Set: Hub-and-Spoke Cryptographic Payment Channels

v0.0.1

Nathan Ginnever: nathan@finalitylabs.io

Abstract

Here we outline the Set-Payment channel protocol (In a later paper we incrementally extend Set to general state over virtual channels). Set is built upon insights from other “layer2” payment scalability solutions for decentralized cryptocurrency ledgers, namely, Lightning Network[2], Raiden[5], state-channels (Machinomy_v2[4], Jeff Coleman[6], etc) and Perun[3] virtual channels. This paper explores a variant of Perun[3] that is built for human interactions (tipping, micropayments for online service, gambling etc) in a payment channel facilitated by a hub. Each connection to the hub is capable of opening many concurrent payment channels with any other peer to the hub without any on-chain transaction costs.

Introduction

Perun[3] virtual channels is a commitment based approach to the Lightning/Raiden network hash-locked transactions when considering a hub and spoke payment network (see Perun Networks[7] for their implementation of routing, we do not explore routing in Set). The main difference provided by the Perun[3] research group over hash-locked hubs is the ability for the intermediary to "drop out" of the balance updates between the parties they are escrowing funds for. The hub is not required to witness the balance updates between the parties since the hub is only concerned with the opening and final states of the channel it is escrowing funds for. This improves bandwidth requirements, privacy, and generally simplifies the payment channel model offered by hash-locked transactions. However, Perun[3] style virtual channels require that the channel may only live for a short period of time. More specifically, all parties involved in a single hop payment must agree to how long they have to make payments. Set channels waives this requirement to allow any party to choose when to close a virtual channel.

Summary of Our Contribution and Its Applications

The main contribution offered by Set-Payment channels is the elimination of the requirement that virtual channels have a validity timeout and must therefore require clients to track when a virtual channel must be closed. In Perun[3], you may not close a virtual channel before or after a validity time, it needs be closed at that time for the system to continue, while Set channels may be closed at anytime both in the happy and byzantine cases. We consider this a different approach that allows for different trade-offs and a system of short lived channels may be more beneficial for machine-to-machine applications (where a protocol is defined in the client for opening/closing channels automatically. This assumes harder liveness requirements than in more human interactive systems). Some applications may benefit from the ability to trust that the intermediary will allow for a payment channel to be open as long as they can in order to collect fees. We also reduce the number of messages required to close a virtual channel in the happy case (todo: do a comparison of protocols to justify this claim) and provide less network traffic to the hub since channels may live forever (or until either Alice, Bob, or Ingrid signal to close). To maintain consistency with Perun research we will extend the same notation. Again assume that Alice and Bob both have "*Ledger Channels*" (lc) opened on the Ethereum[1] (or other efficiently complex stack based) blockchain with an intermediary hub Ingrid. We will construct "*Virtual Channels*" (vc) over these lc channels in the same way as Perun[3] and assume most ledger channel procedures are the same (some are not). Where we deviate is the protocol specification for closing a virtual channel γ constructed "over" a ledger channel β in the byzantine and happy cases laid out below.

Nathan Ginnever: nathan@finalitylabs.io

Channel Validity

Let us first further explore the idea of channel validity provided by Perun[3] so that we may focus on the area of difference in Set payment channels. It is noted that Perun[3] devised the concept of channel validity to ensure that Ingrid (the intermediary) does not have to worry that her coins will be locked up forever in a virtual channel. This is due to the way that the protocol of virtual channel closing is structured. Specifically, it is expected that Ingrid must hear of a **vc-close** before she is able to construct a ledger channel proof of how to settle Alice and Bob's balances and all parties must wait for the virtual channel to expire before any closing operations (off-chain or on-chain) can begin. A pre-agreed upon timeout for the virtual channel ensures that Ingrid may get her coins back when the channel time expires and provides a default final state to the ledger channel. It also provides an easy way to reason about what channels are open and when it is valid to close on-chain in the event that byzantine settling is required. We replace this with merkle trees and an agreement between ledger channel parties of the number of open virtual channel during any given state update. As quoted by Perun[3] "Since they (virtual channel) cannot be closed earlier (validity timeout), there is no separate "closing" procedure for the virtual channels, and instead both opening and closing procedure are described below". We can look deeper into the protocol governing opening/closing of virtual channel constructs below.

