(KOR) BiFi Smart Contract: Interest Model Design

1. Introduction

BiFi는 사용자가 여러 종류의 가상화폐를 예금할 수 있고, 예치된 가상화폐를 담보로 여러 종류의 가상화폐를 대출받을 수 있는 Ethereum Defi 프로젝트이다. 전통적인 은행과 같이 BiFi는 예금자들에게 이자를 제공하며 대출자에게 이자를 받으며 이자율은 시장의 유동성에 의해 자동적으로 결정된다. 만약 전체 예금량이 늘어나면 대출에 유리하도록 이자율이 줄어들고, 전체 대출량이 늘어나면 예금에 유리하도록 이자율이 늘어난다. 한편, 원리합계의 산출은 복리로 계산되며 서비스를 이용자들이 많아질 수록 블록당 복리에 가까워지도록 설계되었다.

2. Assumptions and Terminology

Assumptions

- 예금 및 대출 이자율은 시장 유동성에 의해 자동으로 결정된다.
- BiFi Smart Contract는 사용자에게 다음과 같은 4종류의 BiFi-Action(이하 action)을 지원한다
  - `deposit`: 가상화폐 예치 기능
  - `withdraw`: 예치금 인출 기능
  - `borrow`: 예치금을 담보로 가상화폐 대출 기능
  - `repay`: 대출 자산 상환 기능
- 대출 자산이 예금 자산의 일정 비율을 넘어서는 사용자의 예치금은 청산된다.

Terminology

Global Shared

- H_L: 다른 블록에서 최근 실행된 action의 Block Height
- E_D: 예치량 복리 연산을 위한 Exchange Rate의 누적 곱
- E_B: 대출량 복리 연산을 위한 Exchange Rate의 누적 곱
- D_T: 예치량의 총합
- B_T: 대출량의 총합

- $E_D[u]$: 사용자 u에게 할당된 예금 Exchange Rate(E_D)값
- $E_B[u]$: 사용자 u에게 할당된 대출 Exchange Rate(E_B) 값
- $D[u]$: 사용자 u의 예금 원리 합계
- $B[u]$: 사용자 u의 대출 원리 합계

Sahred in a block

- tH_L: H_L의 임시 변수
- tE_D: E_D의 임시 변수
- tE_B: E_B의 임시 변수

Constants

- $blocksPerYear$: 1년 동안 생성되는 Block 수 (e.g., 2,102,400 in Ethereum)
- r_{min}: 관리자가 지정한 최소 대출 이자율 (e.g., 2%)
- s: 관리자가 지정한 유동성 민감도

3. Calculation Method for Amount with Interest

BiFi implementation은 사용자에게 예치와 대출 기능을 제공하는 여러개의 "Token handlers"를 보유한다. 특히 BiFi가 지원하는 Token마다 deposit, withdraw, borrow and repay 기능을 제공하는 handler가 할당되며, 각 handler는 해당 token의 시장 상태 변수(D_T, B_T, D for users and B for users), 이자율 산출 변수(r_{min}, s)과 원리합계 산출 변수(H_L, E_D, E_B, E_D for users and E_B for users) 등을 저장한다. 이어지는 절에서 위 변수들을 활용하여 이자율을 산출하는 방법과 원리합계를 계산하는 방법을 서술한다.

3.1 Interest Rate Formula

D_T와 B_T 값에 의해 연간 대출 이자율(Borrow Interest Rate, BIR_y)과 연간 예금 이자율(Supply Interest Rate, SIR_y)이 결정된다. BiFi는 연간 이자율을 블록당 이자율(BIR_b, SIR_b)로 분할하여 활용한다.

Utilization

- 전체 유동성에서 대출량이 차지하는 비율을 의미한다.
- Utilization formula: $U = \frac{B_T}{D_T+B_T}$

Interest Rates

- An annual interest rates
  - BIR_y = $r_{min} + U * s$
  - SIR_y = $BIR_y * U$
- An interest rates per block
  - BIR_b = $\frac{BIR_y}{blocksPerYear}$
  - SIR_b = $\frac{SIR_y}{blocksPerYear}$

3.2 Calculating the Amount with Interest

예금과 대출 자산에 대한 이자율 산출 방법은 서로 다르지만 원리합계를 활용하는 방법은 동일하므로 블록당 이자율을 SIR_b과 BIR_b 대신 r이라고 표기하여도 일반성을 해치지 않는다. 아래첨자를 가진 r은 i-th block에서 산출한 블록당 이자율을 의미한다.

