
Яндекс

Яндекс

Использование ClickHouse для мониторинга связности сети

Дмитрий Липин

Мотивация

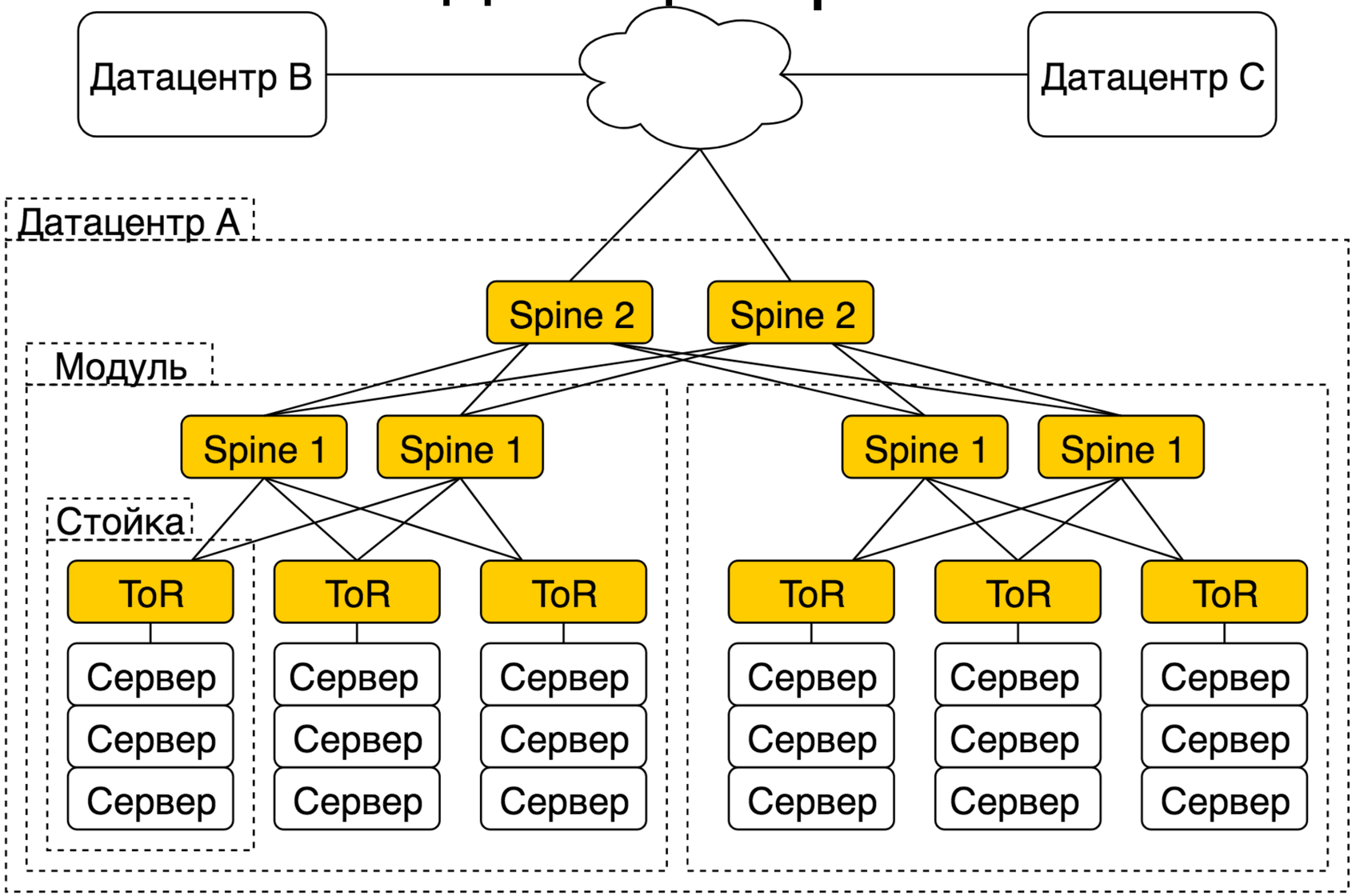
- В масштабах Yandex отказ сетевого оборудования довольно частое явление
  - Необходим инструмент, который бы позволил быстро понять является ли сеть источником проблемы
- Необходимо предоставить сервис заданного качества (SLA)
  - Необходим инструмент, который бы постоянно следил за предоставленными гарантиями
- И в связи с этим упростить поиск и исправление проблем в сети

Объект мониторинга

Устройство сети датацентра

〉 Десятки тысяч серверов подключены к сотням стоек
〉 Сервера в стойке подключаются в так называемый стоечный коммутатор, ToR (top of rack)
〉 Стоечный коммутатор (ToR) подключен к агрегирующим коммутаторам модулей (Spine 1)
〉 Которые в свою очередь связаны ещё одним уровнем коммутаторов (Spine 2) внутри датацентра
〉 Датацентры связаны между собой
〉 Физическая топология такой сети — двухуровневый Clos
〉 Трафик может идти разными путями в зависимости от заголовков пакета, что усложняет поиск и диагностику проблем
〉 Такая схема обеспечивает резервирование Spine, но не ToR

