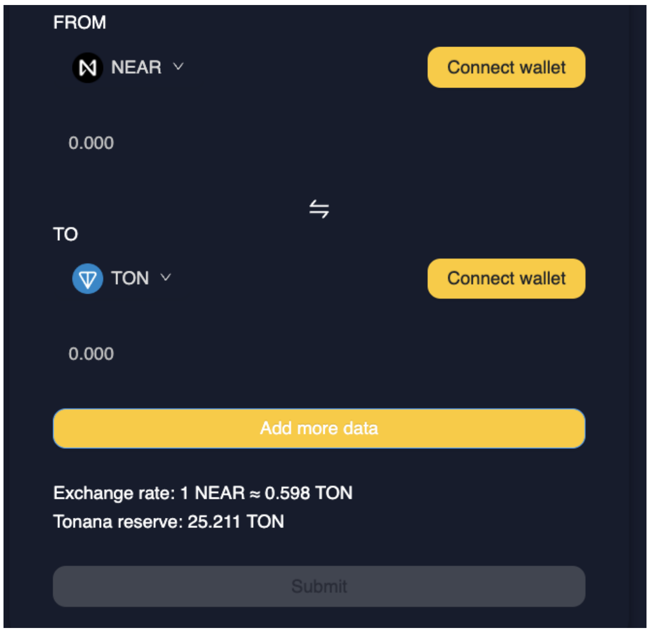
Report on the progress of Milestone #2

TON x NEAR data transfer implementation

As per the assigned task, we have successfully completed the implementation of the TON x NEAR data transfer bridge.

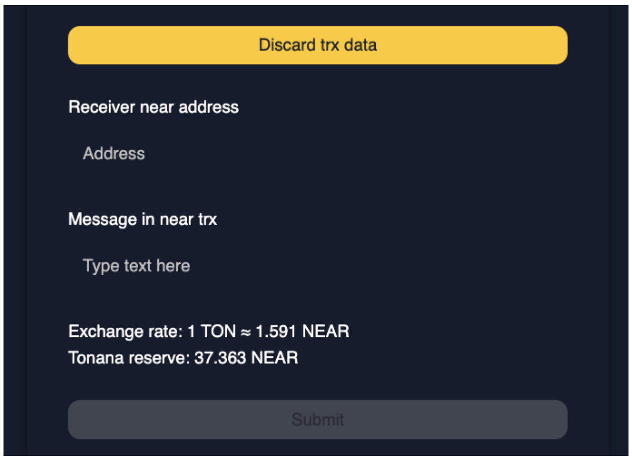
Based on this prototype, other developers can build custom bridge solutions between NFT, GamiFi, DeFi, and DAO projects on Near and TON.

We are working on a common technology, which means that the number of blockchains that can connect to TON in this way will only grow.



Exchange rate: 1 NEAR ≈ 0.598 TON
Tonana reserve: 25.211 TON

This bridge allows for the secure transfer of data between the TON and NEAR blockchain platforms, enabling interoperability between the two networks.



Exchange rate: 1 TON ≈ 1.591 NEAR
Tonana reserve: 37.363 NEAR

User Manual for TON x NEAR Data Transfer Bridge

The TON x NEAR data transfer bridge allows for the secure transfer of data between the TON and NEAR blockchain platforms.

With this bridge, users can easily send or swap tokens between the two networks, as well as transfer other types of data such as **messages between wallets and smart contracts**.

