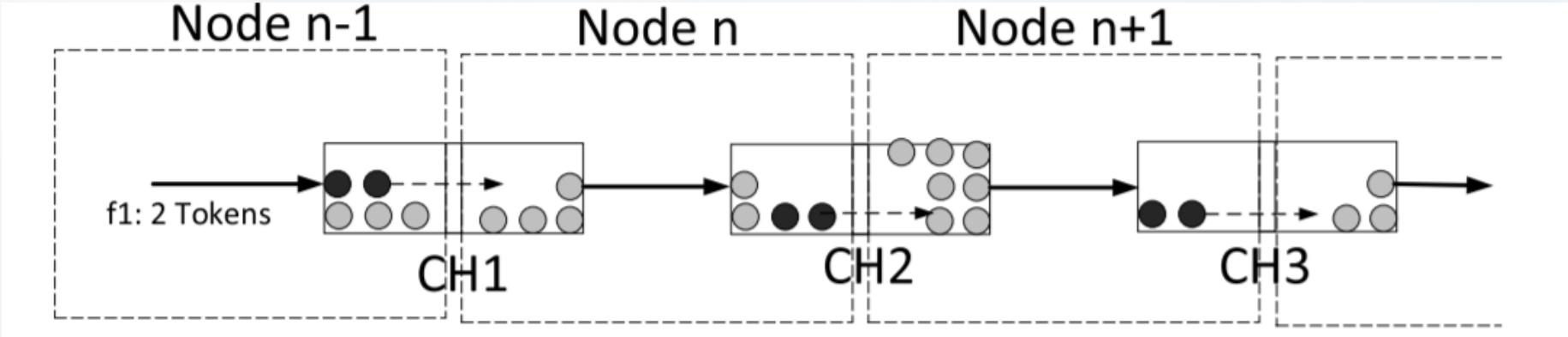
payGo: Incentive-Comparable Payment Routing Based on Contract Theory

Computer Network Research Lab

1 PayGo

Payment Channel Network(PCN)

- Payment Channel Network(PCN)에서 거래는 target이 거래를 받기 전까지 collateral이 발생
- multiple payment channels를 통해 거래를 진행할 경우 payer뿐만 아니라 거래를 도와준 Payment Service Provider(PSP)도 collateral이 발생
- PCN이 원활히 동작하기 위해서는 상기 PSP의 참여가 중요하기 때문에 incentive mechanism이 필요
- 현재는 PSP에게 incentive를 어떻게 부여할지 충분한 연구가 진행되지 않음.



1 PayGo

Contract Theory

- Incentive mechanism을 위해 Contract Theory 적용
- Contract Theory는 employer와 employee 간의 계약을 설계하기 위한 비대칭 정보를 가진 현실 경제에서 널리 사용

  * 정보 비대칭 : 일반적으로 employer가 employee의 특성을 정확히 알지 못한다는 사실을 말한다.

- PCN의 거래에서도 한 명의 payer만 존재하기 때문에 독점시장이 형성되고 payer는 service qualities에 따른 prices를 측정하여 계약 bundle를 제공하고 consumer hub node들이 상기 계약 중 하나를 선택하게 된다.
- 하지만 payer는 여러 consumer hub node들의 service qualities를 알지못하는 정보의 비대칭이 발생한다.
- 따라서 PCN incentive mechanism에 Contract Theory를 적용하여 이러한 비대칭 정보를 극복하고, 필요한 성과와 그에 상응하는 보상을 포함하는 계약을 제공함으로써 PSPs에게 효율적으로 인센티브를 줄 수 있다.

1 PayGo

Contract Theory

- 각기 다른 기여도를 가진 hub nodes의 타입 f를 $\theta_1, \ldots, \theta_F$ 로 분류

  ▷ hub nodes의 타입이 $\theta_1 < \ \ldots \ < \theta_F$로 정렬되어 있다고 가정

- Payer는 타입θ_f의 정보를 알지 못하는 정보의 비대칭 아래 계약 (π, q) -> (incentive, latency)을 작성

- Payer의 Utility는 아래의 공식과 같음.

$$B(\theta_f) = \arg\max_{q} U(\pi, q)$$
$$= V(q(\theta_f)) - \omega\pi(\theta_f), \quad \forall f \in F,$$

$$U_p = \sum_{f=1}^{F} p_f \left(\delta(D_f^{-1} - D_{max}^{-1}) - \omega\pi(\theta_f)\right)$$

