt that the merchant acknowledged exactly this purchase, and on channels the co-signed checkpoints the accepted metered value as it accrued — accepted-payment evidence, not proof of delivery (§6.3).

Two consequences follow. A metered relationship barely has an accounting dispute at all: the payer co-signed the accepted metering as it accrued, checkpoint by checkpoint (quality remains contestable there as anywhere, §13.3). What remains contestable is the one-shot purchase, and it reduces to a single question — was it delivered as described?

C.2 The Category the Design Removes: Unauthorized Payment

Before a dispute layer decides anything, the protocol's authorization model has already removed the largest class of payment dispute. Every PayTP payment is authorized under the payer wallet's own keys and policy — a channel opens and checkpoints under the wallet's signature, a per-payment purchase under keys the wallet holds — and produces a receipt. There is no reusable credential to capture and replay, as a card number is. A payment is therefore wallet-authorized, or it does not exist.

The consequence is that the card-style "I did not authorize this" / "it was not me" chargeback has no surface here, and the buyer's-remorse claims that on card networks travel disguised as non-recognition of a charge lose that disguise. The categories that dominate card chargebacks — third-party fraud and the friendly fraud that shelters behind it — are removed from the protocol path.[25] Payer repudiation does not vanish, but it can no longer masquerade as unauthorized-payment fraud: what remains is a claim about delivery or quality, which the rest of this appendix addresses.

What replaces third-party fraud is a narrower, bounded risk that stays on the payer's own side: a compromised interaction layer requesting unwanted payments within wallet policy, or compromise of the wallet's keys. Both are governed where they belong — by the wallet's budgets, thresholds, and consent rules (§7.2) — and borne as a custody-and-policy matter, not adjudicated after the fact as a merchant chargeback. The residue a dispute layer must handle is therefore only genuine non-performance: the merchant did not deliver, or delivered something other than the quote described.

C.3 Making Delivery Provable Where the Medium Allows

For a class of goods, delivery can be made provable on the wire, so the deciding fact is never contested. For digital goods sold over the connection, the profile's quote commits to the encrypted content and to the hash of the decryption key (binding fields the base v0.1 quote does not carry). The payer verifies the ciphertext against the quote before paying. Payment releases the key, and the release is recorded as objective proof that the promised key was delivered. Non-delivery becomes provable rather than contestable — a merchant that takes payment and withholds the key is caught by the record, and under protected checkout non-release defaults to refund. What the payer cannot confirm until it holds the key is that the committed ciphertext decrypts to the good described (a valid key over a garbage payload would mimic non-delivery) — which is exactly why "it was not as described" is the one dispute that survives for this class, the narrowest there is.

For metered and API relationships, the accounting is already co-signed as it accrues (§C.1); only delivered quality stays contestable, and that the application layer governs.

For shipped goods, delivery is not an event on the connection and cannot be proven on the wire. The profile can bind a carrier reference into the receipt, so an arbiter reads a structured record rather than two competing stories — but carrier evidence is a signal to be weighed, not a proof to be trusted outright, and the physical-goods mechanisms below account for that.

C.4 The Protected-Checkout Mechanism

The reserved dispute-resolution role becomes concrete by reusing the *pattern* of Tier 0's held instance (§5.6) — funds at an immutable, quote-derived address, released by evidence within a window — as a distinct protected-checkout profile. It is not the meed entry itself: a meed instance holds a meed-only entry and divides it among the meed destinations, whereas a protected-checkout instance holds the merchant's *net* and lets the seated agent allocate it between payer and merchant. The role is named in the signed meed vector, provable to the payer's wallet before it pays, and paid its service share by the same machinery that collects every meed (§10.5). A merchant offers protected checkout on the flows where conversion demands it. A ten-cent article pays for no arbitration it does not need.

On a protected purchase the merchant's net sits at the immutable instance for a protection window named in the quote. With no dispute, the instance releases to the merchant at the window's close. On a dispute raised within the window, a registry-listed dispute agent allocates the held amount between payer and merchant under a ruling policy published with the schema: key-release proof decides digital goods, carrier evidence weighs shipped ones, and the undecidable remainder defaults to refund.