In order to use the TON x NEAR data transfer bridge, follow these steps:

- First, access the data transfer bridge interface: app.tonana.org
- In the data transfer form, select NEAR and TON
- Near the "From" field, push connect wallet button of the massage source blockchain
- In the "Receiver address", enter the address of the wallet or contract that you wish to send the data to on the target blockchain
- The next step is to enter the information you want to send between blockchains in the “Message in trx”
- Review the details of your transfer to make sure everything is correct, and then submit the form to initiate the transfer
- Wait for the transfer to be completed, this may take a few minutes

Developer Manual for TON x NEAR Data Transfer Bridge

NEAR → TON

NEAR Source chain. To send a transaction with data:

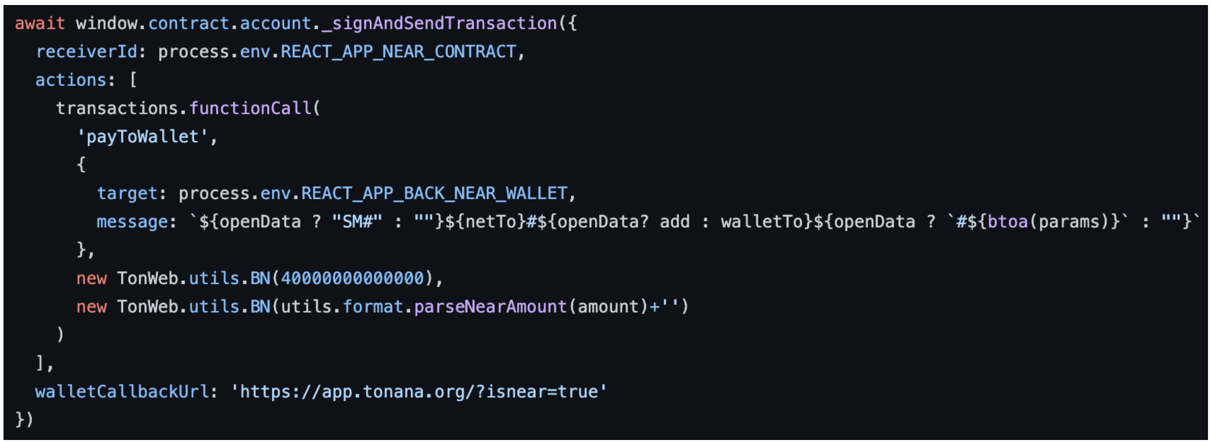
To transfer data from NEAR to TON, you must call the **payToWallet** function on the tonana.near contract. The first parameter is the target. Here you should specify **tonanawallet.near** (the bridge jack from which the transfer will be tracked).

The second parameter is the message. If you want to make a transfer and put the data in, you have to add **"SM#"** as a flag at the beginning of the message line. Then add **"TON#"** which is the destination network.

Then add the address on the destination network (on the tone) and add a **"#"** sign after which the data string comes. In fact, there are no restrictions on the format of the transmitted data - you can put JSON / you can put HEX, or numbers.

The most interesting - you can put HASH data and send it only, and the data to pass unprotected way (just via FTP for example) and target check already HASH data with the hash of the incoming file. There are no limitations here.

Here is an example of how to create a transaction from our Open Source repository:

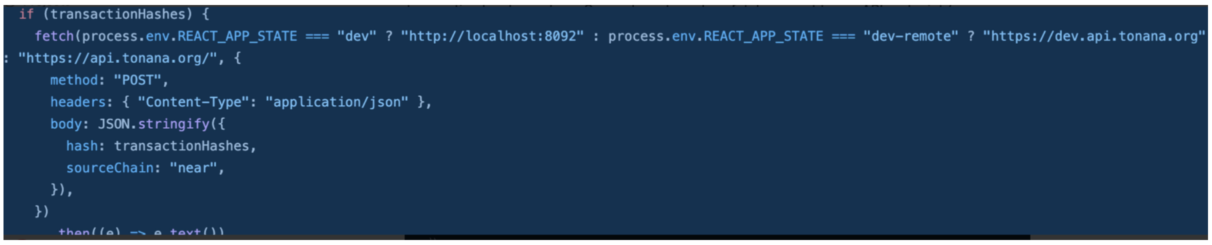


```
await window.contract.account._signAndSendTransaction({
  receiverId: process.env.REACT_APP_NEAR_CONTRACT,
  actions: [
    transactions.functionCall(
      'payToWallet',
      {
        target: process.env.REACT_APP_BACK_NEAR_WALLET,
        message: `${openData ? "SM#" : ""}${netTo}#${openData? add : walletTo}${openData ? `#${btoa(params)}` : ""}`
      },
      new TonWeb.utils.BN(40000000000000),
      new TonWeb.utils.BN(utils.format.parseNearAmount(amount)+'')
    )
  ],
  walletCallbackUrl: 'https://app.tonana.org/?isnear=true'
})
```