Perun Open-vc Protocol

Once all parties $P \in \gamma.$ all-users {Alice, Bob, Ingrid} have signed a message *m* to open a virtual channel in the form of a state update to the ledger channel within 2 rounds, the balance is reduced in each agreement to reflect the bond for the virtual channel in that state update. If it is assumed that opening and closing a virtual channel must take unanimous consent to update the ledger channel β then it could be impossible for Ingrid to close without a validity timeout. It is allowed $\gamma.$ all-users time $\gamma.$ validity + 5Δ +1 to settle the final state of γ. If $\gamma.$ end-users {Alice, Bob} are truely timed out, then the initial state S_0 of γ that Ingrid knows about will win the settlement and the result will be a total refund for $\gamma.$ all-users. We observe that Ingrid could simply supply this initial state S_0 of γ to the ledger channels β_A and β_B before a validity timeout (as this state will be contained in the opening virtual channel agreement of Set channels). Given that Alice and Bob's signatures hold the final verdict of the state update on β_A and β_B to close γ, we break away from the consensus protocol of normal state channels on β_A and β_B such that β.end-users may prove an update that is strictly announcing that they have a higher sequence virtual channel state update to settle the ledger channel with. The problem of updating the state on β_A and β_B has now become one of intent to close γ with a latest state, and we reason this may be settled on-chain. It is possible that Ingrid or a network of bounty hunters has heard of some S_i of γ and may attempt to settle with a more appropriate balance. In either case the net outcome for Ingrid remains neutral as long as the same S_i of γ is provided to β_A and β_B.

Nathan Ginnever: nathan@finalitylabs.io

Set Open-lc/vc and Update Protocol

Here we describe the opening and update protocols for ledger channels and virtual channels. We simply extend more information in the string update annotation that ledger channel participants agree upon during state transitions.

Let $\omega \in \mathbb{N}$ be a version number for β. A version of δ (or a single state), $(\widehat{\delta}, \omega, \alpha, \sigma)$ is a unanimously signed update to β where σ contains both parties signatures in β. The update annotation α string contains information on the agreement happening during a given update. In the Set-Payment protocol, this string contains a root hash of all open virtual payment channels initial state, and a counter marking the number of open virtual channels. A simple hash of a single virtual channel to show the agreement and initial state of an open virtual channel is possible if you only wish to open one at a time. We elect to use a root hash of a merkle tree of open virtual channel for concurrent channel operations.

Nathan Ginnever: nathan@finalitylabs.io

The opening protocol for ledger channels and updating virtual channels is the same as Perun[3] so we omit this part below.

Update ledger channel

1. P (the-initiator) generates a message (**update**, *id*, θ, α) where *id* is some channel δ identifier and waits to hear from their counterparty.
2. Upon receiving an update, P $\in$ δ.end-users "the confirmer" will validate the $\widehat{\delta}$ checking that it is equal to $\widehat{\delta}^{P}$ accept for an updated balance and a reshuffled merkle root $\widehat{\mu}$ that contains the initial state of any open γ. Consider the two cases.
    a. There are no open γ such that μ is equal to an empty byte array.
    b. There are one or more open γ such that μ is equal a byte array representing the root hash of a merkle tree (see merkle tree protocol from Raiden[5] for further details).
3. If P receives (update-ok) message from the environment they produce sig $\widehat{\sigma}$ to confirm the update.

