Clocktower V1

An EVM-compliant Protocol for Reccurring, Timely Payments

Hugo Marx

November 2023

Abstract

Clocktower is an Ethereum Virtual Machine (EVM)-based, decentralized protocol for recurrent payments. Parties who wish to exchange goods and/or services provide payment details off-chain for an initial set-up and third-party agents are financially incentivized to execute payments from master list of transactions. In this way, Clocktower allows for scheduled payments to be reliably processed in the future *without a central processor*. Use-cases would include a variety of financial and commercial possibilities such as mortgage, rent, and bond payments, estate planning, subscriptions for goods and services and many others. This whitepaper will detail the problems solved by Clocktower and a technical overview of the protocol.

The Three Problems

Clocktower solves three problems:

1) Incentivized Polling

Smart contracts have inherent limitations. One of the more important for recurrent payments is that smart contracts cannot actually be aware of time unless acted upon by an externally owned account (EOA). The contract is like a person who is wearing a watch but can only look at it when instructed to do so. This limitation makes standard computer scheduling, such as with a cron jobs, infeasible. Without the ability to schedule transactions in the future, common financial services like payroll and subscriptions are not possible in current decentralized systems.

Clocktower employs a novel incentivized polling mechanism in order to ensure proper execution of future transactions. Each recurrent payment or subscription contributes a small amount to a fee balance for the account. This fee becomes the ongoing reward to those who instruct the contract to check the time (ie, Callers), which occurs through the contract *remit* function. The fee is set at 1% of the payment amount for Clocktower V1. The Caller recieves payment in proportion to the number of payments made as compensation for the effort and the gas costs for the transactions.

The protocol is built for efficiency and to avoid stalling, mainly through minimizing gas costs. All payments are made in-kind to avoid the complexity and gas expenses of DEX conversions. A token minimum is also set by the contract such that the number of payments attempted per remit remains as low as possible.

2) Time Conversion

Humans and computers use different systems of time. Computers mostly use a more simple system called Unix Time, in which a number increments every second past midnight from January 1st, 1970. Unfortunately, the natural world does not increment as cleanly–a full revolution of the Earth around

the sun is actually the equivalent of 365.24 days and so every 4 years, an extra day (ie, leap day) is added to the modern Gregorian calendar on February 29.

Moreover, scheduled recurrent payments are an open-ended time series and one can't easily save an open-ended series with an incrementing number. For instance there's no sane way to save the date "every 5th day of the month" using just unix time. One could theoretically save a long series of calculated dates in seconds after midnight January 1st 1970. But this is inefficient and creates an edge case if the subscription goes longer than the initial series of numbers. A much simpler solution is to use a Gregorian calendar point.

So for each type of frequency of subscription we simply create a range of numbers:

Frequency	Range
Weekly	1 - 7
Monthly	1 - 28
Quarterly	1 - 90
Yearly	1 - 365

These ranges then need to be converted to unix time. The best way to translate the block timestamp incremeting unix number to Gregorian ranges is to use an intermediary incrementing day standard called Julian Days. Using the following code, the Clocktower contract can perform this translation without Oracles or any other external sources,[12].

```
function unixToTime(uint unix) internal pure returns (Time memory time) {

    uint _days = unix/86400;
    uint16 day;
    uint16 yearDay;

    int __days = int(_days);

    int L = __days + 68569 + 2440588;
    int N = 4 * L / 146097;
    L = L - (146097 * N + 3) / 4;
    int _year = 4000 * (L + 1) / 1461001;
    L = L - 1461 * _year / 4 + 31;
    int _month = 80 * L / 2447;
    int _day = L - 2447 * _month / 80;
    L = _month / 11;
    _month = _month + 2 - 12 * L;
    _year = 100 * (N - 49) + _year + L;

    uint uintyear = uint(_year);
    uint month = uint(_month);
    uint uintday = uint(_day);

    day = uint16(uintday);
    ...
```

3) Tri-party edge cases

As previously mentioned, the Clocktower system requires a three-party incentivised polling system for it's decentralized recurrent payments. The three user categories are named for their functions in

a subscription service, although the protocol has many potential use-cases outside of this model. At its core, Clocktower is a series of functions that allow two parties, a Subscriber and a Provider, to orchestrate recurrent payments for a service or good with the help of a third party, an incentivized polling agent referred to as the Caller. While not immediately apparent, the permissionless nature of Clocktower creates gameable edge-cases when a user could play two roles at the same time, and potentially extract value from the third party.