After the user signs the transaction, your application has to notify oracle tones that the transaction has been done.

So you have to make a fetch request to our API endpoint (https://api.tonana.org/). In the body of which you put two parameters - transaction hash and **sourceChain** (in this case it will be "near").

Here is an example of a notification by oracle tonana that the request was made.



```
if (transactionHashes) {
  fetch(process.env.REACT_APP_STATE === "dev" ? "http://localhost:8092" : process.env.REACT_APP_STATE === "dev-remote" ? "https://dev.api.tonana.org"
: "https://api.tonana.org/", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      hash: transactionHashes,
      sourceChain: "near",
    }),
  })
    .then((e) => e.text())
```

The source file where you can find an example of a Near transaction:

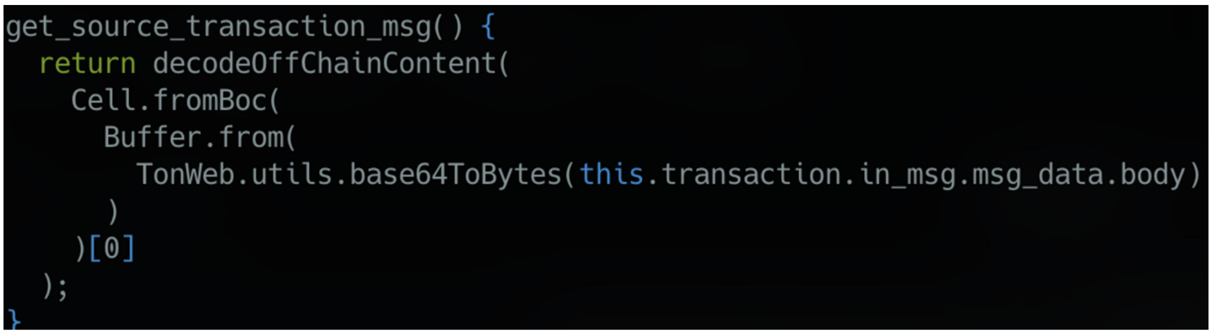
https://github.com/tonanadao/tonana-swap-v1-fe/blob/main/src/logic/transaction/MakeNEARTrx.ts

TON Target chain. To receive data:

After you send a transaction on Near, oracle tones it down, and if it's okay, it will send a transaction on a ton. Inside the transaction, your data will be encoded. To get it you have to pull it out correctly.

Take the transaction that came to your wallet (**this.transaction** is the received needed transaction from the RPC provider). We take **msg_data.body** which is the message of the transaction.

Then we have to decode it correctly. You can see an example in the screenshot.



```
get_source_transaction_msg() {
  return decodeOffChainContent(
    Cell.fromBoc(
      Buffer.from(
        TonWeb.utils.base64ToBytes(this.transaction.in_msg.msg_data.body)
      )
    )[0]
  );
}
```

Where the **decodeOffChainContent** function parses the cell tree by the desired prefix. Your data lies in the cell tree in chunks of up to 127 bytes per cell. So you need to get them correctly and "glue" them back together - that's what the decodeOffChainContent helper function is for.

Here is a link to the helper function:

https://github.com/tonanadao/tonana-swap-v1-fe/blob/main/src/logic/transaction/BOCcontent.ts

After that, you will get a decoded message as a result of this function. It is important to note that the result is the string **"SM#TON#WALLET#DATA"**. From here you can do what you want with it - it is already data on a tone that came to you from Near.