Open virtual channel γ:

1. A (**vc-open**) message is generated triggering γ.end-users to create a state update to γ.Ingrid that contains their opening certificates (signed initial state of γ).
2. P $\in$ γ.end-users lets ($\widehat{\delta}^{P}$, ω^{P}) be the latest state of δ that P is aware of. P generates a new state of the channel that is equal to $\widehat{\delta}^{P}$ on all elements aside from the adjusted balance and α containing a rebalanced merkle root μ with γ_0 contained.
3. γ.Ingrid receives the (**vc-open**) message and waits for both γ.end-users state update (**update**, *id*, θ, α).
    a. γ.Ingrid receives correct (**update**, *id*, θ, α) from both γ.end-users. She emits (vc-opened) and waits to hear a (**vc-close**) message from the environment.
    b. Otherwise she outputs (**vc-not-opened**).
4. P $\in$ γ.end-users waits to hear of an opening certificate from γ.Ingrid in the form of a signature on the ledger channel update presented by γ.end-users. The may now enter an idle state and pass messages over γ and close at anytime.

Figure 1: Procedures for: (A) open virtual channel (B) update ledger channel (C) update virtual channel (omitted)

Nathan Ginnever: nathan@finalitylabs.io

Set Close-lc/vc Protocol

Happy

At any point in time during an open virtual channel γ, any γ.all-users {Alice, Bob, Ingrid} may present a ledger channel update containing a rebalance of the balances of β by applying θ := Win($W^{\gamma.Alice}$, $W^{\gamma.Bob}$) (simply selecting the highest sequence update they know of in γ) to their respective β_A or β_B that will close the virtual channel. It is up to β.end-users to decide if they wish to close the virtual channel with given balance adjustments. In the case of Ingrid, she does not have to wait to hear of updates to both β_A and β_B before she signs as long she is confident that she is signing the latest state of γ. However, if a later state of γ becomes known after Ingrid signs a ledger channel update then she may lose a part of her bond. Ingrid may wait until Alice and Bob send agreeing γ final states in their ledger channel updates before she signs to be sure that her channel balances nets neutral.The update should also come with an adjusted update annotation that removes the open virtual channel data from the merkle root.

Byzantine

We will define a “close-vc” protocol that will give Ingrid an exit path to remove the validity timeout. Our contribution is noticing that you can use a two phase dispute process to ensure that the latest state of a virtual channel is chosen to rebalance ledger channel state after a settlement process. In this case, Ingrid would send her last known state of the virtual channel to the ledger channels, β_A and β_B. This will likely be the initial state of the virtual channel that would just undo the channel opening. It is important that this update to the ledger channel state only be allowed to update a closing virtual channel balance and no other state in the ledger channel (i.e a regular ledger channel balance update transaction or opening a new channel) since the second dispute no longer obeys consensus between {Alice} or {Bob} and {Ingrid} but rather obeys the consensus of {Alice} and {Bob}. The ledger channel is programmed to check this state update for validity since it cannot be assumed correct without the ledger channels β. end-users ({A/B} and {Ingrid}) consensus.

Ledger Channel Settling

There are two functions to update the ledger channel state in byzantine events. They are sequential events and the latest signed γ update must be initiated after settling the last known state of the ledger channel.

Update lc (start lc settlement)

The purpose of updateLC() is to provide both parties a settlement time for determining the latest state of the ledge channel. This will allow any open virtual channel over that ledger channel to then be settled. The updateLC function requires both parties β.all-users {Alice or Bob} and {Ingrid} are present in σ to initiate a challenge period with $\widehat{\delta}$. We observe again that it should take time ... to settle $\widehat{\delta}$.

settleVC

As described above, the purpose of the settleVC() is to allow time for the latest known state of γ to be settled on-chain if the virtual channel initial state is contained in the agreed upon root hash of this update. To determine if a state update to a virtual channel is a progressive iteration of the initial state, we use an identifier that is in the agreed upon initial state. This will later allow a finalization (wake up function) to take the balance parameters and move the balances of Alice/Bob and Ingrid back into the ledger channel. Each higher sequence state of the closing virtual channel presented to the on-chain contract will extend the challenged time by a predetermined length as is with any state channel challenge. At this point Alice and Bob may hold Ingrid's funds in escrow as long as they are willing to pay expensive computational fees for doing so. We could limit the second settlement phase here to only be available for a finite amount of time. This would ensure that Alice and Bob do not sign state updates to the virtual channel infinitely, however the costs of doing so should prevent them from doing this in the first place so we do not limit the challenge period on virtual channel state settlement.

