

# Table of Contents

Introduction	1.1
Билеты	1.2
1. Последовательная и параллельная сложность алгоритмов, информационный граф и ресурс параллелизма алгоритмов.	1.2.1
2. Архитектурные особенности графических процессоров, направленные на массивнопараллельные вычисления. Методы эффективной организации параллельных вычислений на графических процессорах.	1.2.2
3. Основные принципы организации оптических и беспроводных систем передачи данных.	1.2.3
4. Сети хранения данных – архитектура и основные сервисы.	1.2.4
5. Принципы организации и основные достоинства MPLS технологии.	1.2.5
6. Программно-конфигурируемые сети (SDN). Основные принципы, архитектура и преимущества. Протокол OpenFlow. Структура OpenFlow контроллера и коммутатора. Примеры применения.	1.2.6
7. Виртуализация сетевых сервисов (NFV). Основные принципы, этапы развития, архитектура, преимущества. Примеры применения.	1.2.7
8. Качество сервиса в компьютерных сетях: модели распределения ресурсов сети и методы борьбы с перегрузками.	1.2.8
9. Основные подходы математического моделирования КС. Прототипирование КС: преимущества, недостатки, ограничения применимости.	1.2.9
10. Динамическое планирование задач в ИУС РВ. Схемы планирования Rate Monotonic (фиксированные приоритеты) и Earliest Deadline First (динамические приоритеты). Оценка времени отклика задач для схемы Rate Monotonic.	1.2.10
11. Понятие наихудшего времени выполнения программы (WCET). Факторы, влияющие на WCET. Фазы анализа WCET. Использование абстрактной интерпретации для выявления недопустимых путей. Анализ влияния конвейера на время выполнения программы.	1.2.11
12. Архитектура интегрированной модульной авионики (ИМА), её основные преимущества, примеры типов модулей (шина VME). Статико-динамическая схема планирования вычислений в системах ИМА.	1.2.12
13. V-образный жизненный цикл (ЖЦ) программного обеспечения. Основные виды инstrumentальных средств поддержки ЖЦ, их отнесение к фазам ЖЦ. Структура комплекса стендов для поэтапной интеграции ПО и аппаратуры ИУС РВ на восходящей фазе ЖЦ.	1.2.13
14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознавающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии.	1.2.14
15. Ансамбли классификаторов. Основные этапы работы типичного базового классификатора, возможность коррекции на разных этапах. Бэггинг и случайные подпространства. Бустинг. Случайный лес как композиция основных подходов к построению ансамбля.	1.2.15
16. Задача кластеризации как фундаментальная задача интеллектуального анализа данных, сопоставление с операцией группирования и задачей классификации. Различные постановки: разбиение, стохастическая, нечёткая, иерархическая, упорядочивание, однокластерная (последовательная). Примеры методов кластеризации для разных постановок.	1.2.16
17. Дискреционные управление доступом. Модели HRU и Take-Grant. Задача проверки безопасности системы защиты от НСД.	1.2.17
18. Методы аутентификации в сети. Протокол аутентификации Kerberos.	1.2.18

19. Пасивные и активные сетевые атаки (снифинг, спуфинг, МИМ, имперсонация).	1.2.19
20. Коммуникационные протоколы. Ошибки, возникающие при передаче сообщений. Задача надежного обмена сообщениями. Симметричные протокол скользящего (раздвижного) окна: устройство протокола и обоснование его корректности. Протокол альтернирующего бита.	1.2.20
21. Задача маршрутизации. Алгоритм Флойда-Уоршалла построения кратчайших путей в графе. Алгоритм маршрутизации Туэга: описание алгоритма, обоснование его корректности оценка сложности по числу обменов сообщениями.	1.2.21
22. Общие принципы дедуктивной верификации программ. Операционная семантика императивных программ. Формальная постановка задачи верификации программ. Логика Хоара: правила вывода и свойства. Автоматизация проверки правильности программ.	1.2.22
23. Темпоральная логика деревьев вычислений CTL. Синтаксис и семантика CTL. Примеры спецификаций моделей в терминах формул CTL. Темпоральная логика линейного времени PLTL. Синтаксис и семантика PLTL. Свойства живости и безопасности. Ограничения справедливости. Задача верификации моделей (model-checking).	1.2.23
24. Временные автоматы как формальные модели распределенных систем реального времени. Вычисления временных автоматов. Примеры использования временных автоматов для моделирования встроенных систем. Зеноновские вычисления. Синтаксис и семантика Timed CTL. Задача верификации моделей программ реального времени. Программноинструментальное средство верификации моделей программ реального времени UPPAAL.	1.2.24
25. Дискретные цепи Маркова. Метод вложенных цепей Маркова при исследовании систем массового обслуживания.	1.2.25
26. Процессы гибели и рождения. Исследование марковских систем обслуживания с помощью теории процессов гибели и рождения.	1.2.26
27. Понятие антагонистической игры. Верхнее и нижнее значения конечных и бесконечных антагонистических игр. Седловая точка. Необходимые и достаточные условия существования седловой точки. Теорема Фон Неймана о существовании седловой точки у вогнуто-выпуклых функций	1.2.27
28. Понятие потока в сети. Задача о максимальном потоке. Алгоритмы Форда-Фалкерсона и Карзанова. Теорема о максимальном потоке и минимальном разрезе. Сведение задачи составления допустимого расписания с прерываниями для многопроцессорной системы при заданных директивных интервалах к задаче о максимальном потоке в сети.	1.2.28
29. Псевдополиномиальные алгоритмы решения задач: разбиение, рюкзак, расписание для многопроцессорной системы (число процессоров фиксировано).	1.2.29
30. Метод ветвей и границ на примере минимаксной задачи теории расписаний Приближенные алгоритмы решения NP-трудных задач: упаковка в контейнеры, рюкзак, коммивояжер, расписание для многопроцессорной системы, вершинное покрытие. Оценки их сложности и погрешности.	1.2.30

## As i want so that

Все лекции в одном [pdf](#)

## **Билеты для государственного экзамена 2020**

Для подготовки к гос. экзамену.

# 1. Последовательная и параллельная сложность алгоритмов, информационный граф и ресурс параллелизма алгоритмов.

**Последовательная сложность** (serial complexity) алгоритма — число операций, которые нужно выполнить при его последовательном исполнении.

**Параллельная сложность** (parallel complexity) алгоритма — число шагов, за которое можно выполнить данный алгоритм в предположении доступности неограниченного числа необходимых процессоров (функциональных устройств, вычислительных узлов, ядер и т.п.). Параллельная сложность алгоритма понимается как высота канонической [ярусно-параллельной формы](#).

**Ярусно-параллельная форма (ЯПФ)** (parallel form) — это представление графа алгоритма, в котором:

- все вершины разбиты на перенумерованные подмножества ярусов;
- начальная вершина каждой дуги расположена на ярусе с номером меньшим, чем номер яруса конечной вершины;
- между вершинами, расположенными на одном ярусе, не может быть дуг.

*Высота ЯПФ* — это число ярусов. *Ширина яруса* — число вершин, расположенных на ярусе. *Ширина ЯПФ* — это максимальная ширина ярусов в ЯПФ.

**Канонической ярусно-параллельной формой** называется ЯПФ, высота которой на единицу больше длины критического пути (самого длинного пути), а все входные вершины расположены на первом ярусе. Для заданного графа его каноническая ЯПФ единственна.

**Граф алгоритма** (algorithm graph, информационный график) — это ориентированный ациклический мультиграф, вершины которого соответствуют операциям алгоритма, а дуги — передаче данных между ними. Вершины графа алгоритма могут соединяться несколькими дугами, в частности когда в качестве разных аргументов одной и той же операции используется одна и та же величина. Граф алгоритма почти всегда является параметризованным графом. В частности, его вид часто зависит от входных данных.

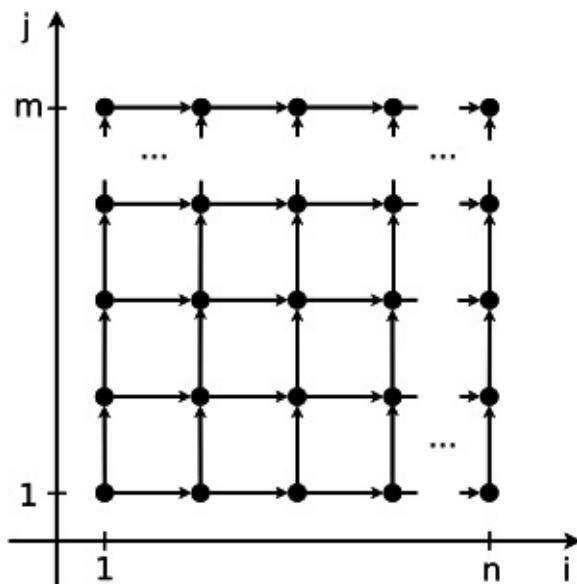


Рисунок 1: Информационная структура одного из вариантов алгоритма решения систем линейных алгебраических уравнений с блочно-двухдиагональной матрицей

Граф алгоритма используется как удобное представление алгоритма при исследовании его структуры, ресурса параллелизма, а также других свойств. Его можно рассматривать как параметризованную информационную историю. Он сохраняет её информативность, при этом обладая компактностью за счёт параметризации. Разработана методика построения графа алгоритма по исходному тексту программ.

*Граф алгоритма:*

- почти всегда является параметризованным графом. В частности, его вид часто зависит от входных данных.
- используется как удобное представление алгоритма при исследовании его структуры, ресурса параллелизма, а также других свойств.
- показывает как устроена параллельная структура алгоритма. Много информации несут разного рода проекции информационного графа, выделяя его регулярные составляющие и одновременно скрывая несущественные детали.
- потенциально бесконечный граф, число вершин и дуг которого определяется значениями внешних переменных.
- потенциально многомерный объект. Наиболее естественная система координат для размещения вершин и дуг информационного графа опирается на структуру вложенности циклов в реализации алгоритма.

Информационная независимость определяет ресурс параллелизма программы. Ресурс параллелизма — это то, насколько хорошо данный алгоритм может быть распараллелен. Таким образом, ресурс бывает хорошим и плохим.

Для описания **ресурса параллелизма** алгоритма (ресурса параллелизма информационного графа) необходимо указать ключевые параллельные ветви в терминах **конечного** и **массового** параллелизма. Далеко не всегда ресурс параллелизма выражается просто, например, через **координатный параллелизм** или, что то же самое, через независимость итераций некоторых циклов (да-да-да, циклы — это понятие, возникающее лишь на этапе реализации, но здесь все так связано...). В данном случае, координатный параллелизм означает, что информационно независимые вершины лежат на гиперплоскостях, перпендикулярных одной из координатных осей). С этой точки зрения, не менее важен и ресурс **скошенного параллелизма**. В отличие от координатного параллелизма, скошенный параллелизм намного сложнее использовать на практике, но знать о нем необходимо, поскольку иногда других вариантов и не остается: нужно оценить потенциал алгоритма, и лишь после этого, взвесив все альтернативы, принимать решение о конкретной параллельной реализации. Хорошей иллюстрацией может служить алгоритм, структура которого показана на рис.1: координатного параллелизма нет, но есть параллелизм скошенный, использование которого снижает сложность алгоритма с  $n \times m$  в последовательном случае до  $(n + m - 1)$  в параллельном варианте.



**Конечный параллелизм (finite parallelism)** — параллелизм, определяемый информационной независимостью некоторых фрагментов в тексте программы.

**Массовый параллелизм (mass parallelism)** — параллелизм, определяемый информационной независимостью итераций циклов программы.

**Координатный параллелизм (coordinate parallelism)** — частный случай скошенного параллелизма, определяемый циклами ParDO, при котором информационно независимые вершины лежат на гипер-плоскостях, перпендикулярных одной из координатных осей.

**Скошенный параллелизм (skewed parallelism)** — параллелизм в пространстве итераций, определяемый поверхностями уровня разверток. Ряд исследователей этим термином пользуется только для случая, когда параллелизм не является координатным.

Все можно найти на алговики — <https://goo.gl/0AtOj7>

Про граф алгоритма (инфо график) — [https://ru.wikipedia.org/wiki/Граф\\_алгоритма](https://ru.wikipedia.org/wiki/Граф_алгоритма)

## Возможные допы

### Закон Амдала

Напрямую связан с темой билета

Пусть необходимо решить некоторую вычислительную задачу. Предположим, что её алгоритм таков, что доля  $\alpha$  от общего объёма вычислений может быть получена только последовательными расчётами, а, соответственно, доля  $1 - \alpha$  может быть распараллелена идеально (то есть время вычисления будет обратно пропорционально числу задействованных узлов  $p$ ). Тогда ускорение, которое может быть получено на вычислительной системе из  $p$  процессоров, по сравнению с однопроцессорным решением не будет превышать величины:

$$S_p = \frac{1}{\alpha + \frac{1 - \alpha}{p}}$$

### Закон Мура

Раз уж заговорили про законы

**Закон Мура** — эмпирическое наблюдение, изначально сделанное Гордоном Муром, согласно которому (в современной формулировке) количество транзисторов, размещаемых на кристалле интегральной схемы, удваивается каждые 24 месяца.

В последнее время, чтобы получить возможность задействовать на практике ту дополнительную вычислительную мощность, которую предсказывает закон Мура, стало необходимо задействовать параллельные вычисления. На протяжении многих лет производители процессоров постоянно увеличивали тактовую частоту и параллелизм на уровне инструкций, так что на новых процессорах старые однопоточные приложения исполнялись быстрее без каких-либо изменений в программном коде. Примерно с середины десятилетия 2000-х годов по разным причинам производители процессоров предпочитают многоядерные архитектуры, и для получения всей выгоды от возросшей производительности ЦП программы должны переписываться в соответствующей манере. Однако не каждый алгоритм поддается распараллеливанию, определяя, таким образом, фундаментальный предел эффективности решения вычислительной задачи согласно закону Амдала.

## 2. Архитектурные особенности графических процессоров, направленные на массивно параллельные вычисления. Методы эффективной организации параллельных вычислений на графических процессорах.

[https://www.iis.nsk.su/files/articles/sbor\\_kas\\_16\\_poletaev.pdf](https://www.iis.nsk.su/files/articles/sbor_kas_16_poletaev.pdf)

[https://vk.com/@physics\\_math-graficheskii-processor-ili-gpu](https://vk.com/@physics_math-graficheskii-processor-ili-gpu)

[http://hpc-education.unn.ru/files/schools/hpc-2014/gpu/Lecture1\\_IntroToGPGPU.pdf](http://hpc-education.unn.ru/files/schools/hpc-2014/gpu/Lecture1_IntroToGPGPU.pdf)

[http://elar.urfu.ru/bitstream/10995/40616/1/978-5-7996-1722-6\\_2016.pdf](http://elar.urfu.ru/bitstream/10995/40616/1/978-5-7996-1722-6_2016.pdf)

[http://www2.rsuu.ru/binary/2632507\\_99.1424349071.26912.pdf](http://www2.rsuu.ru/binary/2632507_99.1424349071.26912.pdf)

### Архитектурные особенности

GPU (Graphics Processing Unit) предназначен для вычислений:

- параллельных по данным: одна и та же операция выполняется над многими данными параллельно,
- в которых отношение вычислительных операций к числу операций по доступу к памяти велико.



Высокая вычислительная мощность GPU объясняется особенностями архитектуры. Современные CPU содержат несколько ядер, тогда как графический процессор изначально создавался как многопоточная структура с множеством ядер. Если архитектура CPU предполагает последовательную обработку информации, то GPU исторически предназначался для обработки компьютерной графики, поэтому рассчитан на массивно параллельные вычисления.

CPU лучше работает с последовательными задачами. При большом объёме обрабатываемой информации очевидное преимущество имеет GPU. Условие только одно — в задаче должен наблюдаться параллелизм.

Графический процессор может выполнить лишь часть операций, которые может выполнить центральный процессор, но он делает это с невероятной скоростью. GPU будет использовать сотни ядер, но для достижения высоких скоростей GPU должен выполнять однообразные операции.

Выбор концепции SIMD для графических процессоров обусловлен тем, что она обеспечивает параллельное использование большого количества «вычислителей» без явного управления ими: распределения задач, синхронизации вычислений и коммуникации между параллельными расчетами.

Тем не менее, центральные процессоры более гибкие, чем графические. Центральные процессоры имеют больший набор инструкций, поэтому они могут выполнять более широкий диапазон функций. Также CPU работают на более высоких максимальных тактовых частотах и имеют возможность управлять вводом и выводом компонентов компьютера в отличии от GPU.

Есть множество различий и в поддержке многопоточности. CPU исполняет 1-2 потока вычислений на одно процессорное ядро, а видеочипы могут поддерживать до 1024 потоков на каждый мультипроцессор, которых в чипе несколько штук. И если переключение с одного потока на другой для CPU стоит сотни тактов, то GPU переключает несколько потоков за один такт.

Различен и принцип работы с памятью у GPU и CPU. Так, все современные GPU имеют несколько контроллеров памяти, да и сама графическая память более быстрая, поэтому графические процессоры имеют гораздо большую пропускную способность памяти, по сравнению с универсальными процессорами, что также весьма важно для параллельных расчетов, оперирующих огромными потоками данных.



В универсальных процессорах большую часть площади кристалла занимают различные буферы команд и данных, блоки декодирования, блоки аппаратного предсказания ветвления, блоки переупорядочения команд и кэшпамять первого, второго и третьего уровней. Все эти аппаратные блоки нужны для ускорения исполнения немногочисленных потоков команд за счет их распараллеливания на уровне ядра процессора. Сами же исполнительные блоки занимают в универсальном процессоре относительно немного места.

В графическом процессоре, наоборот, основную площадь занимают именно многочисленные исполнительные блоки, что позволяет ему одновременно обрабатывать несколько тысяч потоков команд.

## Методы эффективной организации параллельных вычислений на графических процессорах

GPGPU – технология использования графического процессора для выполнения расчетов в приложениях общих вычислений. Это стало возможным благодаря добавлению программируемых шейдерных блоков и более высокой арифметической точности растровых контейнеров, что позволяет использовать потоковые процессоры для неграфических вычислений.

На сегодняшний день технология GPGPU реализована несколькими производителями:

- Khronos Group: OpenCL — язык программирования задач общего назначения, связанных с вычислениями на различных графических и центральных процессорах.
- Компания nVidia: CUDA — технология GPGPU, позволяющая реализовывать на языке Си (или других языках программирования) вычислительные алгоритмы, выполняемые только на устройствах компании nVidia.

Особо значимое ускорение в GPGPU можно получить, если одни и те же инструкции применяются к огромным массивам данных.

Следующим требованием можно назвать отсутствие взаимодействий между обрабатываемыми потоками или «слабое» взаимодействие.

Потоковая обработка данных особенно эффективна для алгоритмов, обладающих следующими свойствами, характерными для задач физического и математического моделирования:

- большая плотность вычислений — велико число арифметических операций, приходящихся на одну операцию ввода-вывода (например, обращение к памяти). Во многих современных приложениях обработки сигналов она достигает 50:1, причем со сложностью алгоритмов увеличивается;
- отсутствие в алгоритмах множественного ветвления;
- локальность данных по времени — каждый элемент загружается и обрабатывается за время, малое по отношению к общему времени обработки, после чего он больше не нужен. В результате в памяти потокового процессора для каждого «вычислителя» можно хранить только данные, необходимые для обработки одного элемента, в отличие от центральных процессоров с моделью произвольно зависимых данных.
- одна и та же последовательность вычислений, применяемая к разным данным
- могут быть разбиты на подзадачи одинаковой сложности (подзадача будет решаться блоком нитей)
- каждая подзадача может быть выполнена независимо от всех остальных, нет потребности в глобальной синхронизации

Возможные случаи параллелизма:

- Параллельные копирование и выполнение кода на GPU
- Параллельное выполнение кода на GPU

Если отсутствует возможность параллельного копирования памяти в обе стороны, то неправильный порядок отправки команд может привести к простою.

