САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Иванищев Василий Олегович

Магистерская диссертация

Распределенная система криминалистического копирования и хранения данных

Направление 02.04.02

Фундаментальная информатика и информационные технологии

Магистерская программа «Технологии баз данных»

Научный руководитель,
кандидат тех. наук,
доцент
Блеканов И.С.

Санкт-Петербург

2018

Содержание

Введение

Актуальность работы

В настоящее время наблюдается большой прогресс в области информационных технологий. Тенденция такого рода способствует развитию программного и аппаратного обеспечения, которое позволяет решить многие задачи в различных сферах деятельности человека. Сейчас можно с большим трудом представить жизнь большинства людей без использования подобных технологий – стали активно использоваться платежные системы, облачные вычисления, запоминающие устройства для записи и хранения важной информации, набирает популярность концепция интернета вещей и т.п. В связи с массовой интеграцией современных компьютерных технологий в области человеческой деятельности при помощи программно-аппаратных средств все чаще совершаются разного рода правонарушения и преступления: нарушения в области авторского и смежных прав, хищения денежных средств, мошенничество, лжепредпринимательство, продажи секретной информации и т.п. Когда в средствах массовой информации сообщается о подобных правонарушениях, зачастую отмечается важность доказательств, собранных с компьютеров.

Такого рода преступления принято называть компьютерными [1]. Наука, занимающаяся исследованием таких правонарушений, называется компьютерной криминалистикой (на англ. computer forensics) [1]. Сам термин, форензика, произошел от латинского «foren», что значит «речь перед форумом». В русский язык это слово пришло из английского. Полная форма этого термина на английском языке звучит следующим образом: «computer forensic science», что дословно означает «компьютерная криминалистическая наука». Согласно определению, компьютерная криминалистика – это прикладная наука о раскрытии преступлений, связанных с компьютерной информацией, об исследовании цифровых доказательств, методах поиска,

получения и закрепления таких доказательств, о применяемых для этого технических средствах [1].

Из-за неуклонного увеличения числа случаев компьютерных преступлений данная область криминалистики становится все более важным объектом для правительства и правоохранительных органов. Трудно представить раскрытие подобных преступлений без современного программно-аппаратного обеспечения, поэтому данная область криминалистики полноценно существует в развитых странах: издан ряд научных трудов (Counterfeiting and Defending the Digital Forensic Process, Alvaro Botas, Ricardo J. Rodriguez; The Art of Memory Forensics, Michael Hale Ligh, Andrew Case, Jamie Levy, AAron Walters и т. д.) имеются учебные курсы, существуют официальные рекомендации, которым необходимо следовать при криминалистической экспертизе.

В других странах компьютерная криминалистика лишь начинает развиваться. К сожалению, несмотря на высокий уровень развития сферы информационных технологий, Россия относится как раз к таковым. Одним из показателей развития является выпуск оборудования для сбора, обработки и анализа цифровых доказательств. Примером может послужить оборудование от таких компаний, как MANDIANT, WindowsSCOPE, Tribble и т.д. В России такого рода обеспечение только начинает производиться, но все чаще – закупается. Данное положение дел объясняется несколькими обстоятельствами: слаборазвитыми теоретическим и прикладным основаниями компьютерно-криминалистической науки, малым количеством доступных публикаций по данному направлению, а также не адаптированной к эффективному обучению основам криминалистической науки системой (высшего) образования.

Основными задачами компьютерной криминалистики являются извлечение и анализ информации, хранящейся в памяти компьютера [2]. Но перед тем, как приступить к анализу извлеченных данных, их необходимо

передать в центр проведения компьютерно-технических экспертиз. К такой передаче данных выдвигается масса требований: высокий уровень надежности, конфинденциальности и скорости. Также необходимо обеспечить целостность изъятых данных. Поэтому в настоящее время подобный обмен информацией осуществляется в основном с использованием запоминающих устройств, что занимает большое количество времени и задерживает расследование.

Несмотря на то, что в России прецедент официально не является источником права, результаты анализа данных, как и сами данные, прошлых судебных дел могут существенно ускорить процесс проверки. Другой потребностью может стать необходимость сбора статистики по данным, полученным в ходе расследования компьютерных преступлений.

На первый взгляд, существует множество инструментов, решающих такого рода задачи по отдельности, но даже решение, основанное на интеграции таких инструментов, не будет удовлетворять выдвинутым требованиям. Это связано с тем, что большинство средств, решающих задачу сбора данных, основаны на клиент-серверной архитектуре, которая не может обеспечить должный уровень отказоустойчивости и масштабируемости. Также среди общих ограничений можно отметить высокую стоимость, отсутствие открытого исходного кода под либеральными лицензиями и, как следствие, невозможность модификации программных компонентов под определенные задачи, платная техническая поддержка и т.п.

Цель работы

Целью данной работы является создание распределенной системы криминалистического копирования и хранения данных в контексте задачи сбора цифровых доказательств.

Задачи работы

Для достижения поставленной цели были сформулированы следующие задачи:

- исследование существующих распределенных система поиска, сбора и хранения данных;
- выбор сетевой архитектуры распределенного программного комплекса;
- разработка прикладного протокола передачи данных;
- тестирование распределенного программного комплекса.

Глава 1. Обзор существующих распределенных систем поиска, сбора и хранения данных

1.1. Классификации сетей

Существуют две основные сетевые архитектуры [3]:

- клиент-серверная,
- одноранговая.

Использование одноранговой архитектуры более приоритетно ввиду большего уровня масштабируемости, автономности и отказоустойчивости.

В некоторых случаях, рассматривают третий тип – гибридный, при котором в сеть добавляется координационный узел(лы) [4]. Но ввиду того, что данный тип мало распространен и не имеет явных достоинств перед остальными типами, опустим его рассмотрение.

Одноранговая архитектура не подразумевает полное равенство узлов, в таких сетях могут находится супер-узлы, позволяющие управлять маршрутизацией и индексацией данных в сети. В связи с этим принято классифицировать одноранговые сети по степени централизации [5]:

- централизованные (на англ. centralized P2P),
- полностью децентрализованные (на англ. pure P2P),
- гибридные (на англ. hybrid P2P).

Существует несколько способов организации связей между узлами, размещения и индексирования ресурсов в оверлейной сети. Поэтому для сетей однорангового типа можно ввести еще одну классификацию [6]:

- структурированные,
- неструктурированные.

В неструктурированных одноранговых системах нет определенной накладываемой на оверлейную сеть структуры – она формируется узлами, которые случайным образом соединяются друг с другом [7]. Поскольку все узлы в сети одинаковые и содержат относительно немного маршрутизационной информации, неструктурированная сеть устойчива к одновременному присоединению или оттоку большого количества узлов [8].

Из недостатков неструктурированной сети можно отметить неэффективный поиск информации и большую нагрузку на сеть.

В дальнейших рассуждениях N будет означать количество участников сети, если не оговорено иное.

В структурированных одноранговых сетях оверлейная сеть имеет определенную топологию, которая гарантирует, что любой узел может эффективно (как правило, за $O(\log N)$ операций) осуществлять процедуру поиска даже очень редкого ресурса. Большинство структурированных одноранговых сетей организованы по принципу распределенной хеш-таблицы (DHT). Для того, чтобы трафик в сети организовывался эффективным образом, каждый узел должен поддерживать в актуальном состоянии список ближайших по какой-либо метрике узлов – это делает структурированную сеть менее устойчивой к высокому притоку/оттоку участников.

Примеры классификации сетей:

1. *HTTP, FTP* – клиент-серверная;
2. *Gnutella 0.4* – одноранговая, полностью децентрализованная, неструктурированная;
3. *Napster* – одноранговая, централизованная, неструктурированная;
4. *Gnutella 0.6* – одноранговая, гибридная, неструктурированная;
5. *Chord, Kademlia, Pastry* – одноранговая, полностью децентрализованная, структурированная.