Finalization Protocol

Once both challenge times have passed a wake up function may be called by anyone that will rebalance the ledger channel state based on the locked in virtual channel state. This will decrease the number of open channels marked in the smart contract from the ledger channel settlement phase. Any remaining open virtual channel may now start a settlement phase off of the settled ledger channel as long as they pass the merkle proof check of inclusion. If at any point the number of open virtual channel reaches 0 (either through closing byzantine virtual channels on-chain, or supplying a happy case closed state with no open virtual channels) then the contract will pay out based on the ledger channel final state and close the ledger channel.

Closing a ledger channel is the same as Perun[3] aside from checking that the state update has an agreed upon closing flag and settles on their being no open virtual channels in the state update, so it is omitted.

Closing virtual channel

1. At any point, P $\in \gamma$.all-users may present updates to β_P channel with identifier γ.subchan(P) that settle γ with W_P := (γ^P, υ^P, α^P, σ^P) the latest known state of γ that P has received from P':= γ.other-party(P).
2. Within a reasonable time Ingrid decides to close the channel γ.
    a. If γ.Ingrid receives correctly formatted message (**vc-close**, W^P , S^P) from both γ.end-users she goes on to step 3.
    b. If she does not she goes on to step 4
3. Party γ.Ingrid learns of the latest state (W^P , S^P) from either γ.end-users or from the contract settlement in step 4. She lets her update to her ledger channel β_P contain a rebalance of μ' and balance updates based on θ := Win($W^{\gamma.Alice}$, $W^{\gamma.Bob}$) state balance. Ingrid emits (**vc-close**) and everyone is back in idle state for other ledger channel operations.
4. Any P $\in \gamma$.all-users may start a Δ forced-reply message to the contract with their latest β_P state ($\widehat{\delta}^P$, ω^P) which contains the initial state of the challenged γ in μ. This forces both β.end-users to close all open virtual channels and settle exit the ledger channel.
5. Once the latest state of merkle roots in β_P is settled, another Δ forced-reply message may present the latest known state of γ, (W^P , S^P).
6. A wake up call is sent to settle the balances of the virtual channel back into the ledger channel on-chain and the number of open channels is decremented. Any other open virtual channels may repeat step 5 until no open channels are left and the ledger channel may be closed.

Figure 2: Procedures for: (A) close virtual channel (B) close ledger channel (omitted)

vc-State Validity Checks

1. Initial state contained in ledger channel merkle root and update state is sequentially higher.
2. Signature on initial and update state match Alice/Bob
3. Balance on update state does not overstep Ingrid's bonds

Example Scenarios

For ledger channel β_P ,

Let $S_0 = (\delta^0, \omega^0, \alpha^0, \sigma^0)$

ω = 0 // lc sequence number

α = { balanceAlice, balanceIngrid, 0x0 } // lc state containing an empty vc root

σ = sigs { Alice/Bob, Ingrid } // Unanimous consent

State = { balanceA, balancel, 0x0, numOpenVc = 0 }

For virtual channel γ ,

Let $S_0 = (\delta^0, \omega^0, \alpha^0, \sigma^0)$

ω = 0 // vc sequence number

α = { balanceAlice, balanaceBob, bondIngrid, addressB, addressA, addressl } // vc state