The bounds are the point. The agent can only route the held amount between the two parties. It can never take it, and once the amount is released there is no clawback. Both sides know at quote time what is held, for how long, judged by whom, and under which published rules. This is the protocol's bounded, negotiated-trust posture (Chapter 7) applied to disputes: a capped, pre-agreed exposure resolved by evidence, not an open-ended reversal right vested in an intermediary.

C.5 Physical Goods without On-Wire Proof

Shipped goods are the case where the deciding fact — did the thing arrive, in the condition described — is off the wire. Because the unauthorized-payment category is already gone (§C.2), the risk a physical protected checkout carries is genuine non-performance — non-delivery, or goods damaged, wrong, or not as described, plus later warranty or statutory claims — a far smaller and more honest class than a card network's dispute load[26], and one the mechanisms below allocate rather than eliminate. Three mechanisms, in order, bound it without an investigation apparatus.

First, the escrow window itself. Dimensioned to cover the delivery estimate plus a short inspection period, it lets "it did not arrive" and "it is not as described" surface while the funds are still held — resolved by release or refund at the instance, which allocates a bounded, pre-agreed loss between the parties (capped at the held amount) rather than the open-ended reversal a card network can impose. The dispute-timing of the medium helps: problems around delivery surface inside the window, though warranty-style defects that appear weeks later fall to off-protocol insurance, not the on-wire role.

Second, carrier evidence the profile binds into the receipt. A tracked-delivery reference is a signal the ruling policy weighs; it tilts a ruling, but does not by itself decide it.

Third, reputation as the backstop (§13.3). Rulings and receipts attach to the merchant's stable public key, so a merchant's own receipt-backed history disciplines the repeat game and lets protection be priced from verifiable record. A purchase whose value exceeds what the window, the carrier signal, and reputation together bound is served by off-protocol insurance, not by the on-wire role.

C.6 Protection as a Market

The on-wire role is a floor, not the whole of buyer protection. Wallets can sell protection on unprotected flows — refunding their user and pursuing the merchant with the signed evidence in hand. Underwriters can price merchant risk from portable receipt history. Reputation systems consume the same signed material (§13.3).

One limit is structural and inherited from the privacy design: merchant-scoped payer keys make a serial false-disputant invisible across merchants (Chapter 8). The cost of that payer privacy is that protection economics concentrate at the wallet — the one seat that sees its own user's pattern across merchants.

C.7 Why the Role Stays Opt-In and Reserved

The role is priced per flow and offered per merchant because most payments need no arbitration and should pay for none (§10.5). The agent's discretion is real but bounded, and seated only by the parties' signed opt-in: it may allocate the held amount between them under the published rules, but it takes no standing role in ordinary payments, cannot redirect the base split, and cannot claw back or seize once the amount is released. That is a far narrower authority than the card network's, which the end-to-end commitment exists to remove — discretion over one opt-in instance, not standing power over the payment path (§10.5). And the role is reserved rather than specified — implementing it extends the protocol, it does not merely configure what v0.1 ships.

The protected-checkout instance form itself (a held instance that carries the merchant's net and an agent-allocation step, which v0.1's meed-only instances do not provide), the quote and receipt fields that bind the delivery evidence, the ruling-policy vocabulary, the carrier-evidence formats, and the window semantics are all the work a future version completes.

Appendix D — The Ecosystem Case (Non-Normative)

This appendix sits outside the technical specification. Chapters 1–14 define a mechanism and take no position on whether PayTP is good for the ecosystem, or is ever adopted. This appendix argues the other question: the case for, and against, for the payments ecosystem adopting PayTP — and, more than PayTP itself, the principles beneath it, that a healthy transactional web is one where every part that creates value can sustain itself.

Two things frame the argument. First, PayTP removes the *necessity* of a rent-extracting intermediary; it does not remove anyone, and it does not decide who has a seat. The market still chooses. Second, the enabling software on the payer-side has had no *direct, protocol-native* way to be paid for carrying payments — which is not to say it is never paid.

D.1 — Why an on-wire incentive at all

A payment format is unlike a data format in the one way that decides everything. Software adopts a data standard because it improves its *own* product; a payment standard improves the *merchant's* revenue, not the carrier's — and carrying it adds counterparty risk, liability, and upkeep. So the open payment standards had no one with a reason to ship them. The meed supplies that reason, on the wire, where a sponsor cannot quietly withdraw it.