Ввиду того, что в рамках рассматриваемой задачи маловероятен высокий приток/отток участников за короткий промежуток времени, а также важна скорость выполнения запросов, низкий сетевой трафик и возможность масштабирования – было принято решение выбрать одноранговую полностью децентрализованную структурированную архитектуру сети.

Большинство одноранговых структурированных систем представляет собой распределенную хеш-таблицу (distributed hash table – DHT) [9]. Наиболее известные представители данного класса:

- *CAN – Content Addressable Network,*
- *Chord,*
- *Pastry,*
- *Kademlia*.

У представленных систем много общего: идентификация узлов, схожие по семантике сообщения протоколов, логарифмическая сложность процедуры поиска ресурса по идентификатору в среднем случае [10-13].

1.2. CAN

Content Addressable Network была разработана в 2001 году в Калифорнийском университете совместно с институтом исследований архитектуры интернета AT&T (AT&T Center for Internet Research at ICSI). В основе архитектуры системы лежит многомерная прямоугольная система координат. Каждый из участников сети имеет свою область ответственности в d-мерном пространстве – виртуальную зону координат (на англ. virtual coordinate zone), которая определяется координатами $2d$ точек d-мерного прямоугольника. На рис. 1.1. представлены виртуальные зоны координат 5 узлов при $d = 2$. Далее для узлов вводится понятие соседства – два узла считаются соседями, если по $d - 1$ координатам интервалы их виртуальных зон совпадают полностью или частично, а по оставшейся координате – они граничат. На рис. 1.1. узлы B и D соседи, A и D – нет.

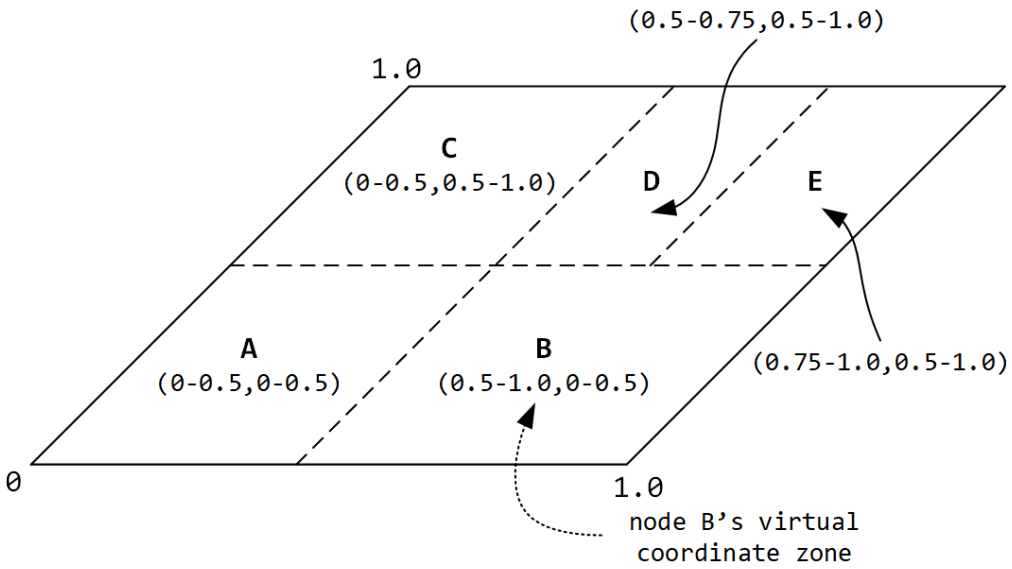

Рис. 1.1. Координатное пространство CAN

Основной структурой маршрутизации узла в CAN является координатная таблица, которая содержит IP-адрес и виртульную координатную зону каждого из соседей. Алгоритм маршрутизации между двумя точками в координатном пространстве заключается в простой последовательной отправке сообщений соседям, чьи координаты ближе к точке назначения.

Главным приемуществом данной одноранговой системы является независимость объема структуры маршрутизации от количества участников. В общем случае каждый участник содержит $2d$ записей в своей таблице. В работе [10] показано, что для d-мерного пространства, разделенного на N одинаковых виртуальных зон, средняя длина пути маршрутизации требует порядка $O(n^{1/d})$ сообщений. Этот факт как раз и является следствием константного объема таблицы маршрутизации. Ввиду требований к эффективности процедуры поиска нельзя в полной мере использовать рассмотренную DHT в проводимой работе.

На первый взгляд может показаться, что если d задавать динамически – в зависимости от количества участников сети, – можно добиться логарифмической сложности поиска. Однако в таком случае мы лишимся

постоянности количества записей в таблице маршрутизации и, следовательно, при изменении состава участников системы придется перестраивать структуры маршрутизации практически всех узлов.

1.3. Chord DHT

Данная система была разработана в 2001 году в Массачусетском технологическом институте. В ее основе лежит технология консистентного хеширования. В качестве идентфикаторов узлов и данных выступают m бит результата хеш-функции SHA-1 от IP-адреса и идентификатора данных соответственно. Число m является параметром системы и выбирается в зависимости количества участников системы таким образом, чтобы избежать коллизий. Идентификаторы в Chord образуют кольцо идентификаторов по модулю 2^m (на англ. identifier circle modulo 2^m). Ключ `k` назначается первому узлу, чей идентификатор равен или предшествует k в пространстве идентификаторов. Такой узел называется преемником (на англ. successor) ключа k и обозначается как `successor(k)`. Поскольку идентификаторы упорядочены на кольце, преемником ключа k является первый узел, встретившийся по часовой стрелке по пути от k (см. рис. 1.2.). По аналогии вводится понятие предшественника (на англ. predecessor) узла – `predecessor(k)`.

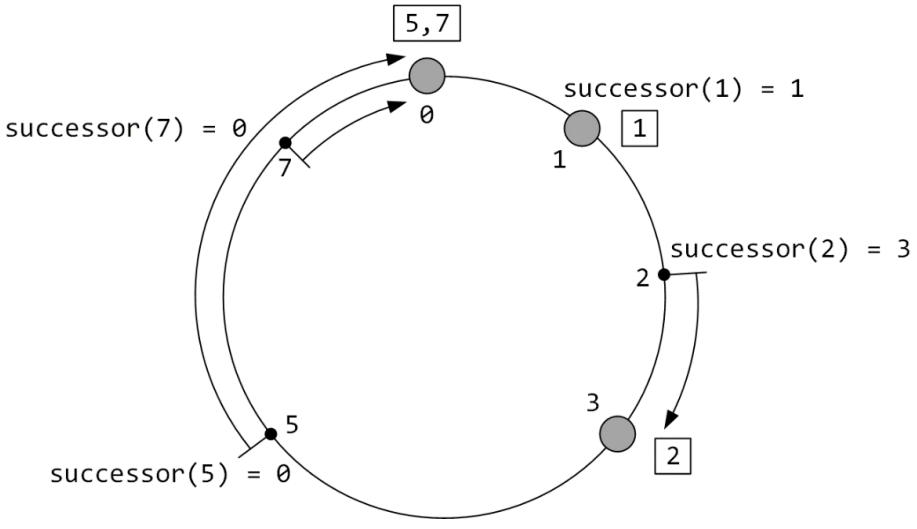

Рис. 1.2. Кольцо идентификаторов по модулю 2^3 Chord.

В работе [11] показано, что в Chord выполнение процедуры поиска имеет логарифмическую сложность, таблицы маршрутизации имеют размер, логарифмечески зависящий от количества узлов, а процедура обновления структур маршрутизации в случае удаления/добавления участника требует в среднем $O(\log^2 N)$ сообщений. Для того, чтобы добиться логарифмической сложности поиска, каждый узел хранит фингер-таблицу (на англ. finger table), содержающую до m записей, каждая из которых содержит информацию о преемнике successor($(n + 2^{i-1}) \bmod 2^m$), где i – номер записи (от 1 до m), n – идентификатор текущего узла, m – параметр системы. В таблице 1.1. представлены данные из фингер-таблицы узла с идентификатором 1 из рис. 1.2.