MPI + OpenMP + CUDA — хорошо гармонируют, так как MPI — ускорение экстенсивным путём, между процессорами, OpenMP — простое ускорение экстенсивным путём внутри процессора, а CUDA — ускорение интенсивным путём однотиповых вычислений.

Один CPU поток может управлять несколькими GPU, для этого нужно лишь переключать контексты.

### 3. Основные принципы организации оптических и беспроводных систем передачи данных.

#### Оптические системы

**Волоконно-оптическая связь** — способ передачи информации, использующий в качестве носителя информационного сигнала электромагнитное излучение оптического (ближнего инфракрасного) диапазона, а в качестве направляющих систем — волоконно-оптические кабели. Благодаря высокой несущей частоте и широким возможностям мультиплексирования пропускная способность волоконно-оптических линий многократно превышает пропускную способность всех других систем связи и может измеряться Терабитами в секунду. Малое затухание света в оптическом волокне позволяет применять волоконно-оптическую связь на значительных расстояниях без использования усилителей. Волоконно-оптическая связь свободна от электромагнитных помех и труднодоступна для несанкционированного использования: незаметно перехватить сигнал, передаваемый по оптическому кабелю, технически крайне сложно.

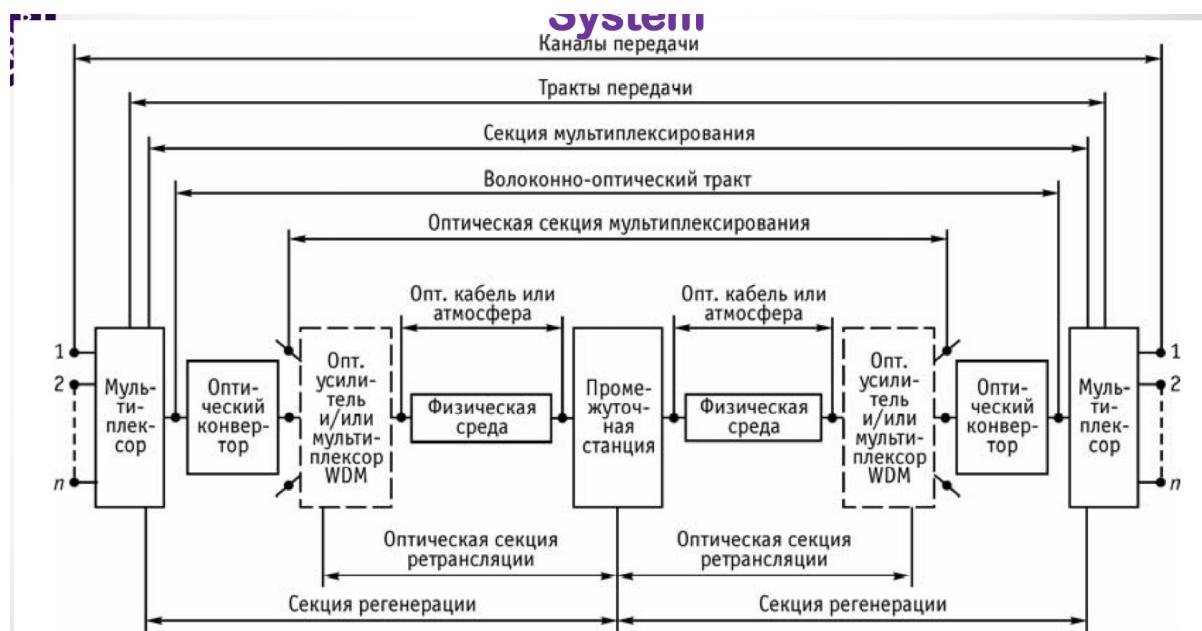


Рисунок 1: Общая схема построения оптических систем.

**OTN** — основная технология построения магистральных волоконно-оптических сетей связи на сегодняшний день, сменившая SDH/SONET (Synchronous Optical NETwork). Аббревиатура OTN расшифровывается как Optical Transport Network («оптическая транспортная сеть»). OTN — это оптическая транспортная сеть, которая обеспечивает передачу и мультиплексирование цифровых данных по волновым каналам DWDM (Dense Wavelength Division Multiplexing, плотное мультиплексирование с разделением по длине волн), обеспечивает передачу по волокну оптических сигналов на разных длинах волн). Технология OTN стандартизована сектором телекоммуникаций Международного союза электросвязи (ITU-T) в декабре 2009 года, см. «ITU-T Recommendation G.709 «Interfaces for the Optical Transport Network (OTN)»».

**Принцип действия** технологии OTN заключается в том, что сигналы различных форматов упаковываются в стандартные контейнеры, которые затем передаются по волоконно-оптической сети. Таким образом, обеспечивается возможность передачи по транспортной сети любых необходимых типов клиентских сигналов (STM, ATM, IP, Fibre Channel, InfiniBand и др.), а также эффективное использование пропускной способности за счет плотной упаковки разнородного трафика.

В заголовки контейнеров может добавляться служебная информация, которая позволяет контролировать прохождение трафика по сети и обнаруживать ошибки работы, а также избыточное кодирование, которое позволяет восстанавливать повреждённый трафик без необходимости его повторной передачи. Технология коррекции ошибок FEC, применяемая в

сетях OTN, позволяет успешно восстанавливать переданный сигнал даже после существенных искажений и затуханий, что даёт возможность строить оптоволоконные магистрали OTN протяжённостью сотни и тысячи километров.

### Структура контейнера OTN

Контейнер OTN строится путём добавления к исходным клиентским данным нескольких заголовков, каждый из которых выполняет свою функцию.

- Во-первых, клиентский трафик разбивается на части нужного размера, после чего к каждой из них добавляется заголовок, описывающий тип трафика. Получившийся блок информации называется **OPU** — Optical Payload Unit, «оптический блок нагрузки». В данном блоке описывается используется ли мультиплексирование и его параметры. Также данный может передавать сигнал для синхронизации. Блок OPU передаётся в неизменном виде из конца в конец сети — т.е. от точки приёма клиентских данных до точки выдачи этих данных клиенту (между конечными (де)мультплексаторами оптического сигнала).
- Во-вторых, к блоку OPU добавляется служебная информация, необходимая для мониторинга прохождения сигнала по сети и управления процессом передачи сигнала. Получившийся блок информации называется **ODU** — Optical Data Unit, «оптический блок данных». Блок ODU также передаётся в неизменном виде из конца в конец сети — т.е. от точки приёма клиентских данных до точки выдачи этих данных клиенту.
- В-третьих, к блоку ODU добавляется избыточное кодирование (**FEC**, Forward Error Correction) и дополнительная служебная информация — для мониторинга, контроля и восстановления трафика на отдельном сегменте сети между двумя транспондерами. Получившийся блок информации называется **OTU** — Optical Transport Unit, «оптический транспортный блок». Блок OTU передаётся в неизменном виде в пределах участка сети, ограниченного транспондерами (т.е. пунктами, где сигнал преобразуется в электронный вид для [3R-регенерации](#)). Таким образом, протокол OTU работает между соседними узлами сети OTN, которые поддерживают функции электрической регенерации оптического сигнала, называемые также функциями 3R(retiming, reshaping and regeneration).

Таким образом, по сети OTN передаются контейнеры OTU, каждый из которых представляет собой «матрёшку», где под несколькими слоями служебных данных скрывается исходный клиентский сигнал. Можно сказать, что клиентский сигнал «завёрнут» в несколько слоёв служебных данных — поэтому технологию OTN называют также «digital wrapper technology», или «optical channel wrapper» (англ. wrapper — обёртка).

G.709 определяет 6 уровней OTN потока

- OPU
- ODU
- OTU
- OCh: Optical Channel. Представляет собой end-to-end оптический путь. Передается на несущей определенного "цвета".
- OMS: Optical Multiplex Section. Секция мультиплексирования. Работает между OADMs (Optical Add Drop Multiplexer). Используется для мониторинга оптического соединения и обнаружения проблем в OTN.
- OTS: Optical Transport Section. Секция передачи. Работает в пределах участка сети, ограниченного двумя сетевыми элементами. Выполняет задачи контроля и обслуживания сети между сетевыми элементами (мультиплексоры, демультиплексоры, оптические коммутаторы и тд.).

Первые три - electrical domain, вторые три - optical domain.

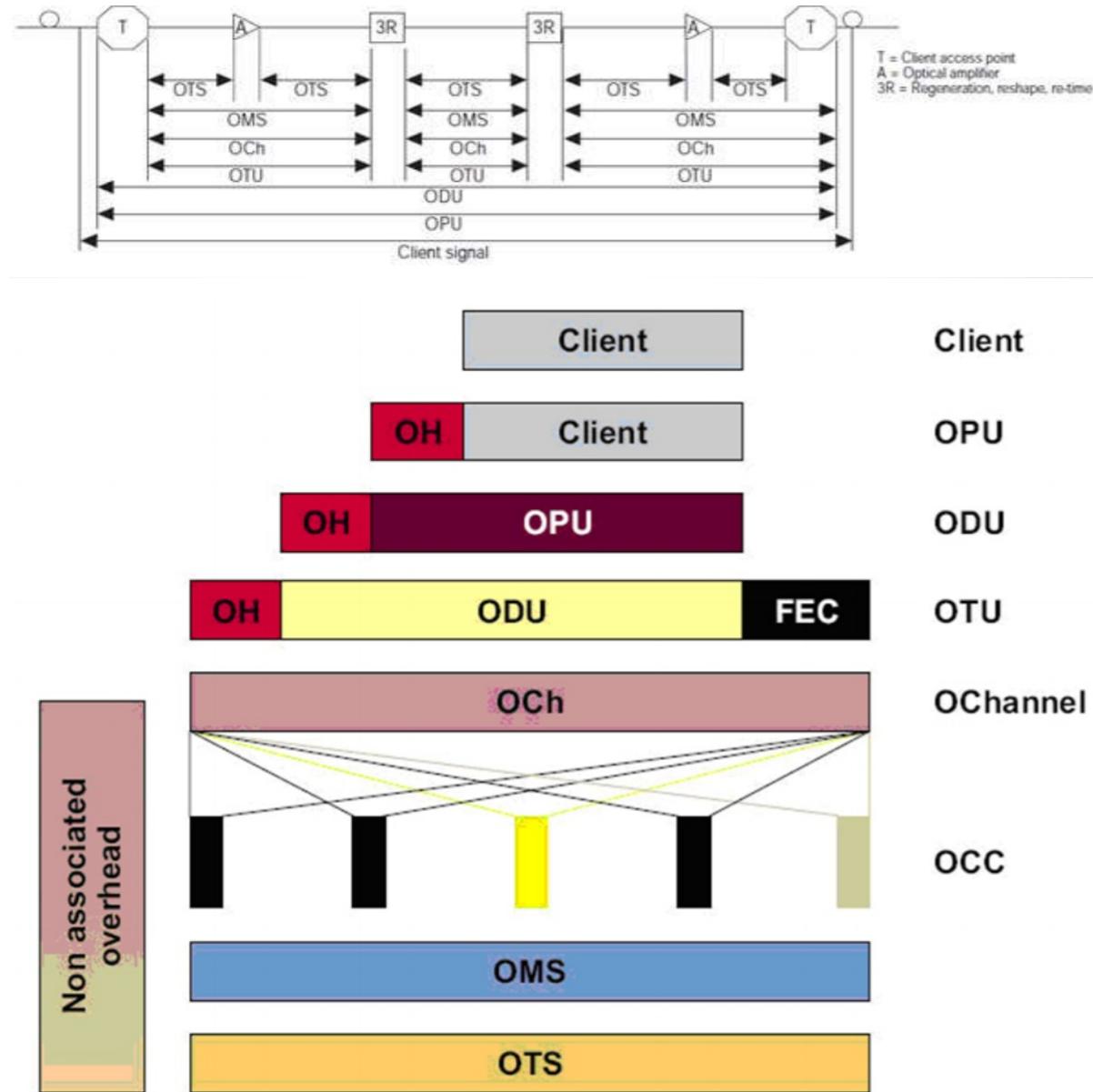


Рис 3 Формирование структуры транспортного потока OTN

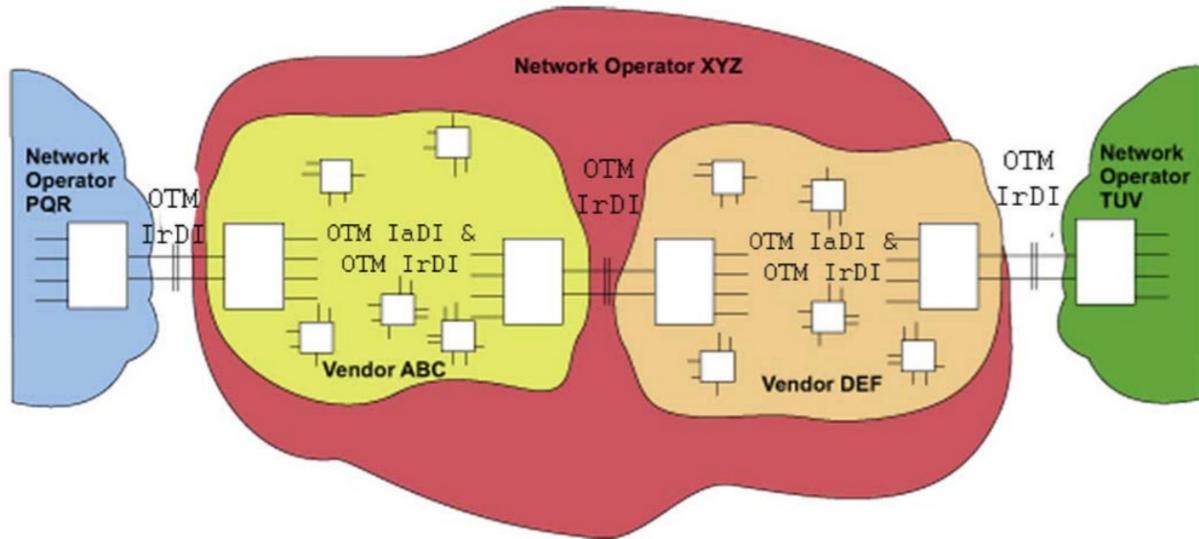
#### FEC

В процедуре прямой коррекции ошибок (FEC) используются коды Рида-Соломона RS(255, 239). В этом самокорректирующемся коде данные кодируются блоками по 255 байт, из которых 239 байт являются пользовательскими, а 16 байт представляют собой корректирующий код. Коды Рида-Соломона позволяют исправлять до 8 ошибочных байтов в блоке из 255 байт. Применение кода Рида-Соломона позволяет улучшить отношение мощности сигнала к мощности шума на 5дБ при уменьшении уровня битовых ошибок с  $10^{-3}$  (без применения FEC) до  $10^{-2}$  (после применения FEC). Этот эффект дает возможность увеличить расстояние между генераторами сети на 20 км или использовать менее мощные передатчики сигнала.

#### Интерфейсы

В рекомендации ITU-T G.872 описаны два типа интерфейсов для OTN: IrDI (Inter-Domain Interface) и IaDI (Intra-Domain Interface). Внешний интерфейс IrDI встречается при присоединении сети одного оператора к сети другого оператора, на внутренних стыках различных подсетей одного оператора и на стыках однотипного оборудования внутри одной подсети.

Внутренний интерфейс IaDI встречается только на стыках однотипного оборудования одной подсети.



**Рис 1. Интерфейсы сети OTN**

[Подробно про структуру заголовков OTN можно прочитать тут](#)

## Беспроводные системы

**Беспроводная вычислительная сеть** — вычислительная сеть, основанная на беспроводном (без использования кабельной проводки) принципе, полностью соответствующая стандартам для обычных проводных сетей (например, Ethernet). В качестве носителя информации в таких сетях могут выступать радиоволны СВЧ-диапазона.

**ALOHAnet** — первая компьютерная сеть передачи данных с пакетной коммутацией, использовавшая в качестве среды доступа к ней беспроводную технологию. Была разработана и введена в эксплуатацию в 1968-1970-х годах группой учёных Гавайского университета под руководством Нормана Абрамсона в рамках исследовательского проекта THE ALOHA SYSTEM, основной целью которого было изучение возможностей использования радиопередачи как альтернативы проводным коммуникациям. В чистой ALOHA вещание, как и в радио, может начаться в любой момент, все устройства слушают и передают на одной частоте, следовательно могут возникать коллизии.

Поэтому придумали **Carrier Sense Multiple Access With Collision Avoidance** (CSMA/CA, множественный доступ с контролем несущей и избеганием коллизий) — это сетевой протокол, в котором выделяются фазы:

1. **Carrier Sense**: перед передачей станция слушает канал, определяя свободен ли он (могут быть скрытые станции)
2. **Collision Avoidance**: если мы услышали, что кто-то передает — мы спим время (часто случайное), затем слушаем снова, если свободно — передаем.
  - RTS/CTS — могутoptionально использоваться для уменьшения негативного эффекта скрытых станций. Явный запрос на передачу (RTS). Передача не начнется пока станция не получит разрешение (CTS) от получателя.
  - Передача данных: так как отправителю сложно обнаруживать коллизии во время передачи, часто используется схема с явным подтверждением передачи кадра (ACK). Если ACK не получен, то производятся повторные попытки (возможно с экспоненциальным backoff-ом).

Беспроводные сети делятся на два типа: локальные (WLAN) и мобильные сотовые сети.

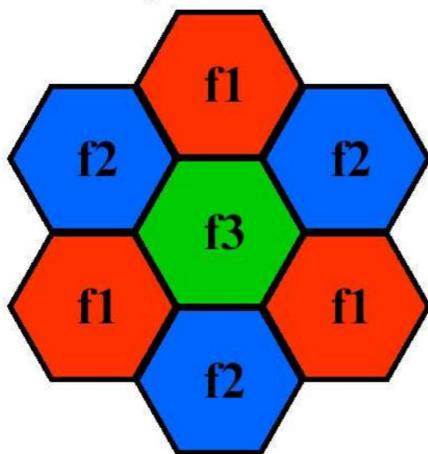
Примеры WLAN: WiFi, WiMAX (сейчас практически нигде не используется)

Сотовые сети: AMPS, GSM, CDMA2000, LTE, etc

**Технологии, использующиеся в беспроводных сетях для увеличения скорости:**

- MIMO
- OFDM (orthogonal frequency division multiplexing)
- OFDMA
- Увеличение ширины канала
- Новые СКК
  - QAM 16, QAM 64, QAM 256, QAM 1024
  - QAM 4096 (думают делать или нет)
  - Номер соответствует количеству значений одного символа (на одной несущей)
    - 64 => 6 бит (можно закодировать)
    - 1024 => 10 бит
  - Упирается в формулу Шеннона
  - Каждый раз нужен в 2 раза меньший SNR
- Агрегирование
- Блочное подтверждение

**Сотовая связь** — один из видов мобильной радиосвязи, в основе которого лежит сотовая сеть. Зона покрытия сети делится на ячейки (соты), определяющиеся зонами покрытия отдельных базовых станций. Основной идеей, на которой базируется принцип сотовой связи, является повторное использование частот в несмежных сотах. Каждая из ячеек обслуживается своим передатчиком с невысокой выходной мощностью и ограниченным числом каналов связи. Это позволяет без помех использовать повторно частоты каналов этого передатчика в другой, удаленной на значительное расстояние, ячейке. Это хорошо иллюстрируется раскраской сот, использующих один диапазон частот, в одинаковый цвет и сот, использующих разные диапазоны, в разные цвета. Таким образом сеть делится минимум на 3 кластера.



