x402-flash Design Doc

Supporting x402 micropayments with zero settlement latency overhead

Abstract

This document outlines an extension to Coinbase's x402 standard by introducing a new scheme `flash`. The scheme allows an API server to respond to requests containing a x402 payment payload without needing to wait for transaction settlement, while guaranteeing eventual payment by requiring the client to lock funds in an escrow contract. The API server that is implementing this scheme will keep track of the total value of transactions that are pending settlement, and can safely fulfill client requests so long as that value is at most smaller than the value in escrow.

Background

x402 specifies a standard for using the dormant `402 Payment Required` HTTP status code to attach cryptocurrency payments to HTTP requests. This enables a wide variety of applications including micropayments and pay-per-use APIs.

While x402 technically supports different payment schemes, the only scheme currently in use is the `exact` scheme, where the client transfers the exact amount of the requested payment for each request. This means that the API cannot respond to the request until the corresponding payment transaction is settled on the underlying blockchain without risking non-payment. The current de facto settlement layer for x402 is the Coinbase Developer Platform's Facilitator service, which boasts 200ms settlement times. However, a latency cost of 200ms (which also does not account for additional time on the wire due to round trips from the facilitator) is unsuitable for many critical use cases such as high frequency trading and data streaming.

In order to support these high-value use cases, a new x402 scheme that does not require blocking on transaction settlement is required.

Design

Overview

The `flash` scheme involves the usage of a Settlement Contract that will proxy transfers of the payment token. This smart contract has two key responsibilities:

1. Hold funds in an escrow account for the client and server. The client that wishes to send x402 requests to the server must lock up some token balance proportional to

the desired transaction throughput prior to sending requests to the server using the `flash` scheme.

2. Proxy the token transfers from the client to the server to settle the payment. If the token transfer fails, then the smart contract will instead withdraw the payment amount from escrow and transfer it to the server's wallet.

This escrow account serves as a buffer that ensures that the server can receive payment for fulfilled requests, even in the case the client wallet doesn't have enough remaining token balance. As a result, the server can safely fulfill requests without needing to wait for the transaction to settle on the blockchain.

Opening escrow

WIP

Request flow

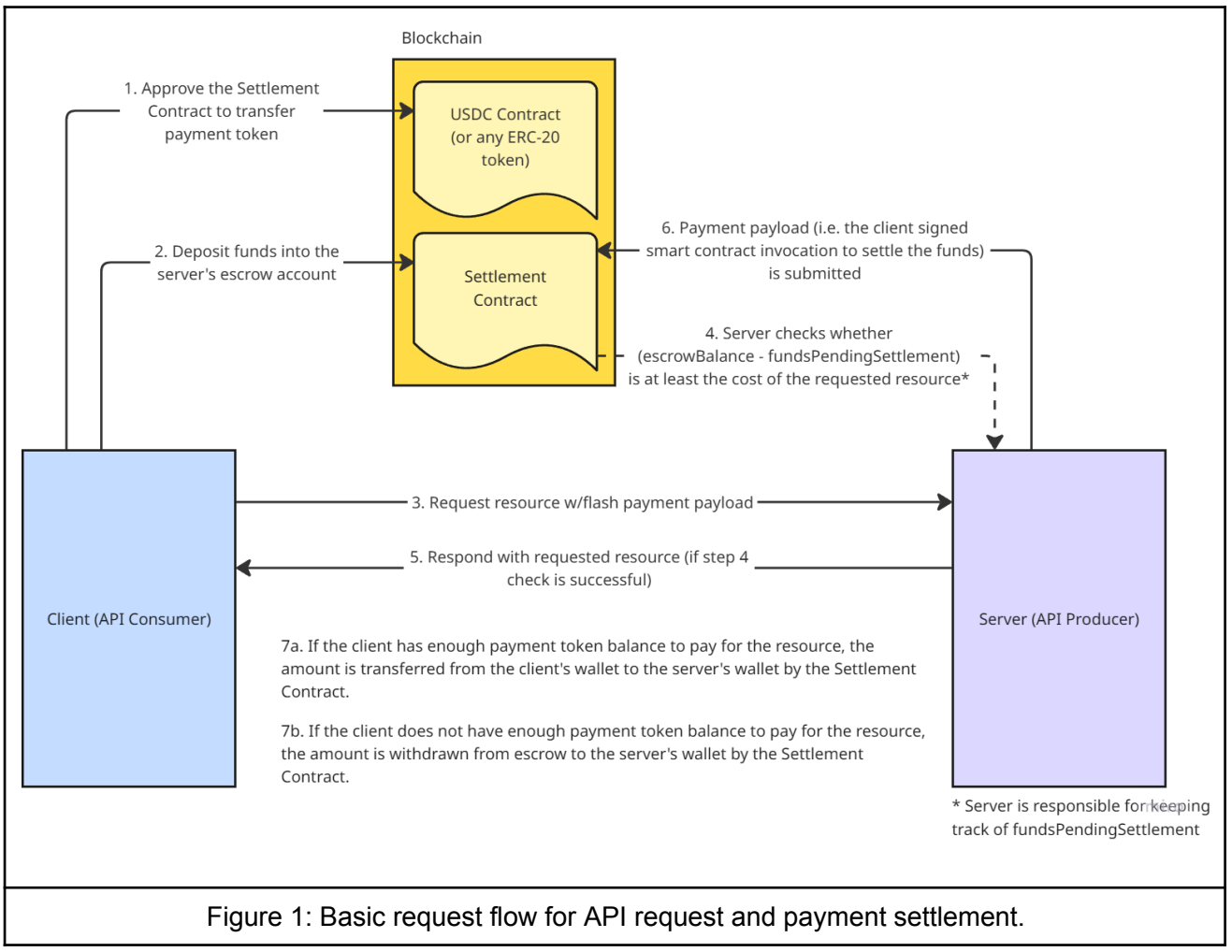

Figure 1: Basic request flow for API request and payment settlement.

Closing escrow

WIP

Settlement contract interface

```
// SPDX-License-Identifier: GPL-3.0
pragma ...;

interface SettlementContract {
    // Called by the client* to setup an escrow balance with the
    // specified server address.
    function openEscrow(address server, address token,
                        uint256 amount, uint64 ttlSeconds) external;

    // Returns the current escrow balance for the client/server pair.
    function openEscrow(address client,
                        address server) public returns(uint256);

    // Called by the client* to transfer the payment amount to the
    // server's wallet.
    function settlePayment(address server, address token,
                           uint256 amount) external;

    // Called by the client* to indicate that they are done sending
    // requests to the server.
    function clientCloseEscrow(address server) external;

    // Called by the server to indicate that the server is done
    // fulfilling API requests from the client (i.e. they are no longer
    // waiting for any funds to be settled.
    function serverCloseEscrow(address client) external;
}
```

Schema

WIP

[x402 type specifications]

```
{
  ...

  // Specify the new ‘flash’ scheme
  scheme: "flash";

  extra: {
    settlementContractAddress: string;
  };
}
```

```
{
  ...

  // For 'flash' scheme payments, the client must provide a signed
  // invocation of the settlePayment method of the Settlement Contract
  payload: <signed tx>;
}
```