Key	Value = successor(Key)
$1 + 2^0 = 2$	3
$1 + 2^1 = 3$	3
$1 + 2^2 = 5$	0

Таблица. 1.1. Фингер-таблица Chord.

Стоит отметить, что данная система также предназначена для выравнивания нагрузки на узлы. Но для поддержкки такого механизма требуется $O(\log^2 N)$ сообщений при каждом изменении состава оверлейной сети, что для наших задач является критичным, ввиду того, что разрабатываемая система должна быть не требовательна к широкому пропускному каналу и стабильному соединению. Еще одна особенность, из-за которого использование Chord в данной работе не представляется возможным, касается степени близости узлов. К сожалению, введенное расстояние не является метрикой, поскольку не выполнена аксиома симметричности метрического пространства. Следствием ассиметричности близости узлов является увеличенное количество конфигурационных сообщений и отсутствие

гарантий, что обратный запрос будет выполнен за то же количество сообщений, что и прямой.

1.4. Pastry DHT

Система Pastry была разработана в 2001 году сотрудниками университета Райса и Microsoft Research Ltd. Организация пространства идентификаторов в одноранговой оверлейной сети Pastry напоминает реализацию Chord: каждому узлу присваивается случайный 128-битный идентификатор (*nodeId*), который используется для указания его позиции в кольцевом пространстве идентификаторов [12]. Но, в отличие от Chord, ответственным за ресурс с ключом k является тот узел, чей идентифкатор является численно ближайшим к k (см рис. 1.3.).

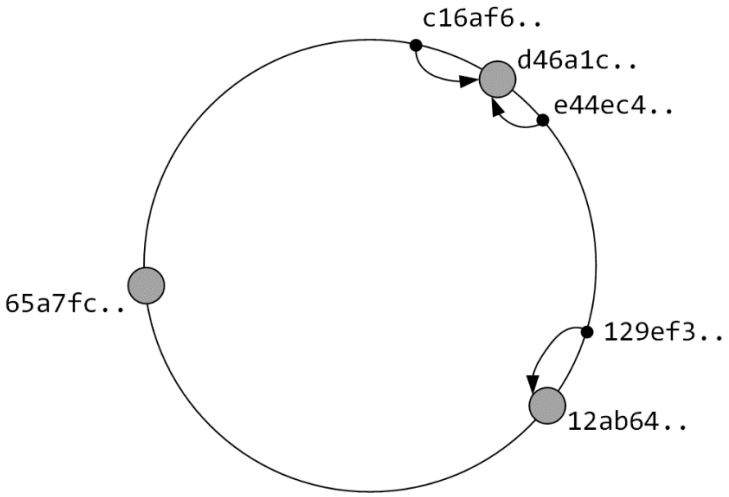

Рис. 1.3. Кольцо идентификаторов Pastry.

Для маршрутизации идентификаторы *nodeId* и ключи рассматриваются как последовательность цифр с основанием 2^b (где b – параметр системы и в классической реализации равен 4). В рамках процедуры поиска система Pastry направляет сообщения к узлу, чей идентификатор численно близок к рассматриваемому ключу. В большинстве случаев на каждом шаге маршрутизации получивший сообщение узел перенаправляет его узлу, у которого общий префикс с ключом больше, по крайней мере, на одну цифру, чем у текущего узла. В случае, если такого узла не оказалось в структуре

маршрутизации, сообщение перенаправляется узлу, длина общего префикса которого аналогична текущему, но *nodeId* численно более близок к рассматриваемому ключу. Таким образом, в большинстве случаев достигается порядок $O(\log_{2^b} N)$ количества сообщений при выполнении процедуры поиска.

Для обеспечения такой сложности поиска каждый участник системы поддерживает в актуальном состоянии структуру маршрутизации, которая состоит из таблицы маршрутизации, множества окрестностей (на англ. neighborhood set) и множества листовых узлов (на англ. leaf set).

Таблица маршрутизации содержит $\log_{2^b} N$ строк с $2^b - 1$ записями в каждой. Каждая запись в i-ой строке относится к узлу, длина общего с текущим узлом префикса *nodeId* которого равна i, в то время как $(i + 1)$-ая цифра принимает одно из $2^b - 1$ значений, за исключением $(i + 1)$-ой цифры *nodeId* текущего узла.

Множество окрестностей *M* содержит информацию о ближайших узлах согласно внешней метрике близости, которая задается при конфигурации системы. Примером такой метрики может стать количество прыжков команды traceroute. Данная подструктура не используется в процессе маршрутизации, она необходима для инициализации и поддержки в актуальном состоянии других подструктур маршрутизации. Также она используется для проверки актуальности узлов (на англ. dead peer detection). Как правило, размер множества *M* равен 2^{b+1}.

Множество листов содержит ближайшие узлы с точки зрения численной близости их *nodeId*, делится на ближайшие «слева» и «справа». Данное множество используется в конечном этапе маршрутизации. Размер множества листов совпадает с размером множества окрестностей.

На рис. 1.4. проиллюстрированы описанные подструктуры маршрутизации для *nodeId* = 10233102 и b = 2.

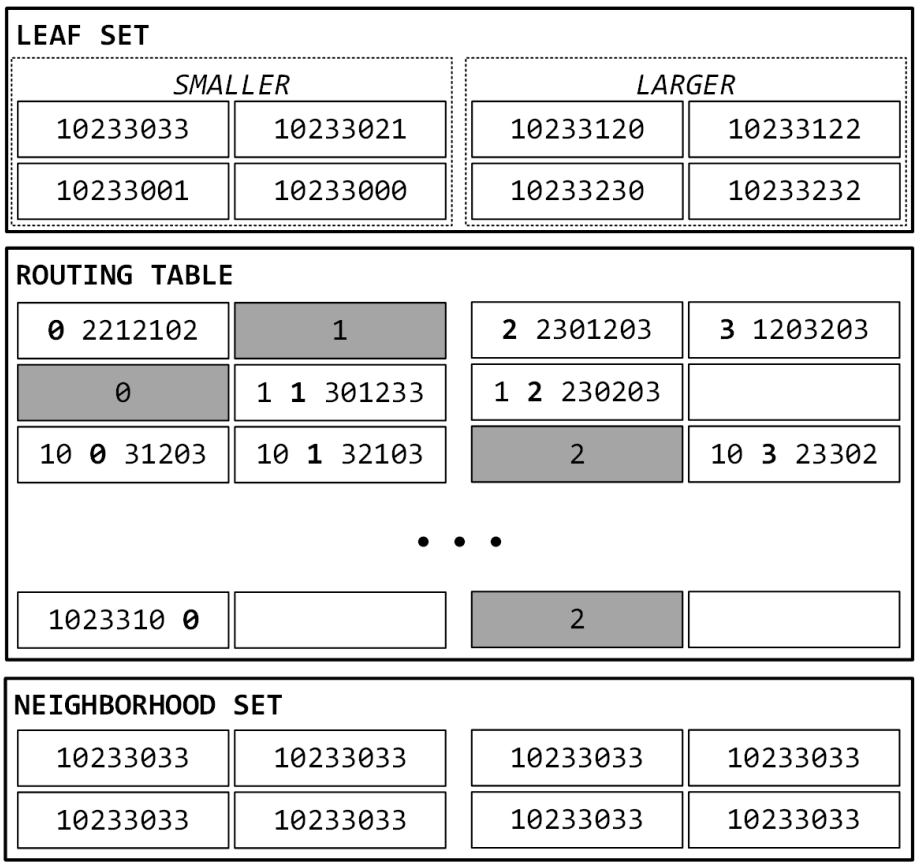

Рис. 1.4. Структура маршрутизации Pastry.

Несмотря на то, что степень близости узлов, используемая на основном этапе маршрутизации, обладает свойством симметричности и позволяет добиться логарифмической сложности поиска, общая сложность процедуры поиска, строго говоря, не логарифмическая. Это объясняется тем, что метрика, применяемая в завершающем этапе маршрутизации, численная и в худшем случае способствует линейной сложности алгоритма поиска. Это усложняет формальное доказательство вычислительной сложности и анализ наихудшего поведения.

