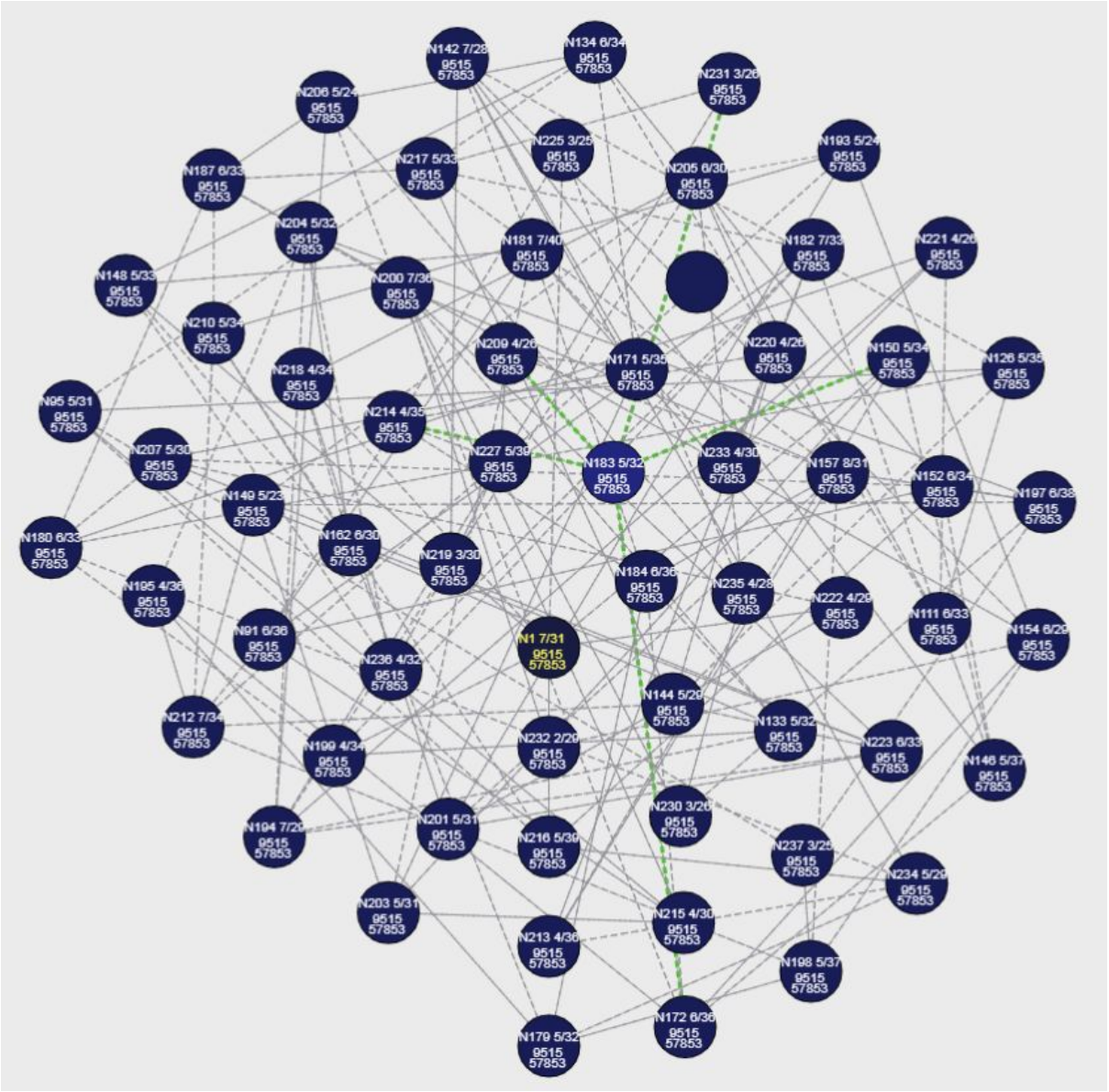
Обновлённый протокол консенсуса TERA - JINN

Протокол Tepa 2.0 основан на новой библиотеке JINN.

