Generalised swap swear and swindle games

Viktor Trón, Aron Fischer

December 31, 2017

Contents

1 Introduction

Public proof-of-resource blockchains implement a decentralised consensus mechanism. Consensus is reached on the validity and ordering of transactions in a state machine. State transitions are triggered by transactions, and clients run a virtual machine to calculate these state transitions. In this process, transactions refer to specific accounts which store programs - called "smart contracts" – and the clients trigger functions of these programs to calculate the state transition. Limited only by the expressive power of these smart contracts, the blockchain can govern a myriad of rules of interaction, and enforce agreements.

In the context of data storage and provision, we can imagine carefully designed economic incentive systems that are driven by transparent smart contracts, which regulate the peer-to-peer data flow in such a way that the emergent properties of the network are beneficial to all of its users. Such a system can act as a fully decentralised application platform, providing useful services while potentially disintermediating trusted third parties on which such services had to rely in the past.

While the Turing-complete blockchain is sometimes refered to as "the final missing piece" of the puzzle to bring the cypherpunk vision of decentralisation to its completion, problems with their scalability are still a major bottleneck preventing mass adoption to real-world problems. These scalability issues include the high transaction costs for micropayments, the inherent speed and volume limitations in current blockchain designs, and are likely to remain problematic even for blockchains implementing more advanced technologies such as "sharding" and "proof of stake".

In recent years, "side chains" and "state channels" have been proposed as technologies that could remedy these shortcomings, and they directly inspired some of the approaches taken in this paper. We introduce a framework to support generic incentive systems for decentralised services. Our framework was first conceived of in the context of designing storage incentives for the "swarm" decentralised content delivery network, but has since been generalised.

As with any state channel, the system relies on the three pillars of communication, registration and enforcement:

1. Peers engage in local exchange (peer-to-peer communication) of data, promises, payments, messages, requests, deliveries etc. We call this the **SWAP** network. Furthermore, peers will relay data between disjoint peers to extend the scope of the network.

2. All nodes in the network pay a security deposit to a smart contract. This is the **SWEAR** registration. This allows them to offer more complex services to their peers.

3. The rules of service provision are enforced by a courtroom – the **SWINDLE** contract – making nodes accountable in case of non-compliance.

Such a system is playfully called a *swap, swear and swindle* game.

In this paper we present the generalised version of this idea and show how the paradigm can serve as the base-layer infrastructure driving decentralised service economies. The tools presented here provide a platform for running digital services that are cheap and scalable without compromising the level of security offered by the public blockchain. The claim is that virtually any digital service can be reinterpreted as a swap, swear and swindle game.

The solutions presented are built up in a step-by-step fashion from very simple and intuitive modules, and most of the components have clear real-world analogues. Our modular approach has the advantage that reasoning about the overall complex system becomes easier, enabling less error-prone implementations.

In section 2 we introduce swap channels, an off-chain peer-to-peer accounting and payment solution for bidirectional services. We start from an off-chain protocol to account for service-for-service exchange between peers, introducing compensation for services and various ways to minimise blockchain interaction yet mitigate liability due to delayed payments.

Section 3 introduces a taxonomy of promissory notes passed between peers in a swap channel and shows how future payment promises can serve as enforceable service contracts. Conditional bonds pay out rewards upon successful delivery and implement positive incentivisation for decentralised services.

In section 4, we present a way to implement service guarantees using an abstract challenge based system inspired by the blockchain-as-a-judge paradigm. The threat of enforceable punitive measures serves as incentive to play by the rules.

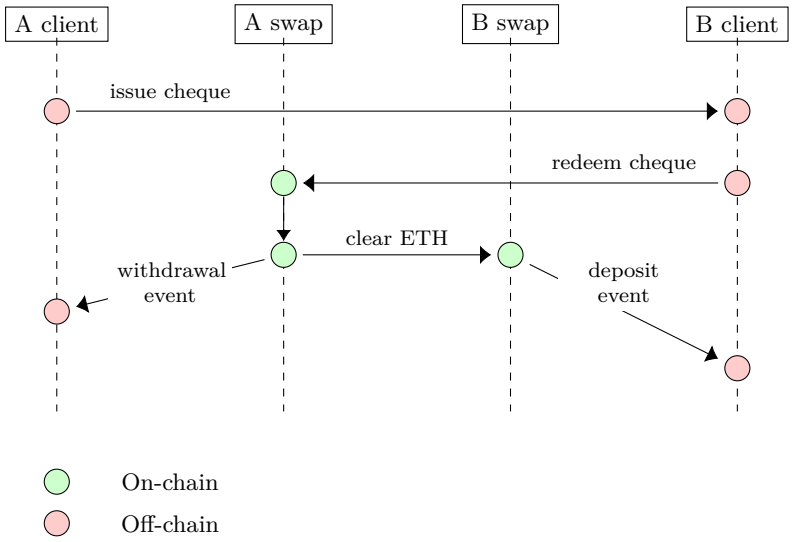

Figure 1: The basic interaction sequence for swap chequebooks

In section 5, we show how swap channels can form a service network and extend the scope of economic interaction between peers both in terms of reach and frequency, yet preserve the security and scaleability offered by swap channels.

Section 6 discusses price signalling and shows a way to eliminate the opportunity cost inherent in advance payments for future services when using appreciating assets like cryptocurrency. We conclude by showing how real world digital services can serve to back stablecoins.

In the appendix you find the implementation details of the smart contracts underpinning the system, as well as detailed examples of their application to data storage insurance and generic off-chain payments.

2 Swap channels

One of the major issues with direct "on-chain" payments in a blockchain network is that each transaction must be processed by each and every node participating in the network, resulting in high transaction costs. One strategy to mitigate transaction costs is to defer payments and process them in bulk. In exchange for reduced cost, the beneficiary must be willing to incur higher risk of settlement failure.

2.1 A simple chequebook

A very simple smart contract that allows the beneficiary to choose when payments are to be processed was introduced in [Tron et al., 2016a]. This *chequebook* contract is a wallet that can process cheques issued by its owner. The cheques are analogous to those in the real-world: the issuer signs a cheque specifying a beneficiary, a date and an amount, gives it to the recipient as a token of promise to pay at a later date. The smart contract plays the role of the bank. When the recipient wishes to get paid, they "cash the cheque" by submitting it to the smart contract. The contract, after validating the signature, date and the amount specified on the cheque, transfers the amount to the beneficiary's account (see figure 1). Analogously to the person taking the cheque to the bank to cash it, anyone can send the digital cheque in a transaction to the owner's chequebook account and thus trigger the transfer.

Since these digital cheques are files and can therefore be copied, care must be taken that the same cheque cannot be cashed twice. Such "double cashing" can be prevented by assigning each cheque given to a particular beneficiary a serial number which the contract will store when the cheque is cashed. The chequebook contract can then rely on the serial number to make sure cheques are cashed in sequential order, thus needing to store only a single serial number per beneficiary. An alternative strategy to prevent double cashing, when repeated payments are made to the same beneficiary, is that the cheques contain the *cumulative* total amount ever credited to the beneficiary. The total cumulative amount that

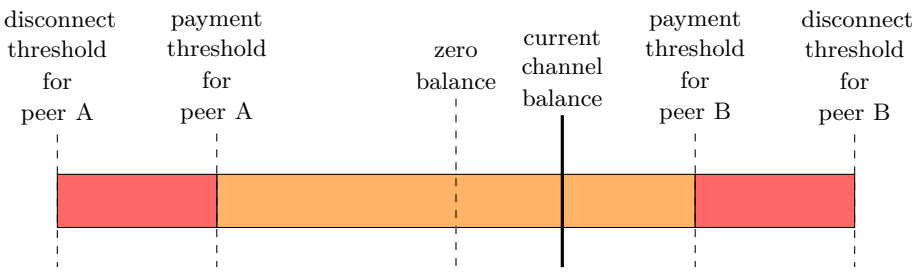

Figure 2: Swap balance and swap thresholds. Zero balance in the middle indicates consumption and provision are equal. The current channel balance represents the difference in uncompensated service provision: if to the right of zero, the balance tilts in favour of A with peer B being in debt, whereas to the left the balance tilts in favour of B with A being in debt. The orange interval represents loss tolerance. If the balance goes over the payment threshold, the party in debt sends a cheque to its peer, if it reaches the disconnect threshold, the peer in debt is disconnected.

has been cashed out is stored in the contract for each beneficiary. When a new cheque is submitted, the contract ignores cheques with amount equal to or less than the stored total, but it will transfer the difference if it receives a cheque with a higher total.

This simple trick also makes it possible to cash cheques in bulk because only the current 'last cheque' need ever be processed. This achieves the reduction of transaction costs alluded to above.

Incidentally, the cumulative amount stored in the contract represents the total of all outgoing payments that have been honoured and thus the contract also serves as a credit history for the owner.

