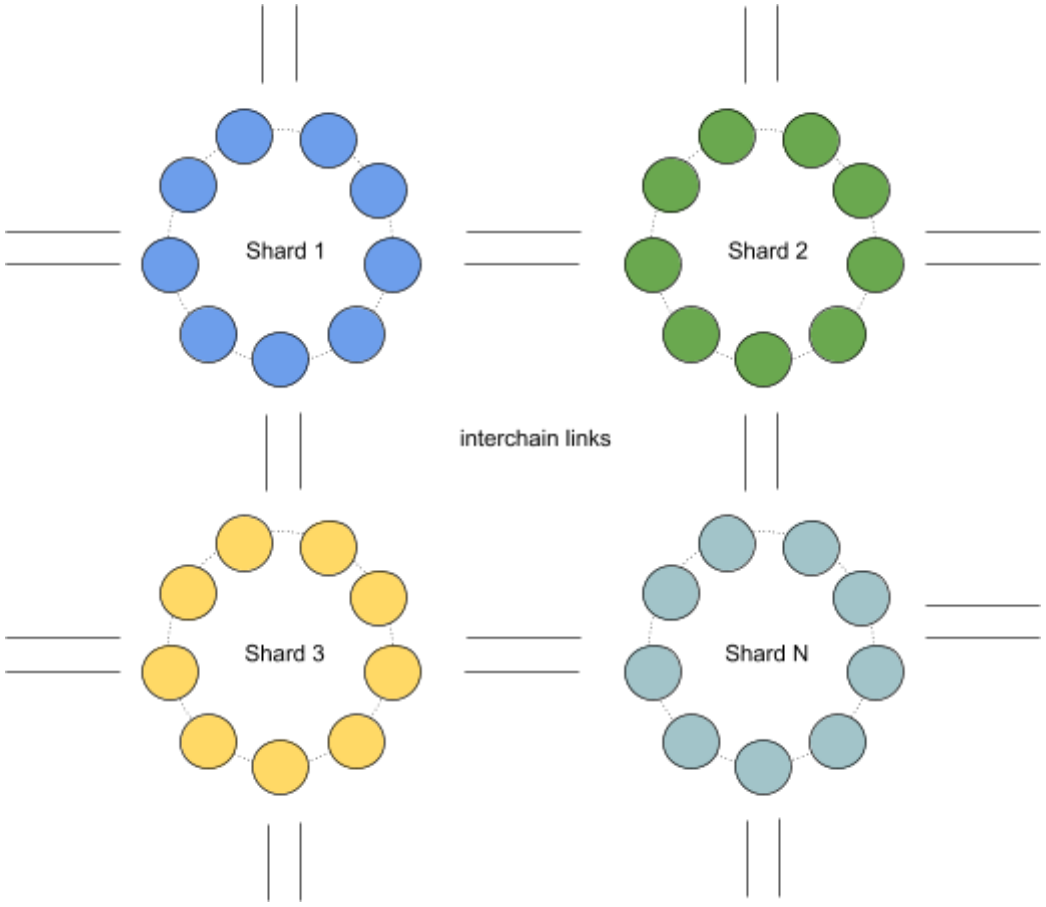
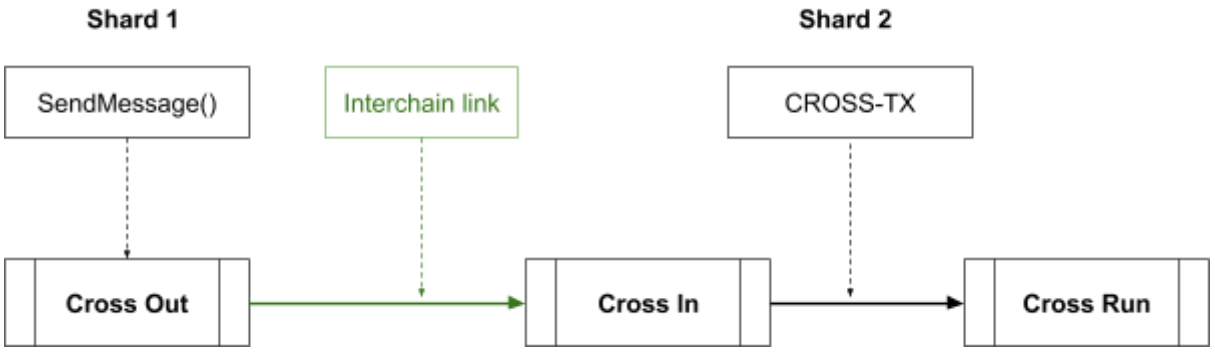
Спецификация транзакций для кросс-шардинговых сообщений

