1 Payment Network Functionality

Functionality $\mathcal{F}_{\mathrm{PayNet}}$ – interface

- from $\mathcal{E}$:
  - (register, delay, relayDelay)
  - (toppedUp)
  - (openChannel, *Alice*, *Bob*, x, *tid*)
  - (checkForNew, *Alice*, *Bob*, *tid*)
  - (pay, *Bob*, x, $\overrightarrow{\texttt{path}}$, `receipt`)
  - (closeChannel, `receipt`, *pchid*)
  - (forceCloseChannel, `receipt`, *pchid*)
  - (poll)
  - (pushFulfill, *pchid*)
  - (pushAdd, *pchid*)
  - (commit, *pchid*)
  - (fulfillOnChain)
  - (getNews)
- to $\mathcal{E}$:
  - (register, *Alice*, `delay`(*Alice*), `relayDelay`(*Alice*), pubKey)
  - (registered)
  - (news, newChannels, closedChannels, updatesToReport)
- from $\mathcal{S}$:
  - (registerDone, *Alice*, pubKey)
  - (corrupted, *Alice*)
  - (channelAnnounced, *Alice*, $p_{Alice,F}, p_{Bob,F}$, *fchid*, *pchid*, *tid*)
  - (update, `receipt`, *Alice*)
  - (closedChannel, `channel`, *Alice*)
  - (resolvePays, *payid*, `charged`)
- to $\mathcal{S}$:
  - (register, *Alice*, delay, relayDelay)
  - (openChannel, *Alice*, *Bob*, x, *fchid*, *tid*)
  - (channelOpened, *Alice*, *fchid*)
  - (pay, *Alice*, *Bob*, x, $\overrightarrow{\texttt{path}}$, `receipt`, *payid*)
  - (continue)
  - (closeChannel, *fchid*, *Alice*)
  - (forceCloseChannel, *fchid*, *Alice*)
  - (poll, Σ_{Alice}, *Alice*)
  - (pushFulfill, *pchid*, *Alice*)
  - (pushAdd, *pchid*, *Alice*)
  - (commit, *pchid*, *Alice*)
  - (fulfillOnChain, t, *Alice*)

Fig. 1.

All players need to register in order to use channels. The registration of *Alice* works as follows: *Alice* inputs her desired delay and relayDelay that will be used for all her future channels. The first denotes how often she has to check the blockchain for revoked commitments and the second defines the minimum time distance between incoming and outgoing CLTV expiries. $\mathcal{F}_{\mathrm{PayNet}}$ then informs $\mathcal{S}$, who sends back a long-lived public key for *Alice*. This key represents *Alice*'s account, from where $\mathcal{F}_{\mathrm{PayNet}}$ can get coins to open new channels on her behalf and to place coins of closed channels. The key is sent to *Alice* who moves some initial funds to it and notifies $\mathcal{F}_{\mathrm{PayNet}}$. She is now registered. The exact logic is found in Fig. 2, which also contains the actions of $\mathcal{F}_{\mathrm{PayNet}}$ related to corruptions.

Additionally, the procedure `checkClosed`() is called after READing from $\mathcal{G}_{\mathrm{Ledger}}$, with the received state Σas input. This call happens every time $\mathcal{F}_{\mathrm{PayNet}}$ READs from $\mathcal{G}_{\mathrm{Ledger}}$. The formal definition of `checkClosed`() can be found in Fig. 9, along with a discussion of its purpose.