The amount deposited in the chequebook (*global balance*) serves as collateral for the cheques. It is pooled over the beneficiaries of all outstanding cheques. In this simplest form, the chequebook has the same guarantee as real-world cheques: none. Since funds can be freely moved out of the chequebook wallet at any time, solvency at the time of cashing can never be guaranteed: if the chequebook's balance is less than the amount sanctioned by a cheque submitted to it, the cheque will bounce. This is the trade off between transaction costs and risk of settlement failure.

2.2 SWAP accounting with chequebook

Nonetheless, even in this simple form, the chequebook proves quite useful. [Tron et al., 2016a] introduces a protocol for peer-to-peer accounting, called *SWAP*. *SWAP* is a tit-for-tat accounting scheme that scales microtransactions by allowing service for service exchange between connected peers (*swap = swarm accounting protocol for service wanted and provided*).

In case of equal consumption with low variance over time, bidirectional services can be accounted for without any payments. Data relaying is an example of such a service, making swap ideally suited for implementing bandwidth incentives in content delivery or mesh networks.

Extended with a delayed payment instrument like the chequebook, swap also offers a way to deal with unequal consumption as well as high variance. In the presence of high variance, or unequal consumption of services, the balance will eventually tilt significantly toward one peer. In this situation, the indebted party issues a cheque to the creditor to return the nominal balance to zero. This process is automatic and justifies swap as *settle (the balance) with automated payments* (see figure 2).

Such cheques can be cashed immediately by being sent to the issuer's chequebook contract. Alternatively, cheques can also be withheld which enables the parties to save on transaction costs. While, strictly speaking, there are no solvency guarantees, a bounced cheque will affect the issuer's reputation as the chequebook contract records it. On the premise that cheques are swapped in the context of repeating dealings, peers will refrain from issuing cheques beyond their balance. In other words, interest in keeping good reputation with their peers serves as an incentive for nodes to maintain solvency.

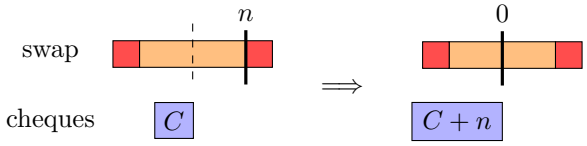

Figure 3: Peer B's swap balance (with respect to A) reaches the payment threshold n (left), B sends a cheque to peer A. B keeps the cheque and restores the swap balance to zero.

2.3 Waivers

If the imbalance in the swap channel is the result of high variance as opposed to unequal consumption, after a period of accumulating cheques the channel balance starts tilting the other way. Normally it is now up to the other party to issue cheques to its peer resulting in uncashed cheques accumulating on both sides. To allow for further savings in transaction costs, it could be desirable to be able to 'play the cheques off against each other'.

Such a process is possible, but it requires certain deep changes within the chequebook contract. In particular, cashing cheques can no longer be immediate and must incur a security delay, familiar from payment channels.

Let us imagine a system analogous to cheques being returned to the issuer. Assume peer A issued cheques to B and the balance was brought back to zero. Later the balance tilts in A's favour but the cheques from A to B have not been cashed. In the real world, user B could simply return the last cheque back to A or provably destroy it. In our case it is not so simple; we need some other mechanism by which B commits not to cash that particular cheque. Such a commitment could take several forms; it could be implemented by B signing a message allowing A to issue a new 'last cheque' which has a lower cumulative total amount than before, or perhaps B can issue some kind of 'negative' cheque for A's chequebook that would have the effect as if a cheque with the same amount had been paid.

What all the implementations have in common, is that the chequebook can no longer allow instantaneous cashing of cheques. Upon receiving a cheque cashing request, the contract must wait to allow the other party in question to submit potentially missing information about cancelled cheques or reduced totals.

We describe one possible implementation below.

To accommodate (semi-)bidirectional payments using a single chequebook we make the following modifications

1. All cheques from user A to user B must contain a serial number.
2. Each new cheque issued by A to B must increase the serial number.
3. A's chequebook contract records the serial number of the last cheque that B cashed.
4. During the cashing delay, valid cheques with higher serial number supersede any previously submitted cheques regardless of their face value.
5. Any submitted cheque which decreases the payout of the previously submitted cheque is only valid if it is signed by the beneficiary.

With these rules in place it is easy to see how cheque cancellation would work.

Suppose user A has issued cheques $c_0 \dots c_n$ with cumulative totals $t_0 \dots t_n$ to user B. Suppose that the last cheque B cashed was c_i. The chequebook contract has recorded that B has received a payout of t_i and that the last cheque cashed had serial number i.

Let us further suppose that the balance starts tilting in A's favour by some amount x. If B had already cashed cheque c_n, then B would now have to issue a cheque of her own using B's chequebook as the source and naming A as the beneficiary. However, since cheques $c_{i+1} \dots c_n$ are uncashed, B can instead send to A a cheque with A's chequebook as the source, B as the beneficiary, with serial number $n + 1$ and cumulative total $t_{n+1} = t_n - x$. Due to the rules enumerated above, A will accept this as

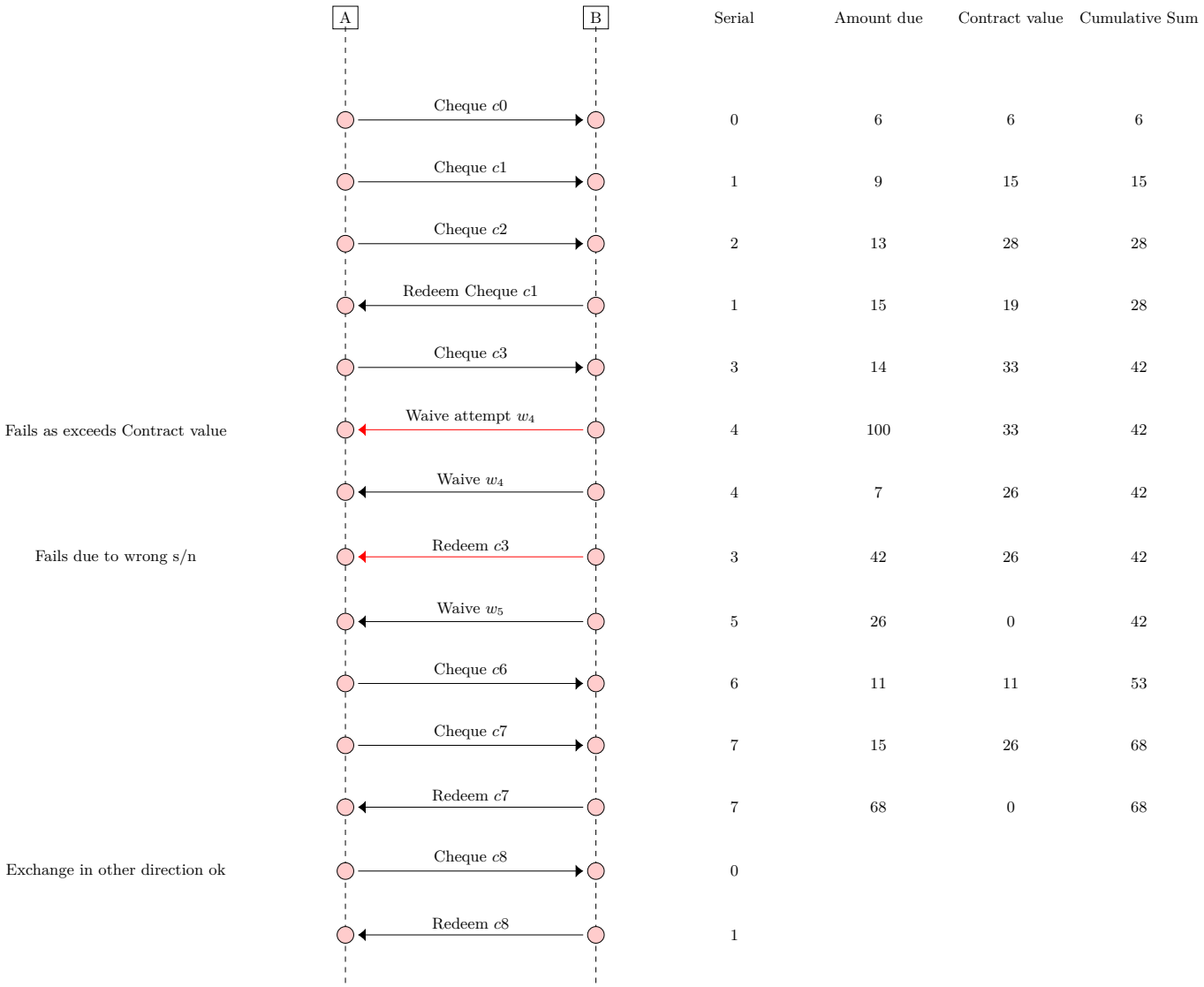

Figure 4: Example sequence of mixed cheques and waivers exchange

equivalent to a payment of amount x by B. In this scenario, instead of sending a cheque to A, B waive part of their earlier entitlement. This justifies swap as *send waiver as payment*.