Смарт-контракты могут обмениваться сообщениями между друг другом, в том числе между разными шардами. Схема шардов:



Если у нас есть нода, которая майнит несколько шардов одновременно, то при создании блока она может добавить в него специальные транзакции, которые содержат информацию о наличии в других шардах сообщений, предназначенных для нашего блокчейна. Такие транзакции сами по себе не участвуют в обмене между нодами до создания блока. Они могут передаваться другим нодам только неотрывно от блока.

Майнеры выполняют голосование за тот или иной состав сообщений в блоках путем создания виртуальных цепочек через специальные транзакции голосования, чем больше блоков содержит цепочка, тем больше достоверность сообщения. При достижении определенного порога, задаваемого изначально, происходит фиксация сообщения в шарде получателе. Фактически майнеры выполняют роль оракулов - свидетелей наличия какой-либо информации в шарде источнике.



Эта механика осуществляется одним методом в шард 1 и тремя видами транзакций в шарде 2:

Шард 1

SendMessage

Встроенный метод, вызываемый из кода смарт-контракта, например при получении денег на счет, являющийся каналом передачи ценностей в другой шард. Сообщения можно передавать только на базовый счет (**Base Account)** смарт-контракта.

Действия:
Все ноды (при обработке): Запись в таблицу исходящих сообщений **CrossOut**
Майнер: Передача по приватным каналам связи в другой шард актуальное состояние таблицы **CrossOut** шарда 1 в таблицу **CrossIn** шарда 2

Шард 1 и 2

NEW-SHARD

Регистрация нового шарда для обмена с данным блокчейном
Формат:

```
Type: "byte",
ShardName: "str5",
Description: "str40"
```

Действия:
Пользователь-инициатор: Отправка транзакции с уникальным именем шарда с которым создается канал обмена, она обязательно должна быть платной, чтобы защититься от DDOS.
Все ноды (при обработке): Занесение нового имени в список каналов.

Шард 2

CROSS-TX

Появление нового сообщения из другого шарда

Формат:

```
Type:"byte",//150
BlockNumFrom: "uint32",
TrNumFrom:"uint16",
ShardFrom: "str5",
AccountFrom: "uint32",
Iteration: "byte",
Mode: "byte",
ShardTo: "str5",
AccountTo: "uint32",
MethodName: "str",
Params: "str"
```

Пример:

```
{
   "Type": 150,
   "BlockNumFrom": 55186,
   "TrNumFrom": 0,
   "ShardFrom": "TERA",
   "AccountFrom": 45,
   "Iteration": 2,
   "Mode": 1,
   "ShardTo": "TEST",
   "AccountTo": 45,
   "MethodName": "SendMyToken",
   "Params": `{"Gate":45,"To":"13","Sum":100,"Description":"Test 1"}`
```

}

Действия:
Майнер: Добавление новой транзакции в блок, если ее еще не было в предыдущих корректных блоках (сравнение таблиц **CrossIn** и **CrossRun** + необработанные блоки)
Все ноды (при обработке): Запись в таблицу **CrossRun**

VOTE-TX

Майнеры строят виртуальные цепочки блоков и тем самым голосуют за состав сообщений. В случае если блок предыдущего майнера не имеет ошибок, от него создается новый блок, в случае если блок содержит хоть одну ошибку - он откидывается, а в качестве предыдущего блока берется последний правильный.

Эта схема идентична голосованию по схеме За или Против в котором вес голосов равен друг другу. При такой схеме атака возможно только при наличии 51% злонамеренных майнеров в течении времени пока идет голосование. Период голосования длится до тех пор пока разница высот цепочки лидера и орфана не достигнет заранее задаваемого значения (например 100 блоков для быстрого канала и 1 млн блоков для очень надежного канала)

Формат:

```
Type: "byte",
ShardArr:[{
   Shard: "str5",
   PrevBlockNum: "uint32",
   PrevTrNum: "uint16"
}]
```

Действия:
Майнер: добавление в блок транзакции с указанием последнего правильного блока цепочки.
Майнер (при обработке): Добавление в приватную базу ссылки на последний правильный блок
.