Functionality $\mathcal{F}_{\text{PayNet}}$ – registration and corruption

```
Initialisation:
    $\texttt{channels}, \texttt{pendingPay}, \texttt{pendingOpen}, \texttt{corrupted}, \Sigma \leftarrow \emptyset$

Upon receiving $(\textsc{register}, \text{delay}, \text{relayDelay})$ from $Alice$:
    $\texttt{delay}\,(Alice) \leftarrow \text{delay}$ // Must check chain at least once every $\texttt{delay}(Alice)$ blocks
    $\texttt{relayDelay}\,(Alice) \leftarrow \text{relayDelay}$
    $\texttt{updatesToReport}\,(Alice)\,, \texttt{newChannels}\,(Alice) \leftarrow \emptyset$
    $\texttt{polls}\,(Alice) \leftarrow \emptyset$
    $\texttt{focs}\,(Alice) \leftarrow \emptyset$
    send $(\textsc{read})$ to $\mathcal{G}_{\text{Ledger}}$ as $Alice$, store reply to $\Sigma_{Alice}$, add $\Sigma_{Alice}$ to $\Sigma$ and add largest block number to $\texttt{polls}(Alice)$
    $\texttt{checkClosed}(\Sigma_{Alice})$
    send $(\textsc{register}, Alice, \text{delay}, \text{relayDelay})$ to $\mathcal{S}$

Upon receiving $(\textsc{registerDone}, Alice, \text{pubKey})$ from $\mathcal{S}$:
    $\texttt{pubKey}\,(Alice) \leftarrow \text{pubKey}$
    send $(\textsc{register}, Alice, \texttt{delay}(Alice), \texttt{relayDelay}(Alice), \text{pubKey})$ to $Alice$

Upon receiving $(\textsc{toppedUp})$ from $Alice$:
    send $(\textsc{read})$ to $\mathcal{G}_{\text{Ledger}}$ as $Alice$ and store reply to $\Sigma_{Alice}$
    $\texttt{checkClosed}(\Sigma_{Alice})$
    assign the sum of all output values that are exclusively spendable by $Alice$ to $\texttt{onChainBalance}$
    send $(\textsc{registered})$ to $Alice$

Upon receiving any message $(M)$ except for $(\textsc{register})$ or $(\textsc{toppedUp})$ from $Alice$:
    if if haven't received $(\textsc{register})$ and $(\textsc{toppedUp})$ from $Alice$ (in this order) then
        send $(\textsc{invalid}, M)$ to $Alice$ and ignore message
    end if

Upon receiving $(\textsc{corrupted}, Alice)$ from $\mathcal{S}$:
    add $Alice$ to $\texttt{corrupted}$
    for the rest of the execution, upon receiving any message for $Alice$, bypass normal execution and simply forward it to $\mathcal{S}$
```

Fig. 2.

The process of *Alice* opening a channel with *Bob* is as follows: First *Alice* asks $\mathcal{F}_{\text{PayNet}}$ to open and $\mathcal{F}_{\text{PayNet}}$ informs $\mathcal{S}$. $\mathcal{S}$ provides the necessary keys and IDs for the new channel to $\mathcal{F}_{\text{PayNet}}$. *Alice* asks $\mathcal{F}_{\text{PayNet}}$ to check if $\mathcal{G}_{\text{Ledger}}$ contains the funding transaction from *Alice*'s point of view. If it does, $\mathcal{F}_{\text{PayNet}}$ activates $\mathcal{S}$, who in turn returns control to $\mathcal{F}_{\text{PayNet}}$. Now $\mathcal{F}_{\text{PayNet}}$ checks that the funding transaction is in the $\mathcal{G}_{\text{Ledger}}$ also from *Bob*'s point of view and in case it does, it notifies $\mathcal{S}$. $\mathcal{S}$ then confirms that to $\mathcal{F}_{\text{PayNet}}$ that the channel is open and $\mathcal{F}_{\text{PayNet}}$ finally stores the channel as open. This last exchange is needed to match the real-world interaction.