This process can be repeated multiple times until the cumulative total is brought back to t_i. At this point all outstanding debt has effectively been cancelled and any further payments must be made in the form of a proper cheque from B's chequebook to A (See figure 4).

2.4 Channel deposits

In what follows we extend the chequebook contract with the functionality of assigning per-user payout guarantees and show how two contracts having swap entries for each other constitute a full payment channel.

2.4.1 Payment channels

A *payment channel* is a smart contract set up by two parties A and B and is used for secure bidirectional peer to peer payments off-chain. The contract has an amount of ether locked up in it that A and B have deposited. As the parties transact, they keep track of their *relative balance*, i.e., which portion of the locked up ether belongs to each party. The two parties A and B exchange signed messages agreeing on the current relative balance using some communication channel. If submitted to the channel contract, such messages change the balance. The messages sent back and forth function as payments between the two parties. Crucially, most of these messages are not submitted to the contract at all. The parties can choose to settle the new balance and record it in the channel contract by submitting the signed message in a transaction. The only time that balance has to be updated on-chain is when one party wishes to withdraw funds from the channel contract. Withdrawing funds requires a two step process.

A chequebook contract implementing a debt waiving procedure like the one outlined in section 2.3 shares a lot of the same features. If user A has deployed such a chequebook and is using it to pay user B, then the cheques act very similarly to the payment messages above, whereas the chequebook contract is analogous to a collection of half payment channels. Key differences to a full payment channel are

1. Payout is not guaranteed.
2. Initially payments can only flow in one direction (from A to B)
3. The balance can never go below zero i.e. in A's favour. This is why we refer to this construct as 'half' of a payment channel. It is analogous to a payment channel in which the initial deposit was made only by A.
4. Balances cannot be updated on-chain without a transfer.
5. All collateral locked to a node in the contract belongs to A; all that B can do is send in cheques to request a payout.

In this section, we show how by adding channel deposits to the chequebook, it is possible to guarantee payouts to individual peers. When two chequebooks mutually have entry of each other, we can consider that as a full payment channel. We introduce a scheme that enables the chequebook owner to reallocate a global balance to individual peers in a flexible way. Then we show how a strict protocol for shadowing outstanding liabilities can result in a novel enhanced full payment channel solution, called *swap channel*.

2.4.2 Hard channel deposit

Since cheques can bounce, payout is not guaranteed (see section 2.1). However, it is possible to add a payout guarantee to the cheques on a per-user basis. The chequebook deployed by user A has a balance of ether that acts as collateral for all outstanding cheques A has issued. We call this the *global balance*. As such, any user B holding a cheque from A has no guarantee that payout will be possible at any particular time in the future.

To add a payout guarantee, we need to make some modifications to the contract. In order to be able to issue guaranteed cheques to user B, the contract would contain some ether that is locked in the contract and acts as collateral for payments from A to B only. Any cheques held by B are thus

<table>
<tr><td>hard
channel
deposit
peer p_0</td><td>hard
channel
deposit
peer p_1</td><td>...</td><td>hard
channel
deposit
peer p_n</td><td>soft
channel
deposit
peer p_0</td><td>soft
channel
deposit
peer p_1</td><td>...</td><td>soft
channel
deposit
peer p_n</td><td>surplus
soft
channel
deposit</td><td rowspan="3">liquid
balance</td></tr>
<tr><td colspan="4">total channel deposit</td><td colspan="5">global liquid deposit</td></tr>
<tr><td colspan="9">global deposit</td></tr>
<tr><td colspan="10">global balance</td></tr>
</table>

Figure 5: Chequebook balances and deposits.

guaranteed to be honoured up to this locked amount, called *hard channel deposit*. In other words, the only way A can reclaim the locked amount is via a two step process that involves a payout request by A followed by a grace period during which B is still able to submit any outstanding cheques. With this mechanism in place, the chequebook is equivalent to a channel in which the entire initial deposit was made by A.

If both A and B have such a chequebook contract, each having some collateral locked up for the other, then this setup is functionally equivalent to a full *payment channel*; with the exception that states can not be saved on the blockchain without an actual transfer. In existing payment channel proposals, the contract can update the *relative balance*, i.e., how much of the tied up collateral belongs to A and how much belongs to B. In the simple chequebook implementation such a re-balancing would involve a transfer from one chequebook contract to the other.[1] The chequebook implementation has the advantage that it can start out simple with features being added gradually as they are needed.

The sum of all (hard) channel deposits is the *total channel deposit*. The global balance can never be less than the total of channel deposits. Channel deposit allocations can be controlled via transactions. Whenever a part of A's balance is assigned to B, the balance is checked and the transaction fails if total channel deposits would exceed the global balance.

Conversely, when the owner attempts to withdraw from the contract, the balance is not allowed to go below the total of channel deposits. The difference between the global balance and global deposit is the *liquid balance*. The total of channel deposits is never higher than the global deposit, the difference is *global liquid deposit* (see figure 5).

The question arises whether there is a way to distribute this global liquid deposit among the swap channels in a cost-effective, flexible yet secure way.[2] At any point in time insolvency can be triggered if total uncashed balance is not covered by the global balance: $\sum_{i \in P} u_i > b$. On the other hand, solvency is guaranteed for peer B if the total of outstanding uncashed cheques from A to B is not greater than the total channel deposit ($u_B <= d_B$). If it is greater ($u_B > d_B$), part of A's liability is unsecured: $l_B = u_B - d_B$. If solvency is not guaranteed, it is reasonable for peer B to cash their cheque. If all peers do this, there is a bank run. If the sum of overruns is not greater than the liquid balance ($\sum_{i \in P} l_i <= l$, the bank run ends well, otherwise the creditors compete for liquidating their entitlement before the liquid balance runs dry. Insolvency is known if there is a peer whose unsecured uncashed balance is higher than the global liquid balance, i.e., $l_B > l$. Solvency is not guaranteed if the unsecured balance is greater than zero. As the unsecured balance increases, so does the potential loss in case of insolvency. Is it possible to secure allocations of the liquid deposit so that it tracks the liabilities?

2.4.3 Soft channel deposit

In the following we describe a construct which gives arbitrarily secure guarantee of solvency on the outstanding liabilities of peer A without explicitly assigning deposits to channels on chain.

Assume that there exist a notion of *epoch*, a fixed settlement period at the end of which B needs to sign off on the total sum of outstanding liability implied in the uncashed cheques issued by A to B. Each time A sends a cheque to B they can offer to reallocate some of the surplus liquid deposit to B. Formally, a *soft channel deposit note* is a signed note with the following fields:

[1] Advanced payment channels such as Raiden have the ability to string together individual channels and compose them into a network. This extension is discussed in detail in section 5.

[2] The global deposit can be implemented by the channel deposit assigned to the owner. Two step delayed withdrawal would fall out from this construct.

- creditor address
- debtor address
- the epoch index
- serial number
- channel balance
- soft channel allocation

If debtor signs it, the note is a *soft channel deposit offer*, if it is signed by the creditor and the note is *soft channel deposit claim*. As A is sending cheques to B, it can continually attach an increased soft channel deposit offer. As a response B accepts the offer by sending A a soft channel deposit claim of the offered amount (or less). Similarly, when B waives uncashed cheques from A, B attaches a (decreased) soft channel deposit claim to the waiver it is sending to A.

The soft channel deposit claim is the amount B would like to see secured/dedicated to them exclusively. Assume that at the end of the epoch A's every active peer has sent at least one claim to A. After A receives soft channel deposit claims from all its active peers for the epoch, the claims are collected in a list (ordered according to the peer index of the beneficiary in A's swap contract). A signs the swarm hash of the (concatenated) list and sends it alongside the list to each peer in a construct called a *soft channel deposit allocation table*.

Upon receiving the soft channel deposit allocation table, B verifies that (1) the amount dedicated to B is no less than the total uncashed balance on matured cheques issued by A to B and (2) the total sum of channel deposits allocated is no greater than the global deposit and (3) each active peer in A's swap contract has a corresponding claim for the current epoch and (4) all the signatures are valid. This process in illustrated in figure 6. If soft channel offers and claims were issued according to the protocol, then (1) is automatically satisfied if A includes B's most recent claim. Points (2-4) make sure that each peer can make sure that the allocations are what the respective peers actually claim and their total does not exceed the global liquid deposit.

If the soft channel deposit allocation table is valid, B can with complete certainty know that the global deposit in A's contract has sufficient funds to cover all outstanding cheques handed to B even if all peers were to redeem all of their outstanding cheques and no further amount is deposited to increase the liquid deposit. Practically, this means then that, after receiving a valid allocation table for the epoch from A, B has no risk of insolvency when holding out on cashing A's cheques.