The claim is that distribution — not custody, not settlement — is the scarce thing. The software that decides whether a payment path ships at all is the bottleneck, so it earns the largest share. That is why the interaction layer is paid the most, ahead of the wallet that arguably carries more effort.

The alignment is meant to hold for every role at once:

- **Interaction layers** (browsers, agent runtimes, engines) gain a protocol-native revenue line where today they have none on payments — and for an open-source framework, a rare one.
- **Wallets** earn protocol revenue for the regulated work they already do, on top of their liquidity business.

- **OS vendors** earn by supporting the protocol and earn nothing by obstructing it.
- **The Development Fund** keeps the specification maintained, audited, and neutral.
- **Merchants** keep 99%, against 97-98% on cards, with no card-style chargeback.

For the merchant, the one percent is a purchase, not a toll. It buys distribution, the software that brings paying customers — the same job card interchange (2–3%) and app-store fees (15–30%) do, at three to thirty times the price[27] — and a feature set (channels, optional dispute resolution) that other transport-level solutions don't provide. And it buys it at a transparent price the merchant can always see: the cap is governed and every share is on the wire.

A lower headline fee is not a lower total cost: a fee buys different things on each rail. Card fees bundle fraud and dispute handling that PayTP leaves optional; app-store commissions bundle billing and distribution. The honest comparison sets what each fee actually buys side by side:

	PayTP	Card rails	App stores
Protocol / platform fee	1.0% (the meed), carved from the merchant	~2–3% + ~$0.30 fixed	15–30%
Buyer protection & disputes	Not bundled — an optional in-protocol service (up to +0.5%), or handled off-protocol	Bundled (chargebacks)	Platform-mediated
Merchant fraud / chargeback liability	None added by PayTP[28]	Merchant bears it	Platform absorbs it
Settlement speed	As fast as the chosen rail	Days; reserves can hold funds for weeks	Typically monthly
Small / machine payments	Yes — sub-cent, via channels	No — the fixed-fee floor rules them out	Limited
Who can decline the sale	The two endpoints	Networks / acquirers (by category, region)	The platform (review, bans)

Raw settlement rails — stablecoins, instant bank transfers — are cheaper still, but they are what PayTP rides on, not a merchant-facing layer: they carry none of the protection, metering, or receipts above. The closest protocol peer, x402, is compared in §D.4.2.

D.2 — Why not zero fees?

Two costs do not vanish at zero. The first is distribution: at zero fees, nothing pays the browser, wallet, or framework to carry payments — so, as D.1 argues, no one does, and the protocol stays a proposal. The second is upkeep: the trust machinery, the maintained specification, the audits and conformance tests. A zero-fee protocol funds neither from the wire, so a sponsor funds both — and a sponsor's governance follows its funder, behind a subsidy it can reprice once everyone depends on it. In early 2026 the leading hosted x402 facilitator began charging per settlement for what it had offered free.

Nor is zero always cheaper. Fee-less HTTP payments settle per request, which at metering granularity costs 10–100× a one-percent meed in rail fees.[29] Only where payments are large and rare is per-request settlement genuinely cheaper — which is why PayTP's Tier 0 speaks it natively rather than competing with it.

So the real choice is never one percent against nothing. It is a fixed, visible, charter-governed price against a subsidy whose terms can change under you.

D.3 — Adoption: why now, and who goes first

Why now, after three decades of failed attempts? Four things arrived together — and PayTP supplies the fifth:

- **Machine payers exist** — AI agents make thousands of tiny buying decisions with no human watching each one, paid for today through clumsy prepaid accounts and invoices.
- **The rails arrived** — instant bank systems, layer-2 channels, on-chain crypto, and stablecoins move money for a small fraction of card cost.[30]
- **Wallets are consolidating** — around one-tap approvals and standing budgets, the piece human payers lean on most.

- **Paying inside a web request is becoming standard** — the per-request groundwork is now Linux Foundation infrastructure.[31]
- **The incentive to carry it exists** — the meed: the direct, on-wire reason to ship a payment standard that every earlier attempt lacked.