Amount with Interest

BiFi는 블록마다 복리 이자를 제공하므로 block i_1부터 block i_2까지 원금 P_{i_1}에 대한 원리합계 A_{i_2}는 다음과 같이 계산 된다.

$$A_{i_2} = P_{i_1} * \prod_{i=i_1+1}^{i_2} (1 + r_i)$$

임의의 시점에서 각 사용자에 대한 원리합계를 위와 같이 계산하려면 모든 블록에서의 r_i 값을 저장하는 것이 요구되어 Storage 비효율이 야기된다.

Exchange Rate (Cumulative Product of Interest Rates)

block i부터 block j까지 (1+r)의 누적곱, ExchangeRate$_{i,j}$를 다음과 같이 정의한다. BiFi smart contract는 `ExchangeRate` 변수를 가지며, 매 블록마다 (1+r)을 기존 `ExchangeRate`

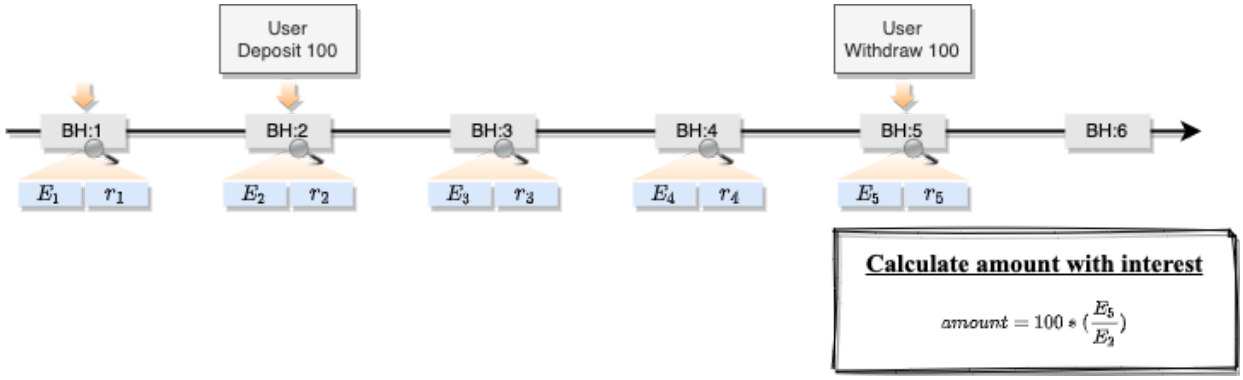
변수에 곱하여 처음부터 현재까지의 (1+r)을 누적하여 곱한 결과를 `ExchangeRate`에 저장한다.

$$Def)\ ExchangeRate_{i,j} = \prod_{i}^{j}(1 + r_i)$$

사용자는 BiFi에 예금이나 대출시점(i_1)에서 `ExchangeRate` 값을 할당 받고, 이자 가산 시점(i_2)의 ExchangeRate 값과 연산하여 A_{i_2}값을 산출한다. (1+r)의 누적곱을 이용하면 모든 블록의 (1+r)값을 저장할 필요가 없으므로 BiFi Storage가 절약되며 사용자의 transaction gas 요금을 줄이는 효과가 있다.

Example of Calculating Amount with Interest

다음 그림에서 E_i와 r_i는 i-th block의 `ExchangeRate`과 이자율이고 A_i는 i-th block에서 사용자의 원리 합계이다. 예를 들어 사용자가 `BH:2`에서 100 token을 예금했을때 `BH:5`에서 사용자의 원리합계는 다음과 같이 계산할 수 있다.