TON → NEAR

TON Source chain. To send a transaction with data:

You have to send a properly assembled transaction to translate the data from TON to NEAR. Our tone is a jack (**EQCyUNjJ6-BJqqhtsiVRn1W4tPAT4jyCDT9DdyRNT_QwrWs9**).

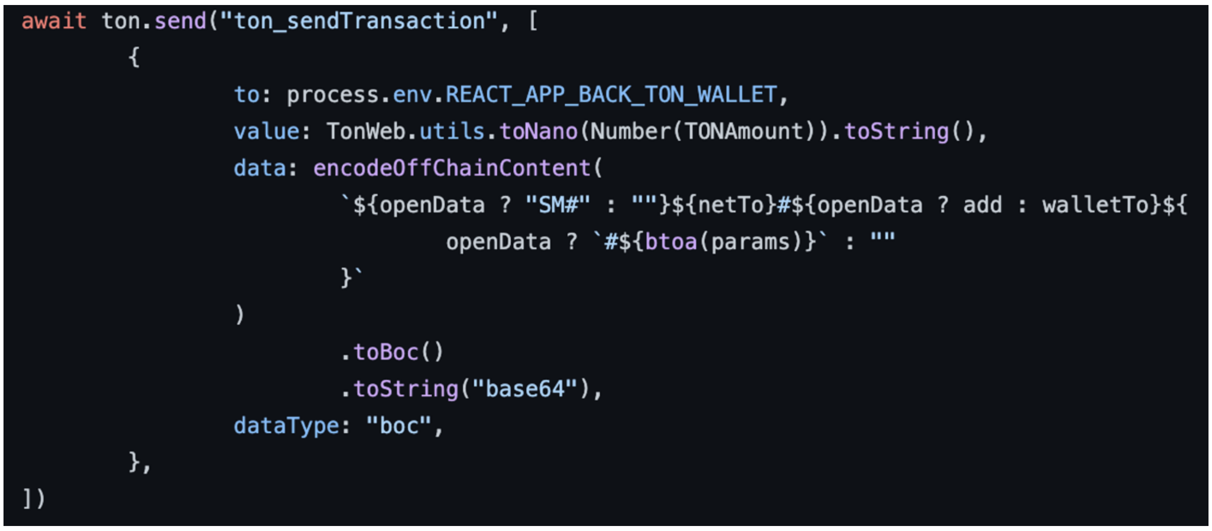
If you want to translate and put data, first you have to assemble a line with **"SM#"** as a flag. Then add **"NEAR#"** which is the destination network. Then you add the address on the destination network (on nir) and add a **"#"** sign after which the data string comes.

In fact, there are no restrictions on the format of the transmitted data - you can put JSON / you can put HEX, or numbers. The most interesting - you can put HASH data and send it only, and the data to pass unprotected way (just via FTP for example) and target check already HASH data with the hash of the incoming file. There are no restrictions here.

After that, the received string should be encoded correctly and put as data transfer. Your data must lie in the cell tree in chunks of up to 127 bytes in one cell. There is a helper function encodeOffChainContent for this (link to it https://github.com/tonanadao/tonana-swap-v1-fe/blob/main/src/logic/transaction/BOCcontent

.ts). The received data should be converted into base64 (in our case, to send a transaction to the user's ton web signature)

Here is an example of transaction creation from our OpenSource repository:



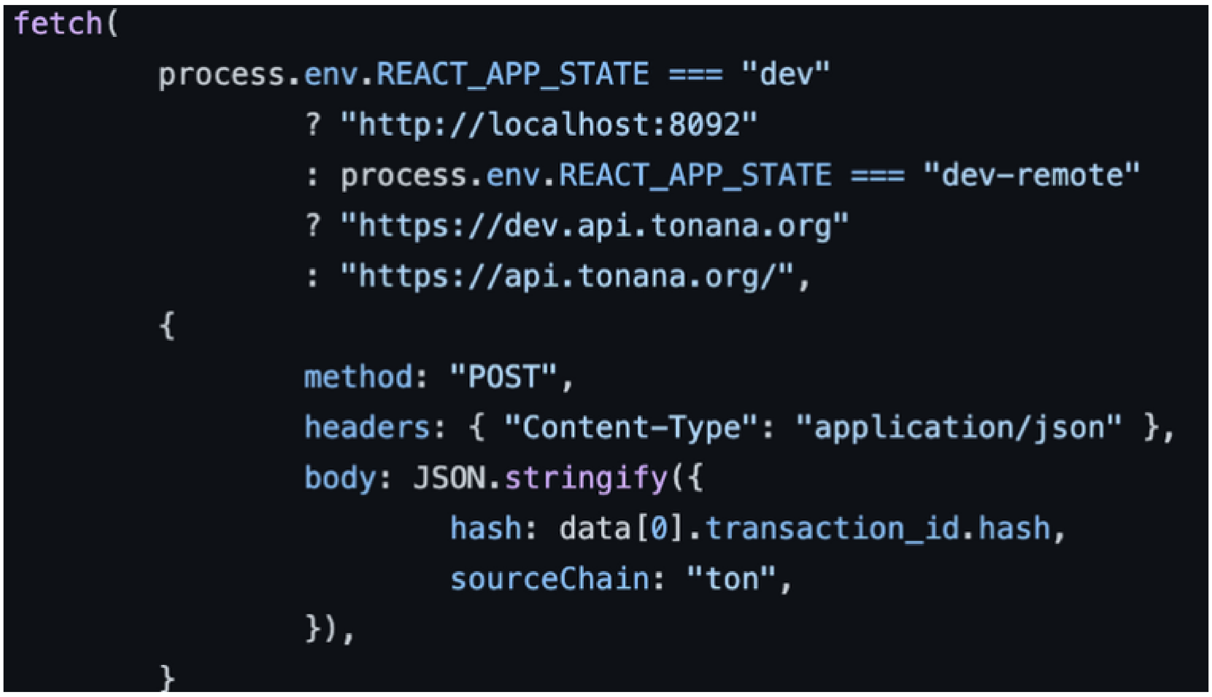
```
await ton.send("ton_sendTransaction", [
        {
                to: process.env.REACT_APP_BACK_TON_WALLET,
                value: TonWeb.utils.toNano(Number(TONAmount)).toString(),
                data: encodeOffChainContent(
                        `${openData ? "SM#" : ""}${netTo}#${openData ? add : walletTo}${
                                openData ? `#${btoa(params)}` : ""
                        }`
                )
                        .toBoc()
                        .toString("base64"),
                dataType: "boc",
        },
])
```

After the user signs the transaction, your application must notify oracle tones that the transaction has been executed.

So you have to make a fetch request to our API endpoint (https://api.tonana.org/). In the body of which you put two parameters - transaction hash and **sourceChain** (in this case it will be "ton").

Here is an example of oracle notifying tonana that the request was made.

NEAR Target chain. To receive data:

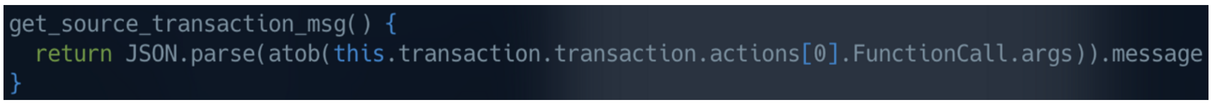


```
fetch(
        process.env.REACT_APP_STATE === "dev"
                ? "http://localhost:8092"
                : process.env.REACT_APP_STATE === "dev-remote"
                ? "https://dev.api.tonana.org"
                : "https://api.tonana.org/",
        {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({
                        hash: data[0].transaction_id.hash,
                        sourceChain: "ton",
                }),
        }
```

After you send a transaction on the tone, the oracle tones it down, and if all is well with it, it will send the transaction on the Near. Inside the transaction, your data will be encoded. To get it you have to pull it out correctly.