Также стоит подчеркнуть сложность структуры маршрутизации, хранящейся на каждом узле системы. Необходимость поддержки второстепенной таблицы листов приводит к увеличению размера общей структуры до $O(2^b \log_{2^b} N)$, времени инициализации новых узлов и усложнению протокола. В связи с этим не представляется возможным использование данной DHT при решении поставленных задач. Однако

возможность задать внешнюю метрику близости узлов очень важна в контексте задачи сбора цифровых доказательств. Поскольку множество окрестностей используется независимо от основого алгоритма маршрутизации, было принято решение использовать данный механизм.

1.5. Kademlia DHT

Kademlia разработана в 2002 году в Нью-Йоркском университете Петром Маймунковым и Давидом Мазьером. Kademlia имеет ряд достоинств, одним из которых является минимальное количество конфигурационных сообщений. Подобная информация распространяется автоматически, как побочный эффект ключевых операций системы [13].

В Kademlia используется базовый подход идентификации узлов – каждый участник системы имеет *ID* из 160-битного пространства ключей. Маршрутизация в системе устроена таким образом, что пары <ключ, значение> хранятся в таблицах узлов, чьи идентификаторы близки к рассматриваемому ключу по метрике, основанной на операции «исключающего ИЛИ» (на англ. XOR). Использование данной функции возможно, поскольку пара <операция XOR, множество идентификаторов узлов> удовлетворяет аксимомам метрического пространства. Многие достоинства Kademlia основаны на применении данной метрики. Например, благодаря ее симметричности, обратные сообщения эквивалентны прямым, что позволяет получить больше полезной информации из поисковых запросов, по сравнению с Chord. Также использование данной метрики позволяет формально доказать тезисы о размерах таблиц маршрутизации и сложности поиска.

Таблица маршрутизации узла в Kademlia делится на части, количество которых, в общем случае, равно размерности пространства идентификаторов (как правило, 160). Каждая из частей ответственна за хранение информации об узлах на расстоянии от 2^i до 2^{i+1}. Рассмотрим случай для узла с идентификатором *0111*. Нулевая часть таблицы будет хранить информацию об

узлах на расстоянии от 1 до 2 (в двоичном виде от *0001* до *0010*), на таком расстоянии по метрике XOR может находиться всего лишь один узел: *0110*. По аналогии, вторая часть таблицы будет содержать узлы на расстоянии от *0010* до *0100*, т. е. узлы *0100* и *0101*. Соответственно, третья часть – узлы *00xx*.

Если рассмотреть пространство узлов в виде двоичного дерева, то становится видно, что таблицы маршрутизации в Kademlia хранят информацию об узлах из каждого поддерева, несодержащего ветвь рассматриваемого узла (см. рис. 1.5.). Это следует из того факта, что в полностью заполненном бинарном дереве идентификаторов расстоянием XOR между двумя идентификаторами является высота наименьшего поддерева, содержащего их обоих.

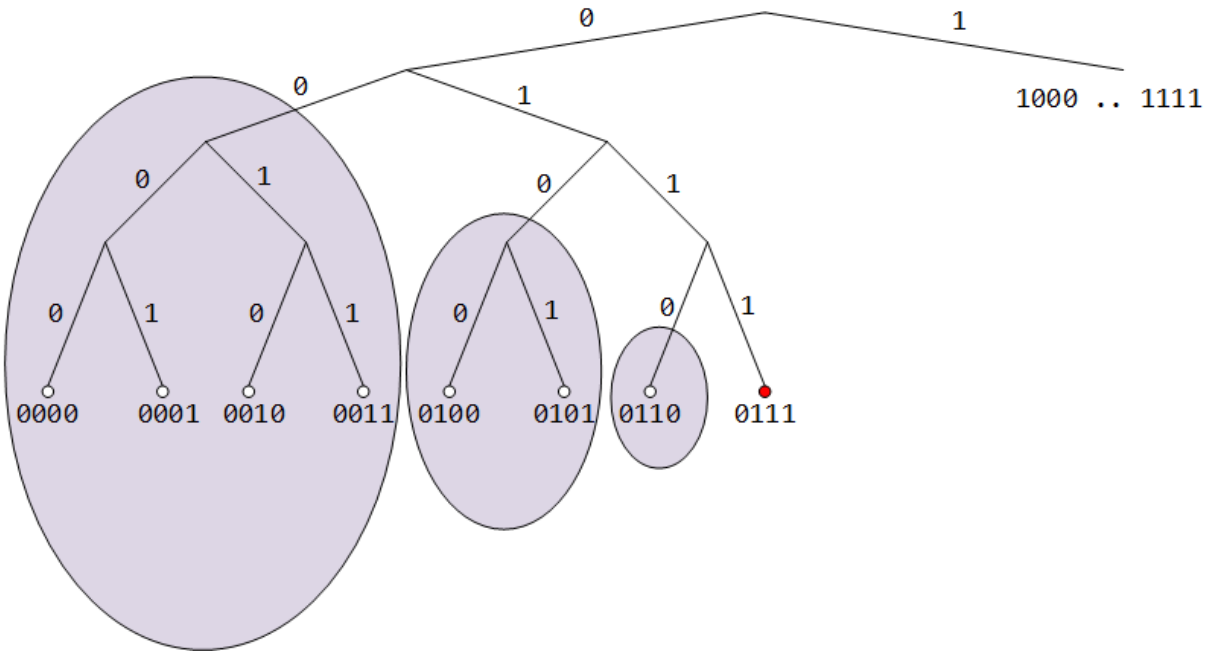

Рис. 1.5. Представление пространства идентификаторов в Kademlia DHT в виде бинарного дерева. Овалами выделены поддеревья, соответствующие частям таблицы маршрутизации узла *0111*.

Протокол Kademlia гарантирует, что каждый узел знает по меньшей мере об одном из узлов в каждом поддереве. Благодаря этому, каждый узел может успешно выполнить процедуру поиска другого узла по его идентификатору. Схема на рис. 1.6 иллюстрирует процедуру поиска узла *0001*,

инициированную узлом *0111*. Черным цветом выделены узлы, информация о которых имеется у рассматриваемого узла. Видно, что рассматриваемая схема удовлетворяет описанным выше требованиям о знании хотя бы одного узла из каждого поддерева.

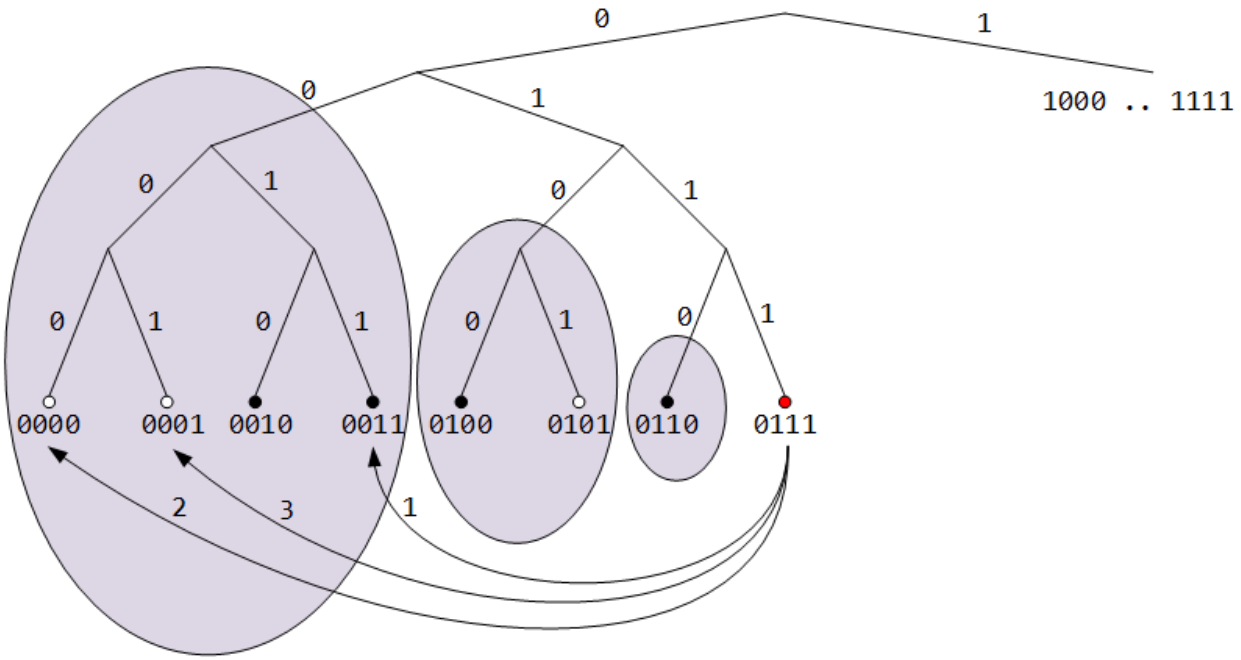

Рис. 1.6. Процедура поиска узла с идентификатором *0001* в Kademlia DHT.

