PRIMER · REVIEW DRAFT V0.30

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

A condensed tour of the protocol — the whole idea in a short read, before the full whitepaper.

PayTP Primer: 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

This is a condensed tour of the PayTP protocol — enough to understand what it is and whether it is worth your time. PayTP is a review-draft protocol design with a reference implementation under active development, not a deployed system yet; what follows describes how it is specified to work. The full specification (~14 chapters plus appendices) carries the normative detail and the proofs. The claims sketched here are developed there in full. An early version of PayTP was first published as PayIP (payip.org) in October 2025.

The one-paragraph version

The Internet has a native way to move data and none to move value. Online payments detour through gateways layered *above* the connection — card networks, processors, platform checkouts — that add fees and delay to every transaction, and let a handful of companies decide what may be sold to whom. That architecture is a historical default, not a law of nature. PayTP moves the payment *into* the ordinary HTTPS connection that already carries the page or the API response: the merchant advertises support in standard headers, the payer's wallet answers, and the payment — its terms, its authorization, its proof — is negotiated inside the same TLS-protected exchange as the thing it pays for, then settles on whatever rail the two sides choose. It is compatible with x402[1] by design, adds no new ports, needs no changes to routers or middleboxes, and introduces no currency or token of its own. What

makes it durable rather than just another proposal is a single deliberate feature: a small, capped, protocol-defined share of each payment — the *meed* — that pays the software which carries it.

One caveat up front, since it colors everything below: whether a fixed, on-wire fee can align the web's software without any central enforcer is an economic hypothesis no deployment has yet tested. What follows is the mechanism and the honest case for and against it — not a claim that it is already proven.

1. The problem

When you pay online today, the checkout hands you off. The page passes you to a separate service — a card network, a payment provider, a platform wallet — and back again before the purchase completes. That detour has three costs that have nothing to do with the payment itself.

It prices out small and machine payments. Card rails carry a fixed fee of roughly thirty cents plus two to three percent per charge[2] — a floor that makes anything under about fifty cents cost more to collect than it earns. So the web has no good home for the commerce that is arriving fastest: software paying as it goes — an AI agent buying data one query at a time, or one online service paying another by the request — in amounts from a fraction of a cent upward.

It is slow and reversible on someone else's terms. Card settlement takes days; a rolling reserve can hold part of the money for months; chargebacks can arrive long after delivery, with the merchant carrying the fraud risk.

It concentrates discretion. A merchant's revenue passes through parties that can, and do, decline entire categories and regions from above.

Three families of remedies have been tried, and each left out at least one thing. **Decentralized currencies** removed the gatekeepers but never became a payment layer *of the web* — each stayed a per-chain island with its own wallets, no way to negotiate payment inside an ordinary web connection, and nothing that paid the web's software to carry it. **Closed-platform wallets** smoothed checkout but fragmented the web into per-platform silos and reproduced the gatekeeping. **Open web-payment standards** — HTTP reserved status code 402, "Payment Required," in the 1990s, and a lineage from Interledger to Web Monetization to x402 has tried to fill it — had the right instinct but each lacked at least one leg: relationship semantics be-

yond the single payment, settlement free of one rail, or a durable reason for the software that must ship it to carry it.[1]

None gave value transfer what the web already gives data: universality, simplicity, and endpoint autonomy — all at once, and not as a subsidy that ends when a sponsor's business model changes.

2. Three commitments

PayTP is built on three commitments. They are the spine of the whole design; everything technical below follows from them.

(1) Masterless, end-to-end. By design, 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, to be administered by a planned PayTP Foundation designed to limit its own power — with separated authority, published rules, and outside appeals. 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 settlement rail beneath — along with liquidity and regulatory compliance — is a service the two sides pick, not a gatekeeper they are forced through. The result is directness: value moving between the two endpoints, with no third party deciding whether it may proceed.

(2) Enabler incentives — the meed. Here is the part earlier open standards never solved. With no owner promoting the protocol for gain, someone still has to *carry* it — the browser, the agent framework, the wallet, the operating system. A data format like HTML ships for free because it improves the shipper's own product at no risk; a payment format does not — it carries counterparty risk, liability, and ongoing upkeep. So the open payment standards never won default support, for want of a reason to. PayTP puts the reason on the wire: the enabling software together earns a fixed **one percent** of each payment, carved from the *merchant's* side so the payer still pays exactly the listed price. It is earned only on payments that software actually carried, never charged on traffic it did not bring, and by charter no single party can raise it. That governed, capped share is the *meed*.