The most immediate question of the system operation is: who pays the fees for the Caller? One might initially think that the provider should pay a portion of their earnings for this purpose. This seems simpler since there is only one provider and they are the one's receiving all the funds for a given subscription. However, there is an attack vector if we use this structure. Since we have no way of knowing the ownership of an ethereum address it is possible that combinations of the subscriber, provider and caller can be the same entity.

If the Subscriber and Caller are the same while the Provider pays the fee, the subscriber could sign up for a subscription and but not fund their account. When the transaction fails the fee would still be extracted from the Provider and paid to the Caller (who is also the Subscriber.) If the fee is higher than the gas costs of signing up for the subscription then the malicious subscriber now has a method of extracting tokens from the provider.

Proration and refunds are also important considerations. Without proration, Providers would be given more than deserved for most initial payments from Subscribers (ie, anyone who doesn't sign up on the exact day of a new cycle). As such, the contract makes a proration calculation with each new signup.

Refunding after a cancellation has also been carefully considered. A given subscription can be terminated unilaterally either by the Subscriber or the Provider. The general rule here is to refund any extra funds to the party *not* doing the cancelling. This eliminates the potential for one party to benefit monetarily from cancelling the service.

The Three Users

This section explores the main actors and lifecycle of the protocol, which can be modeled as a three phase process.
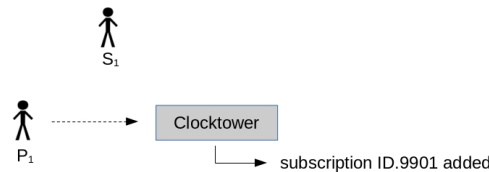
Fig I - Creation

Figure 1: Creation

I) Creation In the creation phase (see Figure 1) a Provider configures a good or service they would like to provide at a fixed interval (weekly, monthly, yearly, etc). This can be done through direct interaction with the contract or, in most cases, through a website providing a simple user interface. Regardless, this process involves a Provider making a function call to the Clocktower contract,

specifying parameters of the subscription includiong the amount of the payment, ERC20 token(s) accepted, description/details of the subscription to be saved in call data, the payment interval, and the due date of the payment.

```
function createSubscription(uint amount, address token, Details calldata details, Frequency frequency
    ...
```

This function also sets a number of validation and anti-griefing parameters before moving forward with subscription creation. A subscription ID is then generated and added to the subscription index of the contract. A subscription can also be edited or destroyed through related contract functions called by the same Provider.

[INSERT FIGURE 1 CONTRACT CREATION DIAGRAM]

II) Initiation After the Provider creates the subscription, the good or service is now available to anyone who would like to set-up recurrent payments. Off-chain, the Provider advertises the service to potential Subscribers who can sign-up via link (A). Again, either through direct interaction with the contract via scripts or more likely, a web portal, a potential Subscriber will make two transactions (B). The first calls the *approve* function to the appropriate ERC-20 contract, which allows the contract to make future draws of the token from the specified EOA. The next transaction will call *subscribe*, which takes the Subscription struct parameters. The contract then makes a number of validation checks, most importantly that there is proper allowance and that there is enough of the token to cover the subscription amount. If valid, the Subscriber is added to the contract index (C)for the EOA and the first payment is made to fill the fee balance. A proration calculation ensures that the Subscriber does not overpay based on the day of the cycle that he signs up:

```
//prorates fee amount

if(subscription.frequency == Frequency.MONTHLY || subscription.frequency == Frequency.WEEKLY){
    fee = prorate(block.timestamp, subscription.dueDay, fee, uint8(subscription.frequency));
        }
else if(subscription.frequency == Frequency.QUARTERLY) {
    fee = prorate(block.timestamp, subscription.dueDay, fee, uint8(subscription.frequency));
    fee /= 3;
    multiple = 2;
}
else if(subscriptions.frequency == Frequency.YEARLY) {
    fee = prorate(block.timestamp, subscription.dueDay, fee, uint8(subscription.frequency));
    fee /= 12;
    multiple = 11;
}
```