JINN - это библиотека, которая позволяет программистам создавать собственные блокчейны по своим правилам консенсусов, в виде настройки параметров, взяв на себя всю сложную “магию” обмена данными. Библиотека разрабатывалась без привязки к конкретному блокчейну. Для связи используется специальный промежуточный модуль-коннектор, который соединяет внешний физический мир с внутренними алгоритмами библиотеки. Он позволяет задавать собственную базу данных, собственные алгоритмы хэширования, алгоритмы валидации транзакций и прочие параметры.
JINN берет на себя все низкоуровневые задачи, такие как: обнаружение соседних нод, построение сети, передача транзакций, оптимизация трафика, синхронизацию цепочек блоков, и наконец достижение консенсуса сети. Эти задачи являются самыми сложные в области блокчейно-строения и они критичны для достижения высокой производительности. Их решение лежит в плоскости разработки асинхронных алгоритмов и поэтому достаточно сложны для разработчиков, которые используют классические приемы программирования. При этом надо заметить, что JINN не решает задачу интерпретации транзакций (например перевод монет). Эта задач возложена на программиста блокчейна, т.к. она более простая и решается стандартными подходами.

Из-за своей высокой абстрактности JINN позволяет моделировать сеть, в этом случае запись блоков не ведется в базу данных, вы выполняется эмуляция записи в памяти. Моделирование позволяет лучше отлаживать алгоритмы и заранее предвидеть “проблемные” ситуации. Библиотека написана на javascript и поэтому одинаково хорошо работает как в серверном варианте (nodejs) так и в клиентском варианте внутри браузера. Работа в браузере позволяет визуализировать модель. Пример вы можете увидеть ниже:



Модель сети
https://terafoundation.org/JINN/model/model.html

Дискуссия:

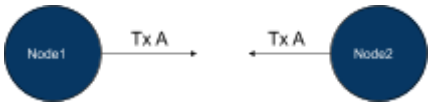
Проблема избыточного трафика

Ноды обмениваются между собой транзакциями, в случае если нода имеет N соединений, каждая нода может отправить или получить N раз одну и ту же транзакцию. Это приводит к избыточному трафику в сети.

В обычных случаях для уменьшения трафика используются буферы. Есть два буфера: буфер полученных транзакций и буфер отправленных транзакций. Алгоритм работы таков: если нода видит что она получила транзакцию от определенной ноды или уже ей ее отправляла, то транзакция ей не отправится - потому что она там есть.

Использование буферов снижает излишний трафик, но есть ситуации в которых они не помогают, рассмотрим следующие сценарии:

1. Две ноды **одновременно** отправляют друг другу одинаковую транзакцию



2. К одной ноде **одновременно** отправляется одна и та же транзакция с разных нод

Проблема следующая - в буферах отправки и получения нет этих транзакций, поэтому обмены совершаются и трафик дублируется.

Метод решения

За несколько фаз (2-3) до отправки транзакций отправляются их короткие хеши, имеющие более чем на порядок меньший размер. Назначение хешей - заполнение буферов отправки и получения такими показателями как это было бы при обмене обычными транзакциями. Таким образом в буферах появляется нужная информация и ноды заранее узнают куда следует отправлять транзакции, а где они уже есть вследствии получения от других соседей и туда отправлять транзакцию не нужно.