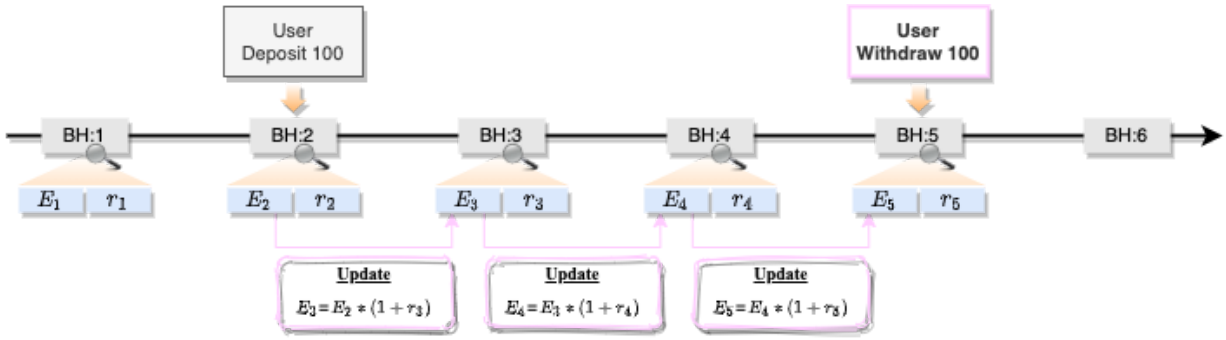


- `BH:5`에서 사용자의 원리 합계는 원금(100)에 (1+r_3), (1+r_4)과 (1+r_5)를 곱한 값이다.
- 한편, $\frac{E_5}{E_2} = (1 + r_3)(1 + r_4)(1 + r_5)$ 이므로 사용자의 원리합계는 원금(100)에 $\frac{E_5}{E_2}$를 곱한 값과 같다.

3.3 Updating ExchageRate

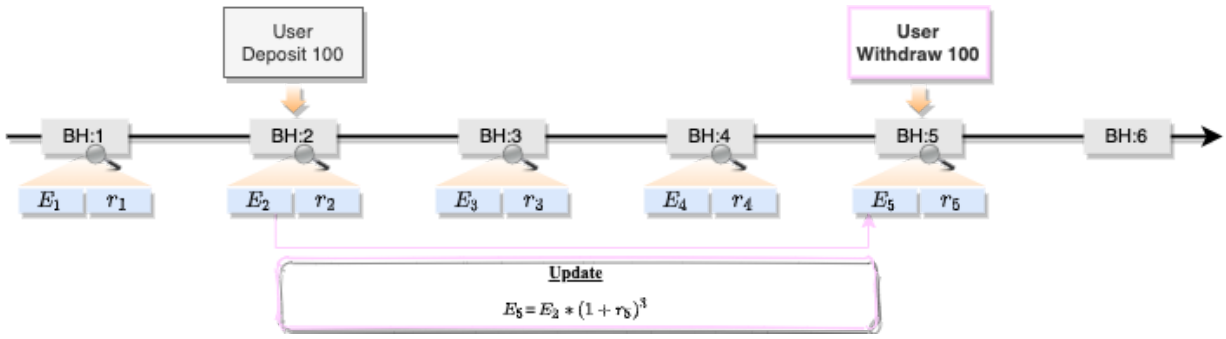
Basic Updating Logic

BiFi contract에 저장된 `ExchangeRate`는 다음 그림과 같이 매 블록마다 기존 값에 (1+r)을 곱하여 갱신한다.



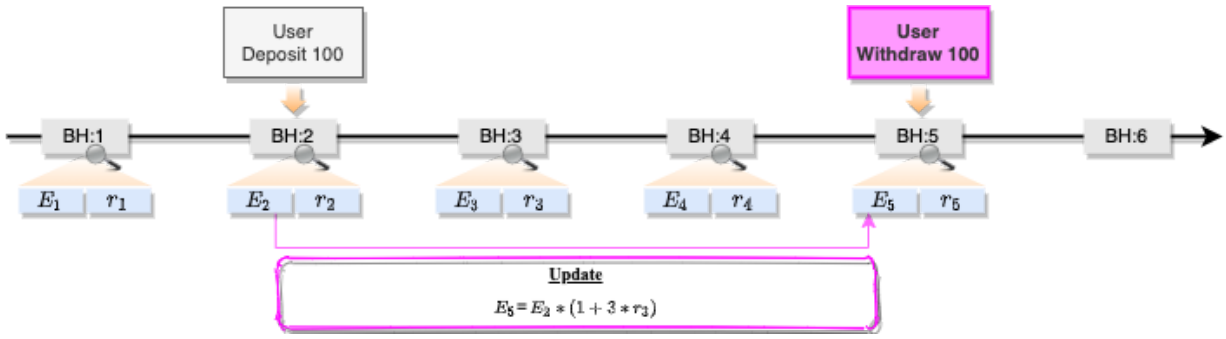
Batch overdue updates

BiFi contract의 state를 변경하는 사용자 transaction(이하 action)이 발생하지 않은 블록에서는 `ExchangeRate`을 업데이트할 수 없으므로, 현재 action은 이전 action 이후 부터 밀린 update를 일괄적으로 수행해야 한다. 한편, 이자율은 D_T와 B_T에 의해 변경되므로 사용자의 action이 없었던 block에서는 이자율이 동일하다 (r_3 = r_4 = r_5). 따라서 `BH:5`의 action은 (1+r_5)를 세번 곱하여 밀린 업데이트를 완료할 수 있다.



