СОДЕРЖАНИЕ

ВВЕДЕНИЕ

За короткий промежуток времени блокчейн технологии стали важной частью цифровой экономики всего мира. На момент написания работы капитализация рынка криптовалют составляет 3.57 триллионов долларов [1].

Важным этапом сетевого взаимодействия является проверка целостности получаемых сообщений. Одним из таких способов являются избыточные циклические коды (cyclic redundancy check). Такие коды позволяют выявлять различные виды ошибок: от единичных и двойных до пакетных ошибок (burst error). Протоколы в блокчейне не стали исключением. Ошибки в них могут привести к большим финансовым потерям.

В работе будут рассмотрены примеры использования избыточных циклических кодов в протоколах криптовалют.

[1] https://www.coingecko.com/en/global-charts

1 Общая характеристика блокчейн технологий

В 2008 году Сатоши Накамото опубликовал работу *Bitcoin: A Peer-to-Peer Electronic Cash System* [1]. В этой статье предлагается технология для реализации цифровых финансов, которая позволяет любым участникам проводить peer-to-peer транзакции, минуя любые финансовые институты. Революционным в данной работе является способ решения проблемы двойной траты (double-spending). Данная проблема отсутствует в случае, если участники сети доверяют третьей стороне для проведения транзакций. Решение заключается в том, чтобы добавление каждой новой транзакции требовало вычислительных ресурсов, а изменение какой-то старой транзакции требовало как минимум количество ресурсов, которые были потрачены на добавление транзакций следующих после неё.

Можно выделить следующие характеристики блокчейн-технологии:

— Неизменяемость. Транзакции в существующем блокчейне нельзя изменить; можно лишь создать разветвление (fork) блокчейна, начиная с некоторого блока.

— Децентрализованность. Состояние блокчейна одновременно хранится у всех участников сети.

— Безопасность. В основе механизмов лежит стойкая криптография.

— Прозрачность. История транзакций доступна всем пользователям.

— Согласованность. Механизмы proof-of-work (bitcoin) и proof-of-stake (ethereum) позволяют добиваться согласованности между участниками сети в вопросе добавляемых в блокчейн транзакций.

Блокчейн технологии могут использоваться для целей, отличных от финансовых:

а) система голосования;

б) цифровая идентификация/цифровая личность;

в) безопасное хранение персональных данных;

г) и многое другое.

2 Циклические избыточные коды

В работе [2] Уильям Уэсли Питерсон предложил способ обнаружения ошибок с помощью циклических кодов. Линейный код K длины n над полем $\mathbb{F}_q$ называется циклическим кодом, если любой циклический сдвиг кодового слова $v \in K$ также принадлежит коду K: $Z(v) \in K$.

Питерсон предложил представлять бинарные сообщения в виде многочленов, где каждый бит соответствует коэффициенту многочлена. Например, сообщение 1101 можно представить в виде многочлена $1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 1$. Для того, чтобы получить проверочные биты, необходимо разделить многочлен, представляющий сообщение, на многочлен-генератор. Остаток деления этих двух многочленов и будет представлять собой проверочные биты. Пример представлен на рисунке 2.1 [1].

```
1 1 0 1 1 1 0 0 | 10101
1 0 1 0 1       | 1110
---------
  1 1 1 0 1
  1 0 1 0 1
  ---------
    1 0 0 0 0
    1 0 1 0 1
    ---------
      1 0 1 0
```

Рисунок 2.1 — Пример деления многочленов

[1] https://habr.com/ru/articles/770014/

3 Циклические избыточные коды в протоколах криптовалют

3.1 Bitcoin

Для хранения состояния блокчейна в виде UTXO (Unspent Transaction Output) в Bitcoin используется база NoSQL база данных LevelDB [3]. Состояния хранятся в виде ключ-значение, где ключом является идентификатор транзакции (TXID), а значением – количество монет. Пример[1] проверки CRC представлен на листинге 3.1.

Listing 3.1 — Проверка CRC блока в LevelDB

```
// Check the crc of the type and the block contents
const char* data = contents.data();  // Pointer to where Read
   put the data
if (options.verify_checksums) {
  const uint32_t crc = crc32c::Unmask(DecodeFixed32(data + n +
     1));
  const uint32_t actual = crc32c::Value(data, n + 1);
  if (actual != crc) {
    delete[] buf;
    s = Status::Corruption("block␣checksum␣mismatch", file->
       GetName());
    return s;
  }
}
```

3.2 Ethereum

Для того, чтобы оптимизировать протокол поиска подходящих узлов в сети Ethereum, в EIP-2124 [4] было предложено в каждом узле хранить его специальный ID. ID состоит из двух полей:

[1] https://github.com/bitcoin/bitcoin/blob/master/src/leveldb/table/format.cc

а) FORK_HASH — CRC32 от хэш-значения блока генезиса (genesis block) и номеров блоков всех предшествующих форков;

б) FORK_NEXT — номер блока предстоящего форка (0, если номер блока неизвестен).