The hard channel deposit can serve as a secure buffer to allow increase in uncashed balance during the current epoch. The sum of the currently allocated soft channel deposit and hard channel deposit is called channel deposit. As long as the uncashed balance does not exceed the current channel deposit, solvency of A to B's claims is guaranteed. If the uncashed balance exceeds this limit, there is no guarantee that A will remain solvent, therefore B can just stop providing service. If the channel deposit does not cover the uncashed balance, the result can be insolvency. If channel deposits cover the amount of uncashed cheques, the difference is called *surplus channel deposit*. As long as there is a surplus, A does not need to make an offer.

If we regard 'accepting and not cashing cheques' as a service then we can define a higher level swap accounting. In a way, the hard channel deposit amount play the role of the payment threshold in simple swap, whereas soft channel deposit offers and claims are somewhat analogous to cheques or waivers. Instead of the dynamic swap cycle of the simple swap, here the cycle corresponds to an epoch.

We can define a strict etiquette for the protocol. In the beginning, the allocated soft channel deposit from A to B is zero. As A sends a cheque to B in the amount of x then it attaches an offer to raise the allocation from zero to x. Since the updates to the soft channel deposits only signalled after the epoch ends, accumulating cheques is risky if there is no guaranteed cover by per-user guarantee. The hard channel deposit mitigates this by effectively backing up those outstanding notes that are issued during the current epoch. This effectively makes the hard channel deposit a throughput restriction on the channel traffic. As a consequence, hard channel deposits are supposed to be chosen so that they reflect the desired per-epoch throughput of the channel.

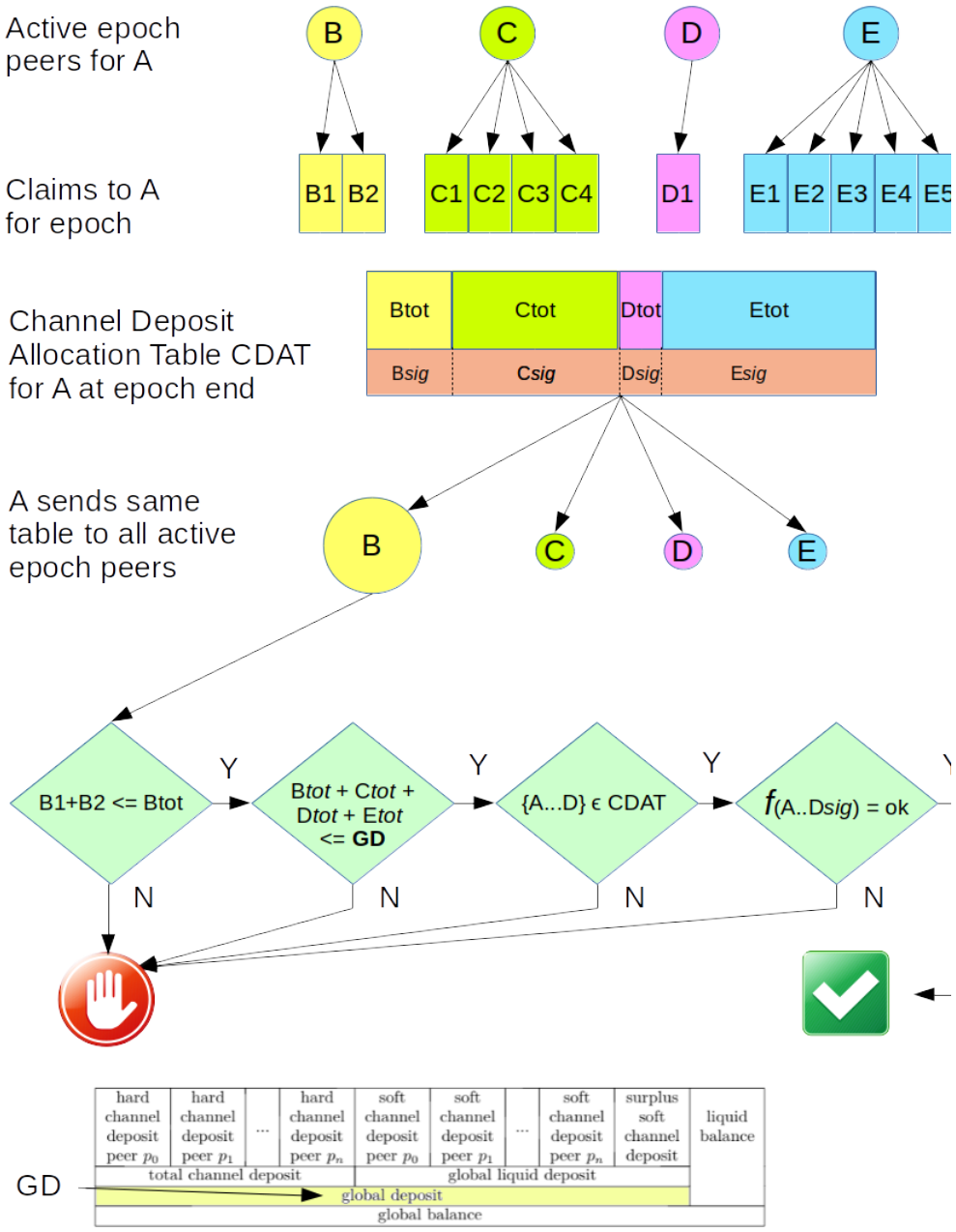

Figure 6: Hard and soft channel deposit

This strict protocol has far-reaching consequences: (1) security, (2) instant payout and (3) unambiguous creditor. Firstly, A's peers cannot lose anything: Assume that B received additional cheques from A during the current epoch and accordingly issued claim to A to raise the soft channel deposit. Imagine that the next soft channel allocation table is omitting A entirely or does not contain the most recent claim amount. However, since the increase is within the limit of the hard channel deposit, B can always enforce 100% solvency. Secondly, if we make the soft channel deposit claim contain the last serial number and swap balance, submitting the inclusion proof of the B's soft channel deposit claim in a valid current allocation table is sufficient to immediately release funds. Another important consequence of the strict protocol that at any one time only one of two connected peers is sending an allocation table to the other. The recipient can use the received one as evidence that the peer claim is zero and include it in their allocation table.

2.4.4 Unique allocation and double spending

So far we just assumed that all peers can give a valid claim at the end of the epoch. If there is no absolute consensus on who participates, peers could be given alternative allocations by A. Two groups of nodes would be made to believe they are the unique descendent of the original group (e.g., the entire set of all peers that has an entry in A's contract). The allocation table signs off on the active participants. If we simply allow the active peer set to change from one epoch to the next, a malicious node can start issuing alternative allocation tables to two sets of peers. Naive nodes in either set would think their set represents the entire active set. Colluding adversaries in both groups would sign off on reduced allocations which then can be used to cover the increased allocations of victims in both groups. The victims incorrectly assume the allocation table is exhaustive, i.e., it contains all peers whose allocation can increase: as long as they will accept their allocation table valid, they can be exploited with such a continuous double spend attack until they decide to settle on the blockchain. To mitigate this attack, one has to make sure that the active peer set does not split. Even though it would be possible for peers to agree on the allocation in a multiparty state channel with unanimous consensus rule, this suffers from the same availability issue as the one it is trying to solve.

Adding new peers to the allocation table is always possible, so is increasing the channel deposits unilaterally. However, dropping peers from a table can be problematic unless we are certain that A does not allocate multiple tables to victims. We propose instead that it is A's responsibility to prove off-chain that they issue a unique allocation table. If we can make sure that A issues a unique resource update for the active set in every epoch, then changing peer set is no longer a problem. Assume that C formerly included in the allocation table is presented with a new table with some earlier peers missing, then they must verify somehow that the table is unique.

A protocol called *mutable resource update notifications* offers a solution: The pull notification version requires the resource owner to send a *resource update chunk* to a number of deterministic but not predictable addresses. These are stored and served exactly as chunks, except that their validity is not verified by checking their address against the swarm hash of their content but the signatures against its uniqueness. Now assume a malicious node A and its colluding peers achieve a situation when they sign on alternative allocation tables to two groups each containing a naive node B and C, respectively. When B and C are in their respective split, then both B and C check A's resource updates at the same time. In order for A to keep up a double spend attack against B and C then B and C must receive the two allocation tables as a result to their query of the resource update chunk. In other words, adversary A needs to make sure each of the two versions is served to the respective peer that expects it. But this is not easy. With nodes incentivised to report multiple signing, the adversary would have to (1) be able to identify which incoming request originates from B and C[3] and (2) bribe all nodes involved in the retrieval to collude. This uses the power of the big network to make it impossible for adversaries to fork on the truth.

In summary, using soft channel deposits enables the chequebook owner to allocate and reallocate funds from their liquid deposit as channel deposits each dedicated to one particular peer. This construct

[3]Due to sender anonymity in retrieval, requests cannot easily be linked to the originators' identity, although direction where the request is coming from can be. The scheme can be strengthened by requiring entire neighbourhoods to sign off on the uniqueness and increase the stake., i.e. if they are caught giving out different updates under the same address, they can be challenged on the blockchain and lose their deposit. See section 4 for how such promises can be guaranteed. Alternatively, multiple addresses for a single update could be required. As the number of required addresses (neighbourhoods) for a single update increases, the probability of uniqueness multiplies, essentially to allow arbitrary level of certainty.