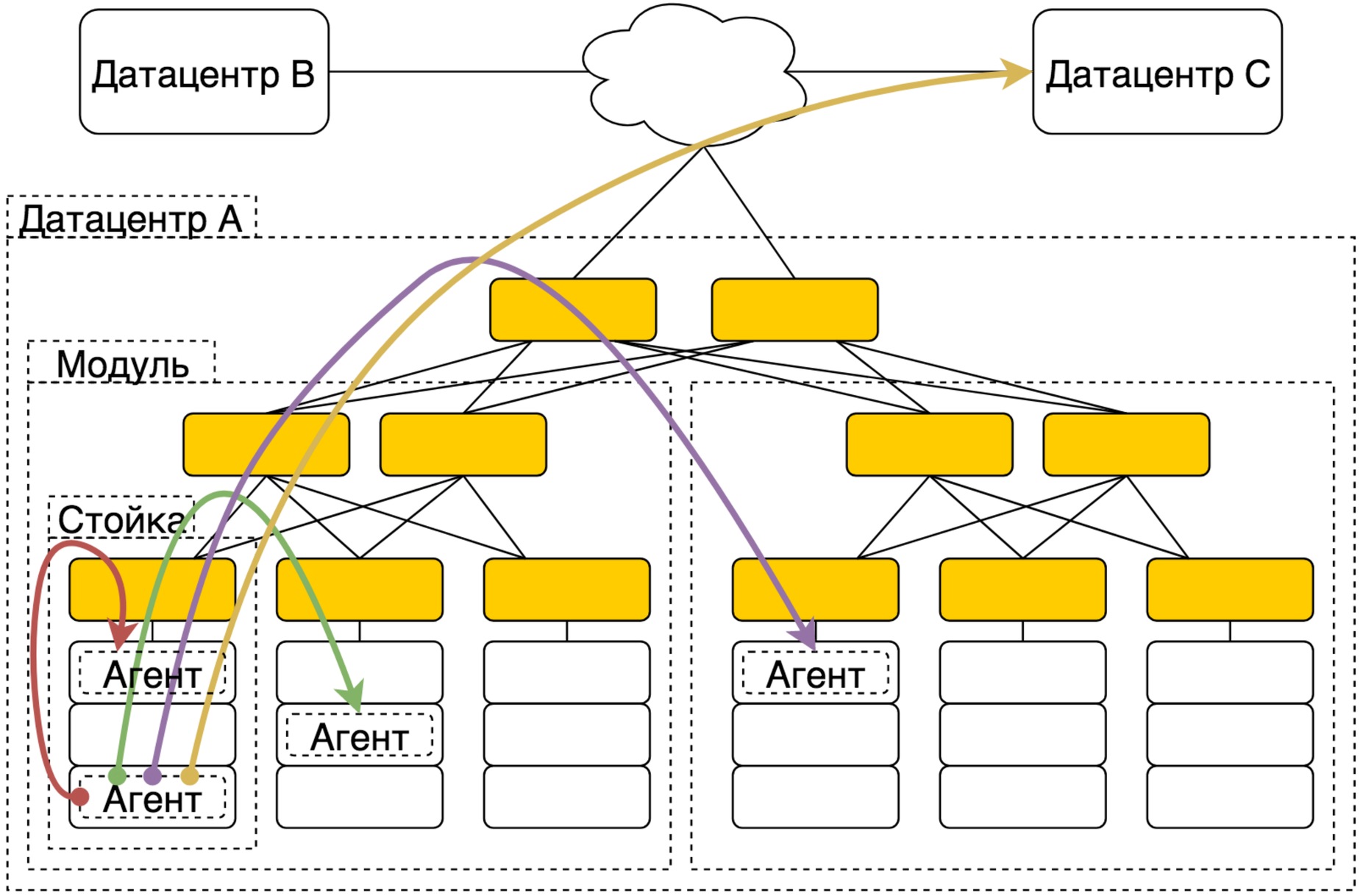
Устройство сети датацентра



“Pingmesh: A Large System for Data Center Network Latency Measurement and Analysis”, SIGCOMM 2015

Агент

〉 Агент ставится только на “bare metal" сервера, но не на ToR
〉 Агент должен быть как можно более легковесным в плане использования ресурсов, поэтому он потребляет менее 1% CPU
〉 С точки зрения агента сеть является “чёрным ящиком”
〉 Каждый агент работает независимо, зная о расположении других агентов, периодически отправляя им UDP пакеты и ожидая от них же ответа
〉 Каждая отдельная проверка связности между двумя серверами является пробой
〉 На каждой итерации случайным образом выбирается список серверов, которые должны быть проверены, причём плотность проб на разных уровнях иерархии сети разная