Важно отметить, почему авторы данного предложения решили использовать CRC32 вместо Keccak256: поскольку узел может в любой момент соврать о своем настоящем ID, то криптографическая хэш-функция не несет никакой ценности.

3.3 The Open Network

3.3.1 Адреса

В соответствие с TEP-2 [5] в конец адресов смарт-контрактов, публичных ключей и ADNL (Abstract Datagram Network Layer) адресов добавляется CRC16 от этих данных. Например, адрес смарт контракта состоит из следующих частей:

а) тип адреса (1 байт);

б) идентификатор Workchain (1 байт);

в) 256 бит адреса смарт-контракта внутри Workchain (32 байт);

г) значение CRC16-CCITT от предыдущих полей (2 байта).

3.3.2 Идентификатор метода

Для того, чтобы получить какие-то данные из смарт-контракта, необходимо вызвать get-методы контракта. Выполнение таких методов происходит вне блокчейна, поэтому они не имеют стоимости. Такие функцию помечаются спецификатором `method_id` [6]. В свою очередь функция, помеченная таким спецификатором, получает идентификатор, который вычисляется следующим образом:

Listing 3.2 — Вычисление идентификатора метода

```
crc16(<function_name>) & 0xffff | 0x10000
```

3.3.3 TL-B

В TON для сериализации сообщений используется язык описания данных TL-B [7]. TL-B схема содержит описание типов и функциональных комбинаторов. Каждый комбинатор имеет конструктор, в котором указывается его идентификатор. Если идентификатор не указан в явном виде, то для его вычисления используется CRC32. Например в листинге 3.3 показано, как определяется комбинатор `get_static_data`.

Listing 3.3 — Комбинатор get_static_data

```
get_static_data query_id:uint64 = InternalMsgBody;
```

Поскольку после имени комбинатора не указывается идентификатор, то он вычисляется следующим образом:

Listing 3.4 — Вычисление идентификатора get_static_data

```
crc32('get_static_data query_id:uint64 = InternalMsgBody')
= 0x2fcb26a2 & 0x7fffffff = 0x2fcb26a2
```

3.3.4 Bag of Cells

Все данные в TON представлены в виде ячеек (Cell). Каждая клетка хранит в себе до 1023 байт данных и до 4 ссылок на другие клетки. Для передачи данных между узлами используется формат Bag of Cells [8]. При сериализации Bag of Cells генерируется CRC32 для контрольной суммы. В листинге 3.5 показано, как происходит проверка контрольной сумма для Bag of Cells при десериализации[1].

Listing 3.5 — Проверка CRC значения при десериализации BoC

```
if (info.has_crc32c) {
```

[1] https://github.com/ton-blockchain/ton/blob/master/crypto/vm/boc.cpp

```
  unsigned crc_computed = td::crc32c(td::Slice{data.ubegin(),
     data.uend() - 4});
  unsigned crc_stored = td::as<unsigned>(data.uend() - 4);
  if (crc_computed != crc_stored) {
    return td::Status::Error(PSLICE() << "bag-of-cells␣CRC32C␣
       mismatch:␣expected␣" << td::format::as_hex(crc_computed)
                                    << ",␣found␣" << td::
                                       format::as_hex(
                                       crc_stored));
  }
}
```

ЗАКЛЮЧЕНИЕ

В результате проделанной работы стало ясно, что избыточные циклические коды являются простым и универсальным инструментом. В протоколах блокчейн они используются как по их прямому назначению — выявление ошибок при передаче данных, так и для генерации уникальных идентификаторов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Nakamoto Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System. — 2009. — Режим доступа: http://www.bitcoin.org/bitcoin.pdf (дата обращения: 19.02.2025).

2. Peterson W. W., Brown D. T. Cyclic Codes for Error Detection // Proceedings of the IRE. — 1961. — Т. 49, № 1. — С. 228–235.

3. Bitcoin Chainstate Parser. — 2019. — Режим доступа: https://github.com/in3rsha/bitcoin-chainstate-parser (дата обращения: 16.02.2025).

4. EIP-2124: Fork identifier for chain compatibility checks. — 2019. — Режим доступа: https://eips.ethereum.org/EIPS/eip-2124 (дата обращения: 16.02.2025).

5. TON Addresses. — 2019. — Режим доступа: https://github.com/ton-blockchain/TEPs/blob/master/text/0002-address.md (дата обращения: 18.02.2025).

6. FunC Language. Documentation. Functions. — 2025. — Режим доступа: https://docs.ton.org/v3/documentation/smart-contracts/func/docs/functions (дата обращения: 18.02.2025).

7. TL-B. — 2025. — Режим доступа: https://docs.ton.org/v3/documentation/data-formats/tlb/tl-b-language (дата обращения: 18.02.2025).

8. Cell and Bag of Cells (BoC). — 2025. — Режим доступа: https://docs.ton.org/v3/documentation/data-formats/tlb/cell-boc (дата обращения: 18.02.2025).