Functionality $\mathcal{F}_{\text{PayNet}}$ – open

```
Upon receiving (openChannel, $Alice, Bob, x, tid$) from $Alice$:
    ensure $tid$ hasn't been used by $Alice$ for opening another channel before
    choose unique channel ID $fchid$
    pendingOpen $(fchid) \leftarrow (Alice, Bob, x, tid)$
    send (openChannel, $Alice, Bob, x, fchid, tid$) to $\mathcal{S}$

Upon receiving (channelAnnounced, $Alice$, $p_{Alice,F}$, $p_{Bob,F}$, $fchid$, $pchid$, $tid$) from $\mathcal{S}$:
    ensure that there is a pendingOpen($fchid$) entry with temporary id $tid$
    add $p_{Alice,F}, p_{Bob,F}$, $pchid$ and mark "$Alice$ announced" to pendingOpen($fchid$)

Upon receiving (checkForNew, $Alice$, $Bob$, $tid$) from $Alice$:
    ensure there is a matching channel in pendingOpen($fchid$), marked with "$Alice$ announced"
    $(\text{funder}, \text{fundee}, x, p_{Alice,F}, p_{Bob,F}) \leftarrow$ pendingOpen $(fchid)$
    send (read) to $\mathcal{G}_{\text{Ledger}}$ as $Alice$ and store reply to $\Sigma_{Alice}$
    checkClosed($\Sigma_{Alice}$)
    ensure that there is a TX $F \in \Sigma_{Alice}$ with a $(x, (p_{\text{funder},F} \wedge p_{\text{fundee},F}))$ output
    mark channel with "waiting for fundingLocked"
    send (fundingLocked, $Alice$, $\Sigma_{Alice}$, $fchid$) to $\mathcal{S}$

Upon receiving (fundingLocked, $fchid$) from $\mathcal{S}$:
    ensure a channel is in pendingOpen($fchid$), marked with "waiting for fundingLocked" and replace mark with "waiting for channelOpened"
    send (read) to $\mathcal{G}_{\text{Ledger}}$ as $Bob$ and store reply to $\Sigma_{Bob}$
    checkClosed($\Sigma_{Bob}$)
    ensure that there is a TX $F \in \Sigma_{Bob}$ with a $(x, (p_{\text{funder},F} \wedge p_{\text{fundee},F}))$ output
    add receipt(channel) to newChannels($Bob$)
    send (fundingLocked, $Bob$, $\Sigma_{Bob}$, $fchid$) to $\mathcal{S}$

Upon receiving (channelOpened, $fchid$) from $\mathcal{S}$:
    ensure a channel is in pendingOpen($fchid$), marked with "waiting for channelOpened" and remove mark
    $\text{offChainBalance}(\text{funder}) \leftarrow \text{offChainBalance}(\text{funder}) + x$
    $\text{onChainBalance}(\text{funder}) \leftarrow \text{onChainBalance}(\text{funder}) - x$
    channel $\leftarrow (\text{funder}, \text{fundee}, x, 0, 0, fchid, pchid)$
    add channel to channels
    add receipt(channel) to newChannels($Alice$)
    clear pendingOpen($fchid$) entry
```

Fig. 3.

When instructed to perform a payment, $\mathcal{F}_{\mathrm{PayNet}}$ simply takes note of the message and forwards it to $\mathcal{S}$. It also remembers to inform the payer that the payment has been completed when $\mathcal{S}$ says so. Observe here that $\mathcal{F}_{\mathrm{PayNet}}$ trusts $\mathcal{S}$ to correctly carry out channel updates. While counterintuitive, it allows $\mathcal{F}_{\mathrm{PayNet}}$ to ignore the details of channel updates, signatures, key and transaction handling. Nevertheless, as we will see $\mathcal{F}_{\mathrm{PayNet}}$ keeps track of requested and ostensibly carried out updates and ensures that upon channel closure the balances are as expected, therefore ensuring funds security.

```
Functionality $\mathcal{F}_{\mathrm{PayNet}}$ – pay
Upon receiving $\left(\textsc{pay}, Bob, x, \overrightarrow{\texttt{path}}\right)$ from $Alice$:
    choose unique payment ID $payid$
    add $\left(Alice, Bob, x, \overrightarrow{\texttt{path}}, payid\right)$ to pendingPay
    send $\left(\textsc{pay}, Alice, Bob, x, \overrightarrow{\texttt{path}}, payid, \textsc{state}, \Sigma\right)$ to $\mathcal{S}$

Upon receiving ($\textsc{update}$, receipt, $Alice$) from $\mathcal{S}$:
    add receipt to updatesToReport($Alice$) // trust $\mathcal{S}$ here, check on resolvePays
    send ($\textsc{continue}$) to $\mathcal{S}$
```