Machine commerce can also break the usual cold-start deadlock. A new way to pay normally needs shoppers, stores, and browsers all at once, so none of them moves first. But a business paying an API funds its own wallet and pays in a single act — no consumer to win over, no browser that must ship support first. A payer and a merchant on compatible software can start between themselves. And later, the same technology powers one-tap payments for people.

D.4 — The objections, answered

D.4.1 — Doesn't the payer's software just push users onto a costlier checkout to earn its fee?#

The objection. *A merchant could ask a higher price on its PayTP path than on a plain one — $100 by plain x402, $101 by PayTP — and pass the one percent to the payer. The payer's browser, wallet, or agent earns its share whenever PayTP is used, so it has a reason to push the payer onto the pricier path. Software that puts its own fee ahead of the user it serves would be distrusted and switched off, and the worry is that this breaks adoption before it begins.*

The reply. The objection rests on there being two prices for the same goods, and that is not how selling normally works. A merchant sets a price; the buyer decides to buy; only then is a payment method used. Charging more for one method than another is the exception — in the card market it is often restricted or banned, with the EU and UK forbidding surcharges on consumer cards and much of the US limiting them. American Express costs a merchant more than Visa, yet the shelf price is the same, because the customers it brings are worth the fee. So in the ordinary case there is no pricier path to push toward — the software only chooses which rail carries the money, and the buyer pays the same either way.

When a merchant does charge more for its PayTP path, the payer's software faces a real conflict, because it earns only when PayTP is chosen. PayTP meets it directly: §10.3 requires conformant software to resolve for the payer, not for its own meed, and because both paths are advertised and signed, the payer can see and control the choice. No protocol can stop dishonest software from steering a user before it is

caught — the risk every browser already carries — but PayTP adds open terms, not concealment.

D.4.2 — Once the largest payments route around it, is what's left big enough to matter?#

The objection. *The largest financial relationships settle directly and skip the one percent, as every network loses its biggest pairs — and a fee-free path, plain x402, sits beside PayTP for the rest. So why would any payment carry the one percent instead of routing around it?*

The reply. Losing the largest flows is how a percentage fee works: above some size, payments move to a cheaper rail, just as they leave the card networks for wires. What is left is what the card networks live on, and it sustains some of the largest businesses on earth. But PayTP is not bare x402 at a markup. It carries what x402 does not — running-tab channels for metering, signed quotes and receipts, optional dispute resolution — value to payer and merchant alike. A merchant accepts the one percent to reach the buyers who want that, the way it accepts card fees. The payer's software prefers PayTP where doing so serves the payer no worse (§10.3). And because the one percent is identical at every seller, the payer pays no more and no seller is favoured. In addition, machine commerce — software paying one small request at a time — is exactly what the channels are for, and a large and growing part of the payments that stay.

D.4.3 — Why would incumbents, who already monetise payments, carry it?#

The objection. *The incentive is new revenue for challengers — a privacy browser, an open-source agent framework, a startup wallet — who earn nothing on payments today. But the dominant browsers are Chrome and Safari, which are Google and Apple, who already monetise payments far above half a percent: their own wallets, and the 15–30% they take inside their app stores. For them, fifty basis points looks like a pay cut. So the incentive proves the case for challengers and quietly assumes the incumbents will follow?*

The reply. The objection mixes two numbers. The 15–30% is an app-store commission on digital goods sold inside apps — a walled category no outside payment method reaches without acceptance by the platform. However, on the open payments PayTP actually addresses, even the largest players earn little: Apple Pay takes about 0.15% of a card transaction[32], and Google Pay earns nothing[33]. Apple and Google, running the browser, the wallet, and the OS, would earn 0.90% under

PayTP. Incumbents can still hold PayTP off their own platforms to guard a platform strategy — but not off machine commerce, servers, and independent software, where no gatekeeper stands in the path. There, the interaction-layer share goes to whatever software drives the payment; a vendor that stays out simply earns none of it.