We take the transaction that came to our **tonana.near** proxy (**this.transaction** is the received needed transaction from the RPC provider).

This proxy transaction was immediately forwarded to your wallet. But without data (just Near). All the data will be in the incoming transaction on **tonana.near**. Pull it out and take the message field.



```
get_source_transaction_msg() {
  return JSON.parse(atob(this.transaction.transaction.actions[0].FunctionCall.args)).message
}
```

After that, you'll get a decrypted message as a result of our bridge. It's important to note, that the result is a string like **"SM#NEAR#WALLET#DATA"**.

Then you can do what you want with it - it is the data on the tone that came to you from the TON.

AMM research

In addition to completing the data transfer bridge implementation, we also conducted research on automated market makers (AMMs).

AMMs are a type of decentralized exchange (DEX) protocol that allows users to buy and sell digital assets in a trustless, peer-to-peer fashion.

Decentralized exchanges (DEXs) generally require liquidity in order to operate effectively. Liquidity refers to the ability of an asset to be bought or sold without affecting its price. In the context of a DEX, liquidity is important because it allows users to easily buy and sell assets without having to wait for a counterparty to be found. Without sufficient liquidity, it can be difficult for users to execute trades on a DEX, as there may not be enough buyers or sellers available at any given time.

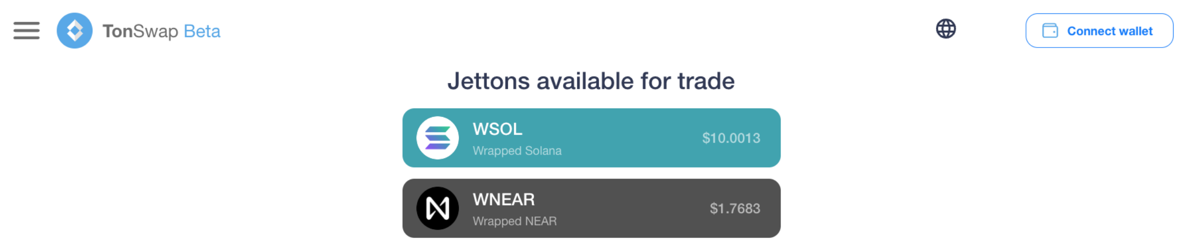
Automated market maker (AMM) decentralized exchanges (DEXs) are already accessible and becoming popular on the TON blockchain, offering users a simple and efficient way to trade digital assets.

Some of the AMM DEXs available on TON include **Megaton Finance, DeDust, tonswap.org, dedust.io, STON.fi, and Tegro DEX.** These platforms offer a variety of trading pairs and use algorithms or smart contracts to determine prices and facilitate trades.

Many of these DEXs also have low fees and fast transaction speeds, making them attractive options for traders.

Tonana as a source of cross-chain liquidity

We are excited to announce that we have already agreed on a **partnership with tonswap.org, STON.fi, and Tegro DEX**! This partnership will allow users on these decentralized exchanges (DEXs) to access and trade wrapped native tokens from other networks.



Jettons available for trade

WSOL	Wrapped Solana	$10.0013
WNEAR	Wrapped NEAR	$1.7683

It means that is possible for decentralized exchanges (DEXs) on the TON blockchain to use the Tonana Bridge as a source of cross-chain liquidity for trading pairs on TON through wrapped assets.

The Tonana Bridge is a blockchain bridge that connects the TON and networks like Ethereum, Aurora, Cosmos, Solana, and Near, allowing users to transfer assets between the two platforms. By using the Tonana Bridge, DEXs on TON could access liquidity from the other networks, potentially expanding the pool of available buyers and sellers for a given asset.

