The Ratcheting AMM (“RAMM”) for Nexus Mutual

[V1.0]

Reinis Melbardis, Anatol Prisacaru, Miljan Milidrag

Summary

The following is a proposed upgrade to the Nexus Mutual protocol with the following aims:

- Bring the price of NXM used by the system and the open market price together again;
- Allow members who wish to exit to do so directly from the protocol at a reasonable price;
- Allow the protocol to capture capital when members wish to provide it; and
- Create positive value for long-term aligned members.

The proposed mechanism is named the *Ratcheting Automated Market Maker (“RAMM”)*, which adapts the constant-product *Automated Market Maker (“AMM”)* mechanism, specifically Uniswap v2.[i]

We propose replacing the current *Bonding Curve* mechanism used for minting and redeeming NXM tokens with the *RAMM*.

We expect to implement and launch the mechanism in two stages: with Stage 1 covering the current scenario where the Mutual is highly capitalised; and Stage 2 adding functionality for the low capitalization scenario when the *Capital Pool* approaches the *Minimum Capital Requirement (“MCR”).*

This version of the whitepaper covers Stage 1 only, with the full Stage 2 mechanics to be added in a later version of the document.

Operation

In summary, the proposed mechanism would work as follows:

- The *MCR Floor* is removed and the *MCR* is driven by the *Total Active Cover Amount* going forward.
- Every time NXM is minted through the *RAMM* smart contract, the price goes up.
- Every time NXM is redeemed through the *RAMM* smart contract, the price goes down.
- This is achieved via two internal *AMM* pools, with the Mutual itself as the only (automated) liquidity provider.
- Minting NXM through the *RAMM* smart contract is disabled below *Book Value* to avoid dilution of other members.

- Redeeming NXM through the *RAMM* smart contract is disabled above *Book Value* to avoid dilution of other members.
- To enable price discovery, there is a ratchet mechanism that moves the spot NXM prices towards the *Book Value* over time from below for the *Below Pool* and from above for the *Above Pool*.
- ETH liquidity is allocated to the *RAMM* pools on an ongoing basis.
    - If liquidity is below a defined *Target Liquidity*, liquidity is added to the pools as long as *Capital Pool* exceeds the MCR.
    - If liquidity is above a defined *Target Liquidity*, liquidity is withdrawn from the pools.
    - Both the *Above Pool* and *Below Pool* use a single liquidity value, and interactions in each pool also affect the liquidity in the other pool.
- An *Internal Price* would be used for interactions with the system that aren't directly related to swapping ETH for NXM, such as valuing NXM in ETH/stablecoins for capacity calculations, staking rewards, and buying cover in NXM. The *Internal Price* is calculated using two separate *Time-Weighted Average Prices ("TWAPs")* based on the spot prices of the *Above Pool* and *Below Pool*, which are then combined to achieve the final *Internal Price*.
- There needs to be a small buffer around *Book Value*, the *Oracle Buffer*, where no swaps are allowed, in order to avoid oracle-related arbitrage opportunities.

TABLE OF CONTENTS

The Ratcheting AMM ("*RAMM*")

Proposed Mechanism Introduction

We propose removing the *Minimum Capital Requirement*[ii] *("MCR") Floor* of 162,425 ETH and have the *MCR* value be driven by actual risk exposures. In conjunction, we also propose decoupling the system NXM price from the *MCR* value and instead let it be set (mostly) by supply and demand by means of an internal *Automated Market Maker ("AMM")* pool.

We propose that Nexus Mutual replaces the current minting and redemption system (i.e., the *Bonding Curve* mechanism) with one where two virtual internal one-sided Uniswap v2-style[iii] pools determine the price at which the Mutual is willing to allow NXM redemptions in value-accretive ranges, complemented by a price ratchet to enable price discovery.

Specific Mechanism Goals

The proposed mechanism has the following aims:

- Bring the price of NXM and the open market valuation together again;
- Allow members who wish to exit to do so directly from the protocol at a reasonable price;
- Allow the protocol to capture capital when members wish to provide it; and
- Create positive value for long-term aligned members.