#### Основные принципы сотовых сетей

- Использование лицензируемого спектра радиочастот (в России выделяется ГКРЧ): для сетей 4G диапазон 900 МГц – 3500 МГц
- Централизованное управление: все передачи пользовательских устройств управляются сетью (базовыми станциями)

**Wi-Fi** — технология беспроводной локальной сети с устройствами на основе стандартов IEEE 802.11. Работает в диапазоне 2.4 ГГц и 5 ГГц. В 802.11ay - 60 ГГц (миллиметровый WiFi). Может работать по двум схемам: с точкой доступа и в режиме ad-hoc. Основной протокол для доступа к среде передачи данных — DCF. DCF использует метод CSMA/CA вместе с алгоритмом двоичной экспоненциальной отсрочки.

#### DCF

Пока канал занят, станция ничего не передает. После освобождения канала станция ждет дополнительного интервала времени *DIFS*. В сетях с большим числом станций передача нескольких станций сразу по окончании интервала *DIFS* может привести к коллизиям, поэтому каждая станция генерирует счетчиком отсрочки (англ. *backoff*). *backoff* — это случайное число в интервале  $(0, CW_{min}]$ , где  $CW_{min}$  — это конкурентное окно (англ. *contention window*). Станция слушает

канал время  $\sigma$ , определённое в стандарте как время пустого слота, и, если канал был свободен, уменьшает счётчик отсрочки на единицу. Если канал был занят, то станция замораживает свой счётчик отсрочки и ждёт пока канал освободится, далее ждёт интервал DIFS и размораживает счётчик отсрочки. Когда счётчик отсрочки достигает нуля, станция передаёт кадр данных.

DCF включает в себя необходимость подтверждения успешного приёма кадра данных, таким образом если станция после отправки своего кадра не получила кадр подтверждения (англ. acknowledgment, ACK), она считает передачу неуспешной.

Если передача была неуспешной (из-за коллизии станций или помех), станция вновь генерирует backoff из интервала  $(0, 2Cw_{min}]$ . Конкурентное окно увеличивается вдвое каждый раз после неудачной попытки передачи кадра данных пока не достигнет значения  $Cw_{max}$ . Если конкурентное окно достигло максимума, станция его не меняет до тех пор пока не будет достигнут предел количества попыток передач (англ. retry limit) кадра данных.

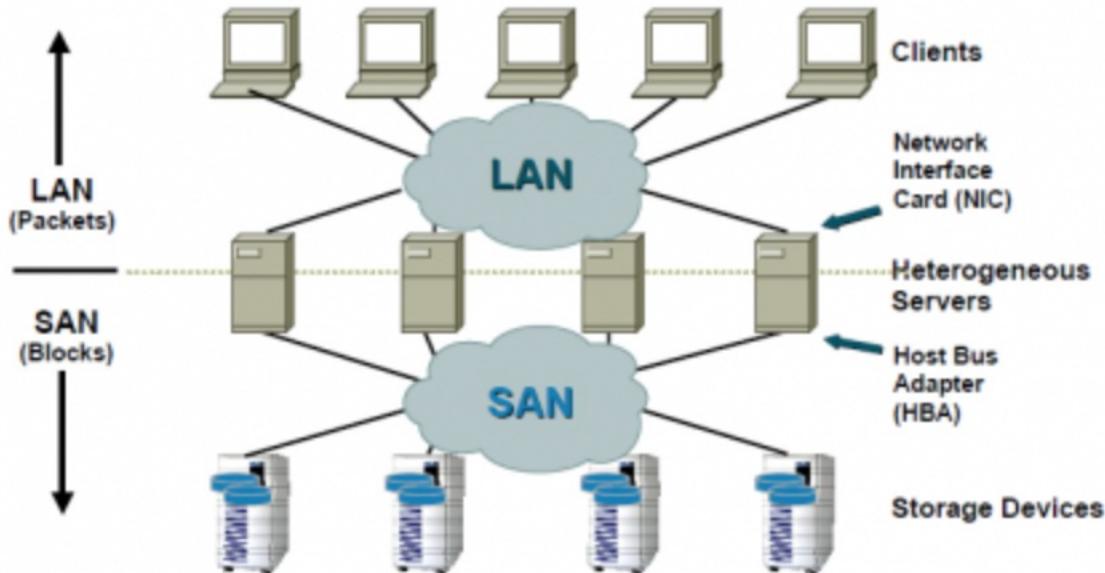
По достижении предельного числа попыток передач кадра данных станция сбрасывает кадр данных и начинает пытаться передавать следующий кадр данных из очереди FIFO. Если это был первый сброшенный кадр данных, станция сбрасывает конкурентное окно до значения  $(0, Cw_{min}]$  и вновь его экспоненциально наращивает. Если же станция сбросила два кадра данных подряд, то все последующие кадры данных передаются на максимальном конкурентном окне до тех пор, пока хотя бы один кадр данных не будет передан успешно. Если кадр данных был передан успешно, то для следующего кадра данных используется минимальное конкурентное окно  $(0, Cw_{min}]$ .

Дополнительно станции могут использовать механизм RTS/CTS, который заключается в предварительной отправке кадров Request-to-Send (англ. Запрос на передачу) передающей станцией и Clear-to-Send (англ. Разрешение передачи) принимающей станцией. Кадр RTS короткий, и попадание в коллизию двух кадров RTS менее болезненно, чем попадание в коллизию двух длинных кадров данных. Если кадр данных слишком короткий, использование RTS/CTS может быть неэффективно — в таком случае используется RTS порог (англ. RTS Threshold), который определяет максимальную длину кадра данных, который будет передан без использования механизма RTS/CTS. В кадрах RTS/CTS дополнительно устанавливается TXOP (англ. transmission opportunity, рус. возможность передачи) — интервал виртуальной занятости канала, в течение которого другие станции должны воздерживаться от начала своей передачи. RTS/CTS позволяет частично (но не полностью) решить проблемы скрытых и засвеченных станций.

## 4. Сети хранения данных — архитектура и основные сервисы.

**Сеть хранения данных** (СХД, англ. Storage Area Network, SAN) — представляет собой архитектурное решение для подключения внешних устройств хранения данных, таких как дисковые массивы, ленточные библиотеки, оптические приводы к серверам таким образом, чтобы операционная система распознала подключенные ресурсы как локальные.

### SAN: What Is It?



#### SAN Is a Dedicated Network for Attaching Servers to Storage Devices

Большинство сетей хранения данных использует протокол SCSI для связи между серверами и устройствами хранения данных на уровне шинной топологии. Так как протокол SCSI не предназначен для формирования сетевых пакетов, в сетях хранения данных используются низкоуровневые **протоколы**:

- Fibre Channel Protocol (FCP), транспорт SCSI через Fibre Channel. Наиболее часто используемый на данный момент протокол. Существует в вариантах 1 Gbit/s, 2 Gbit/s, 4 Gbit/s, 8 Gbit/s, 10 Gbit/s, 16 Gbit/s, 20 Gbit/s.
- iSCSI, транспорт SCSI через TCP/IP.
- iSER, транспорт iSCSI через InfiniBand / RDMA.
- SRP, транспорт SCSI через InfiniBand / RDMA
- FCoE, транспортировка FCP/SCSI поверх «чистого» Ethernet.
- FCIP и iFCP, инкапсуляция и передача FCP/SCSI в пакетах IP.
- HyperSCSI, транспорт SCSI через Ethernet.
- FICON транспорт через Fibre Channel (используется только мейнфреймами).
- ATA over Ethernet, транспорт ATA через Ethernet.

Технологически SAN состоит из следующих компонентов:

- Узлы, ноды (nodes)
  - Дисковые массивы (системы хранения данных) — хранилища (тргеты [targets])
  - Серверы — потребители дисковых ресурсов (инициаторы [initiators]).
- Сетевая инфраструктура

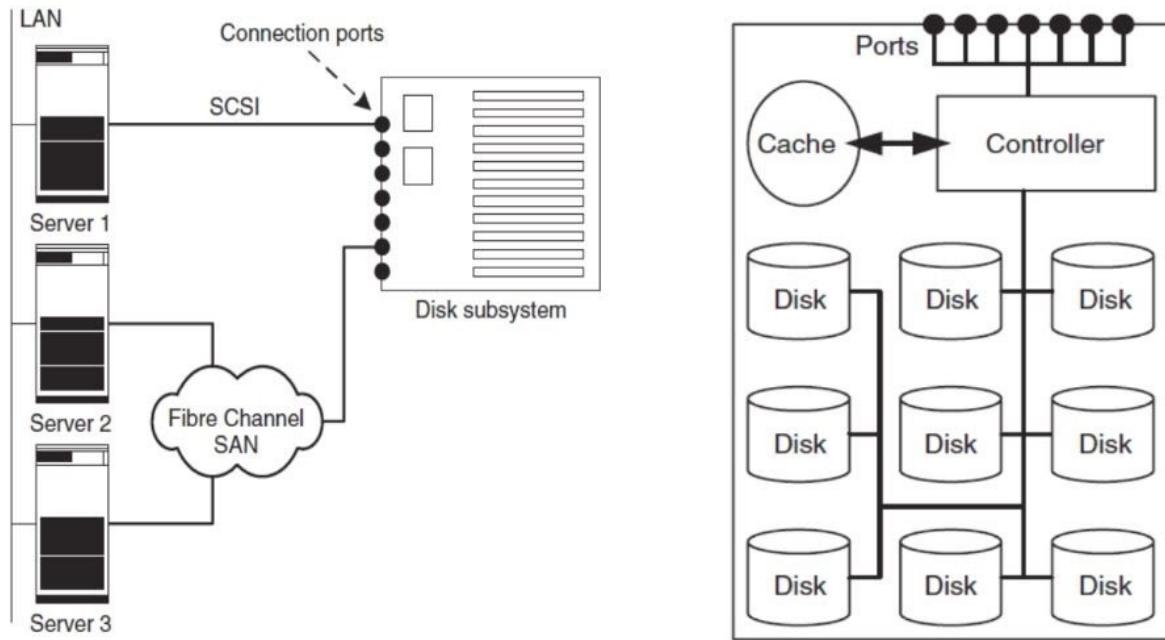
- Коммутаторы (и маршрутизаторы в сложных и распределённых системах)
- Кабели

Для SAN ключевыми параметрами являются не только производительность, но и надёжность. Ведь если у сервера БД пропадёт сеть на пару секунд (или даже минут) — ну неприятно будет, но пережить можно. А если на это же время отвалится жёсткий диск с базой или с ОС, эффект будет куда более серьёзным. Поэтому все компоненты SAN обычно дублируются — порты в устройствах хранения и серверах, коммутаторы, линки между коммутаторами и, ключевая особенность SAN, по сравнению с LAN — дублирование на уровне всей инфраструктуры сетевых устройств — фабрики. **Фабрика** (fabric — что вообще-то в переводе с английского ткань, т.к. термин символизирует переплетённую схему подключения сетевых и конечных устройств, но термин уже устоялся) — совокупность коммутаторов, соединённых между собой межкоммутаторными линками (ISL — InterSwitch Link). Высоконадёжные SAN обязательно включают две (а иногда и более) фабрики, поскольку фабрика сама по себе — единая точка отказа. Фабрики могут иметь идентичную (зеркальную) топологию или различаться. Например одна фабрика может состоять из четырёх коммутаторов, а другая — из одного, и к ней могут быть подключены только высококритичные узлы.

**Различают следующие виды топологий фабрики:**

- **Каскад** — коммутаторы соединяются последовательно. Если их больше двух, то недёжно и непроизводительно.
- **Кольцо** — замкнутый каскад. Надёжнее просто каскада, хотя при большом количестве участников (больше 4) производительность будет страдать. А единичный сбой ISL или одного из коммутаторов превращает схему в каскад со всеми вытекающими.
- **Сетка (mesh).** Бывает **Full Mesh** — когда каждый коммутатор соединяется с каждым. Характерно высокой надежностью, производительностью и ценой. Количество портов, требуемое под межкоммутаторные связи, с добавлением каждого нового коммутатора в схему растет экспоненциально. При определённой конфигурации просто не останется портов под узлы — все будут заняты под ISL. **Partial Mesh** — любое хаотическое объединение коммутаторов.
- **Центр/периферия (Core/Edge)** — близкая к классической топологии LAN, но без уровня распределения. Нередко хранилища подключаются к Core-коммутаторам, а серверы — к Edge. Хотя для хранилищ может быть выделен дополнительный слой (tier) Edge-коммутаторов. Также и хранилища и серверы могут быть подключены в один коммутатор для повышения производительности и снижения времени отклика (это называется локализацией). Такая топология характеризуется хорошей масштабируемостью и управляемостью.

**Архитектура дисковой подсистемы (ДПС)** Все порты ДПС подключаются к контроллеру. Контроллер управляет дисками и кешем. Часть дисков используется для резервирования, поэтому объем памяти, предоставляемый пользователям обычно меньше реального объема. В ДПС может использоваться дублирование каналов связи, для повышения надежности. При выходе из строя одной линии связи, можно переключиться на вторую для доступа к диску. При нормальной работе каналы, по которым происходит доступ к дискам, могут быть фиксированы (запросы к одному диску всегда идут по одному каналу) или выбираться динамически (решение, по какому каналу отправлять запрос принимает контроллер в каждом отдельном случае). См. рисунок ниже.



**Для резервирования данных используются:**

- Горячий дисковый резерв — все диски дублируются, используется активное резервирование
- RAID массивы — избыточный массив независимых дисков.

**Для обеспечения устойчивости работоспособности ДПС используются разные техники:**

- Данные распределяются по нескольким дискам с помощью механизмов RAID и снабжаются избыточными данными (блоки четности).
- На каждом физическом диске данные закодированы кодом Хемминга. Кроме этого диск оснащен подсистемой самодиагностики, которая контролирует частоту ошибок, вибрацию шпинделя и т.д. Это позволяет проактивно прогнозировать отказы диска.
- Каждый диск подсоединен к контроллеру хотя бы через две внутренние шины.
- Контроллер дисковой подсистемы может быть продублирован. Выход одного экземпляра, автоматически будет активизирован следующий экземпляр. Схема Active-Standby.
- Используют периодическое мгновенное копирование для защиты от логических ошибок. Например, создание мгновенной копии данных через каждый час. Тогда в случае сбоя и уничтожения какой-то таблицы, она может быть восстановлена.
- Удаленное зеркалирование (копирование всех данных в удаленную сеть хранения данных) используют от физического уничтожения или повреждения оборудования (катастрофоустойчивость). В сочетании с мгновенным копированием эти сервисы гарантируют сохранение и консистентность данных даже для нескольких виртуальных дисков или дисковых подсистем.
- LUN (Logical Unit Number) маскирование защищает от несанкционированного доступа, упрощает работу системного администратора, защищает от случайных сбоев в работе приложений серверов и их оборудования.

#### LUN (Logical Unit Number) маскирование

LUN не означает отдельный жесткий диск, скорее он определяет виртуальный раздел в RAID-массиве. При этом один и тот же виртуальный раздел массива может иметь разные значения LUN для разных хостов, которым этот LUN назначен. Также возможно наличие на одном хосте одинаковых LUN, принадлежащих разным системам хранения (разным SCSI Target ID).

Таким образом, полный адрес диска (физического раздела жесткого диска) на SCSI-устройстве складывается из SCSI Target ID (уникального для хоста и определяемого драйвером) и LUN, уникального в пределах SCSI-устройства и назначаемого ему в настройках или автоматически по порядку.

**Преимущества** Совместное использование систем хранения, как правило, упрощает администрирование и добавляет изрядную гибкость, поскольку кабели и дисковые массивы не нужно физически транспортировать и перекоммутировать от одного сервера к другому.

Другим преимуществом является возможность загружать сервера прямо из сети хранения. При такой конфигурации можно быстро и легко заменить сбойный сервер, переконфигурировав SAN таким образом, что сервер-замена будет загружаться с LUN'a (LUN, Logical Unit Number - адрес дискового устройства в SAN) сбояного сервера. Эта процедура может занять, например, полчаса. Идея относительно новая, но уже используется в новейших dataцентрах.

Дополнительным преимуществом является возможность на хосте собрать RAID-зеркало из LUNов, которые представлены хосту с двух разных дисковых массивов. В таком случае полный отказ одного из массивов не навредит хосту.

Также сети хранения помогают более эффективно восстанавливать работоспособность после сбоя. В SAN может входить удаленный участок со вторичным устройством хранения. В таком случае можно использовать репликацию — реализованную на уровне контроллеров массивов, либо при помощи специальных аппаратных устройств.

## 5. Принципы организации и основные достоинства MPLS технологии.

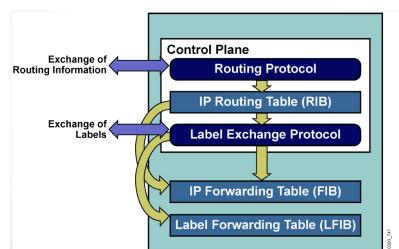
Проблема: протоколам маршрутизации для передачи пакета на следующий хоп нужно смотреть заголовок пакета и локальную таблицу маршрутизации, причем это происходит на каждом хопе. Таким образом на каждом маршрутизаторе по сути нужно иметь полную таблицу маршрутизации и делать лукап по адресу назначения на каждом хопе.

Решение: MPLS — уменьшает оверхед на непосредственно маршрутизацию, может маршрутизировать не только IP.

BTW

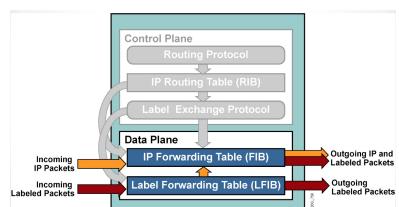
На самом деле сейчас стала широко доступна аппаратная IP маршрутизация, поэтому лучше сказать, что MPLS решает проблему универсальности доставки — совершенно неважно, что находится под меткой — IP, Ethernet, ATM, Frame Relay. В основном сейчас MPLS применяется для VPN и TE. Еще один плюс применения чистого MPLS — возможность сделать так, чтобы не нужно было настраивать BGP на каждом роутере для IBGP, достаточно настроить по меткам.

### MPLS Control Plane



Заполняется новая таблица с метками, метками обмениваются как и IP адресами, заполняется соответствующие таблицы.

### MPLS Data Plane



Маршрутизация может проводится на основе обоих таблиц.

### MPLS устройства

**LSR** — маршрутизатор в домене MPLS. Передает и принимает только пакеты с метками. Может менять метки при передаче пакета.

**Edge LSR** — стоит на границе MPLS домена. Производит установку первой метки при входе в MPLS домен и снятие последней при выходе из домена.