Fig. 4.

The message resolvePays, sent by $\mathcal{S}$, is supposed to contain a list of resolved payments, along with who was charged for each payment after all. For each entry there are four "happy paths" that do not lead to $\mathcal{F}_{\mathrm{PayNet}}$ halting ($\mathcal{F}_{\mathrm{PayNet}}$ halts when it cannot uphold its security guarantees anymore): if the payment failed and no balance is changed, if the charged player is the one who initiated the payment, if the charged player is corrupted or if she has not checked the blockchain at the right times, i.e. was negligent (as discussed in Section **??** and formally defined in Figures 5 and 6). In case the payment was completed in a legal manner, the balance of all channels involved is updated accordingly (Fig. 7). Conversely, $\mathcal{F}_{\mathrm{PayNet}}$ halts if the charged player was not on the payment path (Fig. 5, l. 8), if a signature forgery has taken place (Fig. 5, l. 16), if the charged player has not been negligent (Fig. 5, ll. 19 and 27), or if any one of the individual channel updates needed to carry out the whole payment has not been previously reported with an UPDATE message by $\mathcal{S}$ (Fig. 7, l. 10).

Functionality $\mathcal{F}_{\text{PayNet}}$ – resolve payments

```
Upon receiving (RESOLVEPAYS, charged) from $\mathcal{S}$: // after first sending PAY, PUSHFULFILL, PUSHADD, COMMIT
    for all $Alice$ keys $\in$ charged do
        for all $(Dave, payid) \in$ charged $(Alice)$ do
            retrieve $\left(Alice, Bob, x, \overrightarrow{\texttt{path}}\right)$ with ID $payid$ and remove it from pendingPay
            if $Dave = \bot$ then // Payment failed
                continue with next iteration of inner loop
            else if $Dave \notin \overrightarrow{\texttt{path}}$ then
                halt // Only players on path may be charged
            else if $Dave \in$ corrupted then
                run code of Fig. 7
                offChainBalance $(Bob) \leftarrow$ offChainBalance $(Bob) + x$
            else // $Dave$ honest
                send (READ) to $\mathcal{G}_{\text{Ledger}}$ as $Dave$ and store reply to $\Sigma_{Dave}$
                checkClosed($\Sigma_{Dave}$)
                if $\Sigma_{Dave}$ contains a tx that is not a localCom$_n$ or a remoteCom$_n$ and spends a funding tx for an open channel that contains $Dave$ then
                    halt // DS forgery
                else if $\Sigma_{Dave}$ contains in block $h_{\texttt{tx}}$ an old remoteCom$_m$ that does not contain the HTLC and a tx that spends the delayed output of remoteCom$_m$ then
                    if polls($Dave$) contains an element in $[h_{\texttt{tx}}, h_{\texttt{tx}} + \texttt{delay}\,(Dave) - 1]$ then
                        halt // $Dave$ POLLed, but successful malicious closure
                    else
                        negligent($Dave$) $\leftarrow$ true
                    end if
                else if $Dave \neq Alice$ then
                    calculate IncomingCltvExpiry, OutgoingCltvExpiry of $Dave$ (as in Fig. ??, l. ??)
                    if $\Sigma_{Dave}$ does not contain an old remoteCom$_m$ then
                        if failure condition of Fig. 6 is true then
                            halt // $Dave$ POLLed and fulfilled, but charged
                        else
                            negligent($Dave$) $\leftarrow$ true
                        end if
                    end if
                end if
                run code of Fig. 7
                offChainBalance $(Dave) \leftarrow$ offChainBalance $(Dave) - x$
                offChainBalance $(Bob) \leftarrow$ offChainBalance $(Bob) + x$
            end if
        end for
    end for
```