Operation Summary

In summary, the proposed mechanism would work as follows:

- The *MCR Floor* is removed and *MCR* is driven by the *Total Active Cover Amount* going forward. The current on-chain formula is:

$$MCR = \frac{Total\ Active\ Cover\ Amount}{4.8}$$

- Every time NXM is minted through the *RAMM* smart contract, the spot price increases.
- Every time NXM is redeemed through the *RAMM* smart contract, the spot price decreases.
- This is achieved via internal *AMM* pools, with the Mutual itself as the only (automated) liquidity provider.
- The minimum price for minting NXM from the protocol in exchange for ETH is at *Book Value ("BV")*, where:

$$BV = \frac{Capital\ Pool\ value\ in\ ETH}{NXM\ Supply}$$

This is to avoid dilution of other members. Therefore, the *Above Pool* operates in the price range:

$$spot_a \geq BV * (1 + oracleBuffer)$$

- The maximum price for redeeming NXM to the protocol in exchange for ETH is at *Book Value* to avoid dilution of other members. Therefore, the *Below Poo*l operates in the price range:

$$0 \leq spot_b \leq BV * (1 - oracleBuffer)$$

- To enable price discovery, there is a ratchet mechanism that gradually moves the *Above Pool* and *Below Pool* spot prices towards the *Ratchet Target Values*:

$$Ratchet\ Target\ Values = BV * (1 \pm oracleBuffer)$$

- Liquidity is allocated to the *RAMM* pools on an ongoing basis.
    - If $liq < liq_{target}$, liquidity is added to the pool as long as:

      $Capital\ Pool > MCR + liq_{target}$
    - If $liq > liq_{target}$, liquidity is withdrawn from the pool.
- An *Internal Price (“*ip*”)* would be used for interactions with the system that aren't directly related to swapping NXM for ETH, such as valuing NXM in ETH/stablecoins for capacity calculations[iv], staking rewards[v] and buying cover[vi] in NXM. The *Internal Price* is calculated using two separate *Time-Weighted Average Prices (“TWAPs”)*, $twap_a$ and $twap_b$ based on the spot prices of the *Above Pool* and *Below Pool*, which are then combined to achieve the final *Internal Price*.
- There needs to be a small buffer around *Book Value*, where no swaps are allowed, in order to avoid oracle-related arbitrage opportunities. This is the $oracleBuffer$.

Expected Outcomes of implementing the *RAMM*

Positives

- *Book Value* increases as a result of each swap by definition.
- Capital is captured at prices above *Book Value* and removed from the Mutual below *Book Value*.
- Creates an automatic, efficient buyback mechanism for value-accretive NXM burning.
- Removes the direct dependency between capitalisation levels and the price of the token, decoupling market forces and the Mutual's liability risk management.
- Open market to internal valuation price gap closed via arbitrage opportunities. This allows the Mutual to price risk and manage capacity at market-consistent values.
- Via liquidity and ratchet parameter setting, a variety of outcomes can be achieved in the long term. For example, it is possible to either follow the open market valuation of the

Mutual or act as a price maker by providing significant liquidity at a certain level (e.g., initially providing a lot of exit liquidity up to *Book Value* for those who want to exit now).

Negatives

- Mental anchoring to *Book Value* when it is the ratchet target, especially with regard to the downward ratchet when NXM price > *Book Value*. Possible to manage this by setting appropriate parameters (e.g., lower *Target Liquidity*).
- High number of parameters leads to potential complexity and outcomes difficult to predict, especially immediately after launch.
- Price and capital capture outcomes highly dependent on market forces, including the open market external to the Mutual.

Stage 1 *RAMM* Mechanism Operation Details

MCR Floor

Remove the *MCR Floor* parameter which was set at 162,425 ETH.

After implementation:

$$MCR = f(Cover) = \frac{Total\ Active\ Cover\ Amount}{4.8}$$

Two Pools

We propose replacing the existing *Bonding Curve* with two *RAMM* pools:

1. An Above Pool, which mints NXM tokens and receives ETH.
    - Takes in ETH deposits and distributes NXM.
    - Has an NXM price floor equal to $(1 + oracleBuffer) * BV$, which means no NXM can be minted from the protocol by providing capital below this price.
    - Price is driven up instantaneously by users minting NXM.
    - Price reduces towards the price floor over time via the ratchet mechanism with $ratchetTarget_a = (1 + oracleBuffer) * BV$.
    - ETH Liquidity in the pools increases when users deposit ETH via the *Above Pool*.
2. A *Below Pool*, which burns NXM tokens and distributes ETH.
    - Takes in NXM and distributes ETH.
    - Has an NXM price ceiling equal to $(1 - oracleBuffer) * BV$, which means no NXM can be redeemed from the protocol above this price.
    - Price is driven down instantaneously by users redeeming NXM.