Appoximatation update

오랜 기간 BiFi에 사용자 action이 실행되지 않은 경우 밀린 ExchangeRate 업데이트에 많은 연산 비용이 요구 될 수 있다. action이 발생하지 않은 기간과 관계 없이 연산 효율성을 보장하기 위해 ExchangeRate 갱신 알고리즘을 $(1+r)^n$을 곱하는 대신, $(1+3*r)$을 곱하는 방식을 채택했다. 함수 $y=(1+r)^x$과 $y=1+rx$는 r값이 아주 작을때 거의 동일하게 행동하므로 update 연산을 위와 같이 변경하여도 거의 동일한 결과를 낸다.



3.4 Applying Liquidity Accurately

`ExchangeRate`는 다음과 같이 현재 유동성(D_T, B_T)를 기반으로 업데이트 된다.

1. 현재 D_T, B_T 기반 이자율 r 계산
2. 경과 시간 계산: $\delta =$ `block.number` $- H_L$, (`block.number`: global variable provided on solidity)
3. `ExchangeRate` 업데이트: $E_{next} = E_{prev} * (1 + \delta * r)$.
4. H_L 갱신: $H_L =$ `block.number`

위 프로세스는 한 블록에 action이 많아야 1개 담기는 경우에는 아무런 문제 없이 잘 동작한다. 하지만 한 블록에 여러 action이 포함된 경우에는 `ExchangeRate`이 시장 유동성을 제대로 반영하지 못하게 된다. 블록 내 첫번째 action을 제외한 나머지 action에서는 $\delta = 0$이므로 `ExchangeRate`이 업데이트 되지 않으므로 블록내 두번째 action부터는 같은 블록에 포함된 이전 action이 변경한 유동성을 반영할 수 없다. 결국 이전 모든 action이 누적한 유동성을 기반으로 이자를 산출하는 설계에 위배된다.

Solution: Using Shared Variable in a Block

위 문제를 해결하기 위해서는 블록내 모든 action들이 각기 자신의 이자율을 기반으로 `ExchangeRate`을 각각 업데이트 해야한다. 그렇기 위해서 각 action들은 이전 블록의 마지막 action이 저장한 `ExchangeRate`과 H_A 값을 알고 있어야 한다. 그래서 Bifi는 블록내 첫 action이 지난 `ExchangeRate`(tE_D, tE_B)과 H_L(tH_L)값을 임시 전역 변수에 저장하여 블록내 모든 action에게 알려줄 수 있도록 한다.

4. 청산

4.1 담보 대출 비율(Loan To Value Ratio)

담보 대출 비율(LTV)은 예치금액 대비 대출 금액의 비율로 사용자의 대출 상태를 나타낸다. 금융 서비스는 사용자의 LTV 값이 1이넘기 전에 청산을 실행해야 한다. 사용자의 LTV는 다음 몇가지 이유로 커질 수 있다.

- 예금에 사용된 가상화폐 가격의 하락
- 대출에 사용된 가상화폐 가격의 상승
- 시간이 지남에 따라 대출 원리 합계가 예금 원리 합계를 넘어서는 경우

4.2 청산 대상자와 청산 실행자

청산 대상자

Bifi는 LTV 값이 일정 수준 이상이되는 사용자는 "청산 대상자"로 정의되며 청산 대상자의 담보물은 청산될 수 있다.

청산 프로세스와 청산 실행자

한편, 중앙기관이 없는 금융 서비스는 청산을 스스로 수행할 수 없으므로 보상을 기반한 "청산 실행자"들의 경제를 마련해야 한다. 예를들어 LTV가 0.92 이상이 되는 사용자를 "청산 대상자"로 정의할 수 있고, 청산 실행자는 청산 대상자의 대출을 대신 상환하여 청산 대상자의 예금을 보상으로 얻는다. LTV가 0.92인 청산 대상자의 담보물을 청산한다면 청산 실행자는 담보물의 약 8%의 수익을 기대할 수 있다. 청산 실행자가 대신 상환해주는 비용이 92라면 얻을 수 있는 보상이 100이기 때문이다.