Этапы процедуры поиска, продемонстрированной на рис. 1.6.:

1. При помощи определения расстояния до узла *0001* происходит выбор части таблицы, ответственной за поддерево, в котором хранится искомый узел. Далее, поскольку рассматриваемый узел отсутствует в таблице маршрутизации, из текущей части таблицы выбирается ближайший к нему узел, которому отсылается поисковый запрос (*0011*).
2. В таблице маршрутизации узла *0011* не оказывается узла *0001*, поэтому он осуществляет выбор части таблицы и возвращает ближайший узел *0000*.

3. В таблице маршрутизациии узла *0000* есть узел *0001*, который он и возвращает. Стоит отметить, что данного узла не может не быть в таблице, поскольку данное поддерево состоит из единственного узла.

В Kademlia вводится ограничение на число хранимых узлов в каждой части таблицы маршрутизации – *k*. Поэтому каждую часть таблицы принято называть k-bucket. Данное число является параметром системы, конкретное оптимальное его значение – 20 – было получено эмперическим путем. Таким образом, каждый узел имеет знания об от 1 до *k* узлов в каждом поддереве бинарного дерева.

Информация об узлах в k-bucket отсортирована по времени последнего контакта с узлом – в начале списка находятся узлы, контакт с которыми был раньше, чем с другими. Когда узел в Kademlia получает какое-либо сообщение (запрос или ответ) от другого узла, обновляется соответствующий узлу-отправителю k-bucket. Далее происходит запись отправителя в конец списка, либо его перемещение, если узел-получатель уже знал об отправителе. В случае, если k-bucket полон, узел-получатель последовательно, начиная с первого в спике, пингует узлы до тех пор, пока не найдется неответивший узел. Если такой узел нашелся – он удаляется, и новый узел записывается в конец списка.

Одни из очевидных достоинств такой архитектуры таблиц маршрутизации – масштабируемость и отказоустойчивость. В случае, если произойдет большой приток участников системы, тем самым вызвав подобие DDoS атаки – это не сбросит состояния таблиц маршрутизации, т.к. новые узлы будут добавлены, только если старые станут неактивными. Следующее достоинство заключается в том, что узлы, имеющие большую вероятность закрепиться в сети, останутся в таблицах маршрутизации других участников. Эта особенность основана на экспериментальных данных и вытекает из того, что «старые» узлы хранятся в начале списка k-bucket и не будут вытеснены новыми узлами, пока будут доступны.

Протокол Kademlia состоит из 4 удаленных вызовов процедур (на англ. remote procedure call – RPC): `PING`, `STORE`, `FIND_NODE`, `FIND_VALUE`. `PING` необходим для проверки состояния узла в сети. Запрос `STORE` позволяет разместить информацию на заданном узле. Для повышения доступности информации, `STORE` производится для *k* узлов, ближайших к идентификатору размещаемых данных. Запрос `FIND_VALUE` используется для поиска значения по ключу. Возвращает значение или *k* ближайших к интересующим данным узлов. Как правило, вызывается рекурсивно с двумя условиями выхода: получено значение, или все полученные узлы уже опрошены. `FIND_NODE` отличается от `FIND_VALUE` тем, что всегда возвращает *k* узлов. Чаще всего используется при присоединении нового узла в систему.

Большинство вызовов в Kademlia совершаются асинхронно, а количество одновременно выполняемых запросов – *alpha* – является параметром системы и, как правило, равно трем. Это позволяет уменьшить сетевые задержки путем обхода проблемных узлов и улучшить общее быстродействие системы.

Также стоит отметить количество и популярность приложений, основанных на протоколе Kademlia – среди них Tox, Kad Network, Ethereum, BitTorrent. Учитывая данный факт и остальные перечисленные достоинства, было принято решение остановиться на этой реализации DHT.

Глава 2. Особенности архитектуры

Концепция архитектуры разрабатываемой системы основывается на распределенной хеш-таблице Kademlia. Однако некоторые особенности данной реализации не удовлетворяют выдвинутым требованиям, в связи с чем появилась необходимость модернизировать существующий протокол.

2.1. Локальность сетевых маршрутов

В Kademlia близость узлов определяется операцией XOR над их идентификаторами, а значит не имеет никакого отношения к географической близости узлов. Поэтому было принято решение использовать композитную идентификацию узлов. При таком подходе, идентификатор узла представляет собой пару из 160-битного случайного идентификатора и IP-адреса. Первое значение пары используется в качестве аргумента введенной метрики при выполнении процедуры поиска, как и в других классических DHT-системах. Второе значение играет важную роль при передаче ответственности за данные, а также при проверке актуальности состояния узлов. Следует признать, что метрика, использующая IP-адреса, не всегда выражает связь с географической близостью и шириной пропускного канала между узлами, но в большинстве случаев эта зависимость присутствует и может быть использована.

Стоит заметить, что и IP-адрес и 160-битный идентификатор присутствует в таблице маршрутизации Kademlia, поэтому доработка заключается в изменении структуры данных – вводится дополнительный индекс. В дальнейшем возможно добавление других параметров, отражающих тесную связь с временем задержки обмена сообщениями и скоростью передачи данных между узлами.

Еще один способ поддержки локальности отправляемых запросов с целью уменьшения задержек и увеличения быстродействия системы – это использование дополнительной структуры маршрутизации, подобно

множеству окрестностей в Pastry. Данное множество содержит список ближайших по внешней метрике узлов и находит свое применение в двух случаях:

- dead peer detection – аналогично тому, как это используется в Pastry DHT
- выбор посредника при передаче фрагментов данных в случае отсутствия стабильной связи с центром проведения компьютерно-технических экспертиз.

Выбор способа поддержки внешней метрики является параметром системы, эффективность того или иного способа зависит от количества участников системы, плотности их размещения и т.п.

2.2. Проверка целостности данных и аутентификация

Kademlia не покрывает вопросов аутентификации узлов и проверки целостности передаваемых данных. Существует множество работ, посвященных атакам на Kademlia-based сети [14-16]. Поэтому возникла необходимость в добавлении механизма обеспечения целостности сообщений и аутентификации источника данных.

Было рассмотрено семейство алгоритмов, разработанных для решения таких задач.

MAC

Код аутентификации сообщений – MAC (сокращение от англ. message authentication code) позволяет подтвердить, что сообщение поступило от заявленного отправителя и не было изменено. Представляет собой набор символов, который добавляется к сообщению и проверяется получателем. Данная имитовставка генерируется специальным MAC-алгоритмом, который на вход получает секретный ключ и само сообщение. Принимающая сторона проделывает те же действия и сравнивает результат MAC-алгоритма с принятым кодом.

Поскольку в данном механизме основную роль играет MAC-алгоритм, рассмотрим основные реализации этой технологии.