RUN-TX

При достижении требуемой высоты блоков майнер добавляет в свой блок транзакцию RUN-TX для запуска обработки кросс-шардинговых сообщений, т.е. запускается метод смарт-контракта, который был задан в транзакции CROSS-TX (и кстати эта транзакция задается в переменной *context.Tx*)

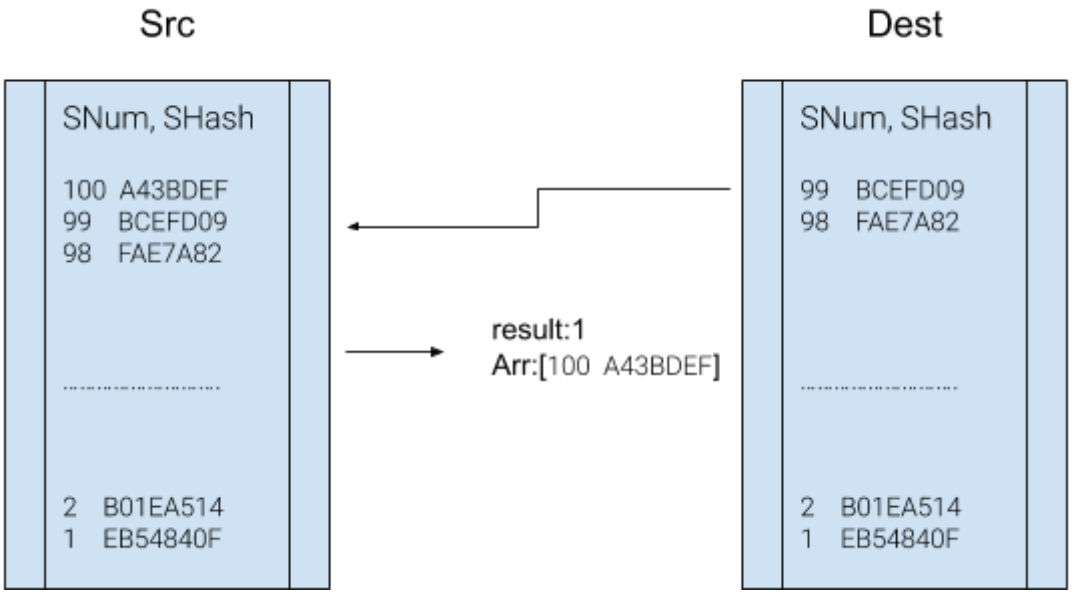
Формат:

```
Type: "byte",
BlockNum: "uint32",
TrNum: "uint16"
```

Действия:
Майнер: Добавление транзакции с теми сообщениями, которые уже можно запустить на исполнение
Все ноды (при обработке): Определение условий запуска (есть требуемая высота блоков, еще не было запуска), запуск смарт-контракта, установка признака.

Синхронизация списков транзакций

Нам нужно достичь одинакового состояния списка кросс-транзакций между нодой источником и нодой приемником. Верхние строки в таких списках постоянно модифицируются из-за свойства блокчейна часто переписывать цепочки блоков при поиске лидера.
Предлагается такое решение: список содержит поля SNum и SHash в которые раздельно для каждого шарда записывается номер по порядку и связанный с предыдущей строкой хэш. Таким образом пара SNum + SHash может являться своеобразным итератором для быстрой синхронизации двух списков в асинхронном режиме.



Логика:

1. На стороне-приемнике **Dest** берем из списка последнее значение SNum + SHash и используя его как итератор передаем его сторону-источник **Src**
2. На стороне **Src** сравниваем значение хэша в строке с номером SNum

    a. Если он совпадает, то возвращаем “result:1” и массив Arr со строками, начиная со строки с SNum + 1 (т.е. которых нет в списке Dst)
    b. Если не совпадает, то возвращаем “result:0”
3. Сторона **Dest**:
    a. При положительном результате загружаем новые данные
    b. При нулевом - уменьшаем номер итератора на величину Delta. При повторных нулевых результатах Delta увеличиваем на 10%, но при условии что ее значение не превышает 1/20 остатка SNum (для предотвращения гигантских шагов назад).