- Price increases towards the price ceiling over time via the ratchet mechanism with $ratchetTarget_b = (1 - oracleBuffer) * BV$.
- ETH Liquidity in the pools decreases when users withdraw ETH via the *Below Pool*.

Parameters

The following section summarises the list of parameters affecting the RAMM spot prices. The example parameters below have been discussed by the community as opening parameters, but are still subject to an on-chain governance vote for approval.

Parameter	Description	Example Value
$Book\ Value$ ("BV")	$\frac{Capital\ Pool\ value\ in\ ETH}{NXM\ Supply}$	0.0217 ETH
liq	ETH liquidity in the pools	5,000 ETH
NXM_a	Notional NXM reserve in the Above Pool	174,216.03 NXM
k_a	Above Pool invariant equal to $liq * NXM_a$	871,080,150
NXM_b	Notional NXM reserve in the Below Pool	526,315.79 NXM
k_b	Below Pool invariant equal to $liq * NXM_b$	2,631,578,950
$spot_a$	Current price at which members can swap ETH for NXM. Equal to $\frac{liq}{NXM_a}$	0.0287 ETH
$spot_b$	Current price at which members can swap NXM for ETH. Equal to $\frac{liq}{NXM_b}$	0.0095 ETH
liq_{target}	Target ETH liquidity	5,000 ETH
$liqSpeed_{out}$	Max amount of ETH that is removed from the pools daily as long as $liq > liq_{target}$	100 ETH
$initialBudget$	Amount of ETH that needs to be injected before the $liqSpeed_{in}$ and $ratchetSpeed_b$ parameters change from Initial State to Long-term State	43,835 ETH

Parameter	Description	Example Value
$fastLiqSpeed_{in}$	Initial State: max amount of ETH that is added to the pools daily. This value is active until an amount of ETH equal to $initialBudget$ has been injected into the pools	1,500 ETH
$liqSpeed_{in}$	Long-term State: max amount of ETH that is added to the pools daily. This value becomes active after an amount of ETH equal to $initialBudget$ has been injected into the pools	100 ETH
$ratchetTarget$	Middle value towards which the $spot$ prices move over time	BV
$oracleBuffer$	Margin to allow for oracle lag when calculating $Book\ Value$ in ETH. Secondary function – create spread.	1%
$ratchetTarget_a$	Value towards which $spot_a$ moves over time	$(1 + oracleBuffer) * ratchetTarget$
$ratchetTarget_b$	Value towards which $spot_b$ moves over time	$(1 - oracleBuffer) * ratchetTarget$
$ratchetSpeed_a$	Daily decrease in $spot_a$ when above $ratchetTarget_a$	$4\%\ of\ ratchetTarget$
$fastRatchetSpeed_b$	Initial State: Daily increase in $spot_b$ when below $ratchetTarget_b$	$50\%\ of\ ratchetTarget$
$ratchetSpeed_b$	Long-term State: Daily increase in $spot_b$ when below $ratchetTarget_b$	$4\ \%\ of\ ratchetTarget$

Operation of the RAMM pools

This section describes the operation of the *RAMM* pools, with some examples.

Note that both the ETH liquidity and the NXM reserves in the pools can be thought of as virtual and act mostly to establish price and price impacts as a result of swaps. The *RAMM* smart contract itself holds no ETH or NXM tokens—the ETH is held in the *Capital Pool* and the NXM

reserves are a balancing item. NXM is burned through member redemptions, even though it increases the number of notional NXM in the *Below Pool*. NXM is only actually added to the total supply via the *RAMM* when members mint tokens through depositing ETH via the *Above Pool*.

Member Mint Swap

- User specifies number of ETH they want to swap for new NXM, n.
- Now there will be $newLiq = liq + n$ ETH in the pools.
- The invariant doesn't change, so $newNXM_a = \frac{k_a}{newLiq}$
- Member receives $x = NXM_a - newNXM_a$ NXM for their n ETH.