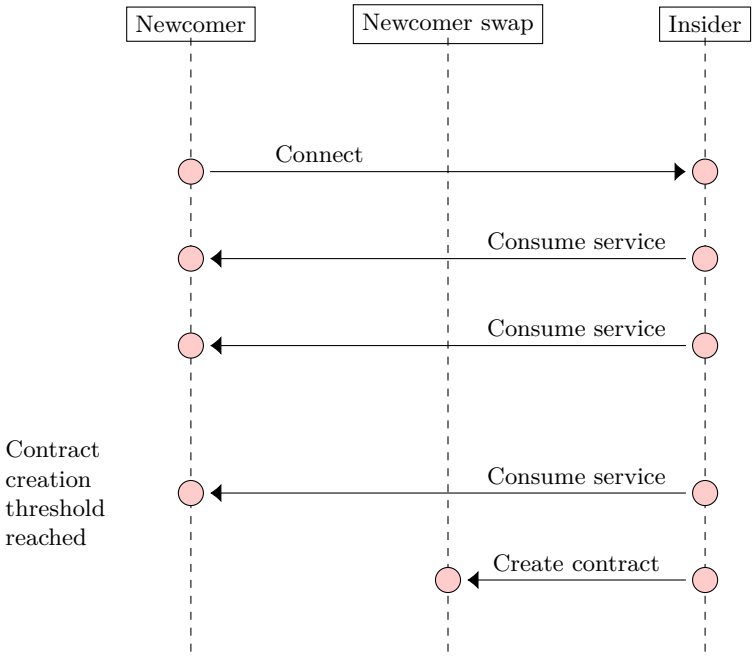

Figure 7: Bootstrapping or how to launch as a swap capable node consuming and providing a service and earn money

offers a way for A's peers to hold out on cashing beyond the hard channel deposit. As long as the overall debt across channels remains below the global channel deposit this allows us to overcome high variance in debt across epochs and peers in a fairly flexible way without blockchain transaction costs, yet provides very strong guarantees as long as there is a consensus among A's peers with regards to the allocation.

2.5 Zero cost entry

Swap accounting can also work in one direction only. If a party enters the system with zero ether (*newcomer*), but connects to peers with funds (*insider*), the newcomer begins to provide the service (and not use it) in order to earn a positive swap balance. Once the payment threshold is reached, the newcomer will be paid and is soon able to deploy chequebook of her own. In short, even without any chequebook contract at all, Swap offers zero cost onboarding to newcomers. Once the payout threshold is reached, the insiders could pay the newcomers on-chain.

If the insider has a chequebook and wishes to pay the newcomer with a cheque, this is also possible, but there is a caveat here in that cashing cheques requires sending a transaction to the blockchain, and therefore: gas.[4] The newcomer will be able to earn cheques for services provided, but will not have the means to cash them. Unless the first payment is an on-chain payment or the node can convince one of its peers to send the transaction for them.

We allow nodes to sign off on a structure, and extend the swap contract with a preprocessing step, which triggers payment to the transaction sender covering the transaction's gas cost plus a service fee.

A newcomer connects to an insider with an existing chequebook contract. The insider consumes the newcomer's services and instead of issuing a cheque, the insider agrees to create a chequebook contract for the newcomer. When the insider's service debt reaches the amount needed for contract creation, the insider sends a transaction to create the contract with newcomer as the owner. Once the newcomer has her own chequebook contract, she is able to consume services and issue cheques to pay for them.

In order to be useful however, the deployed chequebook must also have some positive balance as collateral. Since the newcomer has no ether, it must be the insiders who deposit this starting balance.

[4] Although this restriction may be dropped once ethereum is upgraded to the 'Metropolis' release in which the chequebook contract will be able to pay for the transfer and deduct the cost of the transfer from the payout.

If peers agreed that they want to save on transaction costs, it is reasonable to create the contract with the required initial balance in one go. This would imply waiting out with contract creation until the insider's debt reaches the *cost of bootstrapping* which is the sum of (1) the cost of contract creation, (2) the required starting balance, and (3) possibly some extra service fee to incentivise insiders to provide this kind of service (see figure 7).

To summarise, by serving before consuming, participants can bootstrap their way into swap without the need for funds. Hence swap is justified as (*setting up a wallet as payment*).

3 Promissory notes

A promissory note is a redeemable promise of payment potentially maturing in the future. For example a cheque (see section 2.4.2) is a promise of payment that can be redeemed at any time by 'cashing' the cheque.

We define the general notion of a promissory note as a *promise* that entitles a *beneficiary* to *future payment* under a *condition*. Since any such promise effectively puts the money in escrow, the condition can also be referred to as an *escrow condition*. A promissory note can contain the following fields:

- an index (serial number)
- an amount
- a beneficiary (ethereum address)
- an escrow condition (escrow witness contract address)
- a valid-from time (blockheight or time)
- a valid-until time (blockheight or time)
- a remark

In the simplest case in which no escrow condition or maturation date is given: the note represents a simple cheque as discussed above. A maturation date for the promissory note (expressed as a blockheight or timestamp) can be specified to indicate the earliest possible occasion a note can be redeemed (valid-from) as well as a deadline when the promise expires (valid-until). During the validity period, the promise can be redeemed only if the escrow condition is met. The active period of a note lasts from issuance to expiry. The chequbook contract extended to handle all promissory notes is called a *swap contract*.

3.1 Recurring payments

A cheque represents an adhoc promise to redeem an amount. A transfer of funds is sanctioned by the issuance of a cheque and cheques are issued as needed. There are, however, services where recurring payments are predictable and therefore the installments can be sanctioned in advance. The primary example of this is subscription based services.

The chequebook contract can be expanded to support such services. Authorising a recurring payment (to a contract) would be achieved by signing a blank cheque (a cheque with an unspecified amount) against the beneficiary possibly with a valid-until date. The beneficiary (a smart contract) would then be able to withdraw payments from the chequebook at will. It would be up to the smart contract code of the beneficiary to ensure that only the 'correct' amounts are transferred.

3.2 Bonds

A note with an amount, a beneficiary and a future valid-from date is effectively a bond. The amount represents all outstanding payment obligations that the beneficiary is entitled to once the bond matures. Such a note can either be collateralised (as in section 2.4.2) or it can come without a guarantee (as in 2.1). When such a note is not collateralised at the time of issuance, accepting the bond is equivalent to granting a loan: A sends funds to B and B issues an unsecured bond in return.

note type	fields type	index int256	amount int256	beneficiary address	escrow address	valid-from int256	valid-until int256	remark byte32
cheque		✓	✓	✓			?	?
authorisation			✓	✓			?	?
bond		✓	✓	✓		✓	?	?
conditional bond			✓	✓	✓	✓	?	?
commitment			✓	?	✓	?	?	✓
bounty		✓	✓		✓	✓	?	?
soft channel deposit		✓	✓					?

Figure 8: Taxonomy of promissory notes: '✓' indicates a mandatory field, '?' indicates optional field. Types show the corresponding solidity type to encode in the ABI.

3.3 Conditional bonds and bounties

Conditions for payment that are more complex than a validity period are captured by the escrow condition. A promissory note can specify an address of an escrow contract which is to act as the judge, determining whether the escrow conditions have been met.

A note with an escrow field specified is called a *conditional bond* and can represent a *service request* with the escrow condition defining what constitutes the successful delivery of the service. If no beneficiary is specified, such note is essentially a *bounty* - a payment to be made to the first person who can satisfy the escrow condition. Bonds and bounties with escrow condition are subsumed under *conditional notes*.

Thus, in order for payments to be made, the escrow condition must be met. When the escrow condition is to be verified, the owner, the beneficiary address, and the note id (the hash of the signed note) are passed as arguments to the *testimony* method of the escrow contract. By implementing this method the contract conforms to the *witness* contract interface (section 4). Given the witness contract, the escrow condition is implicitly defined as whatever state makes the escrow witness give a positive testimony.

Figure 8 summarises the various types of promissory notes.

3.4 Conditional bonds and Swap: invoice and cheque

Let us assume that node A issues a conditional note (bond or bounty) to B with expiry (valid-until) time T. T represents the earliest time that A can consider the note unfulfilled (the service undelivered), before that we say that the note is *active*. Let us further assume that B fulfills the condition while the note is active.

Invoking the escrow contract directly, B can of course redeem the note, however, it would be nice if this could be subsumed in the Swap traffic so that no expensive on-chain operation is necessary. To redeem the note in Swap, B notifies A of successful delivery of the service and issues an *invoice*. This invoice contains the note id, the current cumulative swap balance and the serial number of the last cheque. The invoice thus contains the swap balance at the time of delivery as well as the note that is to be redeemed. It serves to indicate that the subsequent cheque will be the one to pay the outstanding amount for the note.