(3) Bounded, negotiated trust. With no network vouching between the two parties, exposure is made explicit instead of assumed. A one-off purchase settles on the rail *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 number it accepted in advance. Where a flow needs more — a refund path, a dispute handler, a guarantor — those are optional, paid roles it can add, not a premium every payment carries.

Whether putting that reason on the wire is the right call — why an incentive at all, why not simply zero fees, and the honest costs — is the ecosystem question this primer takes up in §7, and the full paper argues both sides in Appendix D.

3. How it works on the wire

PayTP adds three things to an ordinary HTTPS connection: a short negotiation in standard headers, a stateless flow for one-off purchases, and a channel for high-frequency relationships. Everything travels inside the TLS-protected connection, so to anything that does not speak PayTP the exchange looks like ordinary HTTPS. If a site does not support PayTP, nothing breaks — you get the normal checkout.

The payer side is deliberately split into two parts with different powers. The **wallet** holds all the keys and enforces the spending policy; the **interaction layer** — the browser, agent framework, or app actually driving the requests — initiates and mediates payments but can never mint payments, extract keys, or spend outside wallet policy. The **merchant** is the value-receiving endpoint, identified by a stable key bound to the TLS certificate the client already verified. And the payer's identity is *merchant-scoped* — the wallet derives a distinct key per merchant, so there is no cross-site identifier to track you by. A one-off purchase needs no PayTP identity at all.

Tier 0 — paying per payment. For a first contact or a one-shot purchase, PayTP is stateless and settles before delivery:

1. The client requests a paid resource, advertising PayTP support.
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 and retries with the proof. On the common baseline rail — the one settlement rail every PayTP implementation supports — the payment goes to a small splitting contract that divides the meed from the merchant's share by construction. On any other rail — Lightning, an instant bank transfer — the mer-

chant's net settles there, while the meed (around one percent) settles as a separate small leg on the baseline rail — funded first, so a failure there never strands the merchant's payment (the full spec details the ordering that leaves neither side hanging). The payer never touches the baseline asset: its wallet sources and settles that meed leg automatically, at the rate fixed in the quote. Carrying the meed on one shared rail — whatever rail the parties pick for the rest — is what lets any two parties settle on any rail while every meed recipient still watches just one place for its income.

4. The merchant verifies that settlement (meed included) is complete, serves the resource, and returns a **signed receipt**. Proofs are single-use, so a lost response cannot double-charge.

Tier 1 — running a channel. When a relationship has frequency — an agent metering an API thousands of times an hour, a reader's regular news site — the two parties open a **channel**. This is the running tab from commitment (3), made concrete: a metering ledger with a negotiated exposure window, in which each payment becomes a compact **value-slice** of about 35 bytes 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 (each side's total exposure cap), the *evidence bound* (how much value may accumulate un-checkpointed before both sides must sign a fresh checkpoint), the settlement thresholds, and the *meed vector* (which software roles are owed what) — and the merchant countersigns.
- **Streaming.** Value-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, much like a TCP receive window.
- **Checkpoint.** At each interval the two sides exchange one statement signed by both — at once the payer's receipt and the merchant's claim — so the amount riding on unsigned trust never exceeds the agreed bound.
- **Settlement.** The debtor settles the accumulated balance on the chosen rail, with proofs covering the creditor's payout and that round's meed. A counterparty treats a settlement that omits the meed as incomplete: the channel closes and the shortfall stands in signed evidence.
- **Close.** Any unused deposit returns — unless a successor channel chains from the final checkpoint and carries it forward, which is how a running tab outlives any single connection.

Settlement itself is rail-agnostic and pluggable: stablecoins, on-chain crypto, Lightning, instant bank rails — whatever the two sides both accept — with the common baseline rail as the guaranteed fallback, so any two parties always have a rail in common. (Which specific rail fills that role is an open decision for the implementation phase and community review; the reference implementation targets Solana as a build choice, not part of the protocol — and its settlement runs against a simulated rail today, not a live adapter.)[3]

At a glance:

	Tier 0 — per payment	Tier 1 — channel
Best for	one-off purchase, first contact	recurring, high-frequency
State	stateless	a metering ledger (the tab)
Per-payment message	full payment + proof	~35-byte value-slice
Settlement	on the rail, before delivery	batched, at negotiated thresholds
Max exposure	a single payment	the signed window or deposit

4. The meed — the distribution split

The meed pays for **distribution**, and distribution — not custody or settlement — is the scarce input. Rails, liquidity, and key custody are competitive commodity services with many suppliers; what is genuinely scarce is a reason for the software that *decides whether a payment path ships at all* — the browser, the agent framework — to carry an open standard, since one that nothing chooses to carry is dead however good the rails beneath it. That software, unlike a merchant-side plugin that already bills its own customer, has no other way to be paid for carrying payments. So the meed is carved from the merchant as a cost of reaching customers, not charged to the payer, and the layer that controls whether payment happens at all earns the most; the full case, with its objections, is §7.