σ = sigs { Alice/Bob, Ingrid } // Unanimous consent

State = { 0x1239, 0, addressA, addressB, addressl, balanceA, balanceB, bondl }

Open vc: For ledger channel β_P ,

Let $S_1 = (\delta^1, \omega^1, \alpha^1, \sigma^1)$

Let α.root = keccak256($\gamma_{init}.\alpha$)

ω = 1

α = { adjustedBalanceAlice/Bob, adjustedBalanceIngrid, 0xd3adbe4f }

σ = sigs { Alice/Bob, Ingrid }

State = {balanceA/B-vcBalanceA/B, balancel-vcBalanceA/B, 0xd3adbe4f..., numOpenVc = 1}

where 0xd3adbe4f... = keccak256(γ_{init})

Nathan Ginnever: nathan@finalitylabs.io

Update vc: For virtual channel γ ,

Let S_0 = { 0x1239, 0, addressA, addressB, addressI, balanceA, balanceB, bondI }

Let S_1 = { 0x1239, 1, addressA, addressB, addressI, balanceA, balanceB, bondI }

...

Close vc: For ledger channel β_P ,

Let S_2 = {balanceA/B+vcBalanceA/B, balanceI+vcBalanceA/B, 0x0, numOpenVc = 0}

Virtual Channel is now closed once S_2 is signed by β .all-users: {Ingrid, Alice} and {Ingrid, Bob}.

Balance Update of Ic state based on vc state

β_A : [Alice $\rightarrow (y'_A + x'_A)$, Ingrid $\rightarrow (y'_I + x'_B)$]

β_B : [Ingrid $\rightarrow (z'_I + x'_A)$, Bob $\rightarrow (z'_B + x'_B)$]

Considerations

Re-entry (back to happy case transactions off chain from a byzantine situation) path for Alice and Bob (if any) if they notice Ingrid tried to settle the vc requires a lot of client side heavy lifting to reshuffle the the Ic state to be accurate to what has already been closed on-chain. Keep in mind Set allows any party to signal to everyone else they wish to close a channel without their consent.

This is assumed to only happen in byzantine cases thus we may assume a punishment if the party trying to close is responded to. Ingrid should wait a sufficiently long enough time before trying to force settle a byzantine vc as to reduce the odds of getting punished arbitrarily.

There are cases when Ingrid's net balance will not be neutral if the force settling of vc state does not settle with the same $\widehat{\delta}$. To handle this, Ingrid should always settle each channel with the same vc state, and if a newer one is presented to either channel then she should update the other accordingly if γ .end-users counterparty is byzantine. She must also be sure not to sign updates to the Ic state that reflect an incorrect state of the vc (don't sign the Ic update to close the vc until both alice and bob agree on final vc-state).

Is it okay to allow any party who initiates an exit to close the channel at anytime. We reason this is acceptable for payment hubs between humans as the intermediary hub Ingrid is incentivised to not close channels before Alice and Bob are ready as fees may be collected for the service.

PoC Solidity Implementation

https://github.com/finalitylabs/set-virtual-channels

References

[1] Ethereum Team. Solidity Documentation, Release 0.4.11.
https://media.readthedocs.org/pdf/solidity/develop/solidity.pdf (2017)

[2] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. Draft version 0.5.9.2
https://lightning.network/lightning-network-paper.pdf (2016)

[3] Perun: Virtual Payment Hubs over Cryptographic Currencies
https://eprint.iacr.org/2017/635.pdf (2018)

[4] Machinomy v2: Generalized State Channels over Ethereum
https://docs.google.com/document/d/13Asq2h79AzZwUidfwI5KxnC7twLNExuCWQ7ZLqksv_I/edit#
(2018)

[5] Raiden Network: Hashlock payment hubs over Ethereum
https://raiden.network/101.html (2016)

[6] Jeff Coleman: State Channels
https://www.jeffcoleman.ca/state-channels/ (2015)

[7] Perun: Foundations of State Channels Networks
https://eprint.iacr.org/2018/320 (2018)

Nathan Ginnever: nathan@finalitylabs.io