Note that in order to maintain the price constant in the *Below Pool* at the same time, the NXM *Below Pool* reserve (and invariant) will be updated, so $newNXM_b = \frac{spot_b}{newLiq}$ and $newK_b = newLiq * newNXM_b$.

Above Pool Ratcheting Down

- $newSpot_a = max(spot_a - ratchetSpeed_a * days\ since\ last\ Above\ Pool\ swap, ratchetTarget_a)$
- The downward ratcheting of price is achieved by increasing NXM_a while keeping liq constant.
- Throughout the time period between member mints, n NXM is added to the NXM reserve in the *Above Pool*, where $n = \frac{liq}{newSpot_a} - NXM_a$.
- The new number of NXM in the virtual pool becomes $newNXM_a = NXM_a + n$.
- The invariant changes to $newK_a = liq * newNXM_a$.
- The price that members can now obtain per NXM, $\frac{liq}{newNXM_a}$ has decreased.

Member Redemption Swap

- User specifies number of NXM they want to redeem, n.
- Now there are $newNXM_b = NXM_b + n$ NXM in the *Below Pool*.
- The invariant doesn't change, so the ETH liquidity becomes $newLiq = \frac{k_b}{newNXM_b}$
- Member receives $x = liq - newLiq$ ETH for their n NXM.

Note that in order to maintain the price constant in the *Above Pool* at the same time, the NXM *Above Pool* reserve (and invariant) will be updated, so $newNXM_a = \frac{spot_a}{newLiq}$ and $newK_a = newLiq * newNXM_a$.

Below Pool Ratcheting Up

- $newSpot_b = min(spot_b + ratchetSpeed_b * days\ since\ last\ Below\ Pool\ swap, ratchetTarget_b)$
- The upward ratcheting of price is achieved by decreasing NXM_b while keeping liq constant.
- Throughout the time period between member mints, n NXM is removed from the NXM reserve in the *Below Pool*, where $n = NXM_b - \frac{liq}{newSpot_b}$
- The new number of NXM in the virtual pool becomes $newNXM_b - n$.
- The invariant changes to $newK_b = liq * newNXM_b$
- The price that members can now obtain per NXM, $\frac{liq}{newNXM_b}$ has increased.

Liquidity Injection

- ETH liquidity in the pools is liq and *Target Liquidity* is liq_{target}.
- If $liq < liq_{target}$ and $Capital\ Pool > MCR + liq_{target}$, increase the liquidity $newLiq = min\big(liq + liqSpeed * days\ since\ last\ observation, liq_{target}\big)$.
- To keep spot prices constant, update the NXM reserves:
  - $newNXM_a = \frac{spot_a}{newLiq}$
  - $newNXM_b = \frac{spot_b}{newLiq}$
- Update the invariants to allow for new liquidity and reserve parameters:
  - $newK_a = newNXM_a * newLiq$
  - $newK_b = newNXM_b * newLiq$

Liquidity Removal

- ETH liquidity in the pools is liq and and *Target Liquidity* is liq_{target}.
- If $liq > liq_{target}$ decrease the liquidity $newLiq = max\big(liq - liqSpeed_{out} * days\ since\ last\ observation, liq_{target}\big)$.
- To keep spot prices constant, update the NXM reserves:
  - $newNXM_a = \frac{spot_a}{newLiq}$
  - $newNXM_b = \frac{spot_b}{newLiq}$

- Update the invariants to allow for new liquidity and reserve parameters:
    - $newK_a = newNXM_a * newLiq$
    - $newK_b = newNXM_b * newLiq$

Implementation Note

For the purposes of implementation in Solidity, the ratcheting and liquidity functions are likely to be combined into a single update of price and liquidity. This function will be called and the parameters updated every time there is a swap or every time there is a request by the system of the *Internal Price*.

Initial vs Long Term State Parameters

A short-term aim of the system upgrade proposed here is to enable a reasonable amount of exit liquidity for those members who have been unable to redeem their NXM directly from the Mutual as a result of the high *MCR Floor* employed by the current *Bonding Curve* mechanism.