CBC-MAC

Данная реализация механизма MAC использует в качестве алгоритма сцепление блоков шифра – CBC (сокращение от англ. cipher block chaining). Использование сцепления блоков обусловлено тем, что длина сообщения может быть произвольной.

HMAC

HMAC – код аутентификации сообщений, использующий хеш-функции (на англ. hash-based message authentication code). В качестве MAC-алгоритма данная функция использует любую итеративную криптографическую хеш-функцию [17].

Как известно, скорость работы хеш-функций выше скорости работы симметричных блочных шифров, используемых в CBC-MAC. В работах [18-19] описаны возможные атаки, проводимые на CBC-подобные реализации MAC-механизма, что говорит о слабой криптостойкости алгоритмов. При этом вероятность атаки на реализацию MAC с использованием хеш-функций равна, а в некоторых случаях ниже, чем вероятность атаки на встроенную хеш-функцию [20]. В связи с этими фактами, было решено использовать HMAC с хеш-функцией SHA-1, имеющую доказанную высокую криптографическую стойкость.

Стоит отметить, что добавление имитовставки во все сообщения протокола может ухудшить быстродействие системы и увеличить процесс инициализаци новых участников, поэтому данный механизм используется только при передаче данных от агентов в центральный узел.

2.3. Прямая передача данных

Для того, чтобы обеспечить возможность быстрой передачи данных напрямую из оперативной памяти агента, необходимо расширить протокол системы. Было введено новое сообщение, нотифицирующее определенный набор участников сети о появлении новых данных. После получения данной нотификации, узел, которому предназначены эти данные – как правило, это центр компьютерно-технических экспертиз, – инициирует процедуру поиска и выполняет копирование данных.

В случае, если отстутствует прямое стабильное соединение между агентом, получившего доступ к данным, и центром, первый последовательно распространяет фрагменты данных среди участников, близких по внешней метрике. Узлы, получившие фрагменты данных, нотифицируют участников, которым эти данные предназначены.

Глава 3. Особенности программной реализации

3.1. Транспортный сетевой протокол

Классическая реализация Kademlia использует на транспортном уровне протокол пользовательских датаграмм – UDP (сокращение на англ. user datagram protocol). Как известно, данный протокол не гарантирует доставку данных, а порядок достижения датаграммами цели не детерминирован. Поскольку разрабатываемая система должна обеспечивать высокую степень надежности передачи и доступности данных, было решено заменить транспортный протокол на TCP для ряда сообщений, среди которых, например, нотификация о получении доступа к новым данным.

Так как в системе используются два транспортных протокола, на каждом узле работает как UDP, так и TCP сервер, каждый из которых ответственен за обработку своих команд. Это незначительно усложняет программную реализацию, но, как было сказано выше, увеличивает надежность.

3.2. Прикладной сетевой протокол

Все сообщения протокола содержат идентификатор сообщения, 160-битный идентификатор узла-отправителя, порт альтернативного сетевого протокола и информацию о связи с центром проведения экспертиз. Вызовы, подразумевающие ответ от получателя, дополнительно содержат случайное значение из пространства идентификаторов (на англ. magic cookie), которое ожидается в ответном сообщении. Сообщения, требующие аутентификации, дополнительно включают в себя HMAC.

Сообщения протокола (в скобках указано значение, используемое в соответствующем поле сетевого пакета):

- `PING(0)` – проверка доступности заданного узла. Повторяет назначение в Kademlia.

- `STORE(1)` – размещение информации о ресурсе, которая является парой 160-битного идентификатора и IP-адреса узла, на котором хранится ресурс, на заданном узле. Повторяет назначение в Kademlia.
- `FIND_VALUE(2)` – поиск заданного значения по 160-битному ключу. Повторяет назначение в Kademlia.
- `FIND_NODE(3)` – поиск k узлов, ближайших к заданному 160-битному ключу. Повторяет назначение в Kademlia.
- `NEW_DATA(4)` – сообщение, целью которого является оповещение специального набора участников сети о появлении новых данных, требующих отправки. В состав сообщения входит метаинформация о источнике данных, список хеш-сумм содержимого фрагментов данных, HMAC.
- `GET_DATA(5)` – запрос на загрузку данных. Инициируется, как правило, центром проведения экспертиз.
- `UPLOAD_DATA(6)` – запрос на передачу данных узлу из множества окрестностей, имеющему стабильную связь с центром. Требует подтверждения перед непосредственной передачей.

На рис. 3.1. изображена организация структуры одного из пакетов, подразумевающего ответ и аутентификацию. Основные составляющие:

`Proto ID` (48 бит) – фиксированный набор битов, позволяющий понять, что принимаемые данные следует интерпретировать как пакет разработанной системы (43 4F 50 45 45 52).

`Source ID` (160 бит) – идентификатор отправителя сообщения.

`Source other port` (16 бит) – порт альтернативного сетевого протокола. Если пакет принят по TCP – то указывается порт UDP и наоборот.

`Message ID` (3 бита) – идентификатор сообщения протокола.

`Master ID` (160 бит) – идентификатор центрального узла, к которому относится отправитель.

`MA` (2 бита) – связь с центральным узлом, к которому относится отправитель. Возможные значения: 0 – нет связи, 1 – есть связь. Остальные значения зарезервированы.

`Magic cookie` (160 бита) – значение, ожидаемое в ответе.

`HMAC` (160 бит) – код аутентификации сообщения.

`Payload` – основное содержимое сообщения.