To use the Tonana Bridge as a source of liquidity, liquidity providers for DEXs on TON could use a process called "wrapping," where users can exchange their assets for a token that represents the underlying asset and can be traded on the TON DEXs. The wrapped asset on TON would be a native token or custom on other blockchains that are backed by the underlying asset on the native chain. Users could then trade the wrapped asset on the DEX or add it to the liquidity pool, with the bridge facilitating the transfer of the underlying asset between the two networks.

Tonana as a source of wrapped TON

One way that Tonana Bridge can be a source of TON coins for Near DEXs is through the process of "wrapping," where users can exchange their TON-based assets or native coin for a token that represents the underlying asset and can be traded on the Near network.

For example, a user on the Near network might want to trade a TON-based asset, but the asset is not directly available on Near. The user could use the Tonana Bridge to wrap the TON asset as a Near-based token, which can then be traded on the Near network. The bridge would facilitate the transfer of the underlying TON asset between the two networks, allowing the user to trade the wrapped asset as if it were native to the Near network.

This process can be beneficial for liquidity providers on the TON blockchain, as it allows them to generate additional revenue by providing TON-wrapped assets for Near. Liquidity providers can earn fees for facilitating these wrapped TON coin trades on NEAR DEXs, and can also potentially earn rewards from the wrapped asset's appreciation in value.

Constant Product Formula for Tonana

Decentralized exchanges (DEXs) are on-chain programs that facilitate the exchange of assets within a single blockchain network. While DEXs can be useful for trading assets within a single network, they are not able to facilitate cross-chain transactions, which are the exchange of assets between different blockchain networks.

To enable cross-chain transactions, special protocols called cross-chain bridges are used. These bridges facilitate the exchange of assets between different blockchain networks, allowing users to trade assets across these networks without the need for intermediaries.

One example of a cross-chain bridge is the Tonana Bridge, which allows users to trade assets between the Solana and TON networks. By using the Tonana Bridge, users are able to trade assets across these networks in a secure and efficient manner.

To facilitate cross-chain swaps, the cross-chain bridges often use liquidity pools that hold assets from different blockchain networks. However, managing the exchange rate between these assets can be a challenge, as the price of assets can fluctuate significantly across different networks. To address this issue, cross-chain bridges can use automated market maker (AMM) protocols that use a constant product formula to maintain a stable exchange rate between assets in the liquidity pool.

The constant product formula, also known as the constant product rule or constant product principle, is a mathematical formula that is used to maintain a constant product of the values of assets in a liquidity pool. The formula is expressed as:

$x * y = k$

Where x and y are the values of the assets in the liquidity pool, and k is a constant value.

To illustrate how this formula works in practice, consider a DEX that operates on the Solana and TON networks and uses the Tonana Bridge to facilitate cross-chain swaps. Assume that the DEX has a liquidity pool that holds 10,000 SOL and 1,000 TON. The initial exchange rate between SOL and TON is 10 SOL per TON, which means that k = 100,000 (10,000 * 1,000 = 100,000).

Now, suppose that the price of SOL increases on the Solana network, while the price of TON remains unchanged on the TON network. This would result in an increase in the value of the SOL in the liquidity pool, while the value of the TON remains the same. To maintain the constant product of the liquidity pool, the cross-chain bridge can use the constant product formula to adjust the exchange rate between the assets.

For example, if the price of SOL increases by 50%, the value of the SOL in the liquidity pool would increase to 15,000 SOL (10,000 * 1.5 = 15,000). To maintain a constant product of 100,000, the DEX would need to decrease the exchange rate between SOL and TON to 6.67 SOL per TON (15,000 / 1,000 = 15; 100,000 / 15 = 6.67).

This process can be repeated whenever the price of either asset in the liquidity pool changes, allowing the DEX to maintain a stable exchange rate between the assets.

The constant product formula can be an effective way to manage exchange rates in cross-chain pools on Tonana Bridge.