Fig. 5. r, windowSize as in Proposition **??**

Absolute delay failure condition

$\texttt{IncomingCltvExpiry} - \texttt{OutgoingCltvExpiry} <$
$\texttt{relayDelay}(Alice) + (2 + r)\,\texttt{windowSize} \vee$
$(\texttt{polls}(Dave)$ contains two elements in
$[\texttt{OutgoingCltvExpiry}, \texttt{IncomingCltvExpiry} - (2 + r)\,\texttt{windowSize}]$ that have a difference of at least $(2 + r)\,\texttt{windowSize} \wedge$
$\texttt{focs}(Dave)$ contains $\texttt{IncomingCltvExpiry} - (2 + r)\,\texttt{windowSize} \wedge$
the element in $\texttt{polls}(Dave)$ was added before the element in $\texttt{focs}(Dave))$

Fig. 6.

Loop over payment hops for update and check

```
for all open channels $\in \overrightarrow{\texttt{path}}$ that are not in any closedChannels, starting from the one where $Dave$ pays do
    in the first iteration, payer is $Dave$. In subsequent iterations, payer is the unique player that has received but has not given. The other channel party is payee
    if payer has $x$ or more in channel then
        update channel to the next version and transfer $x$ from payer to payee
    else
        revert all updates done in this loop
    end if
end for
for all updated channels in the previous loop do
    ensure that a corresponding element has been added to the updatesToReport of each honest counterparty, otherwise halt
end for
```

Fig. 7.

Similarly to payment instructions, when $\mathcal{F}_{\text{PayNet}}$ receives a message instructing it to close a channel (Fig. 8), it takes a note of the pending closure, it stops serving any more requests for this channel and it forwards the request to $\mathcal{S}$. In turn $\mathcal{S}$ notifies $\mathcal{F}_{\text{PayNet}}$ of a closed channel with the corresponding message, upon which $\mathcal{F}_{\text{PayNet}}$ takes a note to inform the corresponding player. Depending on whether the message instructed for a unilateral or a cooperative close, $\mathcal{F}_{\text{PayNet}}$ will either put or not a time limit respectively to the service of the request. In particular, in case of cooperative close, the time limit is infinity (l. 4). As we will see, in case a unilateral close request was made and the time limit for servicing it is reached, $\mathcal{F}_{\text{PayNet}}$ halts (Fig. 9, l. 27). Once more $\mathcal{F}_{\text{PayNet}}$ trusts $\mathcal{S}$, but later checks that the chain contains the correct transactions with $\texttt{checkClosed}()$ (Fig. 9).

Functionality $\mathcal{F}_{\text{PayNet}}$ – close

```
Upon receiving (closeChannel, receipt, pchid) from Alice
    ensure that there is a channel $\in$ channels : receipt (channel) $=$ receipt with ID pchid
    retrieve fchid from channel
    add (fchid, receipt(channel), $\infty$) to pendingClose(Alice)
    do not serve any other (pay, closeChannel) message from Alice for this channel
    send (closeChannel, receipt, pchid, Alice) to $\mathcal{S}$

Upon receiving (forceCloseChannel, receipt, pchid) from Alice
    retrieve fchid from channel
    add (fchid, receipt(channel), $\perp$) to pendingClose(Alice)
    do not serve any other (pay, closeChannel, forceCloseChannel) message from Alice for this channel
    send (forceCloseChannel, receipt, pchid, Alice) to $\mathcal{S}$

Upon receiving (closedChannel, channel, Alice) from $\mathcal{S}$:
    remove any (fchid of channel, receipt(channel), $\infty$) from pendingClose(Alice)
    add (fchid of channel, receipt(channel), $\perp$) to closedChannels(Alice) // trust $\mathcal{S}$ here, check on checkClosed()
    send (continue) to $\mathcal{S}$
```

Fig. 8.