Датацентр B
Датацентр C
Датацентр A
Модуль
Стойка
Агент
Агент
Агент
Агент

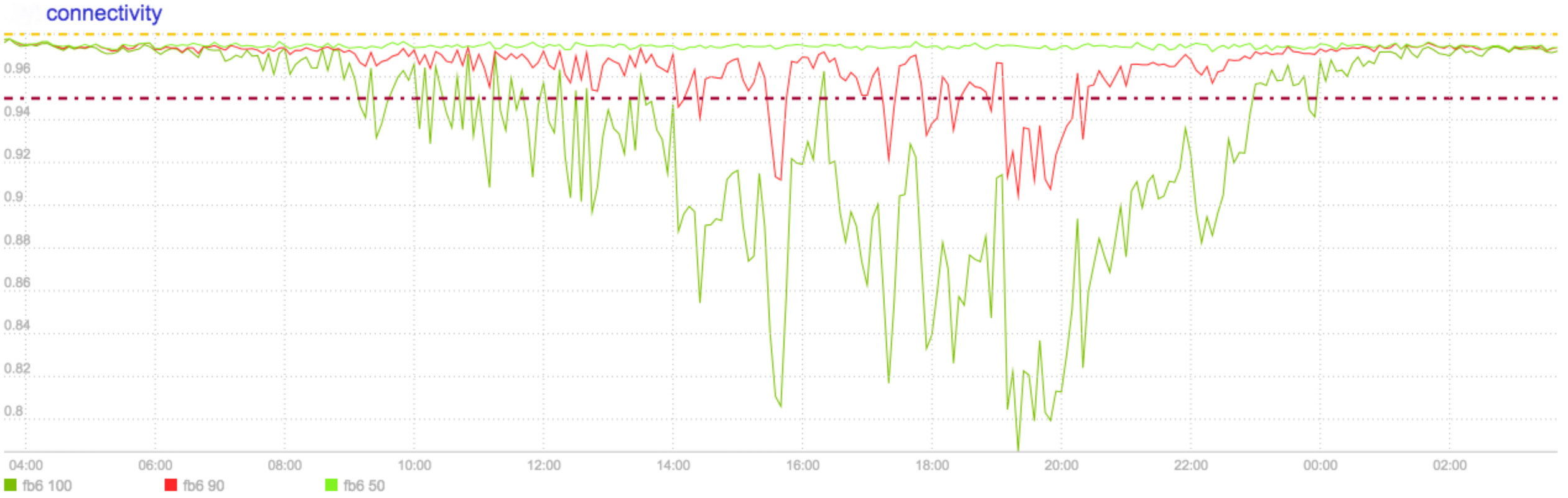
Пробы

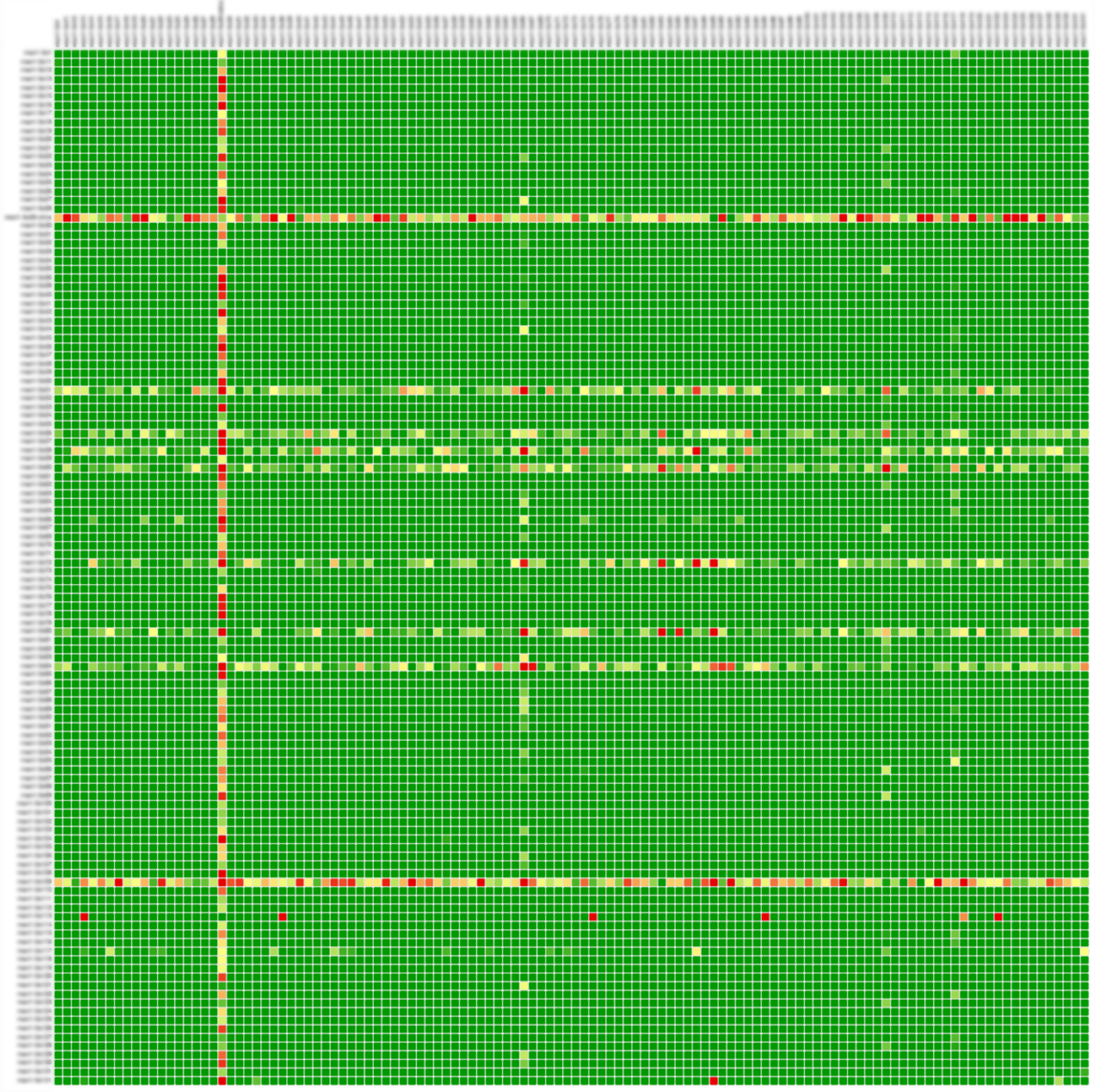
Проба - результат проверки сети агентом. Характеризуется количеством отправленных и принятых в ответ пакетов между выбранной парой хостов и задержкой между ними.


```
{
    "SourceFqdn": "source-server.yandex.net",
    "TargetFqdn": "target-server.yandex.net",
    "SentPackets": 1.0,
    "ReceivedPackets": 1.0,
    "RoundTripTime": 0.0007508,
    "Generated": 1498167250.862623
}
```