Under the initial schema, one percent of each payment — 100 basis points — is split across the software roles that made the payment possible, and the merchant keeps

the other **99%**:

```
Interaction layer (browser / agent framework / app):  50 bp
Wallet (key custody, settlement, compliance):          30 bp
Operating system:                                      10 bp
Development Fund (spec, audits, conformance):          10 bp
                                                      ------
                                                      100 bp   (1.0%)
```

Four properties define the split:

- **It never touches the payer.** The meed comes out of the merchant's side; the shopper pays the listed price. (The rail's own small transfer fee and any currency conversion are separate, as on any rail.)
- **It is fixed, not negotiated.** Every payment under a schema carries the same shares — merchants cannot vary it by user, and every role can forecast its income. Predictability is the point; payment standards succeed through it.
- **It is hard-capped.** Above the 100 bp base, the protocol reserves a further **50 bp (0.5%)** at most, exclusively for *opt-in* service roles a merchant chooses to add — escrow, dispute resolution — and most payments carry none of it. By charter, no coalition can shift the base toward its own members, and changing a schema takes broad ecosystem consensus.
- **Gaming it gains nothing.** The OS's and Development Fund's shares are pinned by schema and registry, so a payer on custom software cannot redirect them to its own address. And because the merchant pays the same 100 bp on every payment — whichever roles are claimed — misreporting moves value to no one and never becomes a discount to the payer: a share for an absent role routes to its pinned fallback (for the OS, an independent open-source fund the Foundation does not control), never to whoever misreported. Approving or denying an OS recipient likewise changes the Foundation's own income by exactly zero — exclusion is never revenue.

Concretely: a reader who spends €10 across their news sites in a month has the publishers collectively receive €9.90, with €0.05 to the browser, €0.03 to the wallet, €0.01 to the OS, and €0.01 to the Development Fund — every party that made the payment possible paid automatically, with no contract between any of them.

5. Where PayTP sits in the landscape

PayTP is not trying to replace payment rails; it is the neutral layer that lets any of them run through the web connections people already trust. Measured against the three commitments, the existing systems each miss at least one:

System	Masterless, end-to-end	Bounded, negotiated trust	Enabler incentives
Card rails	No	No	No
Platform wallets	No	No	No
On-chain / layer 2	Yes (but chain-siloed)	Partial	No
Instant bank rails	No	No	No
x402	Yes	Partial (per-request)	No
MPP	Yes	Partial (per-request)	No

The most important relationship is with **x402**, the HTTP-payments standard now hosted by the Linux Foundation — and it is kinship, not rivalry. PayTP's per-payment tier *is* an x402 flow carrying a `paytp` extension: the merchant keeps its x402 stack 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. Where x402 is ahead — standardization, production deployments, facilitator infrastructure — it is ahead decisively, and PayTP does not try to beat it there.

A newer peer is emerging alongside it: **MPP (the Machine Payments Protocol)**, from Stripe and Tempo Labs — an in-band-402 design submitted to the IETF as a standards-track Internet-Draft. Its core is stricter than x402's — single-use, a receipt header, and on-chain challenge-binding on its recommended EVM method — but it is fee-neutral in the same way, with no distribution meed. PayTP relates to MPP exactly as it does to x402, and a second PayTP embedding over MPP is planned work.[4]

What PayTP commits to, on the wire and under governance, are three things x402's core leaves to the market around it: a durable, capped **distribution meed** owed to the software that enables payment (rather than off-wire sponsor funding that can be repriced); a single standard **relationship model** for recurring commerce (the tab, described the same way whatever rail settles beneath); and a **hardened exchange**: each payment is bound to its signed quote, and delivery is held until the merchant itself confirms settlement. x402 *can* provide both of these — quote-binding and held delivery — through optional extensions and scheme practice; PayTP's contribution is simply to make them mandatory in its own profile rather than leave them to each deployment.[5] This is a compatibility statement, not a critique: making these mandatory on the wire and under a charter is what makes them dependable for software deciding whether to ship a payment path at all.