Алгоритм работы
(модуль jinn-ticket):

Сначала фаза отправки тикетов, потом фаза транзакций
Отправка тикетов по условию:
1) Еще не отправляли
2) И либо не получали, либо получали но это был не новый тикет
При получении тикета устанавливаем спец. флаг:
- FRESH_TIKET когда это новый для нас тикет (т.е. не получали от других нод)
- OLD_TICKET - уже есть в массиве тикетов
Транзакции отправляем только тогда когда значение флага тикета не равно OLD_TICKET (т.е. отправляем когда значение пусто или равно FRESH_TIKET)

Мульти-ячейки в протоколе синхронизации
Таблица асинхронных задач по нескольким лидерам и по диапазону времени

Консенсус в сети достигается путем сообщений от ноды к ноде, в которой содержится максимальное значение, называемое лидером.
На практике блоки имеют достаточно большой размер, поэтому для быстроты синхронизации информация разделяется на части:

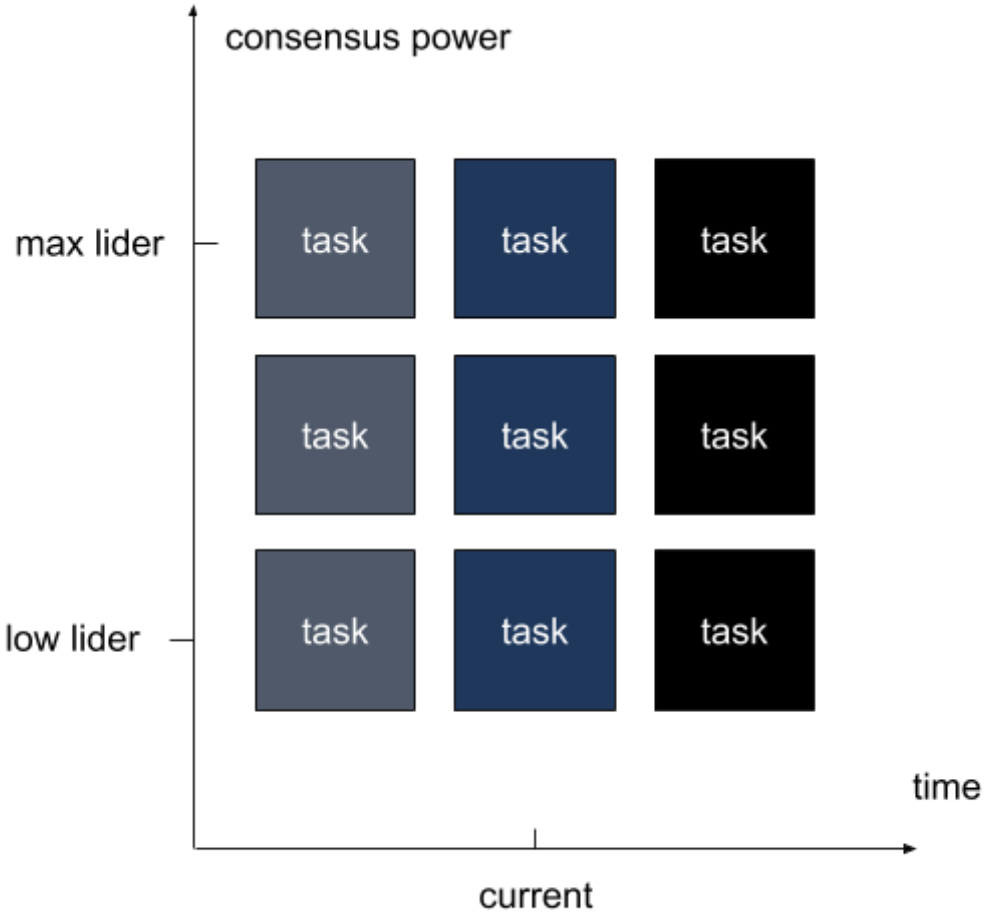
1. хеш
2. список хешей
3. полные данные из которого сформированы хеши

Каждая из частей отправляется поочерёдно и поэтапно, каждая последующая часть отправляется при условии успешной отправки предыдущей части.

Каждый квант информации имеет свою метку времени, а значит есть вероятность некоторого запаздывания или забегания вперед.

В сети возможны проблемы с передачей информации на каждом этапе, например переданы только первые 2 части.