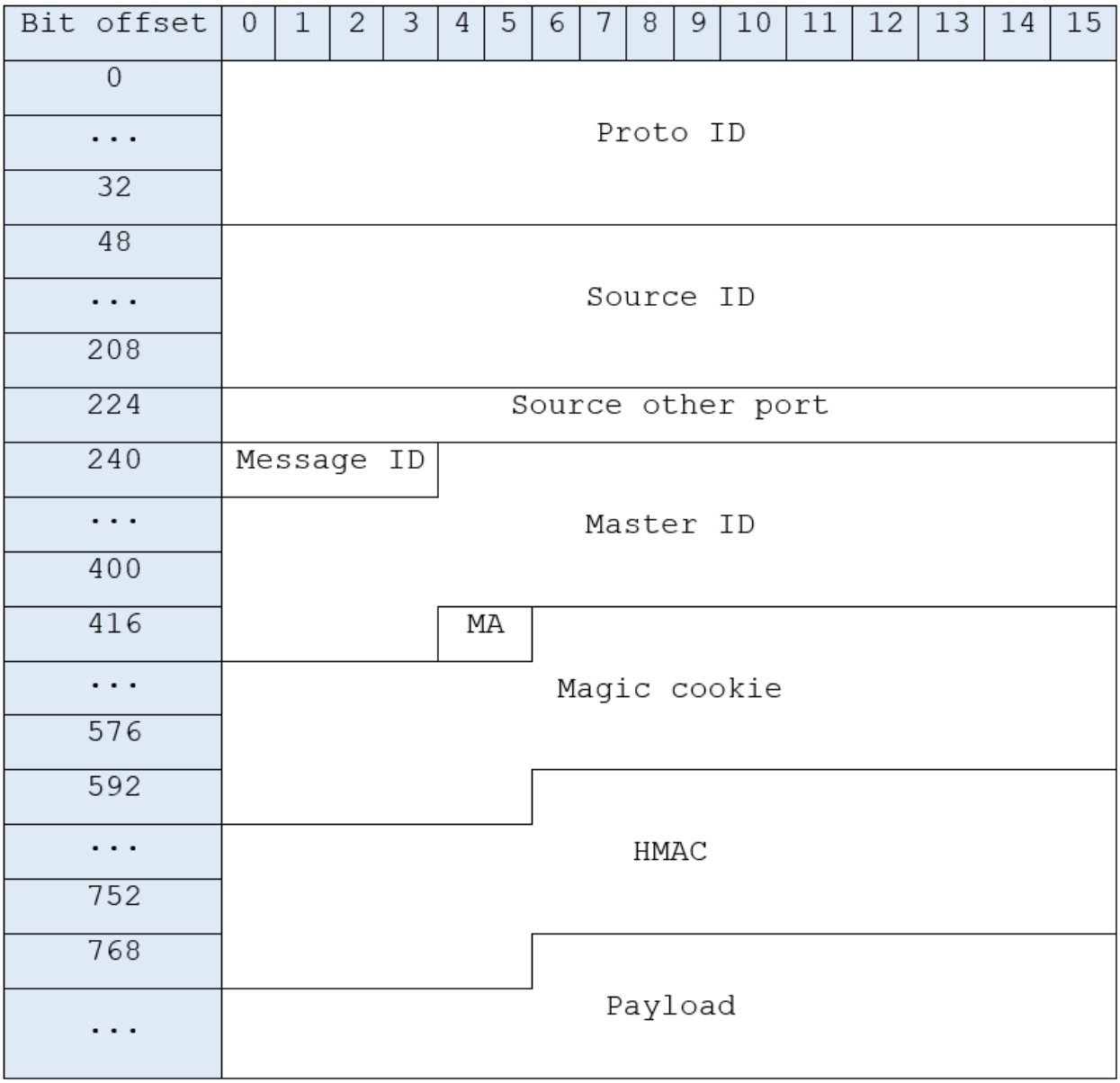

Рис. 3.1. Структура сетевого пакета

3.3. Сценарии работы системы

Перед непосредственным описанием алгоритмов работы системы представим в упрощенном виде основные структуры данных, которые должен поддерживать в актуальном состоянии каждый участник системы.

Структура `contact`, описывающая узел в таблице маршрутизации, содержит (см. табл. 3.1.):

- `ID` – 160-битный идентификатор контакта;
- `TCP/UDP Address`, – конечные точки TCP и UDP;
- `cookies` – значения, ожидаемые в ответном от данного узла сообщениях;
- `Master ID` – идентифкатор центрального узла;
- `Master Access` – степень доступности центрального узла.

name	type
ID	byte[20]
TCP Address	address:port
UDP Address	address:port
cookies	set of byte[20]
Master ID	byte[20]
Master Access	byte

Таблица 3.1. Программная структура узла в таблице маршрутизации

Структура таблицы маршрутизации `routing_state` содержит (см. табл. 3.2.):

- `kBuckets` – основную структуру маршрутизации;
- `Neighborhood Set` – множество окрестностей;
- `Masters Set` – множество узлов, заинтересованных в получении данных (как правило, центры компьютерно-технических экспертиз).

name	type
kBuckets	contact[160][k]
Neighborhood Set	contact[k]
Masters Set	set of contact

Таблица 3.2. Программная структура таблицы маршрутизации

Основная структура участника разработанной системы содержит (см. табл. 3.3.):

- `Node ID` – идентификатор рассматриваемого узла;
- `Configuration` – структура конфигурции, которая содержит TCP/UDP порты и другие параметры системы.
- `Routing State` – таблица маршрутизации.

name	type
Node ID	byte[20]
Configuration	config struct
Routing State	routing_state

Таблица 3.3. Основная программная структура

Для того, чтобы новый участник смог успешно присоединиться к системе, он должен знать IP-адрес по крайней мере одного узла, находящегося в системе. Как правило, подобные узлы называются загрузочными (на англ. bootstrap), и их адреса заранее известны [21]. Информация о bootstrap-узлах передается в систему через структуру `Configuration`.

Присоединение нового участника:

- Регистрация в общем реестре: выдача 160-битного идентификатора и ключа шифрования, используемого в механизме HMAC.

- Отправка сообщения FIND_NODE со своим идентификатором загрузочным узлам.
- Рекурсивная отправка сообщения FIND_NODE узлам, полученным в результате предыдущего запроса. Опрос завершается, если в результате рекурсивных вызовов не появляются новые узлы, либо достигнуто максимальное количество узлов (параметр системы).
- Инициализация структур kBuckets, Neighborhood Set и Masters Set на основе полученного на предыдущих шагах множества узлов.

Отправка данных в исследовательский центр при наличии стабильного соединения между агентом, получившим доступ к данным, и исследовательским центром (см. 3.2.):

- Разбиение целевых данных на фрагменты размера 256 MB, присвоение идентификатора каждому из полученных фрагментов. В качестве идентификатора используется результат вычисления хеш-функции от содержимого фрагмента.
- Для каждого из фрагментов: выбор из структуры kBuckets *k* узлов, ближайших к идентификатору фрагмента по XOR метрике, для отправки сообщения STORE с парой <идентификатор фрагмента, собственная структура contact>.
- Отправка узлам из структуры Masters Set сообщения NEW_DATA, содержащего набор идентификаторов фрагментов ресурса и метаинформацию.
- Заинтересованный в получении данных участник системы выполняет процедуру поиска по полученным идентификаторам.
- Получив список узлов, на которых хранятся фрагменты ресурса, узел последовательно загружает необходимые фрагменты при помощи сообщения GET_DATA.

- После того, как участником получены все данные, идентификаторы полученных фрагментов размещаются на *k* ближайших узлах путем формирования сообщения `STORE`.

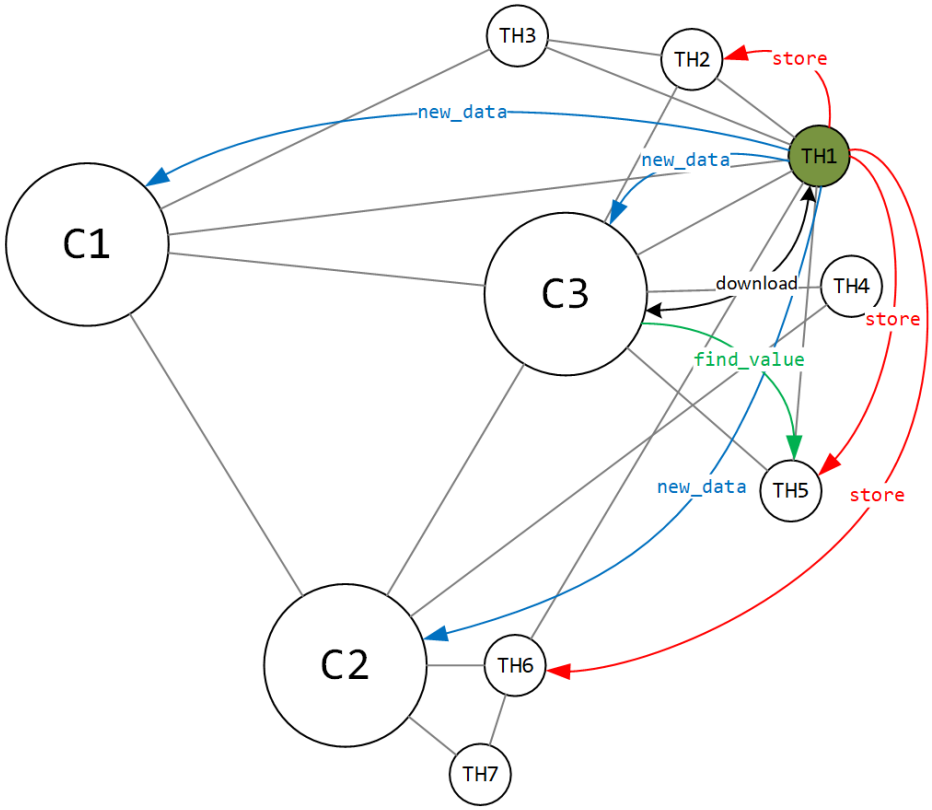

Рис. 3.2. Отправка данных в исследовательский центр. Размещение пар <ключ, значение> в k-buckets ближайших по XOR метрике узлах (красные связи). Нотификация центральных узлов о появлении новых данных (синие связи). Процедура поиска появившихся данных (зеленые связи). Загрузка новых данных (черная связь).

Отправка данных в исследовательский центр при отсутствии прямого соединения между агентом и центром (см. 3.3.):

- Последовательная передача фрагментов данных ближайшему участнику сети, имеющему стабильному связь с центром, путем формирования сообщения `UPLOAD_DATA`. На данном этапе используется внешняя метрика, основанная на IP-адресах.
- Отправка сообщений `STORE` и `NEW_DATA` узлом, получившим фрагмент данных.

