MuonSwap AMM design proposition

The goal is to have cross-chain liquidity, but one issue comes it how to incentive rebalance between chains of liquidity.
A solution that is proposed is to think of every tokenA in a virtual stablepool, every tokenB in another virtual stable pool, and use every token A vs every token B in a UNI-V2 computation.

We have 2 LP on chainA and chainB with token XX and YY.
XXa * YYa = ka
XXb * YYb = kb

Each chain has a weight, wa for chainA and wb for chainB, wher wa + wb=1, p and q are updated each week.
XXa/(XXa + XXb)/wa

Compared to uniswap contract

```
function quote(uint amountA, uint reserveA, uint reserveB) internal pure returns (uint amountB);
```

In the following, x is reservA, and y reserveB

a) User swap XXa for YYa

We input x = XXa and y=(XXa + XXb) * wa in the stablepool computation, spread1= (outputammount-x)/x
We input x= (XXa + XXb) * wb and y= XXb in the stablepool computation, spread2= (outputammount-x)/x
We input x= XXa + XXb and y= YYa + YYb in the stablepool computation, finalAmountOut = outrputAmount * (1 + spread 1 + spread2)

b) User swap XXa for XXb

We input x = XXa and y=(XXa + XXb) * wa in the stablepool computation, spread1= (outputammount-x)/x
We input x= (XXa + XXb) * wa and y= XXa in the stablepool computation, finalAmountOut = (outputammount-x)*spread1/x

c) User swap XXa for YYb

We input x = XXa and y=(XXa + XXb) * wa in the stablepool computation, spread1= (outputammount-x)/x
We input x= (YYa + YYb) * wb and y= YYb in the stablepool computation, spread2= (outputammount-x)/x
We input x= XXa + XXb and y= YYa + YYb in the stablepool computation, finalAmountOut = outrputAmount * (1 + spread 1 + spread2)

d) User add XXa + YYa (provide liquidity)
To incentive, users to add liquidity on the right chain.

We input x = XXa and y=(XXa + XXb) * wa in the stablepool computation, spread1= (outputammount-x)/x
We input x = YYa and y=(YYa + YYb) * wa in the stablepool computation, spread2= (outputammount-x)/x

LPammountReceived = LP * (1 + spread 1 + spread2)

e) User removes XXb + YYb (withdraw liquidity)
To incentive, users to remove liquidity on the right chain.

We input x = (XXa + XXb)* wb and y=XXb in the stablepool computation, spread1= (outputammount-x)/x

We input x =(YYa + YYb) * wb and y=YYb in the stablepool computation, spread2= (outputammount-x)/x

On the same principle as d)
TokenammountReceived = Token * (1 + spread 1 + spread2)

–
In the example I use XXa + XXb, but if there is a chain C, it is replace by XXa + XXb + XXc.