6. What is guaranteed — and what is not

PayTP is honest about its limits, and it is worth stating them plainly. PayTP does **not** claim trustlessness. What it offers is bounded-trust, evidence-backed streaming settlement:

- Each party's maximum loss is a number it chose — the merchant's credit limit, the payer's deposit, or a single payment in Tier 0.
- Everything a dispute could turn on is signed — channel terms and checkpoints by both parties, quotes and receipts by the merchant. What has not yet reached a signature is capped by the agreed evidence bound.
- Because payments are negotiated in-band, the connection's standard TLS keeps network attackers and other web contexts from forging or observing the payment objects, and single-use proofs stop a captured payment from being replayed; settlement visibility then follows the chosen rail.
- Unsettled balances are real, bounded counterparty exposure, and no settlement rail enforces PayTP receipts. Settled value has exactly the finality the chosen rail provides — no more, no less.

7. The ecosystem case

Everything above defines the mechanism; it works whatever you conclude here. This section argues the separate question — whether PayTP is worth adopting — and

gives both sides. The full paper makes the case, with every objection, in Appendix D; here is the short version.

Why not zero fees, and why now. A zero-fee protocol doesn't escape the cost of carrying and maintaining a payment standard — it moves that cost off the wire to a sponsor, whose governance follows its funder and whose terms can be repriced once everyone depends on them (in early 2026 the leading hosted x402 facilitator began charging per settlement for what it had offered free[6]). The wager is that a fixed, visible, charter-governed one percent beats a subsidy that can change under you — and it buys the same distribution the card networks (2–3%) and app stores (15–30%) sell at three to thirty times the price.[2] As for timing: after three decades of failed attempts, four things finally arrived together — machine payers making tiny buys unattended, cheap instant rails,[3] consolidating one-tap wallets, and paying-inside-a-request becoming Linux Foundation infrastructure[1] — and machine commerce breaks the old cold-start deadlock, since a business paying an API funds its own wallet and pays in one act, with no consumer to win over first.

The honest case against. The design has real costs, and the paper names them:

- It doesn't reach the largest flows — the biggest bilateral pairs settle directly and skip the meed, as on every rail — but what remains is the vast multi-trillion-dollar base the card networks live on.
- No master enforces the meed — and none needs to. To avoid it, you just don't use PayTP: plain x402, cards, and direct transfers are all meed-free. On PayTP, an honest counterparty already treats a meed-omitting settlement as incomplete and closes the channel (§3); underpaying it instead takes both parties colluding on non-conformant software — and even then the shortfall stands in signed evidence, costing them standing under the conformance regime and the reputation their signed history carries.
- The Foundation that stewards the meed (funded by 10 bp of all volume) is a real concentration point — made escapable by a covenant that lets any community fork the governance and keep its payments, not eliminated.

And the whole equilibrium is, honestly, an economic hypothesis no deployment has yet tested. Whether these costs are worth the alignment they buy is left to the reader; Appendix D makes both sides of the case.

8. Where to go next

This is a review-draft specification. The initial protocol is designed and a first reference implementation is under active development; real-world validation remains before broad production deployment. Nothing here is a claim about a shipped product.

If this sketch made the shape of the thing click, there are two ways to go deeper. Try the reference-implementation **demos** to watch a value-slice, a channel, and a settlement actually move. And read the **full specification** — Chapter 4 is the map and the right place to start, Chapters 5–7 are the technical core, Chapter 10 defines the meed schema in full, Appendix B explains which parts of the design the three commitments *force* and which are free calibration, and **Appendix D** makes the ecosystem case — for and against — from outside the spec.

Comments and feedback are welcome at martin@paytp.org. The ambition is a web where paying is carried inside the ordinary connection — value moving directly between two endpoints, at any size, with no gateway deciding whether it may.

References

1. The x402 standard, originated by Coinbase and hosted by the Linux Foundation's x402 project — x402.org. Its predecessors in the HTTP-402 lineage include Interledger and Web Monetization / Coil (shut down 2023). ↩
2. Representative published online-card pricing — e.g. Stripe, 2.9% + $0.30 per successful charge. App-store and marketplace fees run 15–30% (Apple, Google Play) — evidence that merchants will pay substantially for delivered customers. ↩
3. Instant account-to-account rails (Pix, UPI, FedNow, SEPA Instant) and stablecoin rails (e.g. Solana, Base) settle at a small fraction of card-rail cost. ↩
4. 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. ↩
5. Why make these bindings mandatory rather than optional: independent analysis of deployed HTTP-payment stacks finds that where quote-binding and single-use

settlement are left to each implementer, some omit them — letting a payment buy the wrong resource, or one authorization be redeemed twice. Li, Wang & Wang, "Five Attacks on x402 Agentic Payment Protocol" (2026); Ling et al., "Free-Riding the Agentic Web: A Systematic Security Analysis of x402 Payments" (2026). ↩

6. Coinbase's hosted x402 facilitator introduced per-settlement pricing on 1 January 2026 (first 1,000/month free, then $0.001 each) — Coinbase CDP docs. ↩