D.5 — The case against

- **It does not reach the largest bilateral flows.** The biggest, most stable pairs settle directly and skip the meed, as payments leave every percentage rail — cards included — above a certain size. What remains however is the multi-trillion-dollar base that the card networks live on; the honest limit is that the meed's reach is bounded to that base, never the whole of global payment volume.
- **No master enforces it.** PayTP is masterless: no gate shuts out non-conformant software or a colluding pair. Collection rests on selection, evidence, and reputation, not on cryptography or a central enforcer (§10.3, §7.6). To avoid the meed, a pair need only transact outside PayTP — plain x402, cards, and direct transfers are all meed-free. They can also stay on PayTP and underpay the meed a channel accrued; the signed vector and checkpoints make that shortfall provable when the evidence is disclosed, and costly to standing — not impossible.
- **The Foundation, and its Development Fund, are a real dependency.** The Foundation sets the schema, curates the OS registry, and controls the Development Fund, which collects 10 bp of all PayTP volume — a treasury that grows large if the protocol takes off. Whoever holds that fund's keys holds real power, and "restricted to specification work" is a charter rule, not a technical lock. The marks-and-fork covenant lets the community replace a captured Foundation and redirect *future* shares, but cannot reclaim a treasury already collected: capture is made costly and escapable, not impossible.
- **Incumbents can wall it off.** A vendor that owns the browser, wallet, and OS can keep PayTP off its own platforms to protect a platform strategy. The incentive reaches machine commerce and independent software regardless, but the largest consumer surfaces are the incumbents' to withhold.
- **Enablers may take on regulatory exposure.** Earning a share of payments can draw the enabling software toward money-transmission or VASP regulation. The partner-wallet model keeps custody and its licensing in the wallet role (§10.4), but the analysis is jurisdiction-specific and unsettled.

- **Collection leans on one shared baseline rail.** Every meed executes on a common rail whose cost, throughput, and availability are a dependency the protocol does not remove — and which rail plays that role is still open (Chapter 11).
- **The incentive is visible on the wire.** The `MEED_VECTOR` and role headers name the payer-side software by its destinations — a metadata cost the meed's own auditability creates (Chapter 8).
- **The split, and the whole equilibrium, are hypotheses.** Whether 50/30/10/10 is the right division is a bet, revisable by governance within the fixed 1% base. More broadly, that the meed is collected at all rests on an economic argument no deployment has yet tested: until running code and real adoption exist, the case for it is reasoned, not proven.

Whether these costs are worth the alignment they buy is the judgment this appendix leaves to the reader. The protocol specification itself takes no position on it.

Appendix — Glossary

Term	Expansion	One-line explanation & PayTP context
PayTP	Payment Transfer Protocol	Open standard embedding payment negotiation, payment objects, and settlement evidence in ordinary TLS-protected web connections — settlement itself runs on the parties' chosen rail; the HTTP profile is v0.1 (Ch 5). The "-TP" names a transfer protocol carried over HTTP/TLS (in the lineage of HTTP, FTP, SMTP), not a new network-layer protocol.
Payment Rail / Rail	—	Any network that actually settles value (stablecoins, Lightning, instant bank systems). PayTP is rail-agnostic; one baseline rail is mandatory-to-implement for interoperability (Ch 11).
Baseline Rail	—	The mandatory-to-implement settlement rail every conformant implementation shares — contract-capable and micro-fee capable, it hosts every PayTP meed execution (Ch 5 §5.6, Ch 6 §6.4, Ch 11 §11.2).
Card Rail	—	Legacy bankcard networks (Visa, Mastercard, AmEx), charging ~2–3% plus fixed fees through gateway-mediated checkout (Ch 3).
Interchange Fee	—	Percentage that card rails skim from every transaction. The PayTP meed — exactly 1% base under schema `0x01` (Ch 10) — is a protocol-level analogue at a fraction of the price.
Tier 0 / Tier 1	—	The two payment modes: Tier 0 is the stateless per-payment profile (x402-compatible, settles before delivery); Tier 1 is payment channels with negotiated exposure windows (Ch 5–6).
Distribution Meed	—	A merchant-enabled reward paid across the software ecosystem that carries the protocol to users — the interaction layer, wallet, OS, and Development Fund — as their standing dividend;