To achieve this aim, the implementation differentiates between two parameters used in the Initial State and the Long-term State. These parameters are:

- $fastLiqSpeed_{in}$ (Initial State) and $liqSpeed_{in}$ (Long-term State).
- $fastRatchetSpeed_b$ (Initial State) and $ratchetSpeed_b$ (Long-term State).

The Initial State parameters will be used at launch. There is also a fixed ETH amount, $initialBudget$, which is depleted at the same rate as liquidity is added to the pools via the *Liquidity Injection* mechanism. Once depleted, the two parameters will switch between Initial State and Long-term State.

Internal Price

Need for an *Internal Price*

Cover and cover fees are denominated in ETH/DAI, but staking rewards and open capacity are denominated in NXM, so there is need for a manipulation-resistant NXM price that is used by the system.

This metric would not affect the price at which members can mint and redeem NXM from the protocol, but would instead be used to:

- Calculate capacity opened up for covers as a result of NXM staking;
- Act as a conversion rate for buying covers using NXM; and
- Act as a conversion rate for rewards assigned to NXM stakers resulting from cover buys.

Parameters

The following section summarises the list of parameters for the *RAMM*-derived *Internal Price*.

Parameter	Description
$twap_a$	Time-weighted average price of the *Above Pool*
$twap_b$	Time-weighted average price of the *Below Pool*
$cumulativePrice_a$	Cumulative price of the *Above Pool*
$cumulativePrice_b$	Cumulative price of the *Below Pool*
$observation$	A single observation contains the cumulative price for each pool and the timestamp for when the cumulative price sample was taken
$granularity$	Number of observations stored at one time by the $twap$ mechanism
$obsIndex$	Index of the observation where each sample for the $twap$ is stored
$periodSize$	Size of time period over which the $twap$ observations are taken
$currentTimestamp$	Timestamp of the current block, in the same units as $periodSize$
p_a	*Above Pool* price allowing for the spot price, calculated as $p_a = min(twap_a, spot_a)$
p_b	*Below Pool* price allowing for the spot price, calculated as $p_b = max(twap_b, spot_b)$
$ipFloor$	Used to set a floor for the *Internal Price*, expressed as a percentage of *Book Value*
$ipCeil$	Used to set a ceiling for the *Internal Price*, expressed as a percentage of *Book Value*
$Internal\ Price$ ("ip")	Final *Internal Price* used by the system, calculated as $ip = max\left(min(p_a + p_b - BV, ipCeil * BV), ipFloor * BV\right)$

Proposed *Internal Price* mechanism

We propose an *Internal Price* which:

- Calculates *TWAP*s based on the trades in the *Above Pool* – $twap_a$ – and the *Below Pool* – $twap_b$ – over a significant period of time.

- Sets the spot prices as limits on the *TWAPs* to incorporate the ratchet and avoid arbitrage opportunities (e.g., members immediately redeeming their NXM rewards at a higher price than was used to mint them).
    - $p_a = min(twap_a, spot_a)$
    - $p_b = max(twap_b, spot_b)$
- Determines the final price used by the system by effectively picking between the *Above Pool* and *Below Pool* depending on which one is being actively used.

  The formula used for this is $ip = p_a + p_b - BV$.

  If a pool isn't actively being used for swaps for a significant amount of time its p_i will sit at the ratchet target level, so will cancel out with the BV term.

  If both pools have had recent active trades, this formula also effectively finds a weighted average between p_a and p_b where the weightings are determined by the distance of each p_i from *Book Value*.

Internal Price Floor and Ceiling

We propose a hard floor and ceiling for the *Internal Price*. Having a floor and ceiling provides a layer of additional protection against manipulation of the *Internal Price*.

These are proposed to be multiples of *Book Value*.

In addition, even if the actual market price sits outside of the (floor, ceiling) range, the floor and ceiling values would increase available cover capacity in low-price scenarios and limit capacity in high-price scenarios. This works in favour of the Mutual in extreme scenarios, as:

- There is still capacity to buy cover even if NXM price is very low, enabling the Mutual to carry on its main purpose; and
- Capacity opened up by temporary extreme upwards price movements is restrained, dampening the exposure the Mutual can take on during these scenarios.

TWAP Technical Implementation Methodology

The oracle model is conceptually similar to the Uniswap v2 oracle[vii] and is based on the same principle. The major difference is the embedded sampling mechanism.