After every READ $\mathcal{F}_{\text{PayNet}}$ sends to $\mathcal{G}_{\text{Ledger}}$ and its response is received, `checkClosed()` (Fig. 9) is called. $\mathcal{F}_{\text{PayNet}}$ checks the input state Σ for transactions that close channels and, in case no security violation has taken place, it updates the on- and off-chain balances of the player accordingly (ll. 6-15). The possible security violations are: signature forgery (l. 17), malicious closure even though the player was not negligent (l. 20), no closing transaction in Σ even though the player asked for channel closure a substantial amount of time before (l. 27) and incorrect on- or off-chain balance after the closing of all of the player's channels (l. 32).

Functionality $\mathcal{F}_{\mathrm{PayNet}}$ – `checkClosed()`

```
function checkClosed($\Sigma_{Alice}$) // Called after every (READ), ensures requested closes eventually happen
    if there is any closing/commitment transaction in $\Sigma_{Alice}$ with no corresponding entry in pendingClose($Alice$) $\cup$ closedChannels($Alice$) then
        add ($fchid$, receipt, $\perp$) to closedChannels($Alice$), where $fchid$ is the ID of the corresponding channel, receipt comes from the latest channel state
    end if
    for all entries ($fchid$, receipt, $h$) $\in$ pendingClose($Alice$) $\cup$ closedChannels($Alice$) do
        if there is a closing/commitment transaction in $\Sigma_{Alice}$ for open channel with ID $fchid$ with a balance that corresponds to receipt then
            let $x, y$ $Alice$'s and channel counterparty $Bob$'s balances respectively
            $\text{offChainBalance}(Alice) \leftarrow \text{offChainBalance}(Alice) - x$
            $\text{onChainBalance}(Alice) \leftarrow \text{onChainBalance}(Alice) + x$
            $\text{offChainBalance}(Bob) \leftarrow \text{offChainBalance}(Bob) - y$
            $\text{onChainBalance}(Bob) \leftarrow \text{onChainBalance}(Bob) + y$
            remove channel from channels & entry from pendingClose($Alice$)
            if there is an ($fchid$, _, _) entry in pendingClose($Bob$) then
                remove it from pendingClose($Bob$)
            end if
        else if there is a tx in $\Sigma_{Alice}$ that is not a closing/commitment tx and spends the funding tx of the channel with ID $fchid$ then
            halt // DS forgery
        else if there is a commitment transaction in block of height $h$ in $\Sigma_{Alice}$ for open channel with ID $fchid$ with a balance that does not correspond to the receipt and the delayed output has been spent by the counterparty then
            if polls($Alice$) contains an entry in $[h, h + \texttt{delay}(Alice) - 1]$ then
                halt
            else
                negligent($Alice$) $\leftarrow$ true
            end if
        else if there is no such closing/commitment transaction $\wedge$ $h = \perp$ then
            assign largest block number of $\Sigma_{Alice}$ to $h$ of entry
        else if there is no such closing/commitment transaction $\wedge$ $h \neq \perp$ $\wedge$ (largest block number of $\Sigma_{Alice}$) $\geq h + (2 + r)$ windowSize then
            halt
        end if
    end for
    if $Alice$ has no open channels in $\Sigma_{Alice}$ AND negligent($Alice$) = false then
        if offChainBalance($Alice$) $\neq 0$ OR onChainBalance($Alice$) is not equal to the total funds exclusively spendable by $Alice$ in $\Sigma_{Alice}$ then
            halt
        end if
    end if
end function
```

Fig. 9.