LSR, которые не Edge будем называть **core LSR**.

Сами Edge по понятному принципу делятся на **Ingress Edge LSR** и **Egress Edge LSR** (вход и выход из MPLS домена соответственно)

Таким образом исходный лукап в таблицу маршрутизации происходит только на Edge LSR. Остальные LSR делают лукап по меткам только в Label Forwarding Table.

## MPLS метки

- Определяют FEC (forwarding equivalence class) — класс эквивалентности маршрутизации — по сути это IP + маска на egress.
- Имеют локальную значимость. Каждый LSR сам выбирает метку и FEC и передает эту информацию другим LSR, инициирует этот процесс edge LSR.

По сути FEC — это группа пакетов, которые передаются по одному пути. MPLS маршрутизация состоит из назначения пакета на FEC, определения следующего хода для FEC.

MPLS метка имеет размер 32 бита из которых:

- Первые 20 — сама метка
- 3 бита — TC (Traffic class) — очередной QoS
- Индикатор S — 1 бит — указывает последняя ли это метка перед IP пакетом (нужно для парсинга)
- 8 бит TTL — копируется из IP пакета

Сама метка находится между 2 и 3 уровнями TCP/IP. То есть IP пакет сначала заворачивается в MPLS, а затем уже в L2 фрейм.

В лекции выделяют:

- Frame-mode MPLS. Именно то, что описано выше, кладем информацию между 2 и 3 уровнями.
- Cell-mode MPLS is MPLS over ATM. Есть такой L2 протокол ATM. Там есть куда куда впихнуть метки в самом протоколе, дополнительные сущности таким образом не нужны.

Для задач маршрутизации обычно достаточно одной метки, но с помощью MPLS можно организовывать всякие сложные вещи, увеличивая количество меток:

- MPLS VPNs (2 метки): Одна метка для маршрутизации, другая для указания VPN (туннеля).
- MPLS TE (2 и больше меток): Тут подразумевается, например, балансировка нагрузки, одна из меток позволяет указать класс (туннель) TE.
- Все вместе: много меток

Последняя метка всегда используется для маршрутизации.

Парсить все это просто. Для L2 пакетов есть EtherType, указывающий, что пакет начинается с метки 0x8847 (на самом деле тут еще указано, что это юникаст), дальше снимаем по 32 бита и смотрим на индикатор S.

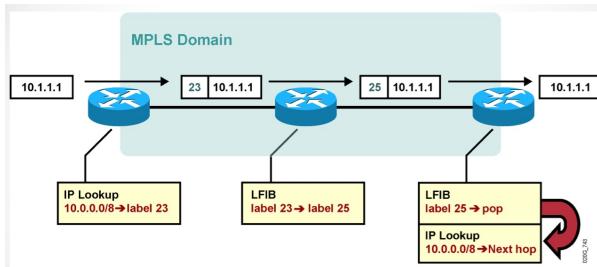
## Операции над MPLS метками

Ранее говорилось, что LSR может менять метки при передаче пакета. Что это, зачем это.

LSR может сделать 3 вещи:

- Вставить одну или несколько меток (Ingress Edge LSR)
- Поменять одну или несколько меток (core LSR)
- Удалить метку (pop) (Egress Edge LSR)

Пример:



## Практическое применение MPLS

Применение:

- Unicast IP routing
- Multicast IP routing
- MPLS TE
- QoS
- MPLS VPNs (на этом кстати фокусировался по хорошему наш курс)
- AToM (раньше про это не было, это возможность завернуть любой L2 в MPLS и другой L2)

Более подробно будем рассматривать только Unicast IP routing (база) и MPLS VPNs (слишком много про это было в курсе).

Для начала нужно разобраться как обнаруживать соседей и передавать метки, мы же все таки маршрутизацией занимаемся.

## Присваивание и распространение меток

### Обнаружение соседей

Метки распространяются благодаря алгоритму LDP (Label Distribution Protocol).

Установление сессии:

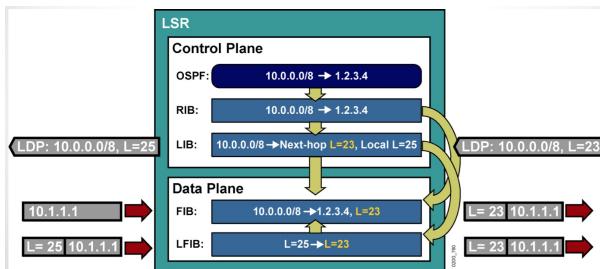
1. Рассылка Hello сообщений (UDP multicast на адреса в той же подсети с TTL 1) на интерфейсы, с MPLS enabled
2. Ответ на Hello сообщение — попытка установить сессию (на самом деле инициатором установки соединения будет тот LSR у которого IP больше)

Сама сессия устанавливается по TCP.

После этого происходит обмен инициализационными сообщениями и начинают посыпаться keepalive сообщения.

Несмежные соседи могут соединяться схожим образом, только вместо multicast — unicast, это называется tLDP, иногда это нужно.

### Распространение меток



Подробного описания нет, но все достаточно очевидно — посылаем соседям те метки и IP адреса, которые у нас есть — при получении заполняем LIB, FIB и LFIB. Отметим, что без отработавшего OSPF это все работать не будет.

Но откуда вообще взять метки? Для этого нужен инициатор, про это ниже.

## Label-Switched Path и построение пути

LSP — последовательность LSR, передающих помеченные пакеты определенного FEC.

Таким образом:

- Строятся LSP исходя из IP маршрутизации (например OSPF)
- LDP распространяет метки на отдельные сегменты LSP.

Все происходит примерно так:

- Egress роутер обнаруживает изменение в таблице маршрутизации
- Выбирает какой нибудь FEC (ip+маска из dst таблицы маршрутизации) и выбирает для нее метку (любую) (так происходит для всех dest)
- Эту информацию он через LDP передает своим соседям
- Соседи генерируют свою метку, заполняют LIB, FIB и LFIB и передают своим соседям
- В итоге построен LSP путь
- Если найдено несколько путей используется лучший по некоторой метрике (как и в других алгоритмах маршрутизации)
- Гифка из СДСМ: <https://gblobscdn.gitbook.com/assets%2F-LIgRTPaaN7wUujKIWEz%2F-LXhL-JzPMdG-IAZwDzA%2F-LXhLLisFxolb93jgcCx%2Fb1371acab49e47c9bab46aa0af23123b.gif>
- RFC: <https://tools.ietf.org/html/rfc5036>

## PHP

**Penultimate Hop Popping** — оптимизация от бога, удаляем все метки на предпоследнем маршрутизаторе, чтобы не делать лишних лукапов в последнем.

Гифка: <https://gblobscdn.gitbook.com/assets%2F-LIgRTPaaN7wUujKIWEz%2F-LfxxzdGb7gGgdgvPhmC%2F-Lfxzrl0AYwgGNeu29Rs%2F2c74499985d7463da7e67beac679038f-1.gif>

## Про сходимость

Все сходится неплохо, но медленнее чем IGP, поэтому предлагается использовать MPLS TE (видимо резервные пути), чтобы избежать проблем при падении линка.

## VPN

С помощью MPLS можно сформировать провайдерский VPN.

Основные понятия для VPN:

**CE — Customer Edge router** — граничный маршрутизатор клиента, который подключен в сеть провайдера.

**PE — Provider Edge router** — граничный маршрутизатор провайдера. Собственно к нему и подключаются CE. На PE зарождается VPN, на нём они и кончаются. Именно на нём расположены интерфейсы, привязанные к VPN. Именно PE назначает и снимает сервисные метки. Именно PE являются Ingress LSR и Egress LSR. PE должны знать таблицы маршрутизации каждого VPN, ведь это они принимают решение о том, куда посыпать пакет, как в пределах провайдерской сети, так и в плане клиентских интерфейсов.

**P — Provider router** — транзитный маршрутизатор, который не является точкой подключения — пакеты VPN проходят через него без каких-либо дополнительных обработок, иными словами просто коммутируются по транспортной метке. P нет нужды знать таблицы маршрутизации VPN или сервисные метки. На P нет интерфейсов привязанных к VPN.

Основные понятия для MPLS:

Центральное понятие здесь **VRF** — Virtual Routing and Forwarding instance — нужна как гарантия чтобы ограничивать раутинг, если разные клиенты используют одни и те же частные сети (например 10.0.0.0/24). VRF — он строго локален для маршрутизатора — за его пределами VRF не существует. Соответственно VRF на одном маршрутизаторе никак не связан с VRF на другом.

MPLS VPN VRF создаются только на тех маршрутизаторах к которым подключены клиенты. Далее принадлежность к конкретному VPN определяется MPLS меткой. Остается вопрос, как найти выходной маршрутизатор и вообще проводить по этой метке раутинг.



Все верно, используем две метки — одну для раутинга на внутренних LSR, вторую для обозначения принадлежности к VPN, она будет использоваться только на Ingress и Egress. Заметим, что в качестве FEC в данной случае будет использоваться адрес последнего LSR в MPLS сети.

Конечно нужно как то выбрать одну метку для одного VPN на Ingress и Egress — для этого используется протокол MP-BGP. Его суть не сильно отличается от LDP.

Итого все вместе:

Маршрутизатор PE навешивает на клиентский трафик две метки — внутреннюю сервисную, которая не меняется до самого конца путешествия и по ней последний PE понимает, какому VRF принадлежит пакет, и внешнюю транспортную, по которой пакет передаётся через сеть провайдера — эта метка меняется на каждом P-маршрутизаторе и снимается на последнем PE или предпоследнем P (вспоминаем PHP). Благодаря наличию сервисной метки и VRF трафик различных VPN изолирован друг от друга как в пределах маршрутизаторов, так и в каналах.

## PS

Если хочется разобраться можно почитать слайды и СДСМ.

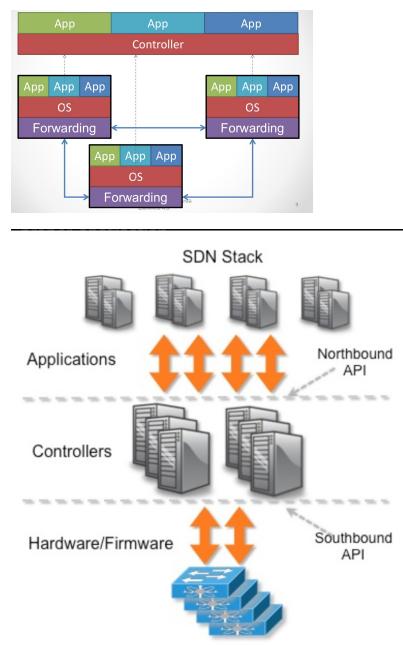
## 6. Программно-конфигурируемые сети (SDN). Основные принципы, архитектура и преимущества. Протокол OpenFlow. Структура OpenFlow контроллера и коммутатора. Примеры применения.

### Основные принципы

Программно-Конфигурируемые Сети (Software Defined Networking/SDN) — это разделение плоскости передачи и управления данными, позволяющее осуществлять программное управление плоскостью передачи, которое может быть физически или логически отделено от аппаратных коммутаторов и маршрутизаторов.

1. Физически разделить уровень управления сетевым оборудованием от уровня управления передачей данных.
2. Перейти от управления отдельными экземплярами сетевого оборудования к управлению сетью в целом — логически централизованное управление.
3. Создать открытый программно-управляемый интерфейс между сетевыми приложениями и транспортной сетью.

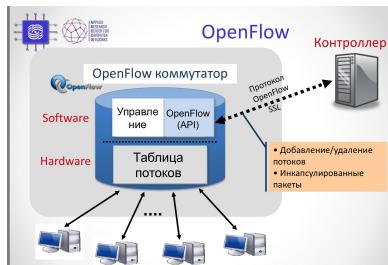
### Архитектура



### Преимущества

- Удешевление оборудования
- Облегчение управления сетью
- Программируемость, открытость, инновации

### OpenFlow



Поддерживается три типа сообщений:

- Сообщения контроллер-коммутатор
  - Конфигурирование коммутатора
  - Управление и контроль состояния
  - Управление таблицами потоков
  - Features, Configuration, Modify-State (flow-mod), Read-State (multipart request), Packet-out, Barrier, Role-Request
- Симметричные сообщения
  - Отправка в обоих направлениях
  - Обнаружение проблем соединения контроллера с коммутатором — Hello,Echo
- Асимметричные сообщения
  - Отправка от коммутатора к контроллеру
  - Объявляют об изменении состояния сети, состояния коммутаторов
  - Packet-in, flow-removed, port-status, error

## Версии OpenFlow (это можно отнести к структуре коммутатора)

В 1.0 была только одна таблица потоков, что приводило к большому росту числа правил. В 1.1 таблиц стало несколько.

Также появились групповые таблицы позволяющие задать ряд действий и выбирать:

- Использование всех
- Одного случайного
- Одного определенного
- Первого живого

Также появилась Meter Table для учета количества трафика.

В версии 1.2 появились сценарии отказоустойчивости для контроллеров — Master / Slave.

Для быстрого матчинга в коммутаторах используется TCAM (Ternary Content Addressable Memory). Это когда у бита целых три значения — 0, 1 и ?, выдается адрес первой подходящей строки. Эта штука работает за константное время.

## Структура OpenFlow контроллера

- Программа, TCP/IPсервер, ожидающий подключения коммутаторов
- Отвечает за обеспечение взаимодействие приложения-коммутатор.
- Предоставляет важные сервисы(например, построение топологии, мониторинг хостов)
- API сетевой ОС или контроллер предоставляет возможность создавать приложения на основе централизованной модели программирования.



## Схема работы OpenFlow

- Реактивный режим
- Проактивный режим

Топология строится с помощью LLDP. Суть в том, что посыпается Packet-Out на один из свитчей (на определенный порт), потом анализируется откуда пришел Packet-In и таким образом строится линк.

## Варианты применения SDN/OpenFlow

Области применения:

- Компании
- Телеком операторы и сервис провайдеры
- ЦОД и облака

Один из примеров применения:

Интеллектуальный Traffic Engineering:

- Выбор оптимального пути
- Реакция на отказ канала
- Резервирование пропускной способности

И так можно придумать еще кучу всего.

## 7. Виртуализация сетевых сервисов (NFV). Основные принципы, этапы развития, архитектура, преимущества. Примеры применения.

Виртуализация сетевых функций (NFV) — это способ виртуализации сетевых служб, таких как маршрутизаторы, брандмауэры и балансировщики нагрузки, которые традиционно выполнялись на проприетарном оборудовании. Эти сервисы упаковываются в виртуальные машины, которые выполняются на обычном оборудовании, что позволяет поставщикам услуг запускать свою сеть на стандартных серверах вместо проприетарных. Таким образом оператор сети перестает зависеть от одного поставщика сетевого оборудования.

С NFV оператору сети не нужно иметь выделенное оборудование для каждой сетевой функции. NFV повышает масштабируемость и гибкость, позволяя поставщикам услуг предоставлять новые сетевые сервисы и приложения по требованию, не требуя дополнительных аппаратных ресурсов.

### Основные принципы

Виртуализация сетевых функций (NFV, Network Function Virtualization) — это технология, которая виртуализирует функциональность физических сетевых устройств, позволяя перенести ее на архитектуру ЦОД (вычислительные сервера, сервера хранения + коммутаторы). NFV программно реализует функции сетевых устройств, которые в дальнейшем запускаются на виртуальной инфраструктуре облака. NFV отделяет реализацию сетевых функций от непосредственного сетевого оборудования.

В NFV это делается при помощи виртуальных функций сети VNF (Virtualized network function). Виртуальная сетевая функция (Virtualized NF, VNF) — программная реализация сетевой функции, которая может быть установлена в виртуальной инфраструктуре. Примеры сетевых функций: DHCP, Firewall, NAT, VPN.

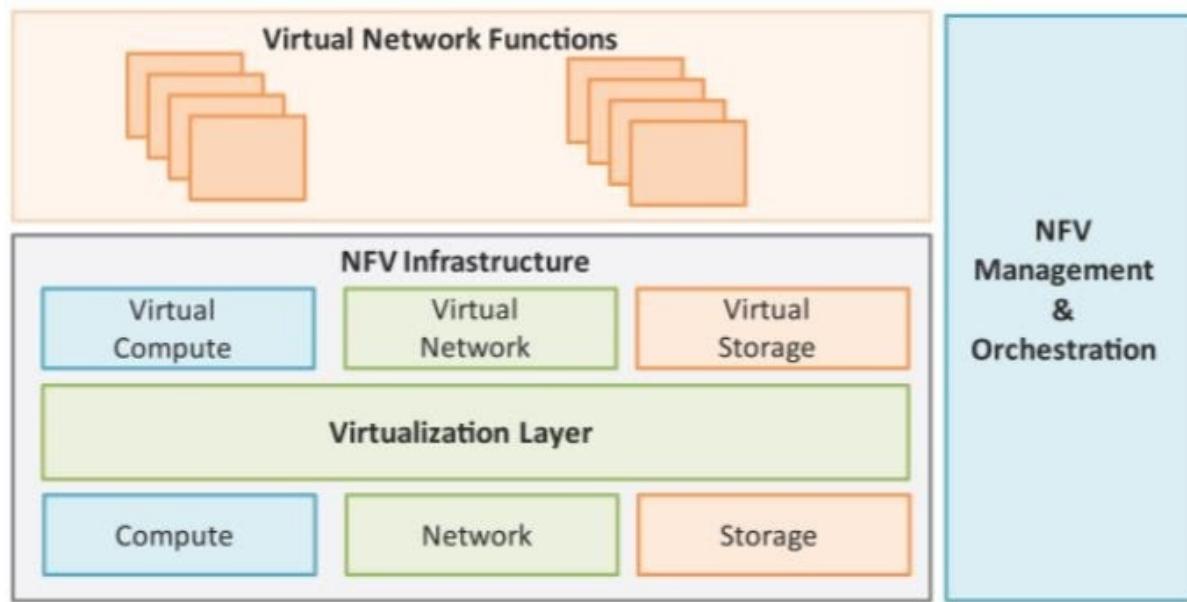
В архитектуре NFV становится возможным осуществлять надежное предоставление сервисов за счет распределенного резервирования ресурсов такой архитектуры, а также сбор данных для индикаторов производительности KPI (Key Performance Indicator) от аппаратных и программных компонентов NFV. Отношения к ПКС

- NFV дружественная к SDN технология, так как NFV позволяет создавать инфраструктуру, к которой применима ПКС архитектура.
- NFV и ПКС взаимовыгодны друг другу, но не зависят друг от друга.
- Сетевые функции могут быть виртуализованы без ПКС, также как и ПКС может быть запущен без NFV.

### Этапы развития NFV

1. Software implementation of network — программная имплементация сетевых функций: переход от быстрых аппаратных реализаций к программным реализациям
2. Network function modules — модули сетевых функций: разделение на функциональные модули, например DHCP, NAT, Rate limiting
3. Implementation in virtual machines — решения для виртуальных машин: переход к использованию виртуальных машин, отсюда появляются все преимущества виртуализации: быстрая доставка новых приложений, масштабируемость, мобильность и другие. Есть гипервизор и много виртуалок.
4. Standard API's between modules — стандартные API для взаимодействия модулей

### Архитектура



Ключевыми элементами NFV архитектуры являются:

- Виртуальная Сетевая Функция (VNF): VNF программная реализация сетевой функции, которая может запуститься на NFV инфраструктуре (NFVI).
- NFV Инфраструктура (NFVI): NFVI включает вычислительные, сетевые ресурсы, а также ресурсы хранения, которые могут быть виртуализованы.
- Управление и Оркестрация (MANO) NFV: NFV MANO фокусируется на всех задачах связанных с виртуализацией, которые возникают на всех стадиях жизненного цикла VNF.