Связность сети

〉 Связность определяется процентом проб с заданным уровнем потерь между парами серверов в сети
〉 Связность интересна в разных срезах, а именно стойках, модулях, датацентрах и между ними
〉 Если бы агент проверял каждый другой агент (P2P), то потребовалось бы 10B проб, но т.к. агенты выбираются случайным образом — проб на несколько порядков меньше
〉 В итоге backend'у необходимо обработать 100k проб в секунду, отправляемых десятками тысяч серверов Yandex
〉 Связность считается над пробами за некоторое скользящее окно, для среза уровня ДЦ окно составляет 2 минуты, в него входит 1.5M проб, показатели рассчитываются каждые 5 секунд


connectivity
0.96
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
04:00
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
00:00
02:00
fb6 100
fb6 90
fb6 50



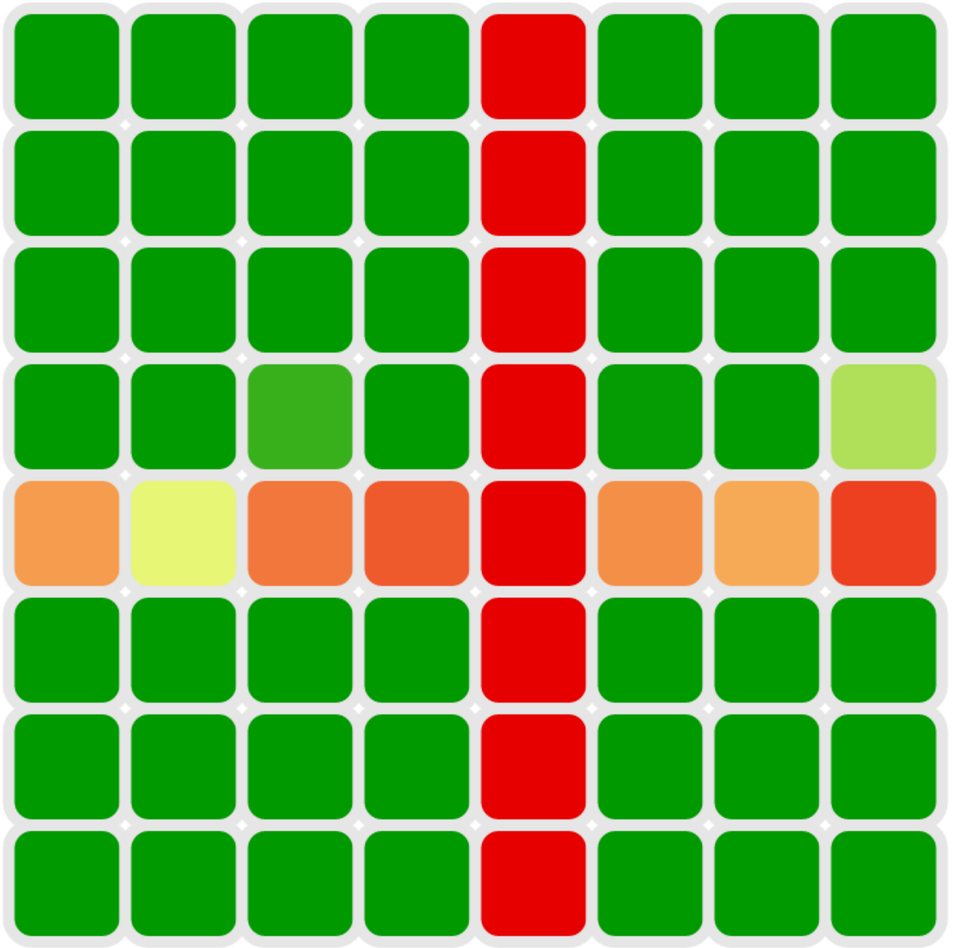
Netmon

〉 Выдерживает выход из строя любого одного датацентра

〉 Отображает состояние сети на каждом из уровней