Term	Expansion	One-line explanation & PayTP context
		carved from the merchant's side on the wire. Exactly 100 bp base, hard-capped at 150 bp, opt-in service seats above (Ch 10).
`MEED_VECTOR`	—	The list of `(role, basis points, destination)` entries signed into channel terms or the Tier 0 quote; validated once per channel or quote — slices carry no meed vector (Ch 5).
Role ID	—	One-byte tag in the `MEED_VECTOR`: `0x10` Interaction Layer, `0x11` OS, `0x12` Wallet, `0x13` Development Fund; `0x14` reserved for dispute resolution (Ch 10).
bp	Basis point	1 bp = 0.01%. The base roles hold exactly 100 bp in every schema; the hard cap is 150 bp (Ch 10 §10.1).
Value Slice	—	PayTP's ~35-byte payment message: sequence number, amount, and a Poly1305 authentication tag — not a signature, and no meed vector (Ch 5 §5.3).
Split Instance	—	An immutable baseline-rail contract at a quote-derived address; money arriving there can leave only along the quoted split. Distinct in form and address from the meed instance (Ch 5 §5.6).
Meed Instance	—	The other baseline-rail contract form: it holds meed-only entries (never a merchant share) and divides each among the meed destinations pro-rata; the funding wallet deploys it before funding an entry (Ch 5 §5.6).
Meed Entry	—	On non-baseline quotes, the reclaimable record — one per quote, keyed by its entry identifier — that the meed transfer funds at the baseline-rail meed instance: released by the merchant's attestation, refundable to the payer if the purchase permanently fails (Ch 5 §5.6).

Term	Expansion	One-line explanation & PayTP context
Registrable Domain	—	The site as a whole rather than one subdomain (example.com, not shop.example.com): payer keys are stable per merchant key and registrable domain (Ch 5 §5.5); the exact public-suffix rules are pinned by the formal specification.
Checkpoint	—	The bilateral evidence object both parties sign: balance, cumulative totals, transcript head — simultaneously the payer's receipt and the merchant's claim (Ch 6 §6.3).
TLV	Type-Length-Value	Self-describing byte layout used for all PayTP objects so unknown fields can be skipped (Ch 5 §5.2).
QUIC	—	Modern UDP transport (RFC 9000); a future transport embedding for PayTP (Ch 13 §13.2) — v0.1 rides HTTP over TLS.
TLS	Transport Layer Security	The encryption layer beneath the HTTP profile; PayTP requires TLS 1.3 or 1.2 (Ch 5 §5.1).
Budget Cap	—	A spending limit enforced by the wallet's policy engine before slices are produced (Ch 7 §7.2).
Destination Pointer	—	Where a meed share or settlement pays out — carried per flow (in `CHANNEL_AUTH` on channels, in the signed quote on Tier 0) for the payer-side roles present, pinned by schema or registry otherwise (Ch 5 §5.4, §5.6, Ch 10 §10.1).
On-Ramp / Off-Ramp	—	Service that swaps fiat ↔ crypto so users can fund or cash out wallets; a wallet-side concern outside the protocol.
Liquidity Hub (FX Hub)	—	Exchange or market maker that swaps between rails and assets so multi-currency relationships can settle; a chosen counterparty, outside the protocol (Ch 6 §6.1).
FX Spread	—	The conversion markup a liquidity hub or wallet charges; outside protocol scope (Ch 7 §7.8).

Term	Expansion	One-line explanation & PayTP context
Channel / Metering Channel	—	PayTP's metering ledger between payer and merchant: a negotiated exposure window, streaming value-slices, bilateral checkpoints, batched settlement (Ch 6). A bounded-trust metering channel — not the collateralized, on-chain-enforced payment channel of a system like Lightning; importing those assumptions misreads it.
ILP	Interledger Protocol	Packetized, multi-asset value routing (Ch 3 §3.5); a potential settlement rail.
L1 / L2	Layer-1 / Layer-2	Base blockchain and its scaling overlay; PayTP settles on whatever rail the parties chose — L2s are common candidates (Ch 3).
ALPN	Application-Layer Protocol Negotiation	TLS extension negotiating HTTP versions; PayTP rides the existing HTTP ALPNs unchanged (Ch 12).
SDK	Software Development Kit	Library for native or hybrid apps to embed PayTP without a full browser (Ch 11).
WebView	—	Embedded browser component inside apps; the embedding app is the interaction layer for the flows it drives (Ch 10 §10.1).
KYC / AML	Know-Your-Customer / Anti-Money-Laundering	Regulatory obligations attaching to whoever holds and moves funds — the wallet role (Ch 10 §10.4).
KYT	Know-Your-Transaction	API that risk-scores settlements against sanctions lists; an optional compliance layer, typically wallet-side.
Settlement Thresholds	—	`TH_value` / `TH_time`, the negotiated triggers for settlement rounds (Ch 6 §6.4).
Collusion	—	Both endpoints transacting outside PayTP to avoid the meed; not cryptographically preventable, economically bounded (Ch 7 §7.6, Ch 10 §10.3).