At all times, the *RAMM* contract holds $granularity = 3$ samples, each an $observation$. Each $observation$ contains a $cumulativePrice_a$, $cumulativePrice_b$ and the timestamp for when the sample was taken.

The $cumulativePrice_i$ are calculated the same way as explained in the Uniswap v2 oracle docs.[viii]

The observations are stored in an array and are referenced by their $obsIndex$. The $obsIndex$ is calculated as:

$$obsIndex = ceil\left(\frac{currentTimestamp}{periodSize}\right) mod\ granularity$$

where $granularity$ is equal to the total number of $observations$ stored.

Each time there is a swap, a lookup for the ip or a trackable NXM supply change:

- liq, NXM_a and NXM_b are updated to reflect the liquidity injection/removal and the ratchet, and $spot_a$ and $spot_b$ are calculated.
- The new $cumulativePrice_i$ are calculated and recorded as the new sample in the relevant $observation$.
- The other $observations$ are updated up to the latest second of their period.

Consulting the oracle to obtain $twap_a$ and $twap_b$ is done by first calculating the current $cumulativePrice_i$ and subtracting the $cumulativePrice(i-2)$ from the $observation$ from two (2) periods ago.

The time-weighted average prices are both calculated as:

$$twap = \frac{cumulativePrice_i - cumulativePrice_{i-2}}{seconds\ between\ observation_i\ and\ observation_{(i-2)}}$$

Given that previous $observations$ are always updated to the latest second, the resulting $twap$ will compare the $cumulativePrice_i$ for up to $2 * periodSize$ in the worst case scenario and $1 * periodSize$ in the best case scenario.

Note on Further Work – Stage 2 implementation

The mechanism described in this document represents the Stage 1 implementation of the *RAMM*, and is designed with the current situation in mind, where the Mutual is highly capitalised relative to the *MCR*.

However, the Stage 1 implementation alone could become problematic as the *Capital Pool* approaches *MCR* (or, indeed, the *MCR* grows to reach the *Capital Pool*). A solution to the potential issues is intended to be implemented as part of the Stage 2 *RAMM* design. A draft adaption of the *RAMM* exists for the range where *Capital Pool* is close to *MCR*, but will be added to a new version of this document as it becomes fully refined and implemented.

The rationale for required modifications is presented in some more detail below.

Rationale and Goal of the Stage 2 Implementation

The *RAMM* Stage 1 solution generally treats *Book Value* (*Capital Pool* / NXM Supply) as:

- The line below which swapping ETH for NXM is not allowed, as the Mutual would be allowing minting of NXM at a price where *Book Value* would decrease and existing members would be diluted.
- Below this line, members can still redeem their NXM for ETH, as the Mutual is happy to allow value-accretive redemptions NXM from members.
- To enable price discovery with the market below this line, the price at which the protocol lets members redeem NXM for ETH moves automatically towards the *Book Value* ceiling.
- This price discovery works well as long as there is sufficient ETH liquidity in the pool so that the *AMM* price movements from members redeeming and the speed of the price ratchet balance each other out to find the right market price.

However, if the amount of capital we have approaches the *MCR* (i.e., the amount the Mutual needs to reliably pay claims), liquidity stops getting injected into the pools.

With low liquidity, it's possible that users stop trading with the Mutual and the *Internal Price* gets pinned to $ratchetTarget_b$, while the market price is below that. This could result in NXM price decoupling from market price, which would again reduce the Mutual's ability to attract capital and cause issues with cover and capacity mispricing.

The Stage 2 *RAMM* adaptation will aim to address the potential discrepancy between the *RAMM* spot price and open market price in this scenario.

---

[i] https://docs.uniswap.org/contracts/v2/concepts/protocol-overview/how-uniswap-works
[ii] https://docs.nexusmutual.io/protocol/capital-pool/mcr
[iii] https://docs.uniswap.org/protocol/V2/concepts/protocol-overview/how-uniswap-works
[iv] https://docs.nexusmutual.io/protocol/capacity/
[v] https://docs.nexusmutual.io/protocol/staking/
[vi] https://docs.nexusmutual.io/protocol/cover/
[vii] https://docs.uniswap.org/contracts/v2/concepts/core-concepts/oracles
[viii] https://docs.uniswap.org/contracts/v2/concepts/core-concepts/oracles