〉 Показывает “сломанные” пробы, приведшие к ухудшению показателей сети

〉 Показывает состояние сети на заданный пользователем момент времени



Реализация

Реализация серверной части

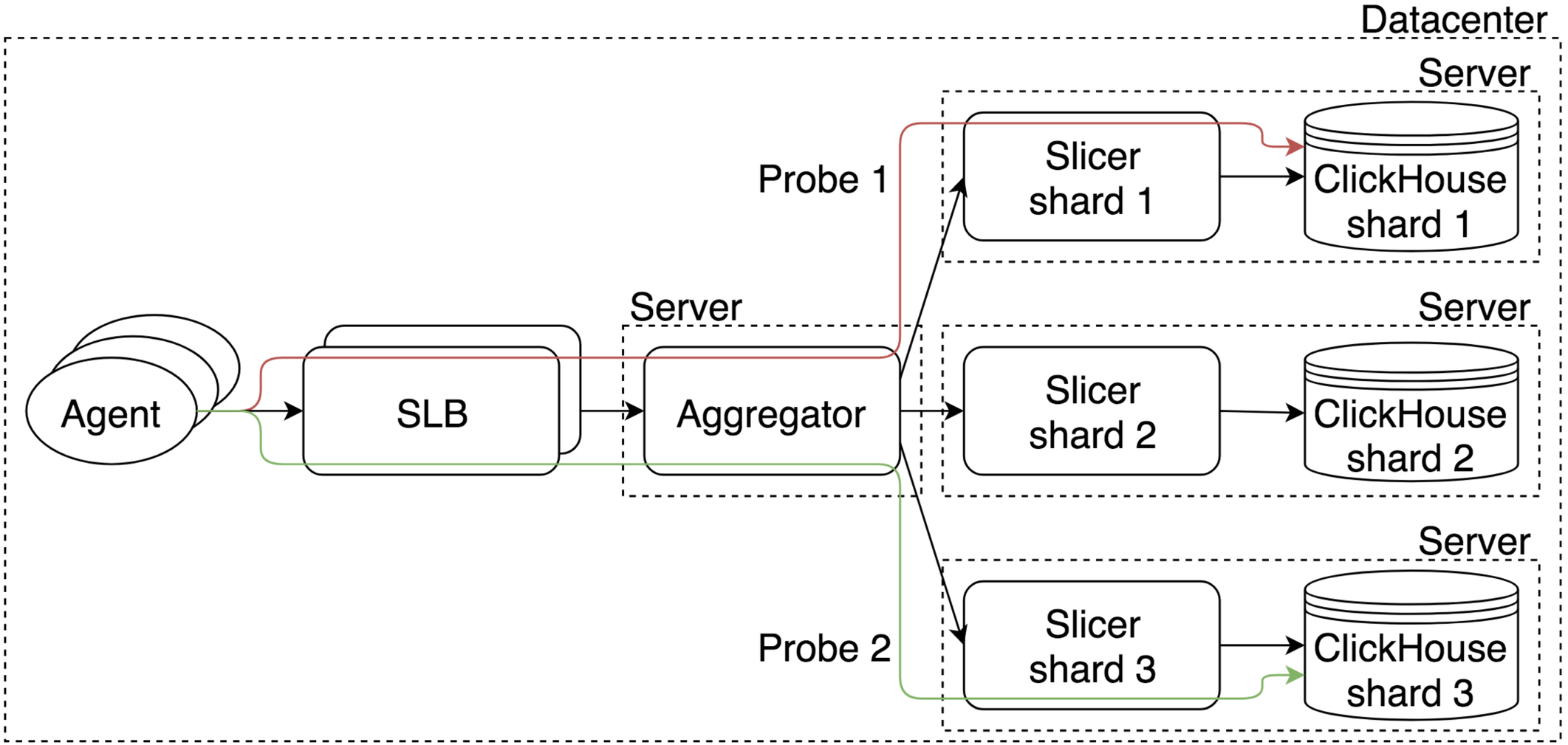
〉 Агенты отправляют пробы в SLB, который распределяет запросы между ДЦ

〉 Всё множество проб принимается агрегатором и распределяется по определённому правилу на независимые подмножества - то есть агрегатор шардирует пробы

〉 Пробы делятся на шарды по паре стойка-источник, стойка-назначение

〉 Соответственно, каждый шард работает со своим объёмом данных, не пересекающимся с другими шардами

Реализация серверной части



Реализация серверной части

〉 От агрегаторов пробы попадают в шарды, каждый из которых записывает свою часть проб в установленный локально экземпляр ClickHouse

〉 Каждые 5 секунд пробы загружаются из ClickHouse обратно в память шарда

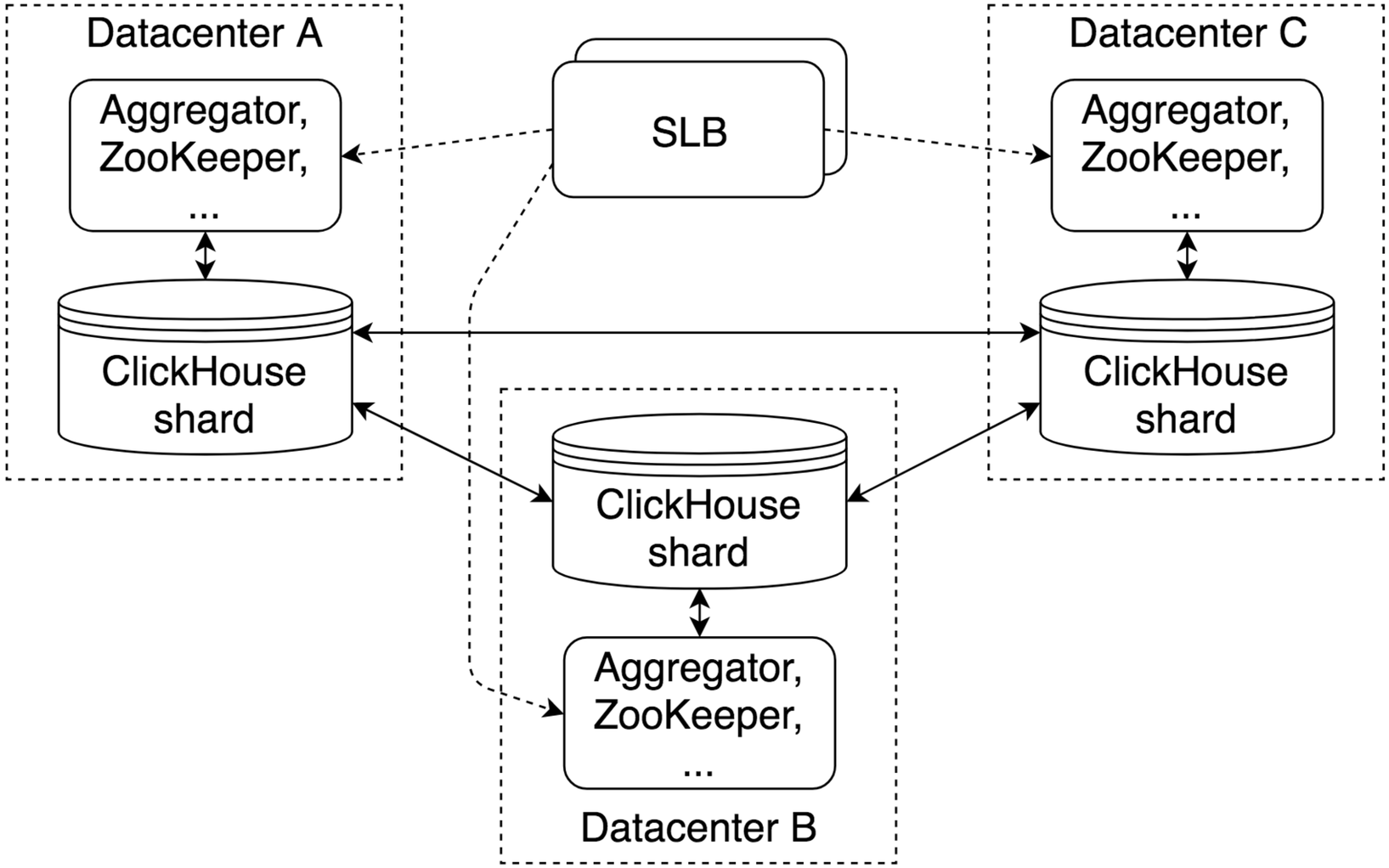
〉 Загруженные в память пробы накладываются на топологию и из этих проб рассчитываются матрицы с показателями сети