**Преимущества** NFV — перенос сетевых функций на виртуальные машины:

- Автоматизация установки и настройки как софта, так и железа
- Поддержка нескольких поставщиков инфраструктуры
- Уменьшение стоимости за счет использования стандартных серверов
- Объединение сервисов в группы
- Целостность облачных приложений
- Восстановление после сбоев

**Варианты применения** Облака:

- NFVI as a service подобно IaaS (infrastructure as a service)
- VNF as a service
- Цепочки сервисов (VNF forwarding graphs)
- Платформа виртуальной сети as a service

Мобильные сети:

- Виртуализация ядра мобильной сети и IMS (не знаю что это)
- Виртуализация базовой мобильной станции

ЦОД:

- Виртуализация CDN

Доступ:

- Виртуализация домашней сети
- NFV фиксированного доступа

## Примеры

### BRAS

- Терминация пользовательских сессий
- Интересует выгода на одного пользователя ~ 1Mbps
- Стоимость существующего решений примерно 10к за 10Gbps => Одно подключение = 1
- С NFV: один сервер может обрабатывать 50Gbps. Стоимость 5к => Одно подключение = 0.1.

### CG-NAT

- Трансляция адресов
- Высокая стоимость существующих решений.
- Экономия: 16 -> 4 -> 2 на подключение

## 8. Качество сервиса в компьютерных сетях: модели распределения ресурсов сети и методы борьбы с перегрузками.

### Качество сервиса

- Нет четкого определения
- (из готовых билетов) Качество обслуживания ( Quality of Service, QoS ) определяет количественные оценки вероятности того, что сеть будет передавать определенный поток данных между двумя узлами в соответствии с потребностями приложения или пользователя.

### Метрики качества сервиса

- Уровень потерь (packet loss) — Доля пакетов, которые были отправлены, но не были доставлены получателю [проценты]
- Пропускная способность (bandwidth) — Количество данных, которые могут быть переданы в единицу времени [байты]
- Задержка (delay) — Время передачи единицы данных по направлению от отправителя к получателю [секунды]
- Вариация задержки, дрожание (jitter) — Разница между минимальной и максимальной задержками [секунды]

### Что такое управление качеством сервиса?

- Методы позволяющие обслуживать разные потоки данных с разным качеством
- Методы распределения ресурсов сети между разными потоками данных
- Методы обеспечения предсказуемого и согласованного поведения сети в условиях постоянно изменяющейся конфигурации
- Методы повышения эффективности работы и утилизации оборудования сети

## Перегрузка

Перегрузка сети в компьютерных сетях и теории очередей — состояние сети, при котором основные показатели качества обслуживания существенно ухудшаются. Возникает, когда на сетевой узел (например, коммутатор) поступает больше данных, чем он может обрабатывать/передавать. Типичные эффекты включают задержку в очереди, потерю пакетов или блокировку новых соединений. Перегрузки могут возникать как на отдельных участках сети (локальные перегрузки), так и распространяться на всю сеть (глобальные перегрузки).

Более формально: перегрузка — это явление, при котором увеличения нагрузки на сеть приводит к снижению пропускной способности (происходит это именно при переполнении очередей).

### Методы борьбы с перегрузкой:

- Буферизация (не уверена, что это тут нужно)
  - На входе: Нет необходимости в сверх-быстрой памяти. Если пакеты из нескольких входных портов начинают конкурировать за один и тот же вход коммутационной фабрики, возникает блокировка пакетов, находящихся за ними — Head Of Line (HOL) Blocking. При равномерном распределении маршрутов передачи пакетов производительность коммутатора равна менее 59% показателя коммутатора с буферизацией на выходе.
  - На выходе
  - На самом деле на сегодняшний день наиболее распространены модели Combined Input Output Queuing (CIOQ)
- управление перегрузкой

### Управление перегрузкой TCP: задачи

- Соединения должны адаптироваться к качеству предоставленной линии связи и стремиться использовать предоставленные ресурсы максимально эффективно
- Соединения должны автоматически распределять пропускную способность разделяемой ими линии связи справедливым образом

Более формально:

Алгоритм управления перегрузкой должен удовлетворять требованиям:

- Эффективность: сумма нагрузок источников (по сути скорость) в bottleneck узле должна быть как можно ближе к пропускной способности этого узла
- Справедливость: говорит о распределении нагрузок в bottleneck узле, есть разные индексы справедливости, например Jain fairness index и max-min.
  - Jain:  $F(r_1, r_2, \dots, r_n) = \frac{(\sum r_i)^2}{n(\sum r_i^2)}$ ,  $r_i \in A_i$ , где максимально справедливое распределение ресурсов достигается при  $F = 1$ .  $r_i$  — нагрузка,  $A_i$  — множество потоков проходящих через bottleneck  $i$ ,  $n = |A_i|$ .
  - Max-min: когда увеличение нагрузки любого из потоков ведет к уменьшению нагрузки другого с меньшей или такой же нагрузкой
- Сходимость: для данного критерия выделяют два аспекта
  - Отзывчивость (responsiveness): скорость с которой нагрузка  $r_i(t)$  достигает критерия эффективности
  - Гладкость (smoothness): величина колебаний, после достижения критерия эффективности

### Управления перегрузкой TCP: принципы работы

- Взаимодействующие с сетью — Сетевые устройства сигнализируют о возникновении перегрузки (TCP/ECN)
- Без взаимодействия с коммутаторами — Перегрузка определяется косвенно (по потере пакета, увеличению задержки и т.д.)
- Реактивные (как правило, loss based) — Детектируют возникновение перегрузок по факту
- Проактивные (как правило, delay based) — Ограничивают пропускную способность соединения, предчувствуя скорую перегрузку

При отсутствии дополнительных сервисных пакетов отправитель получает информацию из ACK-сообщений, поступающих к нему спустя один Round Trip Time (RTT), следовательно отправитель способен адаптироваться к состоянию сети не быстрее, чем за RTT.

### Примеры

- TCP Tahoe
  - медленный старт: начинаем с cwnd=1, потом \*2, до порога slow-start threshold, ssthresh.
  - предотвращение перегрузки (congestion-avoidance): +1
  - таймаут или 3 dupACK: cwnd=1, медленный старт, ssthresh=ssthresh/2
- TCP Reno
- Cubic
- Data Center TCP
- etc

## Модели распределения ресурсов

### Проблема эффективного распределения ресурсов

Проблема обеспечения качества связана с проблемой распределения сетевых ресурсов между потоками данных

- Проблему распределения ресурсов можно формулировать как задачу оптимизации
- Чем больше ресурсов вовлечено в обслуживание потока, тем выше качество его соединения
- Чем большее количество ресурсов позволяет задействовать модель распределения, тем выше эффективность сети, и тем больший уровень утилизации достигается. Чем выше уровень утилизации, тем лучше отношение производительности сети к стоимости сетевой инфраструктуры

Три модели обеспечения QoS:

- Best Effort — никакой гарантии качества. Все равны.
- IntServ — гарантия качества для каждого потока. Резервирование ресурсов от источника до получателя.
- DiffServ — нет никакого резервирования. Каждый узел сам определяет, как обеспечить нужное качество.

### **Модель Интегрированных Сервисов (IntServ)**

*Заблаговременное резервирование ресурсов для потока на всём протяжении от источника до получателя.*

Для модели IntServ требуется сигнализационный протокол (например RSVP), который до установки сессии или начала обменом данными выделит требуемые ресурсы на каждом сетевом устройстве.

Говорят, что IntServ — мертворожденный слон. В некотором смысле современная инкарнация IntServ — это MPLS TE с адаптированной под передачу меток версией RSVP — RSVP TE. Хотя здесь, конечно же не End-to-End и не per-flow.

Мультимедийный трафик в сети:

- Как оградить TCP трафик от мультимедийных данных, передающихся через UDP?
- Как обеспечить качество соединения?
- Гарантированный уровень качества можно обеспечить лишь с помощью резервирования ресурсов — закрепления части ресурсов сети за конкретным потоком данных. Основная идея — прокладывание маршрутов с заданным качеством путём предварительного резервирования ресурсов на оборудовании
- Модель лишь расширяет архитектуру Интернета, сохраняется совместимость с best-effort
- Модель особенно эффективна при многоадресной передаче данных
- Допускаются накладные расходы на предварительное прокладывание маршрута
- всё или ничего — модель или гарантирует соединение нужного качества, или отказывается предоставить какое-либо соединение.

#### **Основные компоненты модели IntServ**

- Классификатор (classifier) — Разделение пакетов на классы обслуживания
- Планировщик (scheduler) — Обеспечение выполнения требований QoS
- Контроль доступа (admission control) — Оценка возможности добавления потоков
- Протокол резервирования ресурсов — Резервирование ресурсов вдоль маршрута

#### **Расчёт и обеспечение качества соединений**

- Распределение по очередям на входящих и исходящих интерфейсах коммутатора
- Использование policing & shaping для формирования нужного профиля потоков
- Установка надлежащих дисциплин сброса пакетов при их постановке в очередь и выборки пакетов из очередей
- Настройка алгоритмов планирования коммутационной матрицы

### **Модель Дифференцированных Сервисов (DiffServ)**

Когда в конце 90-х стало понятно, что End-to-End подход IntServ в IP провалился, в IETF созвали в 1997 рабочую группу «Differentiated Services», которая выработала следующие требования к новой модели QoS:

- Никакой сигнализации.
- Основан на агрегированной классификации трафика, вместо акцента на потоках, клиентах итд.
- Имеет ограниченный и детерминированный набор действий по обработке трафика данных классов.

Стоит обратить внимание, что название DiffServ — это не антитеза IntServ. Оно отражает, что мы дифференцируем сервисы, предоставляемые различным приложениям, а точнее их трафику, иными словами разделяем/дифференцируем эти типы трафика. IntServ делает то же самое — он различает типы трафика BE и Real-Time, передающиеся на одной сети. Оба: и IntServ и DiffServ — относятся к способам дифференциации сервисов.

Модель поведения определяется набором инструментов и их параметров: Policing, Dropping, Queuing, Scheduling, Shaping.

1. Прежде всего нужно определить к какому классу сервиса относится трафик — **классификация (Classification)**.  
Каждый узел самостоятельно классифицирует поступающие пакеты.
2. После классификации происходит измерение (**Metering**) — сколько битов/байтов трафика данного класса пришло на маршрутизатор.
3. На основе результатов пакеты могут окрашиваться (**Coloring**): зелёный (в рамках установленного лимита), жёлтый (вне лимита), красный (совсем берега попутал).
4. Если необходимо, далее происходит полисинг (**Policing**). Полисер на основе цвета пакета назначает действие по отношению к пакету — передать, отбросить или перемаркировать.
5. До того, как пакет попадёт в очередь, он может быть отброшен (**Dropper**), если очередь заполнена
6. После этого пакет должен попасть в одну из очередей (**Queuing**). Для каждого класса сервиса выделена отдельная очередь.
7. На выходе из очереди работает шейпер (**Shaper**), задача которого очень похожа на задачу полисера — ограничить трафик до заданного значения.
8. Все очереди в итоге должны слиться в единый выходной интерфейс. Есть специальный диспетчер (**Scheduler**), который циклически вынимает пакеты из разных очередей и отправляет в интерфейс (**Scheduling**). На самом деле связка набора очередей и диспетчера — самый главный механизм QoS, который позволяет применять разные правила к разным классам трафика, одним обеспечивая широкую полосу, другим низкие задержки, третьим отсутствие дропов.
9. Далее пакеты уже выходят на интерфейс, где происходит преобразование пакетов в поток битов — **серIALIZАЦИЯ (Serialization)** и далее сигнал среды.

В DiffServ поведение каждого узла независимо от остальных, нет протоколов сигнализации, которые бы сообщили, какая на сети политика QoS. При этом в пределах сети хотелось бы, чтобы трафик обрабатывался одинаково. Если всего лишь один узел будет вести себя иначе, вся политика QoS псу под хвост. Для этого, во-первых, на всех маршрутизаторах, настраиваются одинаковые классы для них, а во-вторых, используется маркировка (**Marking**) пакета — его принадлежность определённому классу записывается в заголовок (IP, MPLS, 802.1q). И красота DiffServ в том, что следующий узел может довериться этой маркировке при классификации.

Кратко:

- Каждый маршрутизатор имеет несколько предопределённых классов обслуживания
- Пограничные маршрутизаторы определяют класс потока, маркируют его пакеты dscp метками и проводят traffic conditioning — используют инструменты policing & shaping для установки нужного профиля трафика
- На внутренних маршрутизаторах пакеты с более высоким приоритетом получают большую долю ресурсов, и наоборот

#### Стандартные классы обслуживания DiffServ

- Default Forwarding (DF) — Обычно обслуживается по best-effort
- Expedient Forwarding (EF) — Идёт через очередь с высшим приоритетом, маленькие delay, jitter & loss
- Assured Forwarding (AF) — Идёт через менее приоритетную очередь, охватывает несколько классов с разной политикой сброса при заполнении очереди
- Преимущества
  - Отсутствие внутренней фрагментации

- Высокая степень утилизации оборудования
  - Простота реализации в аппаратуре
  - Хорошая масштабируемость
- Недостатки
    - Не предоставляет гарантий качества
    - Метрики качества не рассчитываются явно
    - Ограниченнное количество классов качества

## 9. Основные подходы математического моделирования КС. Прототипирование КС: преимущества, недостатки, ограничения применимости.

### Методы моделирования

Модель — сущность/объект, который отображает процессы, протекающие в реальных системах с помощью математических или натурных средств. Отражение процессов осуществляется на основе оценки характеристик (зависимостей) или параметров процессов моделируемых систем. Основные условия выбора метода моделирования:

- Постановка задачи
- Составом, характером и объемом исходных данных
- Временем на решение исследовательской задачи

Методы моделирования:

- Натурное или физическое
- Аналитическое
- Имитационное
- Комбинированные методы

**Натурное моделирование** Измерение характеристик осуществляется на исследуемых системах в реальном времени (проведение экспериментов). Данные исследователь получает ведя наблюдение за процессами в реальной системе.

Достоинства:

- Высокая адекватность модели реальной системе
- Высокая точность результатов

Недостатки:

- Высокая стоимость создания модели
- Большие временные затраты
- Необходимость доработки отдельных узлов реальной системы для проведения натурных экспериментов

**Аналитическое моделирование** Модель представляется совокупностью аналитических выражений, которые отражают функциональные зависимости между параметрами реальной системы в процессе ее работы. Аналитические модели применяются для относительно простых систем, для исследования характеристик которых не требуется высокая точность.

Достоинства

- Простота и низкая стоимость модели
- Возможность быстро получить численные результаты

Недостатки

- Большое число допущений и ограничений
- Невысокая точность результатов
- Соответствие результатов определенным условиям
- Большая сложность аналитического описания функциональных зависимостей

**Имитационное моделирование** Метод исследования, при котором изучаемая система заменяется моделью с достаточной точностью описывающей реальную систему и с ней проводятся эксперименты с целью получения информации об этой системе. Экспериментирование с моделью называют имитацией (имитация — это постижение сути явления, не прибегая к экспериментам на реальном объекте). Это метод математического моделирования. Существует класс объектов, для которых по различным причинам не разработаны аналитические модели или аналитические методы решения полученной модели. В таких случаях математическая модель заменяется имитационной. Достоинства:

- Высокая адекватность между физической сущностью описываемого процесса и его моделью
- Возможность описать сложную систему на достаточно высоком уровне детализации
- Значительно больший охват исследования, чем у аналитического моделирования
- Отсутствие ограничений на зависимости между параметрами модели
- Возможность оценки функционирования системы не только в стационарных состояниях, но и в переходных процессах (режимах)
- Получение большого числа данных об исследуемом объекте
- Наиболее рациональное отношение “результат-затраты” по отношению к аналитическому и физическому моделированию

Недостатки

- Относительно большая сложность создания модели
- Необходимость высокой квалификации исследователя для написания модели
- Необходимость проведения верификации и валидации данных моделирования
- Индивидуальность реализации. Для широкого применения модели необходимо сделать детальное описание ее построения

**Комбинированные методы** Модель представляется как комбинация методов моделирования. Наиболее широко применяются имитационно-аналитические модели. Степень применения методов моделирования определяет исследователь, исходя из поставленных задач, имеющихся ресурсов и времени на проведение исследовательской работы.

## Основные подходы математического моделирования КС

### Детерминированное сетевое исчисление

Сетевое исчисление (Network Calculus) — это совокупность математических результатов, которые позволяют исследовать граничные значения характеристик функционирования таких сложных технических систем, как сети связи, цифровые электрические цепи, конкурирующие программы.

Представление в виде сети обработчиков:

- Обработчик — логически целостный компонент, выполняющий преобразования потоков данных

Описание обработчиков:

- Характеристики производительности — Принципы мультиплексирования

Описание нагрузки — множества потоков данных, поступающих в систему:

- Маршрут передачи данных
- Характеристики интенсивности

Накопительная функция — зависимость количества переданных данных от времени (ясно, что монотонно неубывает).

### Основные определения

- Функция прибытия  $A(t)$  описывает зависимость суммарного количества данных, поступивших на обработчик от времени

- Функция отправки  $D(t)$  - зависимость количества переданных данных потока от времени
- Каждый обработчик может быть описан перечислением пар вида  $\langle A(t), D(t) \rangle$
- Отставание(backlog)  $b(t)$  - выражает количество данных, находящихся внутри обработчика
- Период отставания - промежуток в течение которого функция отставания строго положительна
- Задержка (delay)  $d(t)$  - время прохождения через обработчик той порции данных, которая поступила на него в момент времени  $t$ .



Кривые нагрузки определяются профилями трафика после shaping'a & policing'a.



## Результаты



## Применение Network Calculus

- Separated Flow Analysis
- Линейное программирование

## Стохастическое сетевое исчисление

Идея: позволить нарушения кривых нагрузки и сервиса с некоторой вероятностью, следовательно теряем точность результата.

Принципы:

- Задать распределение для поступающего трафика и обслуживания обработчиков
- Получить афинную оценку
- Связать афинную оценку с погрешностью оценки
- Сохраняем подходы из детерминированного сетевого исчисления

## Прототипирование

Прототипирование КС — это моделирование КС с помощью “почти реальных” виртуальных машин/контейнеров и тд. В таком случае мы строим сеть из полноценных сетевых устройств, пусть даже с некоторым уровнем абстракции. Это может быть как ручками построенная сеть из виртуальных машин или контейнеров (lxc, docker), так и mininet/maxinet.

## Linux LXC Контейнеры

LXC используют Cgroups для изоляции ресурсов. Cgroups - это механизмы виртуализации и изоляции, которые поддерживаются ядром Linux начиная с версии 2.6.24. Cgroups позволяют обеспечить сетевыми интерфейсами, таблицами маршрутизации и ARP-таблицами процессы в рамках одной операционной системы. Это один из видов виртуализации на уровне ОС, позволяющий запустить множество однотипных процессов в изолированном и ограниченном по ресурсам окружении.