POLL is a request that every player has to make to $\mathcal{F}_{\text{PayNet}}$ periodically (once every `delay` blocks, as set on registration) in order to remain non-negligent. In a software implementation, such a request would be automatically sent at safe time intervals. When receiving POLL (Fig. 10), $\mathcal{F}_{\text{PayNet}}$ checks the ledger for maliciously closed channels and halts in case of a forgery (l. 6) or in case of a successful malicious closing of a channel whilst the offended player was non-negligent (l. 11). If on the other hand a channel has been closed maliciously but the offended player did not POLL in time, she is marked as negligent (l. 13).

Functionality $\mathcal{F}_{\text{PayNet}}$ – poll

```
1:  Upon receiving (POLL) from $Alice$:
2:      send (READ) to $\mathcal{G}_{\text{Ledger}}$ as $Alice$ and store reply to $\Sigma_{Alice}$
3:      add largest block number in $\Sigma_{Alice}$ to polls($Alice$)
4:      checkClosed($\Sigma_{Alice}$)
5:      if $\exists$channel $\in \Sigma_{Alice}$ that contains $Alice$ and is closed by a tx that is not a
    commitment transaction then
6:          halt // DS forgery
7:      end if
8:      for all channels $\in \Sigma_{Alice}$ that contain $Alice$ and are maliciously closed by
    a remote commitment tx (one with an older channel version than the
    irrevocably committed one) in block with height $h_{\texttt{tx}}$ do
9:          if the delayed output (of the counterparty) has been spent then
10:             if polls($Alice$) has an element in $[h_{\texttt{tx}}, h_{\texttt{tx}} + \texttt{delay}(Alice) - 1]$ then

11:                 halt // $Alice$ wasn't negligent but couldn't punish
12:             else
13:                 negligent($Alice$) $\leftarrow$ true
14:             end if
15:         end if
16:     end for
17:     send (POLL, $\Sigma_{Alice}$, $Alice$) to $\mathcal{S}$
```

Fig. 10.

The last part of $\mathcal{F}_{\text{PayNet}}$ (Fig. 11) contains some additional "daemon" messages that help various processes carry on. PUSHFULFILL, PUSHADD and COMMIT are simply forwarded to $\mathcal{S}$. They exist because the "token of execution" in the protocol does not follow the strict order required by UC, and thus some additional messages are needed for the protocol to carry on. In other words, they are needed due to the incompatibility of the serial execution of UC and the asynchronous nature of LN.

FULFILLONCHAIN has to be sent by a multi-hop payment intermediary that has not been paid by the previous player off-chain in order to close the channel. The request is noted and forwarded to $\mathcal{S}$. getNews requests from $\mathcal{F}_{\text{PayNet}}$ information on newly opened, closed and updated channels.

Functionality $\mathcal{F}_{\text{PayNet}}$ – daemon messages

```
Upon receiving (pushFulfill, pchid) from Alice:
    send (pushFulfill, pchid, Alice, state, $\Sigma$) to $\mathcal{S}$

Upon receiving (pushAdd, pchid) from Alice:
    send (pushAdd, pchid, Alice, state, $\Sigma$) to $\mathcal{S}$

Upon receiving (commit, pchid) from Alice:
    send (commit, pchid, Alice, state, $\Sigma$) to $\mathcal{S}$

Upon receiving (fulfillOnChain) from Alice:
    send (read) to $\mathcal{G}_{\text{Ledger}}$ as Alice, store reply to $\Sigma_{Alice}$ and assign largest block number to $t$
    add $t$ to focs(Alice)
    checkClosed($\Sigma_{Alice}$)
    send (fulfillOnChain, $t$, Alice) to $\mathcal{S}$

Upon receiving (getNews) from Alice:
    clear newChannels(Alice), closedChannels(Alice), updatesToReport(Alice) and send them to Alice with message name news, stripping fchid and h from closedChannels(Alice)
```

Fig. 11.

References