Term	Expansion	One-line explanation & PayTP context
IETF / W3C	Internet Engineering Task Force / World Wide Web Consortium	Standards bodies where future formalization may be pursued (Ch 13).
`PayTP-Error`	—	Header carrying PayTP error codes on 402/409 responses, including the Tier 0 meed-execution errors (Ch 5 §5.7).
TAG	(PayTP TLV `0x02`)	The Poly1305 authentication tag on every slice — an integrity tag under the channel session key, not a signature (Ch 5 §5.3, Ch 6 §6.3).
MAC	Message Authentication Code	Shared-key authenticator (e.g. Poly1305).
HKDF	—	Key derivation per RFC 5869: the session key derives from the sealed session secret, salted by the payer↔merchant relationship digest, and per-slice keys derive from it (Ch 5 §5.5).

Status of This Document

This document is a draft specification published for public review and comment; its publication establishes prior art for the mechanisms it describes.

Version numbering: This specification document is a review draft with the version given on the title page. The protocol it describes is PayTP v0.1; future protocol versions may be described in subsequent drafts.

Implementation status: A first reference implementation is under active development; the protocol is not yet production-proven or independently validated. The specification leads, and the implementation follows it.

Comments and contributions are welcome. Please direct feedback to:

Email: martin@paytp.org

Website: paytp.org

Note: PayTP was originally published as PayIP (payip.org), October 2025.

References

1. Interledger (ILP/STREAM) — packetized, asset-neutral value routing. ↩
2. Web Monetization / Coil — the streaming-payment experiment (Coil shut down 2023). ↩
3. The x402 standard, originated by Coinbase and hosted by the Linux Foundation's x402 project — x402.org; spec at github.com/x402-foundation/x402. ↩
4. Representative published online-card pricing — e.g. Stripe, 2.9% + $0.30 per successful charge. ↩
5. Card-network market structure — the Nilson Report is the standard reference for card-network and acquirer volumes. ↩
6. App-store and marketplace developer program terms — the Apple App Store and Google Play. ↩
7. Enterprise Lightning-liquidity platforms — e.g. Lightspark. ↩

8. Instant account-to-account rails — Pix (Banco Central do Brasil), UPI (NPCI), FedNow, and SEPA Instant (European Central Bank). ↩
9. LSAT / L402 — Lightning-invoice access credentials over HTTP 402, from Lightning Labs. ↩
10. Brave / BAT — the Basic Attention Token browser-native advertising and tips flow. ↩
11. "QUIC Bitcoin: Fast and Secure Peer-to-Peer Payments and Payment Channels", IEEE Future Networks World Forum (2022) — embeds Bitcoin transactions directly in the QUIC handshake. ↩
12. MPP (Machine Payments Protocol) — from Stripe and Tempo Labs; a payment-method-agnostic HTTP-authentication scheme submitted to the IETF as the draft-httpauth-payment Internet-Draft (2026). Specifications at github.com/tempoxyz/mpp-specs. ↩
13. AP2 (Agent Payments Protocol) — ap2-protocol.org, contributed to the FIDO Alliance. ↩
14. UCP (Universal Commerce Protocol) — ucp.dev; Google- and Shopify-led. ↩
15. ACP (Agentic Commerce Protocol) — Stripe agentic-commerce documentation; the Stripe/OpenAI/Meta Shared Payment Token. ↩
16. Coinbase's hosted x402 facilitator introduced per-settlement pricing on 1 Jan 2026 (first 1,000/month free, then $0.001 each) — Coinbase CDP docs. ↩
17. Independent security analyses document these exact failure modes in deployed x402 stacks — auditing official SDKs and live endpoints, they find cross-resource substitution, duplicate-settlement/replay, and authorization/binding failures: Li, Wang & Wang, "Five Attacks on x402 Agentic Payment Protocol" (2026); and Ling et al., "Free-Riding the Agentic Web: A Systematic Security Analysis of x402 Payments" (2026). ↩
18. For a systematization of agent-to-agent payment designs across discovery, authorization, execution, and accounting — and their recurring gaps (weak intent binding, payment-service decoupling) — see Zhang et al., "SoK: Blockchain Agent-to-Agent Payments" (2026). ↩
19. PayTP's cryptographic building blocks are standard, off-the-shelf primitives, with exact suites and parameters pinned by the formal specification: TLS 1.3 (RFC 8446) for the transport (TLS 1.2, RFC 5246, the floor), Ed25519 (RFC 8032) for signatures, HKDF (RFC 5869) for key derivation, Poly1305 (RFC 8439) for slice