[INSERT FIGURE 2 SUBSCRIBER INITIATION DIAGRAM]

III) Incrementation After the initial transactions have occurred, there are no further requirements on the Subscriber, other than keeping his EOA balance sufficient to cover the costs of the recurrent payments. An incentivized polling agent known as a Caller, is key to the extension of recurrent payments into the future. The Caller role is simple: call a single function, *remit* , on the Clocktower contract (A). This is the mechanism through which the contract becomes 'time aware.' *remit* calculates the current day (the most atomic unit of a recurrent payment in Clocktower_V1) and confirms that the contract subscription index has not yet been checked on this day. If it has already been checked, the function terminates and an error code is returned. If the index has not yet been checked, the contract loops through all subscriptions to see if any payment is due. Those payments that are due and have sufficient fee balances, are remitted to the appropriate

Provider (C) and the fee is paid to the Caller (B).

There are a few other important parts of the *remit* function. First, in cases where a subscriber does not have sufficient fee balance to cover the full fee for the recurrent payment, the contract performs a special transaction where it refills the fee balance, much as it did when the Subscriber first signed-up for the service, in the Initiation phase. This feefill will need to occur regularly during the life of the recurrent payment, depending on the frequency of the subscription. Importantly, in the event that a Subscriber's balance has fallen below the level of the recurrent payment amount but can still cover the fee and/or feebalance, the contract will continue to perform these actions–this ensures that the Caller is not penalized for Subscribers with low balances.

If a Subscriber's balance then falls below the level of the subscription fee, *remit* goes into failure mode. The Subscriber is automatically removed from the subscription list and the remainder of the fee is sent to the Provider of the subscription.

3) Fees and Refunds

To keep fees and refunds fair, the Clocktower uses a system of proration. The first payment in a subscription fills the fee balance which can then be used to pay fees through future payments.

```
A) Proration. Show calcs

   --- first payment holdback

B) The three users. Examples of exploits
   -- fee balances
   -- contract as source of dynamic refund policy

C) Conditional Refunds. Show table
```

Lifecycle

— note about use of subscription terms but can be applicable to any sort of regular payment. Use of Provider, Subscriber, Caller. But in a different context could be Payer, Receiver.

Goals

Decentralization
While centralized payment facilitators are the norm, they bring with them a myriad of problems, from high fees to censorship. However, with a decentralized blockchain contract there is no point of failure, no censorship and no arbitraty gatekeeping.

Immutability
A system that cannot be changed is a system that cannot be censored.

Easy of Use
In the past setting up your own subscription service has been too difficult for normal users. But with Clocktower, if you can use a decentralized app, you can create or join a subscription.

Inexpensive
Traditional payment networks have had the advantage of being able to charge high fees due to sizable fixed costs of network build-out. By building on the existing backbone of EVM compliant blockchains, we believe we can eventually undercut the existing networks.

No Oracles
An oracle is a third party data source. Unfortunately, oracles have been manipulated to steal funds. This protocol will not use these data sources as they subject users to unnecessary risk.

Minimum tokens in contract
Hackers hack where the money is kept. Traditional contracts have become targets largely because they rely on the "vault" model where all value is stored within the contract. We seek to turn this model on its head by seeking to hold as little value as possible in the contract. This makes the contract less of a target and allows users to keep their own funds secure in their own wallets.

No protocol token
We believe a protocol should never need its own token to work. A token needed for functionality creates friction for the user when they have to convert it and can lead to inflationary tokenomics. If Clocktower ever issues its own token, it will be used soley for governance purposes.

Your Privacy is Your Business As a simple protocol operating on fully transparent blockchains, Clocktower does not attempt to create privacy and instead leaves this responsibility to the parties involved in the transactions.

1. bokkypoobah (2019). BokkyPooBahsDateTimeLibrary. https://github.com/bokkypoobah/BokkyPooBahsDateTimeLibrary.

2. Navy, U.S. Converting between julian dates and gregorian calendar dates. https://aa.usno.navy.mil/faq/JD_formula.