〉 Матрицы с шардов собираются воедино агрегатором, который показывает их пользователю

Репликация между датацентрами

〉 Каждый шард имеет по одной реплике в каждом из трёх датацентров
〉 Используется встроенная в ClickHouse репликация, при этом реплицируются только пробы, ничего более
〉 В каждом ДЦ поднят ZooKeeper, нужный для хранения общего состояния как ClickHouse, так и самого сервиса
〉 В случае если реплика начинает отставать, сервис в этом датацентре закрывается целиком и запросы от пользователей и агентов распределяется между оставшимися
〉 То же самое происходит в случае недоступности backend'a
〉 Переключение нагрузки достигается с помощью SLB

Репликация между датацентрами



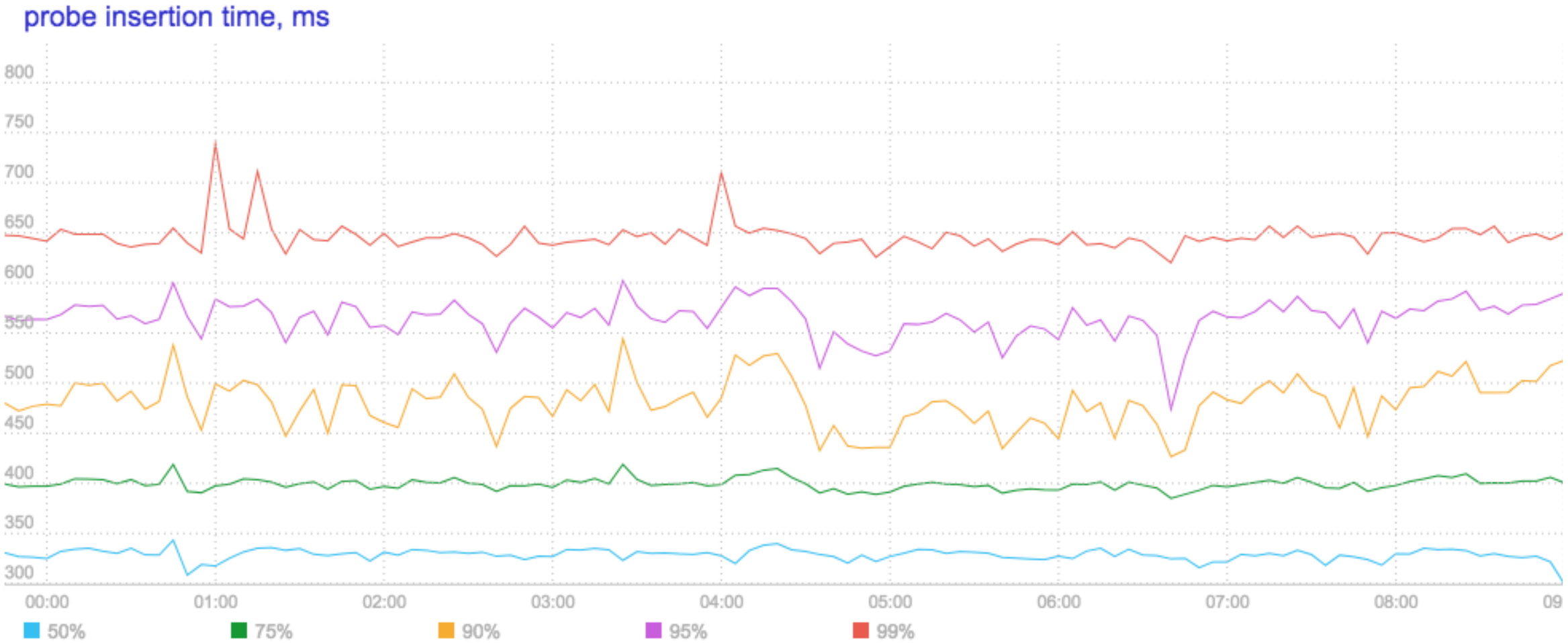
Запись проб в ClickHouse

〉 Для репликации используются таблицы с движком *ReplicatedMergeTree*
〉 В имена таблиц с пробами добавляется текущая дата, что позволяет хранить пробы за нужный интервал времени.
〉 Соответственно пробы записываются в нужную таблицу каждым шардом
〉 Запись проб производится пачками как можно бóльшего размера, обычно это десятки тысяч проб
〉 В фоне периодически проверяется список существующих таблиц, добавляются новые / удаляются старые таблицы
〉 За сутки обрабатывается более 8B проб, которые занимают на диске около 300GB