- ω : price sensitivity parameter

- p_f : prior distribution of type-θ

- D_{max} : is maximum payment delay, payment delay가 lock time을 초과하면 utility는0

1 PayGo

Contract Theory

- Consumer는 자신의 utility를 최대화하는 계약을 선택

$$(\hat{\pi}(\theta_f), \hat{D}(\theta_f)) = \arg\max_{\pi, D} U_c(\pi, D)$$
$$= \theta_f \pi(\theta_f) - \omega' C(D_f^{-1})$$
$$\forall f \in F,$$

- 계약의 목표인 payer의 utility를 최대화하기 위해서 아래의 두 조건을 따른다.
  - Individual Rationality (IR) : U_c가 0보다는 커야 consumer가 계약을 선택

$$\theta_f V(\pi(\theta_f)) - \omega' C(D_f^{-1}) \geq 0, \quad \forall f \in F,$$

  - Incentive Compatible (IC) : 더 높은 type-θ 계약을 수행한 consumer가 더 높은 utility를 얻는다.

$$\theta_f V(\pi(\theta_f)) - \omega' C(D_f^{-1}) \geq \theta_f V(\pi(\theta_{f'})) - \omega' C(D_{f'}^{-1}),$$
$$\forall f, f' \in F, \quad f \neq f'.$$

1 PayGo

Contract Theory

$$\max_{(\pi,D)} U_p = \sum_{f=1}^{F} p_f \left(\delta(D_f^{-1} - D_{max}^{-1}) - \omega\pi_f\right),$$

s.t. IR constraint (11) and IC constraint (12)

$$\sum_{f}^{F} p_f \pi_f \leq \Pi$$

$$\forall f, f' \in F, \quad f \neq f'.$$

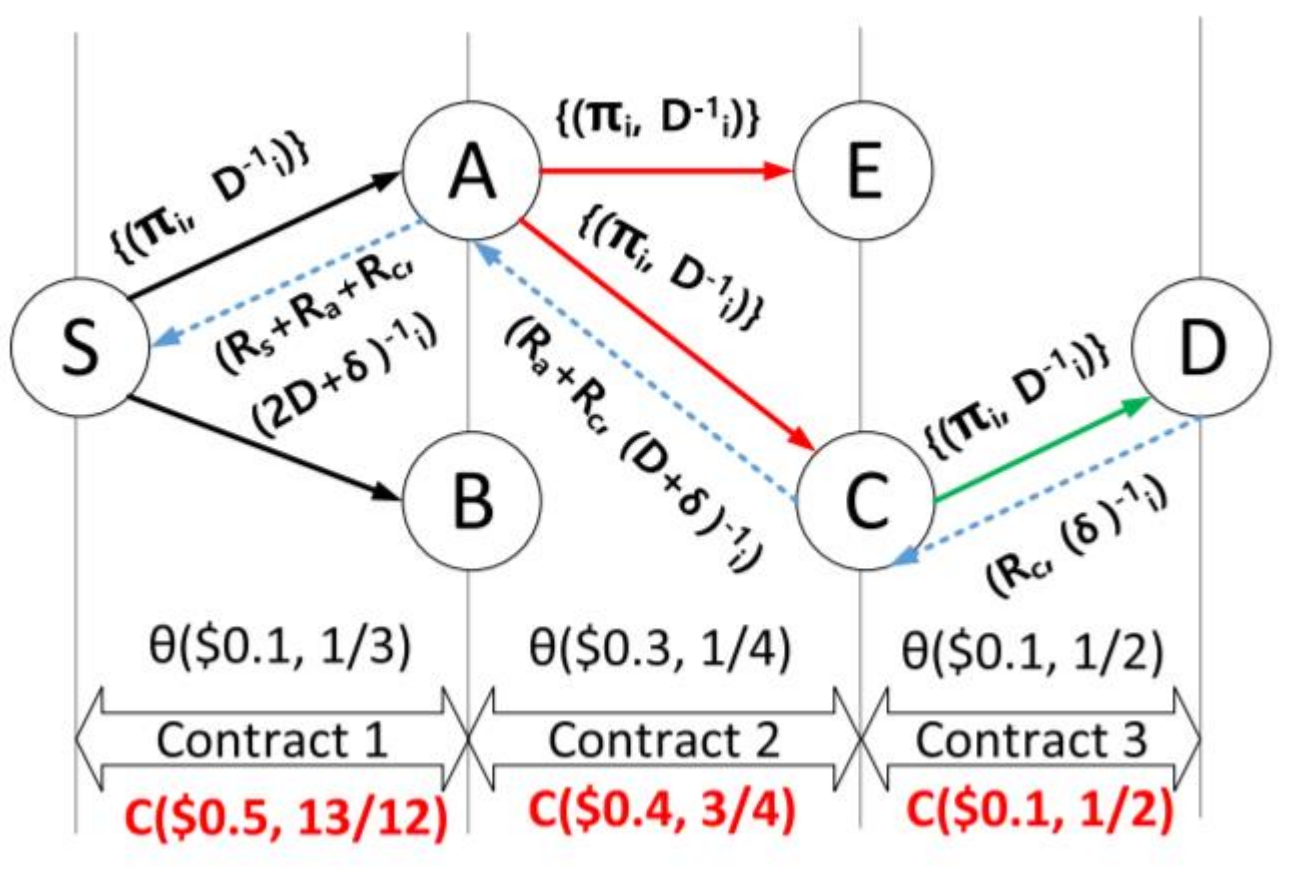
- Π : maximum fee로, blockchain transaction fee로 제한된다.