Используемые операции с сетевым пространством имен:

1. Создание сетевого пространства имен
2. Ассоциирование интерфейса с сетевым пространством имен
3. Конфигурирование интерфейса в сетевом пространстве имен

## Mininet

Техники, подобные Cgroups, позволяют Mininet создавать в пространстве ядра или пользователя коммутаторы, OpenFlow-контроллеры и хосты, и взаимодействовать в рамках моделируемой сети. В качестве виртуальных коммутаторов используется адаптированная реализация OpenvSwitch'а. Основная функциональность Mininet реализована на Python, за исключением некоторых утилит написанных на Си. Практически любая произвольная топология может быть описана с помощью специального синтаксиса на Python. В интернете можно найти множество интересных лабораторных работ на базе mininet, решающих различные задачи. Например реализация простого маршрутизатора.

## Docker

Докер — это открытая платформа для разработки, доставки и эксплуатации приложений. Позволяет отделить ваше приложение от вашей инфраструктуры. Позволяет запускать практически любое приложение безопасно изолированное в контейнере.

Что использует:

- Пространство имен (namespaces)
- Control groups (контрольные группы)
- Union File System

*(Дальше придумывала сама, добавляйте свои пункты, если есть идеи)*

Преимущества

- Высокий уровень детализации модели без использования физических устройств
- Легкость создания модели

Недостатки

- Сложность моделирования больших сетей (вроде по оценкам Антоненко mininet может создать ~1000 хостов максимум)
- Можно исследовать только те характеристики, которые есть в используемых абстракциях сетевых устройств

Ограничения применения

- Насколько я понимаю, нельзя устанавливать необходимые характеристики линков между сетевыми устройствами (например, пропускную способность)

## 10. Динамическое планирование задач в ИУС РВ. Схемы планирования Rate Monotonic (фиксированные приоритеты) и Earliest Deadline First (динамические приоритеты). Оценка времени отклика задач для схемы Rate Monotonic.

### Динамическое планирование задач в ИУС РВ

В ИУС РВ есть два режима активации задач:

- Периодические задачи (time driven) – задача автоматически активируется ядром через регулярные интервалы времени;
- Апериодические задачи (event driven) – задача активируется при наступлении какого-либо события (например, вызывается прерывание).

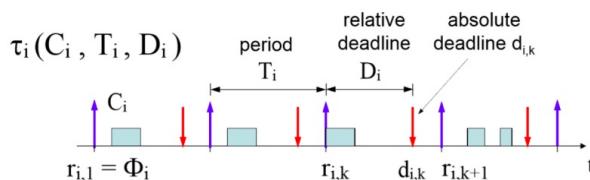
Так как в ИУС РВ много периодических задач с разной скоростью вызова, а также много аperiодических событий, в один и тот же момент времени для выполнения могут быть доступны несколько задач. Порядок выполнения таких задач определяет планировщик. Его работа оказывает влияние на:

- Время отклика задачи;
- Задержку выполнения задач и jitter;
- Время выполнения задачи (так как вытеснение задачи уничтожает кэш);
- Преодоление перегрузки;
- Оптимизацию использования ресурсов;
- Сохранение энергии, затрачиваемой процессором.

Для каждой задачи  $t_i$  определены следующие характеристики:  $C_i$  – время выполнения задачи в наихудшем случае (WCET);  $D_i$  – относительный дедлайн задачи;  $T_i$  – время между активациями задачи (период) (может быть просто минимальным временем между активациями, если период не фиксированный); в некоторых системах  $D_i = T_i$

Каждый экземпляр задачи (т.е. задача, выполненная внутри своего экземпляра периода), называется работой;

- $r_{i,k}$  время активации  $k$ -ой работы задачи  $t_i$ ;  $d_{i,k}$  - абсолютный дедлайн  $k$ -ой работы задачи  $t_i$ ;  $r_{i,k} = \Phi_i + (k - 1) * T_i$ ;  $d_{i,k} = r_{i,k} + D_i$



Далее: набор задач спланирован  $\equiv$  построено расписание выполнения задач, при котором каждая из них завершается до своего дедлайна; необходимо проверить существование расписания до запуска в run-time.

### Rate Monotonic (фиксированные приоритеты)

Суть схемы планирования с фикс. приоритетами:

- каждая задача имеет фикс. (статический) приоритет, определенный до запуска системы (до run-time);
- задачи выполняются в порядке, заданном их приоритетами (1 - самый отстойный приоритет);

В Rate Monotonic приоритеты задач определяются след. образом: если дан набор задач с известными периодами, то каждой задаче ставится уникальный приоритет по следующему принципу: чем меньше период, тем выше приоритет ( $T_i < T_j \Rightarrow P_i > P_j$ ).

Утверждение (на курсе не доказывалось): если какой-то набор задач может быть спланирован при помощи схемы планирования с фиксированным приоритетом (с вытеснением задач, т.е. выполнение низкоприоритетных задач может прерываться выполнением высокоприоритетных задач), то данный набор может быть спланирован и с Rate Monotonic.

#### **Earliest Deadline First (динамические приоритеты)**

Запущенные задачи выполняются в порядке, определяемом их абсолютными дедлайнами  $d_{i,k}$ . Следующая задача для запуска имеет ближайший абсолютный дедлайн. Хотя относительные дедлайны (значение  $D_i$ ) известны заранее, абсолютный дедлайн считается во время выполнения задачи, поэтому эта схема описывается как динамическая.

#### **EDF vs FPS (Fixed Priority Scheduling)**

- FPS (а RM это подвид FPS) проще реализовать, так как приоритеты статичны.
- EDF – динамический и требует более сложной системы, которая будет иметь больше накладных расходов на расчёты во время планирования.
- В FPS проще добавить задачи без дедлайнов (просто назначить наименьший приоритет)
- Проще учитывать другие факторы в отношении приоритетов, чем в отношении дедлайнов;
- Во время перегрузки FPS более предсказуем (дедлайн могут пропустить низкоприоритетные задачи), а EDF непредсказуем (задачи могут начать пропускать дедлайн по принципу домино);
- Но EDF лучше организует процессорное время.

#### **Оценка времени отклика задач для схемы Rate Monotonic**

Для ответа на вопрос «является ли данный набор задач планируемым» для схемы Rate Monotonic используется анализ времени отклика. Анализ времени отклика задач состоит из следующих шагов:

1. Считается интерференция ( $I_i$ ) для задачи  $i$  от более высокоприоритетных задач  $I_i = \sum_{T_k < T_i} C_k$
2. Вычисляется время отклика как  $R_i = C_i + I_i$
3. Проверяется, что  $R_i \leq D_i$ ; если верно для всех задач, то планируемо.

Интерференция задачи  $k$  на задачу  $i$  на интервале времени  $[0, R_i]$ :  $I_{i,k} = \lceil \frac{R_i}{T_k} \rceil * C_k$

Интерференция высокоприоритетных задач на задачу  $i$ :  $I_i = \sum_{T_k < T_i} \lceil \frac{R_i}{T_k} \rceil * C_k$

Тогда время отклика считается как,  $R_i = C_i + \sum_{j \in hp(i)} \lceil \frac{R_i}{T_j} \rceil * C_j$ , где  $hp(i)$  – набор задач с приоритетом выше, чем у задачи  $i$ .

Такое уравнение решается рекуррентным соотношением  $w_i^{n+1} = C_i + \sum_{j \in hp(i)} \lceil \frac{w_i^n}{T_j} \rceil * C_j$

Набор значений  $w_i^0, w_i^1, \dots, w_i^n, \dots$  монотонно не убывает. Когда  $w_i^n = w_i^{n+1}$ , решение найдено.  $w_i^0$  должно быть не больше, чем  $R \sim i \sim$  (например, 0 или  $C_i$ ).

# **11. Понятие наихудшего времени выполнения программы (WCET). Факторы, влияющие на WCET. Фазы анализа WCET. Использование абстрактной интерпретации для выявления недопустимых путей. Анализ влияния конвейера на время выполнения программы.**

Простейшая вычислительная задача:

- входные данные доступны в момент старта
- выходные данные готовы в момент завершения
- нет блокировок в процессе выполнения
- нет синхронизации или обмена данными в процессе выполнения
- время выполнения зависит только от: входных данных и состояния задачи в момент старта

## **Наихудшее время выполнения программного кода (WCET)**

– это максимальное время, которое требуется для выполнения данного фрагмента кода, в данном контексте (входные данные, состояние), на заданном аппаратном вычислителе. WCET используется в формуле оценки времени отклика задач для RM:  $C_i$ .

(есть еще BCET – Best-case execution time - нижняя оценка времени выполнения)

Цель анализа WCET – оценить сверху время выполнения фрагмента кода. Оценка должна быть:

- Безопасной (недопустимо ошибаться в меньшую сторону)
- Точной (завышенность приведёт к излишнему резервированию ресурсов системы)
- Затраты на ее анализ д.б. разумными

## **WCET нельзя просто замерить, т.к.:**

- замер времени на ВСЕХ путях выполнения невозможен
- при определении тестовой выборки могут быть упущены редкие сценарии
- выбранные тестовые данные могут не породить самую длинную трассу
- внутреннее состояние процессора на момент старта измерений может НЕ БЫТЬ наихудшим

## **Факторы, влияющие на WCET:**

- Возможные пути (последовательности действий) выполнения задачи, определяются:
  - Семантикой программного кода (спецификой реализации, в т.ч. аппаратно-зависимой семантикой)
  - Входными данными, возможными в данном контексте вызова программы
- Длительность выполнения каждого действия на каждом возможном пути выполнения, определяются
  - Аппаратной реализацией команд процессора
  - Состоянием аппаратных средств, влияющих на тайминги (кэш-память, конвейер и т.п.) (как связанные с самой задачей, так и связанные с внешними факторами (состояние на момент старта, вытеснение задачи))

Длительность выполнения пути:

- В простом случае – длительность каждого действия константа.
- В реалистическом случае – длительности варьируются (конвейер, кэш-память, параллелизм процессора).

## **Фазы анализа WCET:**

11. Понятие наихудшего времени выполнения программы (WCET). Факторы, влияющие на WCET. Фазы анализа WCET. Использование абстрактной интерпретации для выявления недопустимых путей. Анализ влияния конвейера на время выполнения программы.

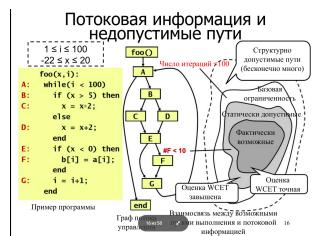
---

1. Анализ потоков: ограничить (сверху) число выполнений различных фрагментов программы (в основном, анализ программной составляющей) (даёт верхнюю оценку, которая корректна для всех возможных трасс) *Примеры предоставляемой информации*: ограничение на число итераций цикла, ограничение на глубину рекурсии, недопустимые пути выполнения. *Источники информации*: статический анализ программы и ручные аннотации кода.
2. Низкоуровневый анализ: ограничить (сверху) время выполнения различных фрагментов программы (сочетание анализа программной и аппаратной составляющих), большая часть исследований по WCET посвящена этой фазе. *Использует*: модель целевой аппаратуры (не требуется моделирование всех подробностей, при этом необходимо безопасно сверху оценить все задержки при работе аппаратуры) *Применяется*: к скомпилированному двоичному коду (исполняемой программе)
3. Вычисление: объединить результаты анализа потоков и низкоуровневого анализа, чтобы получить верхнюю оценку WCET *Примеры подходов*: расчёт по синтаксическому дереву (дерево обходится снизу-вверх), расчёт по путям выполнения, неявный перебор путей (IPET).

#### **Использование абстрактной интерпретации для выявления недопустимых путей** (фаза анализа – Анализ потоков)

Анализируется граф потока управления, множество путей программного выполнения имеет вложенную структуру: все фактически возможные пути  $\leq$  статически допустимые (например,  $if(x < 23)\{ \}$ )  $\leq$  базовая ограниченность (например, число итераций цикла  $< 100$ )  $\leq$  структурно допустимые пути (бесконечно много); Требование базовой ограниченности: для каждого цикла должно быть известно (вычислено или задано) ограничение на число итераций. Недопустимые пути (например, найденные на основе  $if$ ) исключаются из множества статически допустимых путей.

11. Понятие наихудшего времени выполнения программы (WCET). Факторы, влияющие на WCET. Фазы анализа WCET. Использование абстрактной интерпретации для выявления недопустимых путей. Анализ влияния конвейера на время выполнения программы.



### Абстрактная интерпретация (АИ):

1. Ограничивает число итераций циклов и выявляет недопустимые пути
  - Вычисляет безопасную (расширенную) оценку множества значений каждой переменной для различных точек выполнения программы
  - В ходе АИ, переменной сопоставляется не конкретное значение, а множество значений («абстрактное» значение)
2. Программа «выполняется» с использованием абстрактных значений переменных
3. Результат выполнения: безопасная (расширенная) оценка множества допустимых путей выполнения
  - Все фактически допустимые пути входят в полученное множество
  - Также в него могут входить некоторые фактически НЕ допустимые пути
  - Пути, не вошедшие в полученное множество, гарантированно не допустимы

2.1

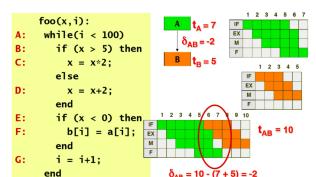
**Анализ влияния конвейера на время выполнения программы** (фаза анализа – Низкоуровневый анализ) Простой конвейер: *Instruction fetch* (выборка команды) -> *instruction decode* (декодирование команды) -> *execution* -> *memory access* (загрузить / сохранить значения в/из памяти) -> *write back* (запись результата)

- В идеале: коэффициент ускорения равен числу ступеней конвейера
- Фактически: между командами есть зависимости по данным; «слом» конвейера при неверном предсказании ветвления; ожидание данных из памяти

Виды конвейеров: отсутствует / скалярный (один конвейер) / VLIW (несколько конвейеров, статическое планирование из загрузки на уровне компилятора) / Суперскалярный (несколько конвейеров, внеочередное выполнение команд (процессор может на ходу переставить порядок команд))

Чтобы учесть задержки в случае простого конвейера – в графе потока управления ребрам присваиваются отрицательные веса:

Задержки: простой конвейер



**Учет совместного влияния кэша и конвейера:** Анализ влияния конвейера должен брать на вход результаты анализа влияния кэш-памяти

- Команды помечаются попаданием/промахом в кэш
- Попадания/промахи влияют на задержки в конвейере

## **12. Архитектура интегрированной модульной авионики (ИМА), её основные преимущества, примеры типов модулей (шина VME). Статико-динамическая схема планирования вычислений в системах ИМА.**

Федеративная ИУС РВ (более старое поколение):

- специализированные блоки (по назначению/ по архитектуре)
- ПО различных подсистем на разных блоках (изоляция по памяти, нет конкуренции за выч. ресурсы)

*Недостатки:*

- система неоднородна (блоки сильно различаются)/
- модули одного блока тесно интегрированы друг с другом
- модули разных блоков слишком изолированы
- низкая переносимость и повторная используемость ПО
- низкая отказоустойчивость и реконфигурируемость (блок выходит из строя целиком)

**Архитектура ИМА и ее основные преимущества:**

- логически единый распределённый вычислитель (единая архитектура, унифицированные модули, унифицированные программные интерфейсы)
- разделение вычислительных ресурсов между ПО различных подсистем.

*Проблемы:* конкуренция за процессорное время; изоляция по памяти.

*Решение:* каждой подсистеме - раздел.

Для ИМА характерно:

- Стандартное API со стороны ОС.
- Статическое разделение времени, памяти и ресурсов.
- Унификация: вычислительные модули, сетевое оборудование и протоколы
- Интеграция: программное обеспечение, потоки данных
- Виртуализация: процессоры, память, сеть

*Преимущества:* надёжность, переносимость, возможность повторного использования, модульность, упрощение верификации и сертификации.

описание ИМА:

- Модули всех блоков равноправно подключены к среде обмена данными => система – «облако» модулей
- Сервисная шина – последовательная, относительно медленная (помехоустойчивость), может объединять все модули системы ИМА
  - CAN bus - Controller Area Network – сеть контроллеров
- Трафик по сервисной шине минимален (низкоуровневые данные о состоянии, простейшие команды вроде вкл/выкл модуля)
- Высокая отказоустойчивость и реконфигурируемость
  - модуль вышел из строя => заменить его может модуль из другого блока
  - поддержка виртуальных каналов => возможность миграции вычислительных задач

**Примеры модулей (это без привязки к VME):**

1. Модуль процессора данных - функциональный модуль общего назначения. Задачи: обработка данных, выполнение

вычислительных задач, принятие управленческих решений.

2. Модуль ввода-вывода - функциональный модуль специального назначения. Задачи: прием/выдача сигналов по «унаследованным» бортовым интерфейсам, преобразование унаследованных форматов сообщений в стандартный формат ARINC 653.
3. Модуль графического контроллера - функциональный модуль специального назначения. Задачи: построение изображений на основе данных, полученных от вычислительных задач, обработка входных видеоизображений, прием/выдача изображений по оптическим видеоканалам.
4. Модуль коммутатора FC - предназначен для обеспечения взаимодействия в сети Fibre Channel
5. Модуль источника питания - вспомогательный модуль, обеспечивающий вторичное питание модулей с требуемыми характеристиками.

#### Шина VME:

- Параллельная шина с арбитражем
- Реализует прямой доступ к памяти модулей
- Объединяет модули в блок (крейт)
- Разрядность шины данных: 32 или 64 бита
- Пропускная способность:
  - 40 Мбайт/с (VME32)
  - 80 Мбайт/с (VME64)
  - до 320 Мбайт/с (VME64 в блочном режиме, на одну передачу адреса – несколько передач данных)

VME - Адресная шина данных с арбитражем и прерываниями (напр., использовалась как процессорная шина Motorola 68000);

Структура шины VME:

- адресная шина
- шина данных
- шина арбитража
- шина прерываний

#### Роли модулей нашине VME:

- *Ведущий* (Master) - может инициировать передачу данных
- *Подчинённый* (Slave) - отвечает на запросы от ведущего
- *Источник прерывания* (Interrupter) - модуль, способный формировать прерывание (обычно – подчинённый)
- *Обработчик прерывания* (Interrupt handler) - модуль, способный обрабатывать прерывания (как правило, одноплатный компьютер)
- *Арбитр* (Arbiter) - модуль, управляющий доступом к шине и осуществляющий мониторинг обмена по шине.  
Устанавливается в слот №1

#### Процедура передачи данных на VME:

1. Ведущий устанавливает запрос на передачу данных на шине арбитража (Ш.А.) – при этом ведущий устанавливает на Ш.А. «свою» линию запроса в активное состояние (логический 0)
2. Шина освобождается от текущей передачи данных =>
  - арбитр определяет, какие линии запроса активны на Ш.А.
  - арбитр выбирает ведущего, которому отдать шину, и устанавливает в активное состояние линию Ш.А. «доступ дан» для этого ведущего
3. Получив доступ к шине, ведущий устанавливает:
  - на шине данных: значения передаваемых данных (в случае отправки); разрядность данных – не больше разрядности шины данных
  - на шине адреса: номер подчинённого устройства, адрес в памяти подчинённого устройства, разрядность передаваемых данных (8, 16, 32 бита; также 64 бита для VME64), признак «чтение» или «запись»

4. Подчинённое устройство:

- на Ш.А. признак «чтение» => устанавливает на шине данных значения данных заданной разрядности с заданного адреса своей памяти
- на Ш.А. признак «запись» => записывает по заданному адресу своей памяти данные заданной разрядности с шины данных

**Прерывания VME:**

- Прерывание – способ для подчинённого устройства оповестить какое-либо из ведущих устройств о необходимости обмена данными
- Запрос прерывания выставляется на одной из 7 линий шины прерывания
- Ведущие устройства сами разбираются, кому адресован запрос

Блочная передача данных:

- Поддерживается в VME64
- В начале обмена задаётся адрес и число передаваемых блоков данных
- Выполняется передача заданного числа блоков данных (каждый блок - не шире шины данных) => значительная экономия времени на задании адреса

**Недостатки VME:**

- Медленная шина (по современным меркам)
- Параллельная шина (много линий на материнской плате, сложность повышения частоты работы)
- Невозможно одновременное выполнение нескольких обменов данными

**Статико-динамическая схема планирования** *Статико* – stands for статическое расписание окон. Границы окон одинаковы для всех ядер одного модуля. В одном окне могут выполняться задачи только одного раздела, между окнами для разных разделов происходит переключение контекста; *Динамическое* – stands for планирование работы в конкретном окне. При этом учитываются: очередь выполнения, приоритеты, вытеснение, ожидание входных данных.