Задача непрерывно находить максимального лидера. Для этого предлагается такая модель:



Matrix of tasks in a consensus algorithm for finding leader in a decentralized network.

Одновременно мы работаем с множеством задач общий смысл которых - это получение и передача информации.

По меткам времени (слева направо):

- С прошлыми отметками времени (на случай если мы забежали вперед)
- Актуальные отметки времени
- Со временем из будущего (на случай если вдруг мы отстали)

По уровням лидерства:

- Максимальный лидер
- Средний лидер
- Слабый лидер

Логика работы со временем:

- Если мы получили информацию из будущего, то обрабатываем ее дважды:
    - вместе с другими задачами из будущего
    - повторно при наступлении текущего времени
- Текущую информацию обрабатываем всегда
- Из прошлого, обрабатываем в рамках задач из прошлого времени

Обработка это: получение информации, сравнение с другой информацией, отправка далее.

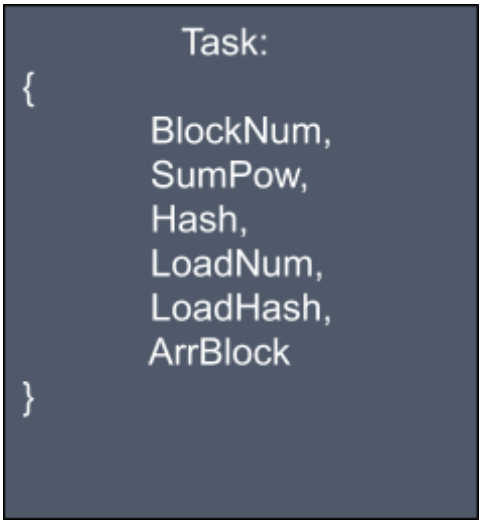
-> Универсальное правило: каждую задачу выполняем в рамках своего времени

Задача обрабатывается если она есть в таблице актуальных задач (время в рамках дозволенного предела) и уровень лидерства не ниже минимального.

При этом остается открытым вопрос финальности лидера (когда записывать информацию в базу данных), поэтому правило такое:

1. Обработка идет по матрице задач
2. Если получена вся информация вплоть до уже существующей в базе данных информации
3. Если нет в базе данных записи более приоритетной задачи
4. Обработка задач не из будущего

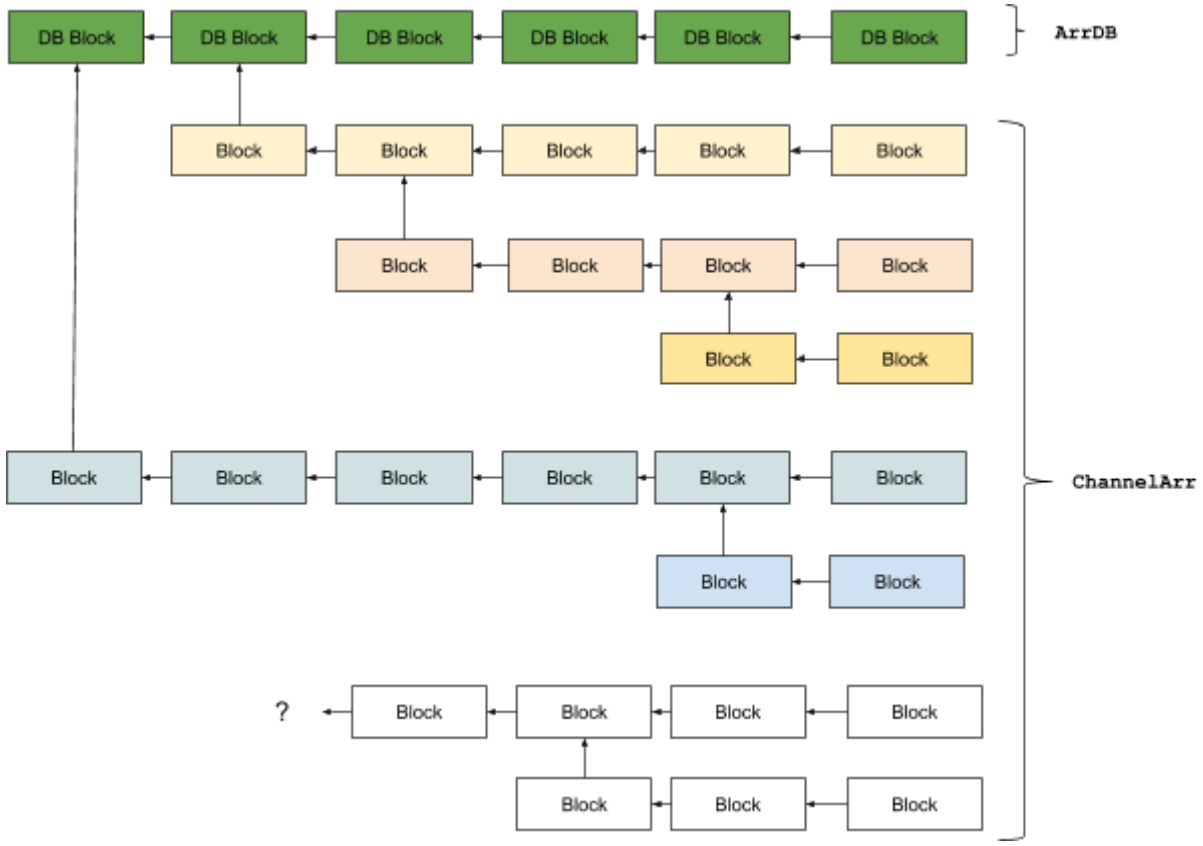
Структура элемента:



Номер актуального блока (текущего времени)
Приоритет загрузки
Хеш

1. Загружаемый номер блока
2. Загружаемый хеш
3. Список загруженных блоков

Полученная информация объединяется в виде дерева блоков. Набор возможных вариантов загрузки блоков. На рисунке видно, что нижние две цепочки не имеют связей с базой данных:



Порядок загрузки цепочки:

1. Загрузка хеша блоков (пока не будет загружен хеш блока, который есть в базе данных)
2. Загрузка данных блоков
3. Запись цепочки в базу данных

Хранение и ротация адресов нод сети
(модуль jinn-connect-addr)
Позволяет последовательно по уровням обходить список нод сети.

Свойства:

- Ротация нод на большом пространстве адресов (практически неограниченном)
- Итератор обхода хранится вне этого списка (внешний, т.е. на вызывающей стороне)
- Если число нод на одном уровне более MAX_LEVEL_NODES, то старые элементы перезатиратстся
- Ноды для попадания их адреса в список рассылки адресов должны выполнить POW

Защита от двойного подключения нод
(модуль jinn-connect)

Если две ноды имеют публичные интернет адреса, то есть вероятность что они одновременно подключатся к друг другу. В этом случае нужно распознать такой случай и оставить только одно подключение.
Алгоритм работы:
Метод HANDSHAKE содержит в качестве параметра ip+port, он используется для образования 32 байтного уникального числа IDArr. Если ноды видят что у них было уже подключение (Child) с таким же значением, то значит это одна и та же нода. В этом случае сравниваются массивы Engine.IDArr и Child.IDArr между собой и в зависимости от этого удаляется входящее или исходящее подключение с этой нодой.

Алгоритм синхронизации глобального времени
(модуль jinn-timing)

Важно чтобы в каждой ноде время было синхронно для того чтобы одновременно создавать новый блок. Для этого подсчитывается статистика полученных меток времени, рассчитывается среднее значение, откидывается шум - значения выходящие за среднеквадратичные отклонения и далее принимается решение о корректировки часов (вместо непосредственной корректировки системных часов выполняется изменения константы DELTA_CURRENT_TIME).
Для защиты от атак в качестве меток времени принимается время прихода максимального хэша за определенный период времени.