- Последующие этапы аналогичны предыдущему сценарию.

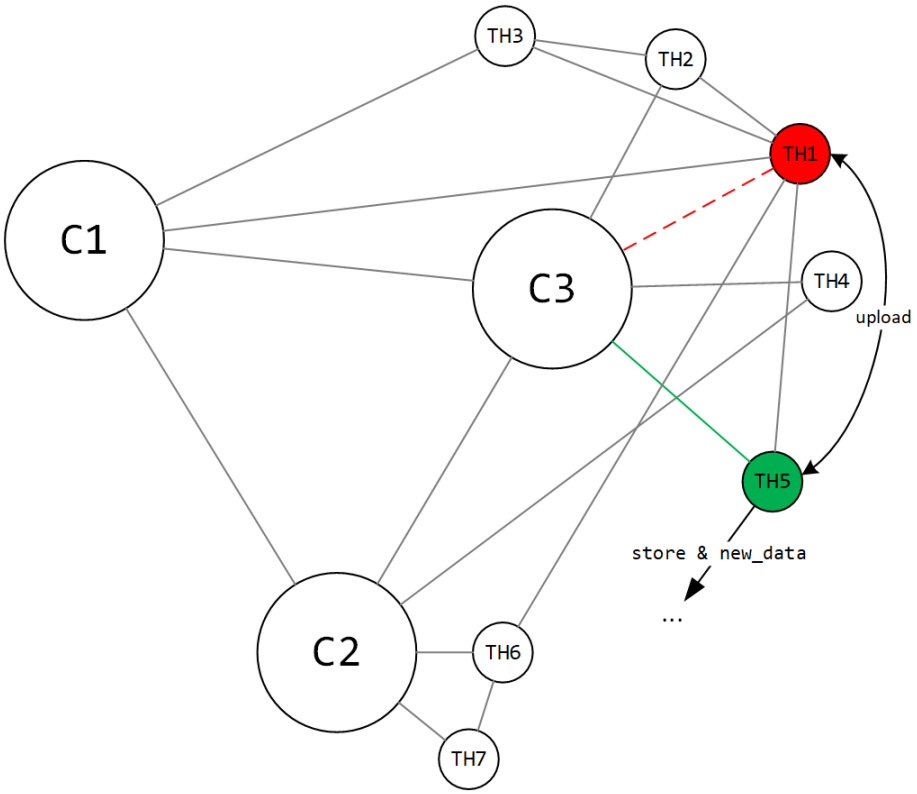

Рис. 3.3. Отправка данных в исследовательский центр С3 через посредника.

Получение ресурса в системе:

- Получение хеш-значения в процессе нотификаций от агентов.
- Выполнение процедуры поиска при помощи рекурсивных вызовов FIND_VALUE.
- В случае успешной процедуры поиска установка соединения с найденным узлом с целью загрузки ресурса.

3.4. Тестирование

В связи с тем, что при разработке системы не были затронуты основные принципы маршрутизации Kademlia, которые были апробированы во множестве высоконагруженных приложений, тестирование включало только функциональную часть – передачи данных через посредника.

Проверка разработанной системы производилась при помощи программного продукта виртуализации VirtualBox. Сконфигурированная

система состояла из 4 участников: 1 центр проведения экспертиз, 1 загрузочный узел и 2 агента. После успешной инициализации участников в сети была искуственно разорвана связь одного из агентов с центром при помощи настроек межсетевого экрана. Далее была осуществлена передача целевых данных через второго агента.

Таким образом, разработанная система удовлетворяет требованиям высокой доступности данных, обеспечивает проверку целостности передаваемых данных, не подвержена DDoS-атакам и имеет возможность поиска с логарифмической вычислительной сложностью от количества узлов в системе. Одной из уязвимостей данной системы является множество загрузочных узлов, как и в любой другой одноранговой системе, поэтому по сравнению с центральными узлами и агентами к ним выдвигаются более строгие требования к времени непрерывной работы. Другим недостатком можно назвать высокую сложность и большое количество параметров системы. Это является следствием запланированной унификации системы и интеграции в качестве бекенда в другие информационные системы.

Заключение

Результаты работы

В данной работе рассмотрены классификации одноранговых сетей. Исследованы главные представители систем типа распределенная хеш-таблица. Представлен прикладной протокол передачи данных, позволяющий с высоким уровнем надежности, скорости и безопасности производить сбор цифровых доказательств и эффективный поиск по ним.

В дальнейшем планируется провести апробацию представленного распределенного программного комплекса в реальных условиях.

Следующим шагом станет расширение области применения разработанной системы. Это позволит увеличить плотность оверлейной сети, в следствие чего увеличится эффективность поиска и доступность ресурсов.

Список литературы

1. Федотов Н.Н. Форензика – компьютерная криминалистика. Москва: «Юридический мир», 2007. 360 с.
2. Kent K., Chevalier S., Grance T., Dang H. Integrating forensic techniques into incident response // Special Publication (NIST SP) - 800-86, 2006
3. Peer-to-peer, https://en.wikipedia.org/wiki/Peer-to-peer
4. Darlagiannis V. Hybrid Peer-to-Peer Systems // Peer-to-Peer Systems and Applications. Springer, 2005. p. 353
5. Schollmeier R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications // First International Conference on Peer-to-Peer Computing, 2001
6. Filali I., Huet F. Dynamic TTL-Based Search in Unstructured Peer-to-Peer Networks // 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010
7. Filali I. A Survey of Structured P2P Systems for RDF Data Storage and Retrieval, Springer, 2011, p. 21
8. Guirat F.B., Filali I. An efficient data replication approach for structured peer-to-peer systems // International Conference Telecommunications (ICT), 2013
9. Ranjan R., Harwood A., Buyya R. Peer-to-peer-based resource discovery in global grids: a tutorial // IEEE Communications Surveys & Tutorials, Volume 10, Issue 2, 2008
10. Ratnasamy S., Francis P., Handley M., Karp R., Shenker S. A Scalable Content-Addressable Network // In Proceedings of ACM SIGCOMM 2001, 2001
11. Stoica I., Morris R., Karger D., Kaashoek M. F., Balakrishnan H. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications // ACM SIGCOMM Computer Communication Review, Volume 31, Issue 4, 2001

12. Rowstron A., Druschel P. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems // IFIP/ACM International Conference on Distributed Systems Platforms, 2001, pp. 329-350
13. Maymounkov P., Mazieres D. Kademlia: A peer-to-peer Information System Based on the XOR Metric, 2002
14. Lee Y., Koo H., Choi S. Advanced node insertion attack with availability falsification in Kademlia-based P2P networks // 14th International Conference on Advanced Communication Technology (ICACT), 2012
15. Lee Y., Kim K., Roh B. H. DDoS Attack by File Request Redirection in Kad P2P Network // International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2012
16. Baumgart I., Mies S. S/Kademlia: A practicable approach towards secure key-based routing // International Conference on Parallel and Distributed Systems, 2007
17. RFC 2104 - HMAC: Keyed-Hashing for Message Authentication, http://www.faqs.org/rfcs/rfc2104.html
18. Knudsen, L.R. Chosen-text attack on CBC-MAC // Electronics Letters, Volume 33, Issue 1, 1997
19. Wulamarisman C. R., Windarta S. Distinguishing attack and second preimage attack on Mini-AES CBC-MAC // International Conference of Advanced Informatics: Concept, Theory and Application (ICAICTA), 2014
20. Bellare M. New Proofs for NMAC and HMAC: Security without Collision-Resistance // Advances in Cryptology – Crypto ’06, Springer-Verlag, 2006
21. Saxena N., Tsudik G., Yi J. H., Admission Control in Peer-to-Peer: Design and Performance Evaluation // Proceedings of 1st ACM workshop on Security of ad hoc and sensor networks, 2003