**Сквозное планирование вычислений и информационного обмена:** *Дано:*

- Описание вычислительной системы с архитектурой ИМА
  - вычислительные средства (модули, процессоры в их составе)
  - сеть передачи данных (структура сети, транспортный протокол)
- Описание рабочей нагрузки
  - на вычислители: задачи (сгруппированы в разделы)
  - на сеть: сообщения, передаваемые между задачами

*Требуется:*

1. Распределить вычислительную нагрузку (привязать разделы к процессорным ядрам)
2. Построить конфигурацию сети (виртуальные каналы, их маршруты и характеристики)
3. Построить расписание окон для каждого ядра

**Вычислительные ресурсы и рабочая нагрузка на них:**

- Вычислительные ресурсы
  - набор модулей
  - модуль -> набор и типы процессорных ядер
  - ядро -> верхняя граница загрузки
- Рабочая нагрузка на вычислительные ресурсы (вычислительная нагрузка)
  - набор задач
  - набор разделов

- задача -> период, приоритет, WCET (для типа процессора), принадлежность к разделу
- раздел -> допустимые ядра

### **Сеть передачи данных и рабочая нагрузка на нее**

- Сеть передачи данных
  - набор коммутаторов
  - структура каналов (коммутатор-модуль, коммутатор-коммутатор)
  - пропускные способности каналов
  - транспортный протокол + соответствующая ему схема оценки задержек и джиттера передачи сообщений через сеть
- Рабочая нагрузка на сеть
  - набор сообщений
  - сообщение -> задача-отправитель, задача-получатель, размер
  - период передачи сообщения определяется периодом задачи-отправителя
  - Длительность передачи сообщения через сеть зависит от конфигурации сети (=> заранее не известна, может быть оценена снизу)

**Распределение вычислительной нагрузки** *Дано:* Описание вычислительной системы; Описание рабочей нагрузки

*Требуется:* Построить привязку разделов к процессорным ядрам *Оптимизируемый критерий:* Трафик между модулями -

> min *Ограничения:* Каждый раздел привязан к одному ядру, допустимому для него; Загрузка каждого ядра не превышает максимально допустимой

**Построение конфигурации сети** *Дано:* Описание вычислительной системы; Описание рабочей нагрузки; Распределение вычислительной нагрузки (=> модуль-отправитель и модуль-получатель для каждого сообщения); *Опционально:*

максимально допустимые длительности и джиттеры передачи сообщений через сеть (индивидуально по сообщениям)

*Требуется:* Построить систему виртуальных каналов; *Оптимизируемый критерий:* нет *Ограничения:* Непревышение пропускной способности каналов; Непревышение максимально допустимых длительностей и джиттеров

**Построение расписания окон** *Дано:* Описание вычислительной системы; Описание рабочей нагрузки; Распределение вычислительной нагрузки; Длительности передачи сообщений через сеть; Минимально и максимально допустимые длительности окон; *Требуется:* Построить расписание окон для каждого ядра (Ядро -> набор окон; Окно -> время начала, время завершения, раздел) *Оптимизируемый критерий:* нет *Ограничения:* окна для каждого ядра не перекрываются; к каждому окну привязан единственный раздел; длительность каждого окна не меньше минимально допустимой и не больше максимально допустимой; границы окон для всех ядер одного модуля должны совпадать; все работы выполняются в рамках директивных сроков, задаваемых периодами (при длительностях, равных WCET);

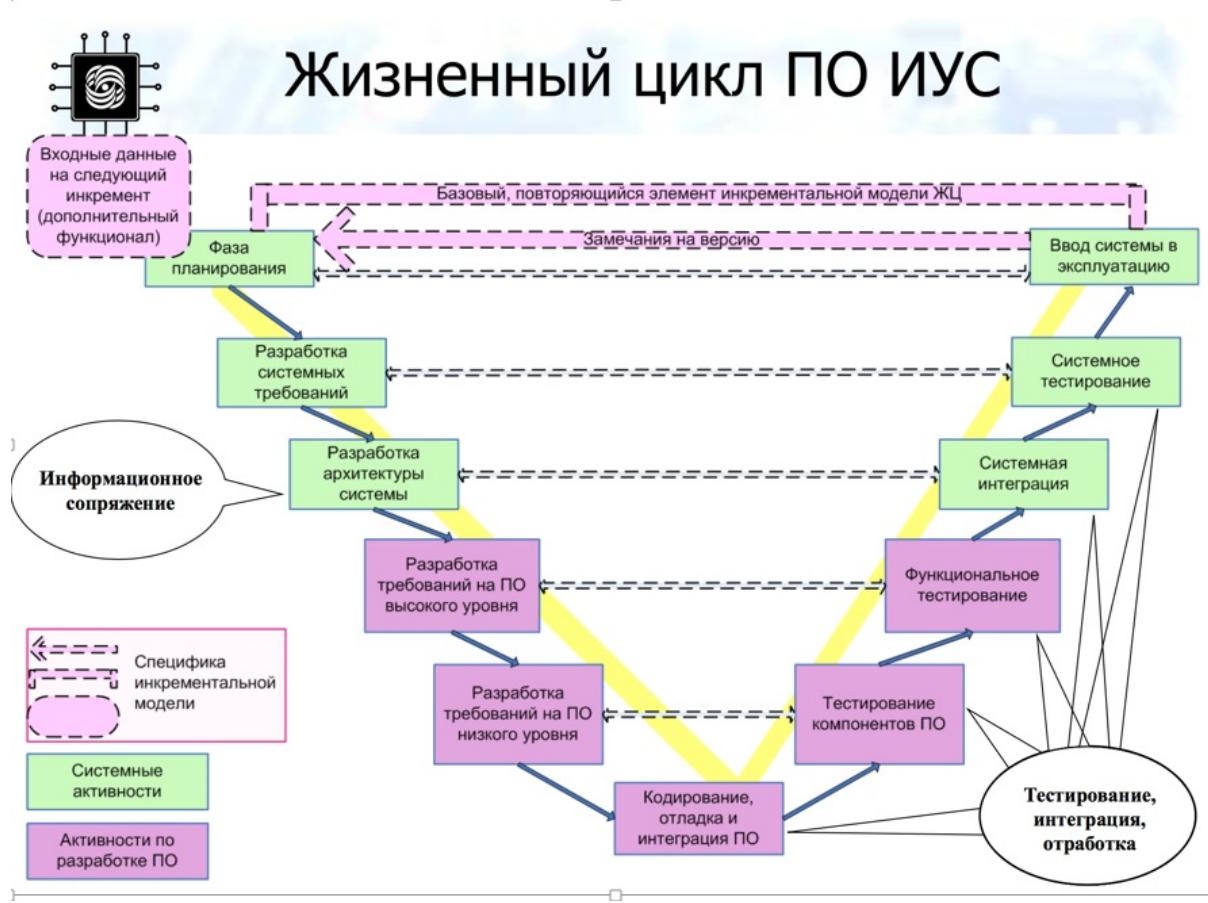
**Соблюдение директивных сроков проверяется имитационным моделированием работы динамического планировщика**

## 13. V-образный жизненный цикл (ЖЦ) программного обеспечения. Основные виды инструментальных средств поддержки ЖЦ, их отнесение к фазам ЖЦ. Структура комплекса стендов для поэтапной интеграции ПО и аппаратуры ИУС РВ на восходящей фазе ЖЦ.

Информационно-управляющая система (ИУС) — вычислительная система верхнего уровня, обеспечивающая:

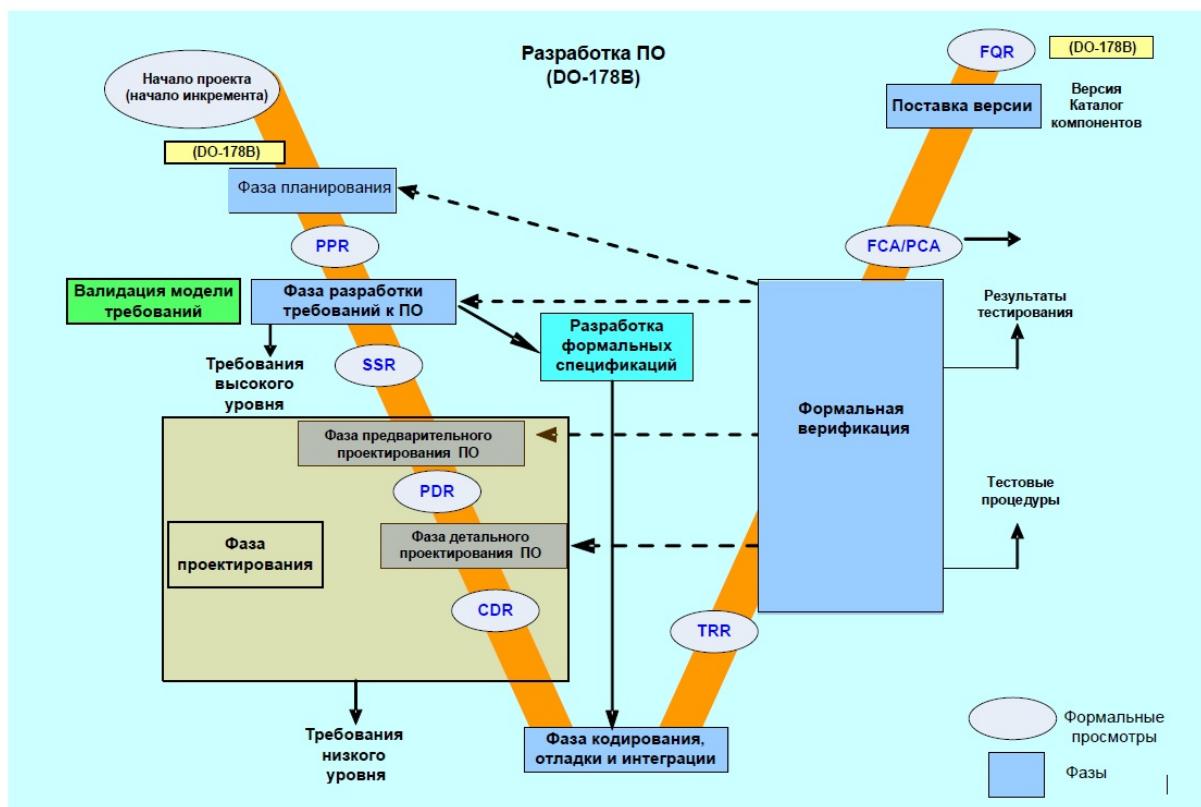
- функциональную и информационную интеграцию составных частей управляемого объекта
- взаимодействие между объектом и оператором

V-образный ЖЦ на рисунке:



Фазы ЖЦ на V-цикле (про вехи не спрашивают в билете).

## Фазы и вехи на V-цикле



### Основные виды инструментальных средств поддержки ЖЦ, их отнесение к фазам ЖЦ

на мой взгляд, достаточно знать сам вид и фазу + мб примеры. Остальное добавила - просто для понимания

#### Средства поддержки разработки требований Фазы: разработка требований Необходимая функциональность:

- создание и хранение требований, отслеживание истории
- связывание требований с версиями документов и ПО
- прослеживаемость требований на: Низкоуровневые требования; Формальные спецификации; Код; Тесты

Примеры средств: IBM DOORS, Borland CaliberRM, SyBase PowerDesigner

Средства версионного/ конфигурационного контроля Фазы: проектирования; кодирования; Версия = вся совокупность документов: Требования, спецификации, код, тесты,...; Часть конфигурации борта Необходимая функциональность:

- Версионирование совокупности документов (в т.ч. атомарность изменений)
- Поддержка ветвей истории
- Поддержка групповой разработки, в т.ч. разграничения доступа
- Обновление документов в реальном времени

Примеры: CVS, Subversion, git, IBM ClearCase

Средства отслеживания проблем и изменений Фазы: формальная верификация (и как следствие исправление баг на остальных фазах)+ фазы, где идет в параллель свое тестирование (проектирование, кодирование, интеграция и тд)

Необходимая функциональность:

- Поддержка структуры продукта и процесса
- Настраиваемый формат сообщения о проблеме
- Настраиваемый ЖЦ сообщения, поддержка согласования

13. V-образный жизненный цикл (ЖЦ) программного обеспечения. Основные виды инструментальных средств поддержки ЖЦ, их отнесение к фазам ЖЦ. Структура комплекса стендов для поэтапной интеграции ПО и аппаратуры ИУС РВ на восходящей фазе ЖЦ.

---

- Поддержка групповой разработки, втч разграничения доступа
- Интеграция со средствами управления версиями

Примеры: Bugzilla, Trac, IBM ClearQuest

**Средства поддержки сопряжения подсистем ПО** Фазы: кодирование, интеграция; проектирования

- Средства автоматизации проектирования бортовых интерфейсов: Балансировка загрузки каналов; Формирование набора сообщений; Построение расписаний обмена (канал с централизованным управлением); Построение системы виртуальных каналов (сеть на основе коммутаторов);
- Средства автоматизации интеграции ПО: Использование унифицированных структурных компонентов ПО;

**Средства автоматизации проектирования индикационных форматов** Фазы: кодирование; проектирования

Индикационный формат = набор графических элементов + правила поведения *Необходимая функциональность*:

- Редактирование в графической форме, WYSIWYG
- Поддержка библиотеки элементов
- Поддержка автономного тестирования
- Генерация кода в формате для целевого устройства

Примеры: SCADE Display, VAPS, САПР ИФ

**Средства проектирования алгоритмов** Фазы: кодирование; проектирования; формальной верификации *Необходимая функциональность*:

- Поддержка обоих видов формального представления (потоковая обработка данных, конечный автомат)
- Графическое описание
- Тестирование и пошаговая отладка на уровне модели
- Верификация на основе формальных методов
- Сертифицированный кодогенератор

Примеры: Telelogic Rhapsody, SCADE Suite, Simulink

**Средства поддержки верификации и тестирования бортового ПО** Фазы: формальной верификации

- Тестирование на целевой платформе
  - Недопустимость инструментирования
  - Тестирование через каналы бортовых интерфейсов
  - Тестирование требований реального времени
  - Интерактивное тестирование индикационных форматов
  - Многоэтапное тестирование
  - Сопровождение интеграции подсистем КБО

*Необходимая функциональность*:

- Поддержка стандартов бортовых интерфейсов
- Многомашинные конфигурации
- Выполнение тестов в реальном времени
- Автоматическое и интерактивное тестирование
- Пакетный режим
- Формирование отчётов, прослеживаемость требований

Примеры средств: Rational Test RealTime, VectorCast, средства разработки ЛВК

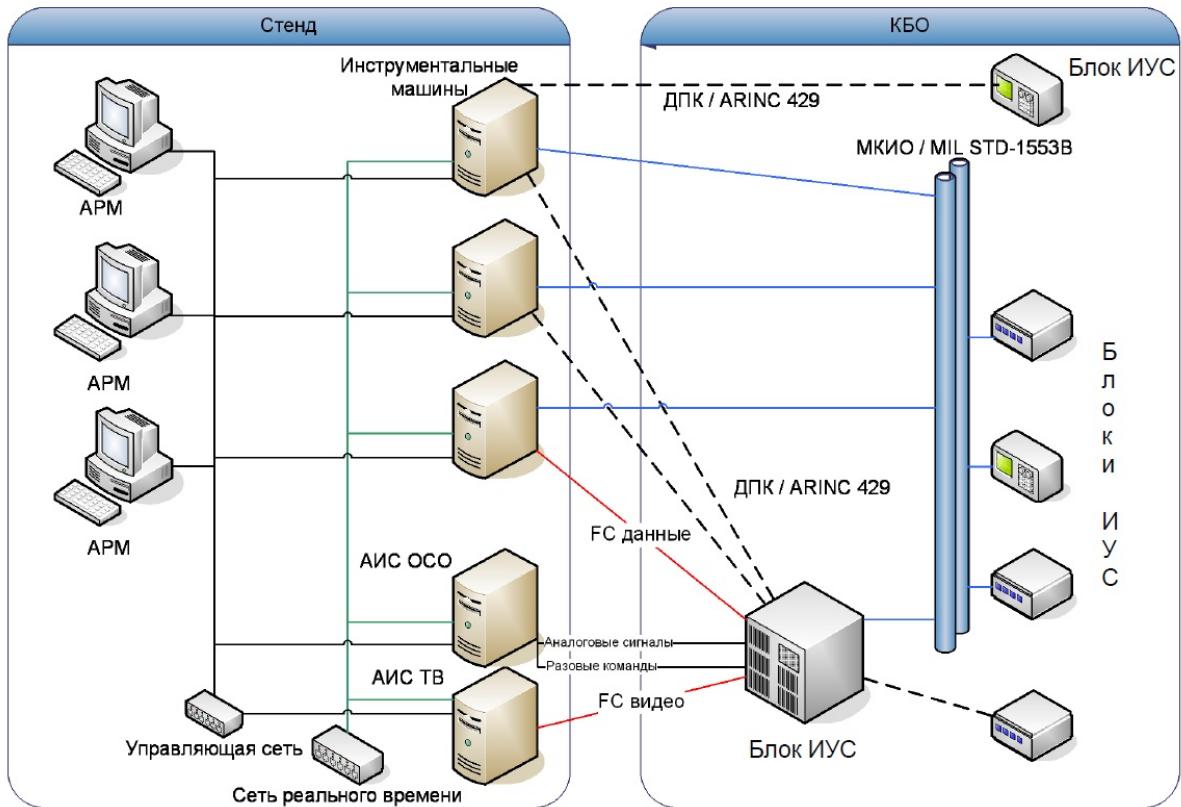
**Структура комплекса стендов для поэтапной интеграции ПО и аппаратуры ИУС РВ на восходящей фазе ЖЦ**

Комплекс средств тестирования ИУС (на картинке)

- Разработан в Лаборатории вычислительных комплексов ВМК МГУ

- Предназначен для тестирования устройств ИУС через каналы бортовых интерфейсов (КБИ)
- Функционирует на ПК под управлением ОС Linux, в состав которых входят адAPTERЫ КБИ
- Поддерживает распределенное выполнение тестовых сценариев
- Удовлетворяет перечисленным выше требованиям (очень кратко перечислены в пункте про "Средства поддержки верификации и тестирования бортового ПО")
- Положен в основу семейства стендов тестирования, отработки и интеграции ИУС