Запись проб в ClickHouse

```
CREATE TABLE probes_20170619
(
    SourceFqdn String,
    SourceLocation …,
    TargetFqdn String,
    TargetLocation …,
    DropRatio Float64,
    RoundTripTime Float64,
    Generated DateTime DEFAULT now(),
    GeneratedDate Date DEFAULT today()
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/probes_20170619', '{replica}',
  GeneratedDate, (GeneratedDate, Generated, SourceLocation, TargetLocation), 8192)
```

Запись проб в ClickHouse



Время вставки проб, 99% составляет 650 мс

Чтение проб из ClickHouse

〉 Каждый шард периодически перечитывает все пробы за указанный временной интервал, границы которого постоянно сдвигаются вперёд. Интервалы накладываются друг на друга, т.к. репликация асинхронная

〉 На каждом шарде создаются таблицы с движком *Merge*, нужные для чтения проб за произвольные интервалы времени

〉 Для чтения проб со всех шардов используется таблица с движком *Distributed*, благодаря которой ClickHouse самостоятельно соберёт данные с шардов

Чтение проб из ClickHouse

```
SELECT
  SourceFqdn,
  TargetFqdn,
  toUInt64(Generated),
  DropRatio,
  RoundTripTime
FROM probes  // или distributed_probes для чтения истории
WHERE (
  Generated <= toDateTime('2017-06-24 14:08:06') AND GeneratedDate <= toDate('2017-06-24')
  AND Generated >= toDateTime('2017-06-19 14:07:51') AND GeneratedDate >= toDate('2017-06-19')
)
```

Чтение проб из ClickHouse

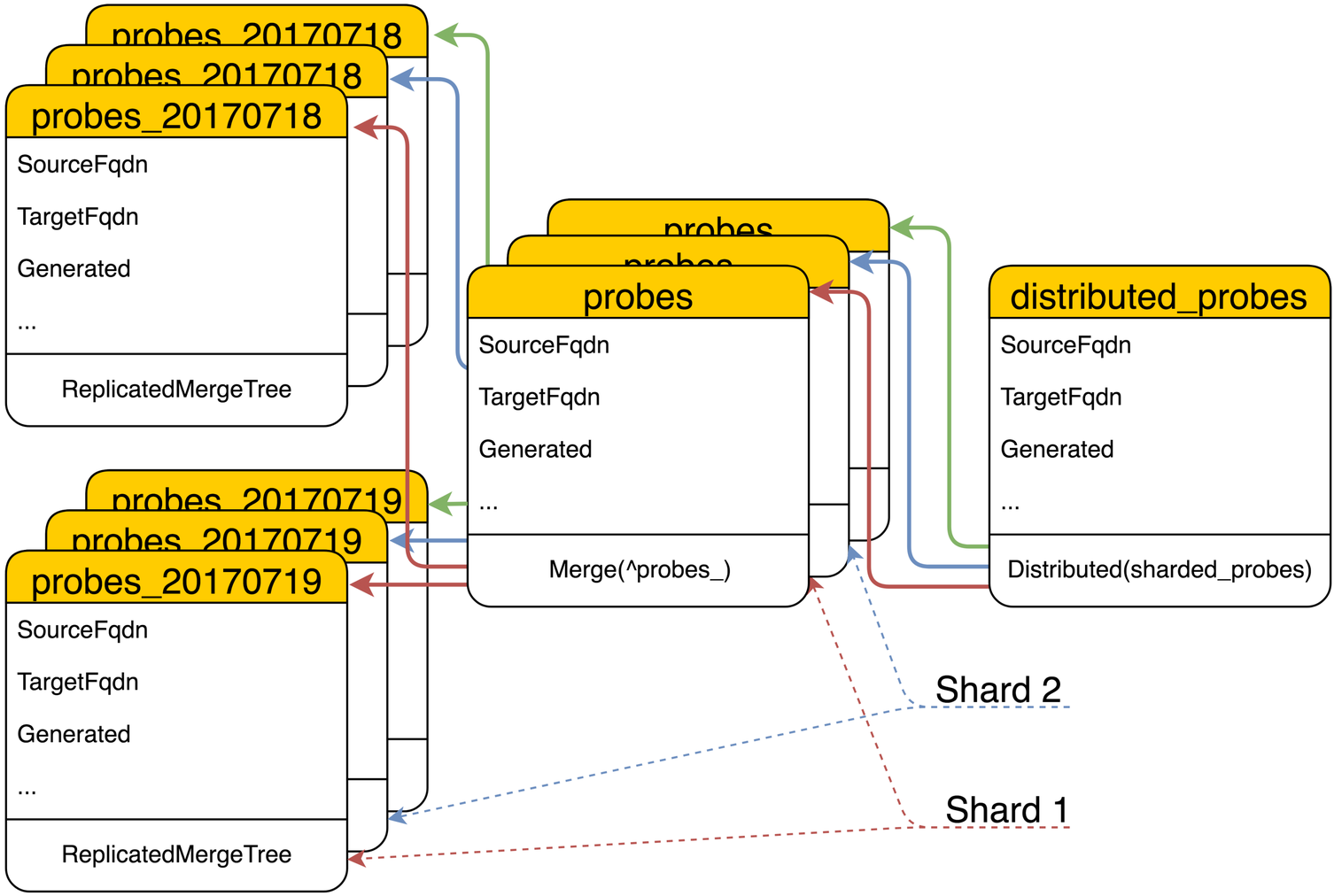
〉 Приложение обращается к случайному шарду ClickHouse, на каждом из них создана таблица *distributed_probes* с движком *Distributed*