As a response, A is expected to send a cheque with an updated channel balance reflecting the payment of the invoice; i.e., the amount in the conditional note is added to the cumulative total. The remarks field references the conditional note id or the invoice id. If A refuses to do this, B can still send the conditional note in a transaction to the swap contract on-chain. When the witness validates the delivery condition, and the testimony is positive, the amount is escrowed and a grace period starts. During the grace period, A can respond by sending in the appropriate cheque to the contract as proof that the cheque was issued. The contract then also acts on the cheque as usual.

Conditional notes correspond to service requests. If peer B accepts and decides to act on it, it is prudent of B to require A to guarantee total liability over all active bonds it issued. If the sum of this total and the uncashed balance exceeds the channel deposit, there are no solvency guarantees. In these cases, B can accept bonds (as loan requests) or can expect A to top up the channel deposit or redeem

some cheques with A's contract as beneficiary. For the scenario, where usage of these services is not simultaneous but alternating, soft channel deposit allocations are ideal.

Let us assume A and B both maintain an ordered list of active conditional notes (bonds and bounties) issued by A. Each time A issues a conditional note or pays for a fulfilled one or one expires, the list is updated.

Consider the liabilities when conditional notes enter the swap. B's uncashed balance with respect to A is defined as the sum of (the amounts on) all outstanding notes from A to B (c_B) and the total amount of unchashed cheques (u_B) active at the beginning of the next epoch. If the total uncashed balance ($t_B = c_B + u_B$) is not greater than the total channel deposit (the sum of hard and soft channel deposits), the swap contract is guaranteed to be solvent even if all conditional notes get fulfilled (and therefore redeemable) and all peers decide to cash out on their as yet uncashed cheques. If it is greater, there is unsecured liability, defined as $l_B = t_B + u_B - d_B$. The peer is potentially insolvent if $\sum_{i \in P} l_i > b$, assuming all conditional notes become fulfilled all peers cash out and no extra amount is deposited on the contract.

If conditional notes are to be secured, B needs to treat conditional notes the same way as uncashed cheques. The total amount of active conditional notes is added to the total of uncashed cheques and following the protocol for soft channel deposit allocation, A shadows this amount with the allocations.

4 Service guarantees

In the previous section we introduced tools to compensate service providers. While rewards are paid out only if delivery is successful, such positive incentivisation may be insufficient if explicit guarantees of future delivery are required.

In this section we describe a system of service contracts and dispute resolution. Service contracts allow providers (or a set of providers) to offer quality and/or delivery guarantees to customers. First we describe how to give service guarantees (swear), then discuss how they can be enforced (swindle). Finally we link together the three contracts by showing how the swap interacts with swear and swindle.

4.1 Commitment and litigation

Without loss of generality, we can conceptualise a generic service as described by a game contract on the blockchain (see later in 4.2).

Nodes indicate their *commitment* to providing a service by signing against the game contract either in a transaction or issuing a signed note offchain. By committing to the game contract providers can be challenged if they don't comply with the rules.

Litigation on the other hand, allows customers to initiate an on-chain court procedure in case they suspect foul play. Such accountability is crucial for the design of scalable decentralised service economies.

We will describe the trial process for dispute resolution and how it evaluates the evidence against the accused provider. The verification is the same as the one used in the context of sanctioning outpayments when the escrow condition is evaluated after service delivery. However, if a provider is proven guilty by the court on the grounds of not complying with the rules specified in the service contract, various punitive measures will be automatically enforced. In most cases, this comprises losing some deposit that provider previously locked up as collateral to back their promise.[5] By registering on a service contract and putting up a deposit if required, providers *swear* to play by the rules. This deposit is locked up and used later as compensatory insurance transferred to the customer, redistributed or burnt.

4.2 Trial

If a service provider is suspected of non-compliance, they can be challenged by opening a case with the swear contract analogous to filing a case and have it tried in the courtroom. An abstract trial is composed of stages of litigation, at each stage the court procedure calls witnesses and, depending on their testimony, advances the trial into the next stage. Witnesses can refuse testimony if evidence is not (yet) available in which case the trial is paused until the grace period allowed to submit evidence is

[5]Instead of or in addition to losing their deposits, nodes would also suffer loss of reputation. If any demonstrated violation of terms is recorded on the public blockchain, users can expect clean histories as a measure of reputation.

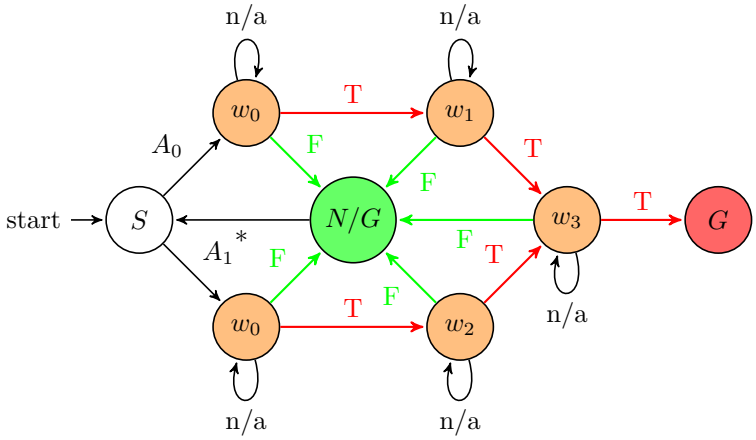

Figure 9: Trial stages represented as a finite state automaton describing a service contract. Upon receiving an accusation (A_0 or A_1) in a transaction, the case contract moves to the first litigation state. The orange circles represent states where a witness contract is called, the following state depends on the outcome of the witness testimony (the return value of the *testimony* function call). The other nodes call the internal *process* function: to pay compensation if the challenge is refuted (N/G) or enforce the sentence in case of a guilty verdict (G).

over. Eventually the trial concludes with a guilty or not guilty verdict followed by automatically enforced punitive measures in the former case or compensation to the accused in the latter.

The specific service contract describes the rules of the service by assigning particular expert witnesses to trial stages. These witnesses evaluate evidence submitted to them in relation to the case and give testimony if asked by the judge. The request–response process between the judge and the witnesses can thus be standardized even though each witness may require different data to be submitted to them as evidence.

To implement such an abstract dispute resolution system, the trial is described by a *finite state automaton* the states of which correspond to stages of litigation. The transitions are labelled with various outcomes of evaluating evidence submitted to the respective expert witness associated with the stage (true, false, N/A). A witness is a smart contract that implements the *testimony* function which takes as arguments the provider address, the plaintiff address, and the note id (the hash of the signed note) representing the service request.

The *swindle* contract orchestrates the transition through the stages by calling the witness contracts, handles grace periods to control the deadline for submitting evidence (challenges or their refutations) to eventually reach a guilty/non-guilty verdict on the grounds of witness testimonies. In case of a guilty verdict, the deposit sworn on is forfeited and the peer's registration is terminated as executed by the swear contract.

4.3 Enforcement

When a guilty verdict is reached, the swindle contract calls back to the swear contract to enforce the sentence, the forfeiture of the game deposit. The swear contract acts as executor of this decree, in a way similar to how a court order is observed by the agencies in a position to enforce sentences.

The contract that handles the deposits and registers which node plays which game is called *swear*. The swear contract is a registry of commitments, a mapping from identities to game contracts.[6] An entry is created if a provider signs off on a service game contract in an act of *commitment*. Essentially, the swear contract only needs to record how much deposit each commitment promised and let the swap channel handle the deposit. The swear contract can ask the swap if the provider has insufficient funds dedicated as swear deposits. This is simply implemented by channel deposit allocation: nodes allocate

[6] One can imagine not completely open swear registries which curate a whitelist of safe or recommended games. Similarly to an appstore, these contracts could catalogue service games.

Figure 10: Swap, Swear and Swindle contracts and their interactions:

the appropriate amount as channel deposit with the swear contract as beneficiary.[7] Commitments are implemented as a signed note containing an amount, the swear contract address as the escrow witness, the games contract address in the remark field, optionally a beneficiary and validity dates. Commitments are a subtype of conditional bonds which when submitted to the swap, call out to the swear contract as its witness and starts a new case against the issuer. Interpreted as a service request, a commitment authorises the transfer of a balance up to the amount (actual amount determined by the witness) and asks the swear contract to handle all service complaints against the provider. The swear contract implements the witness interface and when its *testimony* function is called, it prepares the case and authorises the swindle contract to lead the trial. When the guilty or not-guilty verdict is reached, the swindle contract calls back to the swear contract that sets out to execute the sentence with a call to *execute* giving a case id as argument.

The sentence starts with transferring the deposit amount to the deposit handler contract calling *execute* with the case id.

In this section we connected the dots and showed how swap swear and swindle interact. The swap contract can validate promissory notes so it can itself serve as witness for the swindle contract. When a commitment is sent to the swap contract, it calls the swear contract as an escrow witness which in turn opens a case and sanction the swindle contract to lead a trial. Once the verdict is reached, the sentence is executed by the swear contract which has authorisation to withdraw the deposit from the swap contract that holds it in the form of hard channel deposit. Figure 10 gives an overview of these interactions.