authentication and ChaCha20-Poly1305 (RFC 8439) within HPKE for channel-secret sealing, and SHA-256 (FIPS 180-4) for hashing. ↩

20. The exposure is concrete in comparable designs: in x402 an agent embeds payment metadata — resource URLs, descriptions, and reason strings — in each request, visible to the payment server and the facilitator before settlement. See Stantchev, "Hardening x402: PII-Safe Agentic Payments via Pre-Execution Metadata Filtering" (2026). ↩

21. Candidate baseline rails — smart-contract settlement networks with regulated-stablecoin liquidity: Solana, Ethereum layer-2s Arbitrum and Base; emerging payment-specific chains from Stripe (Tempo) and Circle (Arc). ↩

22. cf. research on TEE-backed, trustless atomic service channels: Li et al., "A402: Binding Cryptocurrency Payments to Service Execution for Agentic Commerce" (2026) escrows payment and binds it to service execution via TEE-assisted adaptor signatures. ↩

23. The EU Digital Identity (EUDI) Wallet, under the eIDAS 2.0 regulation, is designed for privacy-preserving selective disclosure — e.g. proving "over 18" without revealing date of birth; see the European Commission. ↩

24. QUIC (RFC 9000) and WebTransport are the transports these future embeddings build on. ↩

25. Card chargebacks are dominated by fraud and cardholder-dispute categories — friendly (first-party) fraud now drives the majority of e-commerce disputes and has become the leading global fraud type. Card-network dispute taxonomies group disputes as fraud and consumer-dispute reason codes (Visa, Mastercard); the friendly-fraud share is tracked in industry data such as the Chargebacks911 Chargeback Field Report. ↩

26. On the scale of the card dispute load: US chargeback volume is projected near 146 million transactions / $15.3 billion in 2026, at an estimated all-in merchant cost around $110 per chargeback (Mastercard, Chargebacks911). ↩

27. Card interchange ≈ 2–3% (Stripe as a representative published rate); app-store and marketplace fees 15–30% (Apple, Google Play) — 3–30× a 1% meed. ↩

28. None added by the PayTP layer itself: with settlement before delivery, there is no chargeback mechanism. On a rail that permits later recall, the merchant bears that rail-level reversal risk exactly as it would outside PayTP (§7.8, Chapter 11). ↩

29. Per-request settlement rail costs — the cheapest current production rails (Solana ≈ $0.0001–0.001, Base ≈ $0.002–0.02 per transfer); at sub-cent-to-few-cent metering granularity a fixed per-request fee runs 10–100× a 1% meed. ↩

30. Instant account-to-account rails (Pix, UPI, FedNow, SEPA Instant) and stablecoin rails settle at a small fraction of card-rail cost. ↩

31. The x402 standard, originated by Coinbase and hosted by the Linux Foundation's x402 project — x402.org. ↩

32. Apple Pay charges the card issuer ≈0.15% of a US credit-card transaction (about $0.15 on $100), and roughly half a cent on debit; the fee falls on the issuing bank, not the merchant or cardholder. ↩

33. Google Pay takes no comparable issuer fee. After Apple's ≈0.15% arrangement, Visa and Mastercard made their card-tokenization service free and barred wallet providers from charging issuers for it, so Google — arriving after that change — earns no equivalent per-transaction cut (PYMNTS). ↩