〉 В конфигурации каждого шарда ClickHouse указан список всех шардов данного ДЦ

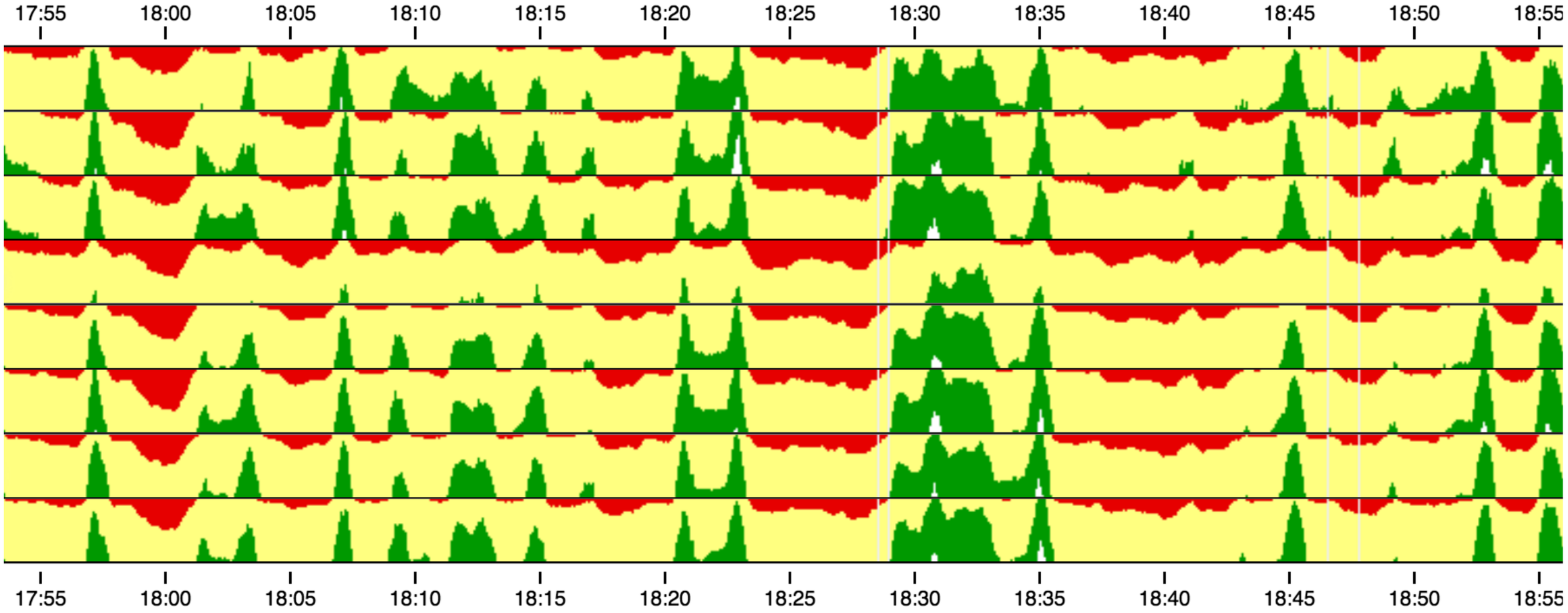
〉 При чтении из *distributed_probes* ClickHouse самостоятельно читает и агрегирует данные из таблицы *probes* с каждого шарда данного ДЦ

〉 Каждый шард в свою очередь при чтении из *probes* читает данные из таблиц с именами *probes_XXXXXXXX*.

Чтение проб из ClickHouse



Запись истории связности в ClickHouse



Запись истории связности в ClickHouse

〉 Логика работы с таблицам устроена похожим на пробы образом, но история **не** реплицируется и в каждом ДЦ пишется независимо

〉 Для удобства чтения в первичный ключ добавлены поля, по которым легко найти все дочерние серии для заданной пары датацентров или модулей

〉 На каждом шарде хранится свой набор серий, чтение производится с помощью таблиц с движком *Distributed*

〉 В историю записываются данные по связности и задержкам на каждом уровне иерархии, для стоек это означает 1.5M точек, генерируемых каждые 30 секунд

〉 Все данные, хранимые в ClickHouse, занимают 4TB диска

Опыт эксплуатации

〉 В каждый компонент встроены мониторы, которые в случае проблем закрывают данную реплику от клиентов. Один из таких мониторов проверяет состояние репликации таблиц
〉 ClickHouse начинает “притормаживать” запросы на вставку если лаг репликации растёт, на этот случай задержка добавляемая к времени выполнению запроса была уменьшена
〉 В случае переналивки сервера с ClickHouse нужные таблицы создаются автоматически, для восстановления данных нужно лишь установить флаг *force_restore_data* для ноды в ZooKeeper
〉 Были подняты пороги на количество “побитых” файлов для автоматического восстановления БД после “жесткой” перезагрузки сервера

Ссылки

〉 Microsoft. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis.
〉 Facebook. NetNORAD: Troubleshooting networks via end-to-end probing.
〉 Google. Localizing packet loss. In a large complex network.

Вопросы?

Спасибо!

Дмитрий Липин
Руководитель группы

 dldmitry@yandex-team.ru

 +7 921 383-36-18