5 Service networks

In this section we describe how the tools for peer-to-peer accounting and data exchange can be used to drive the incentives for distributed digital services. Assume that a set of nodes form a network that provides a decentralised service. Insured chunk storage in swarm, arbitrary remote payments or database insertion are examples of such services. A *service task* is defined as an instance of service provision. Storing a particular chunk, paying an amount to another node or inserting an entry to an index are examples of tasks of the respective network service.

In what follows we will work with the assumption that the participating nodes in a service network are connected in a *kademlila topology* and can relay messages to other nodes using kademlia deterministic routing based on direct devp2p transport layer for each hop. In other words service networks are composed of a subset of nodes within swarm.

Scalability of swap channels is based on the premise that directly connected nodes engage in long-term repeated dealings and can afford setting up contracts on the blockchain to secure their interaction.[8] A *swap channel network* enables *indirect transactions* by relaying tasks and deliveries using swap-channel transactions on every hop. This makes it possible to preserve the scalability and security of swap, yet extend the scope of transactions both in terms of frequency and reach, i.e., enabling *ad-hoc one-off interactions* between *any two non-connected nodes*. Service networks with global provision guarantees (introduced in section 5.2) further improve the scope by offering guaranteed market making and delivery in a direct immediately settling swap transaction. As a result, service users can just *request and disappear*.

5.1 Incentivisation for relaying

[This section may not be part of this article, but the one on relaying etc]

Let's assume that nodes A,B,C,D are participant peers of a service network such that A-B, B-C and C-D are direct connections with a swap channel. Assume further that A intends to do some ad-hoc

[7] If the grace period for withdrawal from hard channel deposits is implemented by a call to the channel beneficiary, then we can handle in an elegant way the cases where deposits are locked as collateral for services with arbitrary validity period.

[8] Given the semi-permanent connections of the TCP based kademlia of swarm, peers interact with $O(log(n))$ peers where n is the number of nodes in the network.

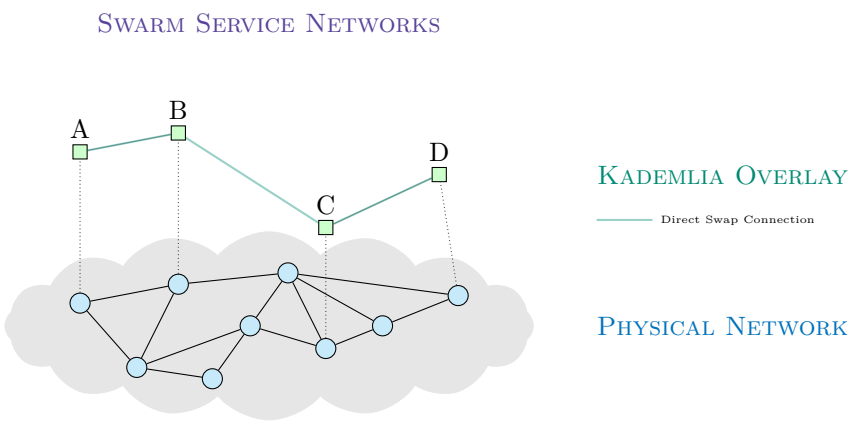

Figure 11: Swap channel network: peer to peer network of nodes with semipermanent TCP connections organised in a kademlia topology. Given nodes have an associated identity owning a chequebook contract, each live connection defines a swap channel.

one-off business with D. To this end, A issues a conditional bond specifying an escrow condition that some data be obtained from D constituting the proof of delivery of the task.

When A sends the conditional bond to B as beneficiary, B receives the conditional bond and issues the same bond only this time with C as beneficiary. C is directly connected to D, so it just relays the same to D who issues a receipt (or takeover proof). Each of these steps is implemented as swaps of conditional notes. The moment C receives the receipt, it is incentivised to pass it back to B as an invoice to prompt settlement. The same is true for B and any number of intermediate relaying nodes (see figure 12).

In general, we define an *indirect transaction* as a chain of swaps between directly connected peers. The success of a such indirect transactions between two nodes is dependent on whether and how such a chain can be found. Our assumption is that the nodes have semi-permanent connectivity and form a network topology where routing between any two nodes is guaranteed, e.g., kademlia. As a result, we can assume that as long as all nodes on the chain have a healthy kademlia connectivity, the prerequisite for finding a route is satisfied.

In order to incentivise relaying nodes to take part in such a network, a transaction fee for every hop needs to be introduced. We stipulate that the transaction fee for one swap is proportional to the logarithmic distance that the hop spans. As a consequence, the simple rational strategy to maximise profit will incentivise nodes to find the closest node when relaying as well as maintain a healthy connectivity needed for successful routing. Furthermore, the entire transaction fee can be precalculated as proportional to the distance between sender and remote beneficiary and therefore can be offered in advance when issuing the conditional bond.

This construct is equivalent to message relaying with *certified delivery*, except that nodes are required to have funds to cover the amount of the conditional bond. For live connections in a service network to be operational, parties agree to always having in-channel capacity in the amount of X. With soft channel deposits, such throughput restrictions can easily be relaxed and adjusted even on an on-demand basis.

In addition to swap channels, connected nodes operate *provable data exchange*. Assume that two nodes participate in a particular service that involves relaying objects (such as requests and responses). The content addresses (swarm hash) of consecutive objects sent in one direction constitute a *data stream*. Both peers save this stream to swarm and maintain an index mapping the hashes to offsets. The upstream peer (sender) calculates the swarm hash of this index and periodically signs it against the current blockheight or timestamp as well as the data stream identitfier. This *handover state* is periodically sent to the downstream peer (receiver) who verifies it and preserves it until the next one. The downstream peer periodically countersigns the handover state together with an initial offset or timestamp. This *takeover state* is then sent to the upstream peer. For instance, the handover state obtained from the upstream peer can be used to prove to third parties that an object X was handed over to the downstream

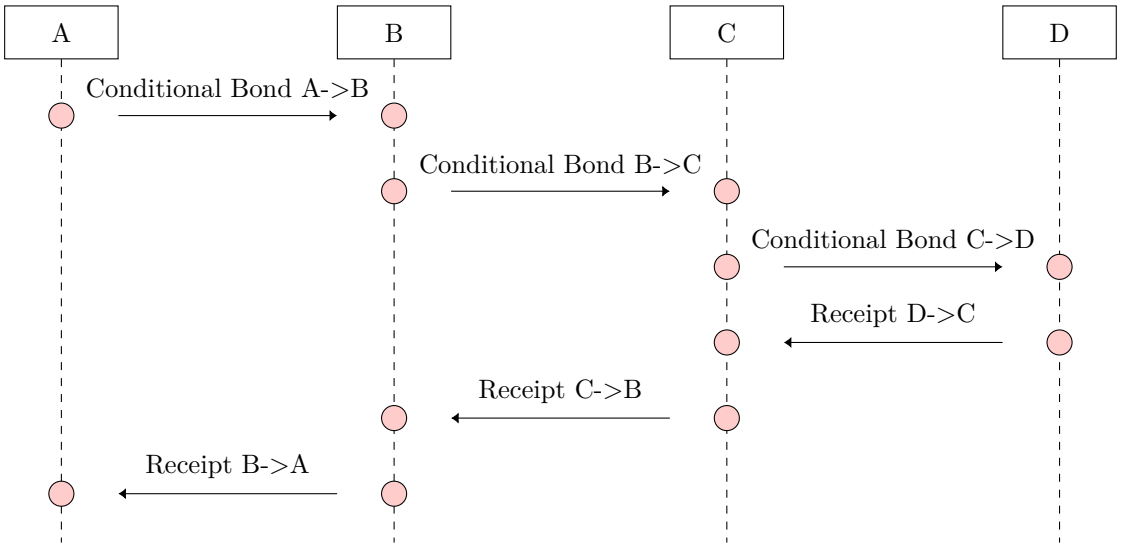

Figure 12: Sequence of conditional bond exchange and receipt forwarding in a service network

peer. This can be done by giving an inclusion proof of the hash of X.[9]

On top of the rewards peers get for relaying, we can introduce additional incentives to guarantee that messages reach the recipient. We can assume that relaying nodes register on a service contract they promise to relay messages in a data stream towards the recipient. If nodes have a stake to lose, punitive measures can be imposed on nodes that fail or refuse to relay messages in the data stream.