  따라서 incentive는 blockchain cost보다 작다.

1 PayGo

Contract procedure in PayGo

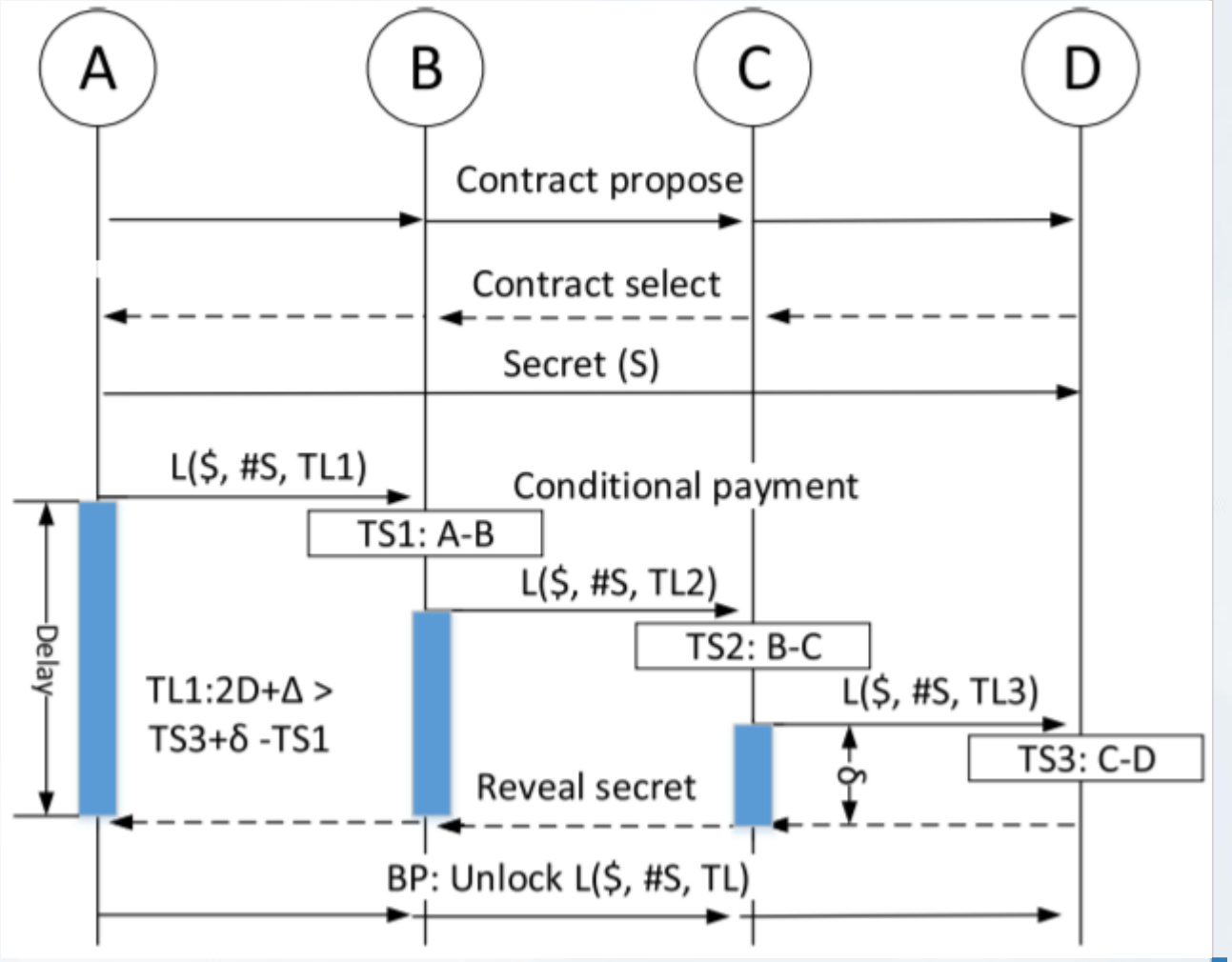
- Source node뿐만 아니라 중간 hub node도 contract theory를 바탕으로 계약을 만들어 next hop를 결정한다.
- Node C 또한 payee D에게 secret를 빠르게 받기 위해 minimum incentive로 single contract을 만들어 제공한다.
- Payee D까지 전달되면 계약을 선택하면서 payer S까지 올라오게 된다. 이때 incentive와 delay는 누적된다.