Take our earlier example of A sending a message to remote node D, and assume that B, C, and D are all registered relaying nodes that are online. If there is reason to believe that the message did not reach D, A can challenge B by simply providing the takeover proof for the message. B can defend itself by providing takeover proof from C, thereby shifting the blame to C for blocking the delivery. C in turn can show takeover proof from D to refute the challenge. It can also happen that C indeed did not forward the message because there was no node closer to D among its peers. But this means that D is a nearest neighbour of C, in which case D should have been connected to C. Therefore D can be challenged to prove that it was connected to C. D can respond to the challenge by showing handover state obtained from C dated after the message delivery deadline. This in turn is a challenge to C to show the inclusion proof of the message against the handover state. This pattern is called *finger pointing* and constitutes the investigative part of litigation which results in identifying the node ultimately responsible for failed delivery by not relaying. Finger pointing essentially implements *guaranteed delivery*.

5.2 Uniform resource allocation and market making

In the previous section we just assumed that A knew about D (from an independent source). In case there are other nodes as well that can in theory perform the same task, a bounty is sent to potentially multiple addresses of candidate providers and an implicit market making mechanism is obtained by certified delivery.

Assume the tasks of a service network can be provided by any participating node using the same amount of resources. Given a network of willing providers, a system can be designed called *global resource allocation by network topology (GRANT)* such that tasks are distributed more or less uniformly across the network.

For instance, this is the case with storage in swarm: each task of storage provision involves storing a fixed-size chunk of data. We index the chunks with their content address (swarm hash) and allocate the storage task to nodes whose address is closest to the chunk address. Since both nodes and chunks are

[9] Upstream peer can be challenged to give an inclusion proof.

uniformly distributed in the same address space, this strategy implements uniform load balancing. We can generalise this allocation strategy for any task if requirements are uniform across tasks by simply indexing the task requests with their hash.

When a service user issues a bounty to request a service task, the bounty is sent towards the task's address using kademlia routing. The node responsible for servicing the task is dynamically defined as the closest node at any point in time. If hops use provable data streams, the incentives can make sure that the request (task bounty note) reaches the appropriate nodes and thereby are granted the opportunity to deliver the task.

For continuous services such as chunk storage, this means that as nodes leave and join the network, the node responsible for a task can change over time. With data exchange proofs however, we can enforce that nodes synchronise historical task requests and therefore the allocation can be taken for granted. In other words, the network can perform dynamic load-balancing of previously submitted future tasks even though the network is changing.

In order to benefit from the allocated tasks by cashing the bounties on delivery, the distribution should be incentivised. In particular relaying nodes are not supposed to withhold tasks in hope of cashing the bounty themselves. To guarantee that downstream peers do get forwarded all the tasks and bounties, upstream peers can be challenged when they cash a bounty: if there is a valid handover proof of connection (synchronisation) for the relevant period after the bond was issued, upstream peer can be challenged for not relaying it (but withholding). Conversely, the bounty outpayment can be challenged if the candidate provider is not the closest node to the task, i.e., a downstream peer closer to the task address can show connectivity during the relevant period. Withholding if proven could result in the same punitive measure as failing to deliver.

Service nodes that are capable of staying online can choose to register and put up a deposit as collateral. One consequence of GRANT is that for each task, there is a node (or set of neighbouring nodes) that is responsible for delivering the task and can ultimately be held liable for non-delivery. Using the swindle scheme for litigation, if proven guilty a node's collateral is forfeited. Since individual punitive measures are a very effective incentive, service guarantees can be given independent of the positive incentivisation (the reward for delivery implicit in the bounty).[10]

When the original issuer of the task request receives the receipt for its bounty (takeover proof) from its immediate peer, they have all that is needed to start finger pointing after the deadline for the service task passed. In other words, service users have *instant guarantees* that the service will be provided for the task they request, even though it is unknown at the time by which node.

This type of service network is called *warranted automatic service provision (WASP)* network. Since one can thus obtain instant enforceable service guarantees in one swap exchange, WASP networks allow users to *request and disappear*. Applying this scheme to swarm insured storage, given a WASP network of storage nodes, swarm users can *upload and disappear*.

5.3 Atomic swap

The typical market making mechanism is the swap of request, offer, agreement. This pattern is described as a conditional note issued by A specifying an escrow that requires a second conditional bond referring to the first one sent back. If this is countersigned by A, the testimony is positive, otherwise N/A. This construction can be abstracted out as a generic connector witness analogous to the logical AND operator. It validates two conditional notes and checks their mutual commitment to binding them atomically.

[10] Since the amount of tasks (and therefore capacity requirements) are roughly uniform for each participant node, catastrophic failure of an entire node can cause limited damage. Since the required deposit is meant to collateralise this risk, it is reasonable to set it to a fixed amount across all provider nodes.

6 Pricing in service networks

6.1 Reverse auction for competitive bounties

6.2 Signalling capacity overload

6.3 Service networks for stablecoins

[?, 2014] [Decker and Wattenhofer, 2015] [Poon and Dryja, 2015] [Prihodko et al., 2016] [Tremback and Hess, 2015] [Bonneau et al., 2014] [Tron et al., 2016a] [Tron et al., 2016b] [Maymounkov and Mazieres, 2002] [Heep, 2010] [Malavolta et al., 2017] [Chiesa et al., 2017] [Heilman et al., 2016] [Green and Miers, 2016] [Miller et al., 2017] [McDonald, 2017] [Ferrante, 2017]

References

[?, 2014] ? (2014). Bitcoin micropayments, a new enabling technology. *Bitcoin Magazine*.

[Bonneau et al., 2014] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J. A., and Felten, E. W. (2014). Mixcoin: Anonymity for bitcoin with accountable mixes. In *International Conference on Financial Cryptography and Data Security*, pages 486–504. Springer.

[Chiesa et al., 2017] Chiesa, A., Green, M., Liu, J., Miao, P., Miers, I., and Mishra, P. (2017). Decentralized anonymous micropayments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 609–642. Springer.

[Decker and Wattenhofer, 2015] Decker, C. and Wattenhofer, R. (2015). A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer.

[Ferrante, 2017] Ferrante, M. D. (2017). Ethereum payment channel in 50 lines of code. Technical report, Medium blog post.

[Green and Miers, 2016] Green, M. D. and Miers, I. (2016). Bolt: Anonymous payment channels for decentralized currencies. *IACR Cryptology ePrint Archive*, 2016:701.

[Heep, 2010] Heep, B. (2010). R/kademlia: Recursive and topology-aware overlay routing. In *Telecommunication Networks and Applications Conference (ATNAC), 2010 Australasian*, pages 102–107. IEEE.

[Heilman et al., 2016] Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., and Goldberg, S. (2016). Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. Technical report, NDSS.

[Malavolta et al., 2017] Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., and Ravi, S. (2017). Concurrency and privacy with payment-channel networks. *?*

[Maymounkov and Mazieres, 2002] Maymounkov, P. and Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer.

[McDonald, 2017] McDonald, J. (2017). Building ethereum payment channels. Technical report, Medium blog post.

[Miller et al., 2017] Miller, A., Bentov, I., Kumaresan, R., and McCorry, P. (2017). Sprites: Payment channels that go faster than lightning. *arXiv preprint arXiv:1702.05812*.

[Poon and Dryja, 2015] Poon, J. and Dryja, T. (2015). The bitcoin lightning network: Scalable off-chain instant payments. Technical report, https://lightning.network.

[Prihodko et al., 2016] Prihodko, P., Zhigulin, S., Sahno, M., Ostrovskiy, A., and Osuntokun, O. (2016). Flare: An approach to routing in lightning network. Technical report, ? version 0.1 July.

[Tremback and Hess, 2015] Tremback, J. and Hess, Z. (2015). Universal payment channels. Technical report, ?

[Tron et al., 2016a] Tron, V., Fischer, A., A, D. N., Felföldi, Z., and Johnson, N. (2016a). swap, swear and swindle: incentive system for swarm. Technical report, Ethersphere. Ethersphere Orange Papers 1.

[Tron et al., 2016b] Tron, V., Fischer, A., and Johnson, N. (2016b). smash-proof: auditable storage in swarm secured by masked audit secret hash. Technical report, Ethersphere. Ethersphere Orange Papers 2.

A Swap contract

B Swear and swindle contract

C Swarm storage insurance

D Scaleable node-to-node payments

E Glossary

bounty a conditional bond without beneficiary 14

certified delivery message delivery with signed receipt from recipient 18

chequebook a wallet smart contract that allows cashing cheques 3

commitment 15, 16

conditional bond promise for future payment conditional on witness testimonies 14

conditional notes 14

epoch periodic interval of settlement for soft channel deposits 8

escrow condition condition describing the delivery condition needed to release payment. 13

finger pointing litigation scheme allowing upstream peers to shift blame to downstream peer" 19

global balance the balance of the chequebook contract 4, 7

global liquid deposit 8

global resource allocation by network topology (GRANT) a strategy to allocate service tasks in a balanced way. 19

handover state root hash of an outgoing data stream signed against downstream peer and time 18

hard channel deposit 8

indirect transactions chain of swaps linking local to remote node 17

insider participant in a service network with swap contract and ether holdings 12

invoice accompanies task delivery requesting payment 14

kademlila topology a scale free network topology that has guaranteed path between any two nodes in $O(log(n))$ hops 17

liquid balance 8