1 PayGo

Payment procedure in PayGo

- Locked Transfer ($amount, #S, expiry timer)
- TL : 잦은 on-chain access를 방지하기 위해 contract delay보다 높게 설정
- Locked Transfer를 받으면 contract과 start time를 확인
- Target은 Locked Transfer를 받으면 end time을 설정 (TS3 + σ)하고 Reveal Secret 통해 전달
- 상기 start time과 end time으로 delay 계산



Locked Transfer message

Field name	Description	Example of initial values
nonce	integer number	1
cr	contract round as a payment_identifier	1
expiration	T_{exp}	$T_{start} + D_s + \delta + \Delta$
next node	recipient	H_c
target	payee node	account address
source	payer node	account address
locksroot	a root hash of an updated merkle tree; add a leaf of a new transaction	leaves += #(M_{locked} , target, T_{exp} #S, π_s, π_a, T_{start})
secrethash	secret hash	#S
cumulative transferred_amount	$M_{trans,i}$ from node i	0
locked_amount	M_{locked}	$0 + M + \pi_s$
amount	payment amount	M
s_contract	a selected contract	$\{\pi_s, D_s\}, \quad s \in F$
a_contract	auxiliary contracts	$\{\pi_a, D_a\}$ $\forall a < s,\ a \in F$, s=selected contract
start_time	T_{start}	now

Listing 3. Solidity codes for calculating reward and rebuilding a merkle tree

```
    // get from leaf included in locksroot.
    function calculateLockedAmount (
    uint256~locked_amount, uint256~expiration, bytes32~secrethash, address
       target, uint256~start_time,
    uint256[] memory incentive_bundle, uint256[] memory delay_bundle) view
       internal returns(bool, uint256)
    {
          uint256~reveal_time;
          uint256~final_incentive;
          uint256~final_delay;
          bool unsettled;
          uint256~end_time = secret_registry.getEndtime(secrethash, target);

          if (msg.sender == target) {
               end_time = secret_registry.setEndtime(secrethash, target,
                  start_time);
          }
          reveal_time = secret_registry.getSecretRevealTime(secrethash, target);

          if (reveal_time == 0 || expiration <= reveal_time) {
               locked_amount = 0;
          }
          else if (end_time = 0) {
               unsettled = true;
          }
          else {
               final_delay = end_time-start_time;
               for (uint256~i=0; i<incentive_bundle.length; i++){
                    if (delay_bundle[i] >= final_delay) {
                         final_incentive = incentive_bundle[i];
                         break;
                    }
               }
               // incentive_bundle[0] = selected_incentive
               locked_amount = locked_amount - incentive_bundle[0] +
                  final_incentive;
          }
          return (unsettled, locked_amount);
    }
}
```

Q.A