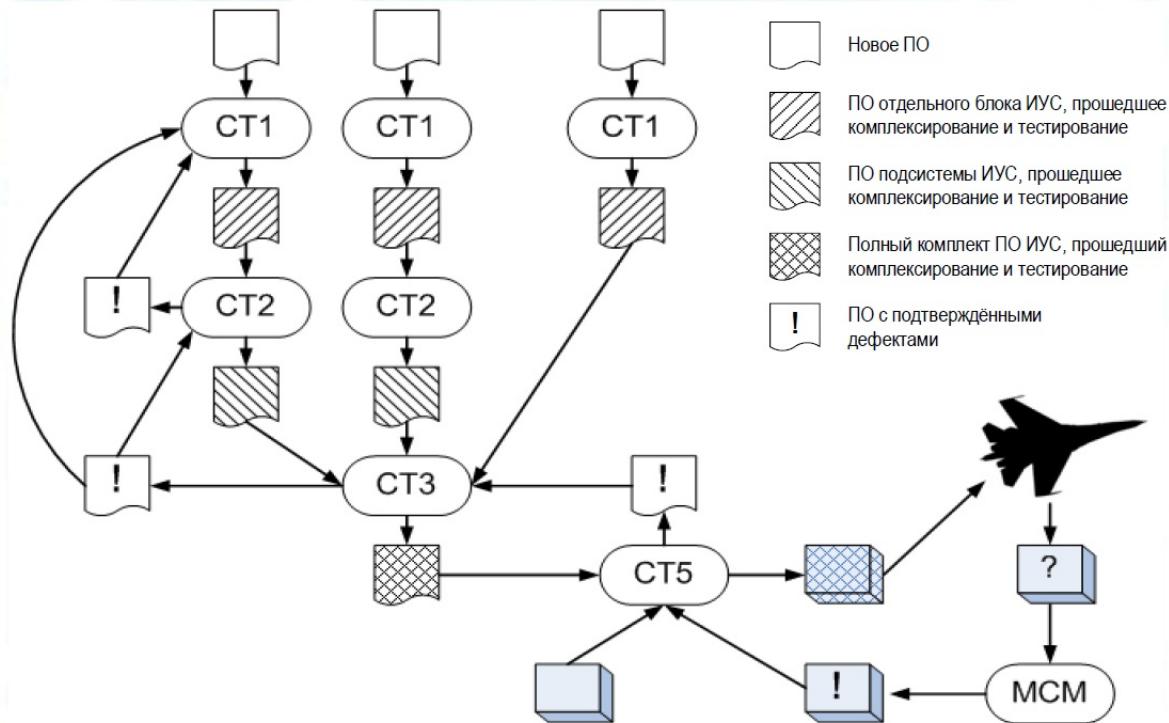
## Архитектура стенда тестирования ИУС



про интеграцию и ее верификацию, думаю, что имели ввиду вот эти 2 картинки



## Технологический цикл применения семейства стендов отработки ИУС



### Программное обеспечение (ПО)



Новое ПО



ПО отдельного блока ИУС, прошедшее комплексирование и тестирование



ПО подсистемы ИУС, прошедшее комплексирование и тестирование



Полный комплект ПО ИУС, прошедший комплексирование и тестирование



ПО с подтверждёнными дефектами

### Обозначения

### Стенды



Стенд тестирования и отладки ПО отдельного блока ИУС



Стенд тестирования и отладки ПО нескольких сопряжённых блоков ИУС



Стенд комплексирования и приёмосдаточных испытаний ИУС



Стенд приёмосдаточных испытаний серийных комплектов ИУС



Мобильная рабочая станция мониторинга и тестирования блоков ИУС

### Аппаратура



Новые серийные образцы блоков ИУС



Комплект ИУС, предположительно имеющий дефекты



Комплект аппаратуры и ПО ИУС, прошедший комплексирование и приёмосдаточные испытания



Блок ИУС с выявленными дефектами

21

13. V-образный жизненный цикл (ЖЦ) программного обеспечения. Основные виды инструментальных средств поддержки ЖЦ, их отнесение к фазам ЖЦ. Структура комплекса стендов для поэтапной интеграции ПО и аппаратуры ИУС РВ на восходящей фазе ЖЦ.

---

14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии.

Арчиловское: <https://drive.google.com/drive/folders/0B3XUBeyJ27tAaHBQSm56LU1zMVE>

## 14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии

### Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий)

Среди задач обучения с учителем можно выделить 2 основные:

- классификация
- регрессия

**Классификатором** называется следующее отображение

$$\hat{c} : X \rightarrow C, \text{ где } C = \{C_1, C_2, \dots, C_k\} - \text{конечное множество классов}$$

Проблема обучения для классификации -- обучить такую аппроксимацию  $\hat{c}$  истинной помечающей функции  $c$ .

Примеры классификации: определить кошку, собаку или какое-либо еще животное на фотографии (многоклассовая); определить есть ли кошка на фотографии или нету (бинарная).

Для классификаторов оценить качество можно построив **матрицу ошибок**. Предположим, что некоторые данные можно разделить на 2 класса (бинарная классификация). Таким образом, матрица будет выглядеть следующим образом

	$y = 1$	$y = 0$
$\hat{y} = 1$	True Positive	False Positive (ошибка 1 рода)
$\hat{y} = 0$	False Negative (ошибка 2 рода)	True Negative

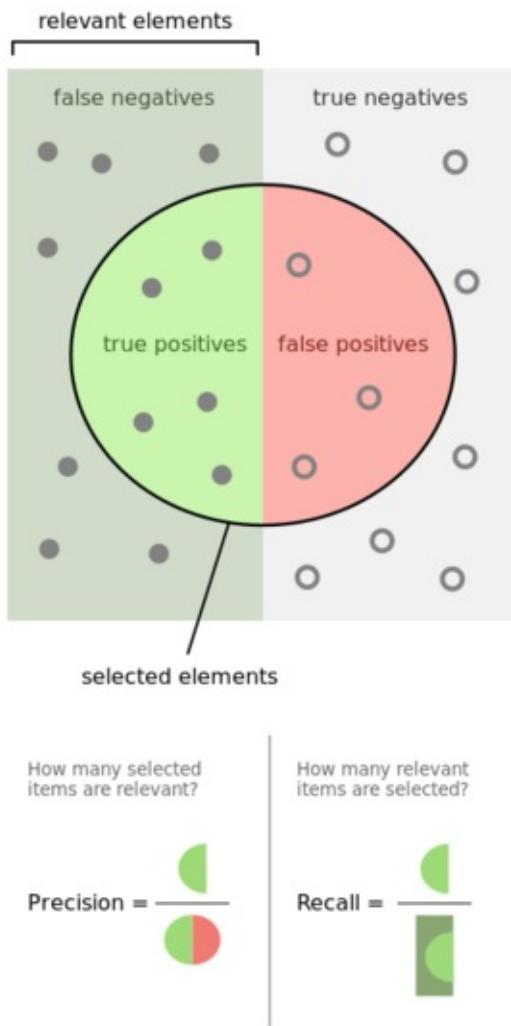
$y$  -- истинный класс объекта

$\hat{y}$  -- предсказанный класс классификатором

Соответственно, на диагонали подобной матрицы будет находиться количество объектов распознанных правильно. В остальных местах -- ошибки. Отмету также, что подобную матрицу можно построить и для многоклассовой классификации ( $|C| > 2$ ).

В зависимости от постановки исходной задачи ошибки 1 и 2 рода показывают разное. Например, в задаче по определению оттока абонентов, ошибкой первого рода будет принятие лояльного абонента за уходящего, так как наша **нулевая гипотеза** состоит в том, что никто из абонентов не уходит, а мы эту гипотезу отвергаем. Соответственно, ошибкой второго рода будет являться "пропуск" уходящего абонента и ошибочное принятие нулевой гипотезы.

14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии.



По такой матрице можно рассчитать различные **показатели качества классификатора** на размеченной выборке:

- accuracy (доля правильных ответов алгоритма) =  $\frac{TP+TN}{TP+FP+TN+FN}$  -- почти не используется из-за проблем с несбалансированными классами.
- precision (точность) =  $\frac{TP}{TP+FP}$
- recall (полнота) =  $\frac{TP}{TP+FN}$

**Precision** можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а **recall** показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм. Precision и recall не зависят, в отличие от accuracy, от соотношения классов и потому применимы в условиях несбалансированных выборок.

<https://habr.com/ru/company/ods/blog/328372/>

[https://en.wikipedia.org/wiki/Precision\\_and\\_recall](https://en.wikipedia.org/wiki/Precision_and_recall)

Рассмотрим задачу регрессии. **Оценочной функцией** называется отображение:

$$\hat{f} : X \rightarrow \mathbb{R}$$

Проблема обучения регрессии заключается в построении оценочной функции по примерам  $(x_i, f(x_i))$ .

Примеры: задача поиска оценочной функции для индекса Доу-Джонса или для FTSE 100, исходя из выбранных экономических показателей. Так же подобная функция может называться гипотезой.

14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии.

Регрессионные модели рассчитываются путем применения функции потерь к **невязкам**  $f(x) - \hat{f}(x)$  (для упрощения  $y - \hat{y}$ ) -- отличие предсказания  $\hat{y}$  от истинного значения  $y$

## Несколько популярных функций потерь $\mathcal{L}(\hat{y}, y)$ --

MAE -- Mean Absolute Error =  $\frac{1}{m} |y - \hat{y}|$

$$\text{MSE} \text{ -- Mean Squared Error} = \frac{1}{m} (y - \hat{y})^2$$

где  $m$  -- количество элементов  $x$ .

(у этих функций потерь есть еще некоторые вероятностные интерпретации + гипотезы накладываемые на параметры, но давайте не будем об этом)

**Риск**, ассоциированный с гипотезой  $\hat{f}(x)$ , определяется как математическое ожидание функции потерь:

$$R(h) = \mathbb{E}[\mathcal{L}(\hat{f}(x), y)]$$

Высшей целью алгоритма обучения будет являться поиск такой гипотезы  $\hat{f}^*$  в фиксированном классе функций  $H$ , для которой подобный риск  $R(\hat{f})$  минимален:

$$\hat{f}^* = \underset{\hat{f} \in H}{\operatorname{argmin}} R(\hat{f})$$

[https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F\\_%D1%8D%D0%BC%D0%BF%D0%B8%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B3%D0%BE\\_%D1%80%D0%B8%D1%81%D0%BA%D0%B0](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%8D%D0%BC%D0%BF%D0%B8%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B3%D0%BE_%D1%80%D0%B8%D1%81%D0%BA%D0%B0)

-----

Текст для дискуссии про эту часть билета:

Не знаю, что означает "средние и эмпирические" вот из билетов:

Эмпирические методы оценки обобщающей способности - методы основанные на контрольной выборке.

Средние операционные характеристики распознавания - ??? - возможно оценка обобщающей способности на самой обучающей выборке???

Предполагаю что эмпирически -- глазами посмотреть красиво/некрасиво (пример: фильтр в инстаграмме). Но инфы как-то мало.

-----

## Проблема переобучения

Основным способом поиска закономерностей (процесса обучения) является поиск в некотором априори заданном семействе алгоритмов прогнозирования  $M' = A : X' \rightarrow Y'$  алгоритма, наилучшим образом аппроксимирующего связь переменных из набора  $X_1, \dots, X_n$  с переменной  $Y$  на обучающей выборке, где  $X'$  – область возможных значений векторов переменных  $X_1, \dots, X_n$  (известные переменные);  $Y'$  – область возможных значений переменной  $Y$  (прогнозируемая величина).

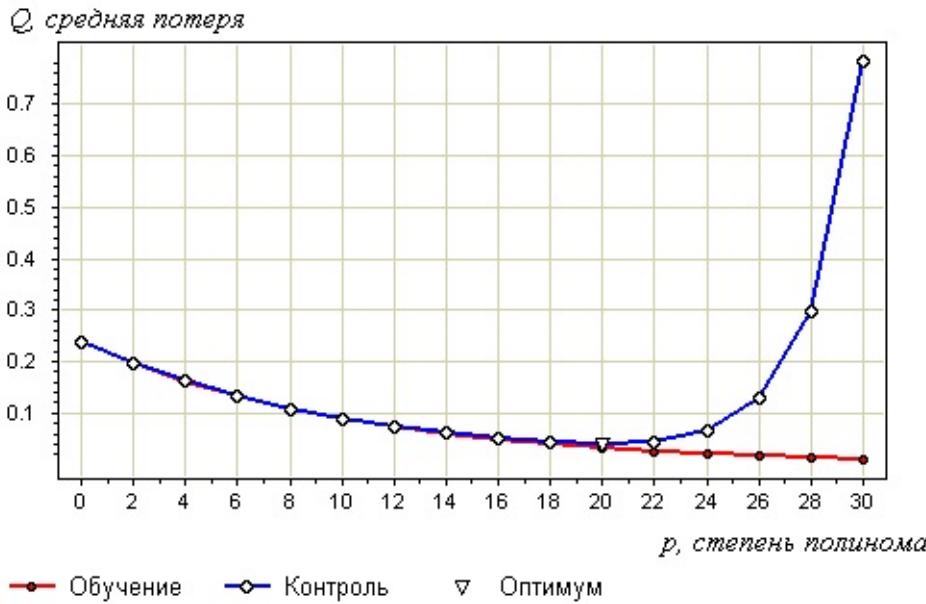
Расширение модели  $M' = A : X' \rightarrow Y'$  всегда приводит к повышению точности аппроксимации на обучающей выборке. Однако повышение точности на обучающей выборке, связанное с увеличением сложности модели, часто не ведет к увеличению обобщающей способности. Более того, обобщающая способность может даже снижаться. Различие между точностью на обучающей выборке и обобщающей способностью при этом возрастает. Данный эффект называется **эффектом переобучения**.

**Переобучение, переподгонка** (overtraining, overfitting) — нежелательное явление, возникающее при решении задач обучения по прецедентам, когда вероятность ошибки обученного алгоритма на объектах [тестовой выборки](#) оказывается существенно выше, чем средняя ошибка на [обучающей выборке](#). Переобучение возникает при использовании избыточно

14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии.

сложных моделей.

*Обучение по прецедентам, или индуктивное обучение*, основано на выявлении общих закономерностей по частным эмпирическим данным.



Пример:

Рассмотрим задачу аппроксимации вещественной функции  $y(x) = \frac{1}{1+25x^2}$  по обучающей выборке из 50 точек  $X^m = \{x_i = 4^{\frac{i-1}{m-1}} - 2\}$ . Это равномерная сетка на отрезке  $[-2, 2]$ .

В качестве модели рассмотрим полиномы заданной степени  $p$ :

$$a(x, w) = w_0 + w_1 x + \dots + w_p x^p$$

В качестве метода обучения возьмём **метод наименьших квадратов**:

$$\sum_{i=1}^m (y(x_i) - a(x_i, w))^2 \rightarrow \min_w$$

Таким образом, функция потерь квадратична:  $\mathcal{L}(a, x) = (y(x) - a(x))^2$ .

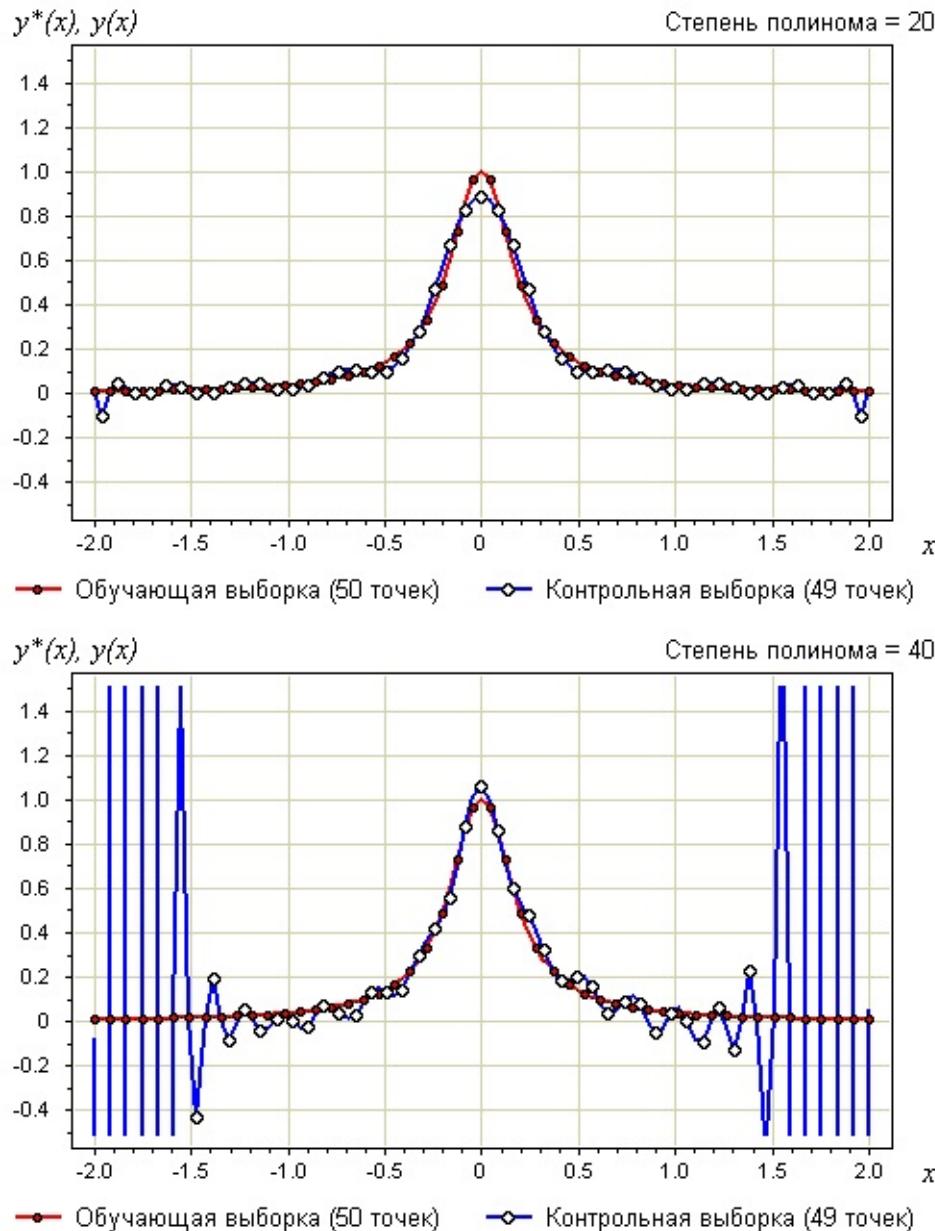
Возьмём контрольную выборку — также равномерную сетку на отрезке  $[-2, 2]$ , узлы которой находятся в точности между узлами первой сетки:  $X^k = \{x'_i = 4^{\frac{i-0.5}{m-1}} - 2\}$ .

Зададимся вопросом: что будет на контрольной выборке при увеличении степени полинома  $p$ ? Степень связана с числом свободных параметров модели, то есть играет роль сложности модели.

Ниже показаны графики самой выборки и аппроксимирующей функции:

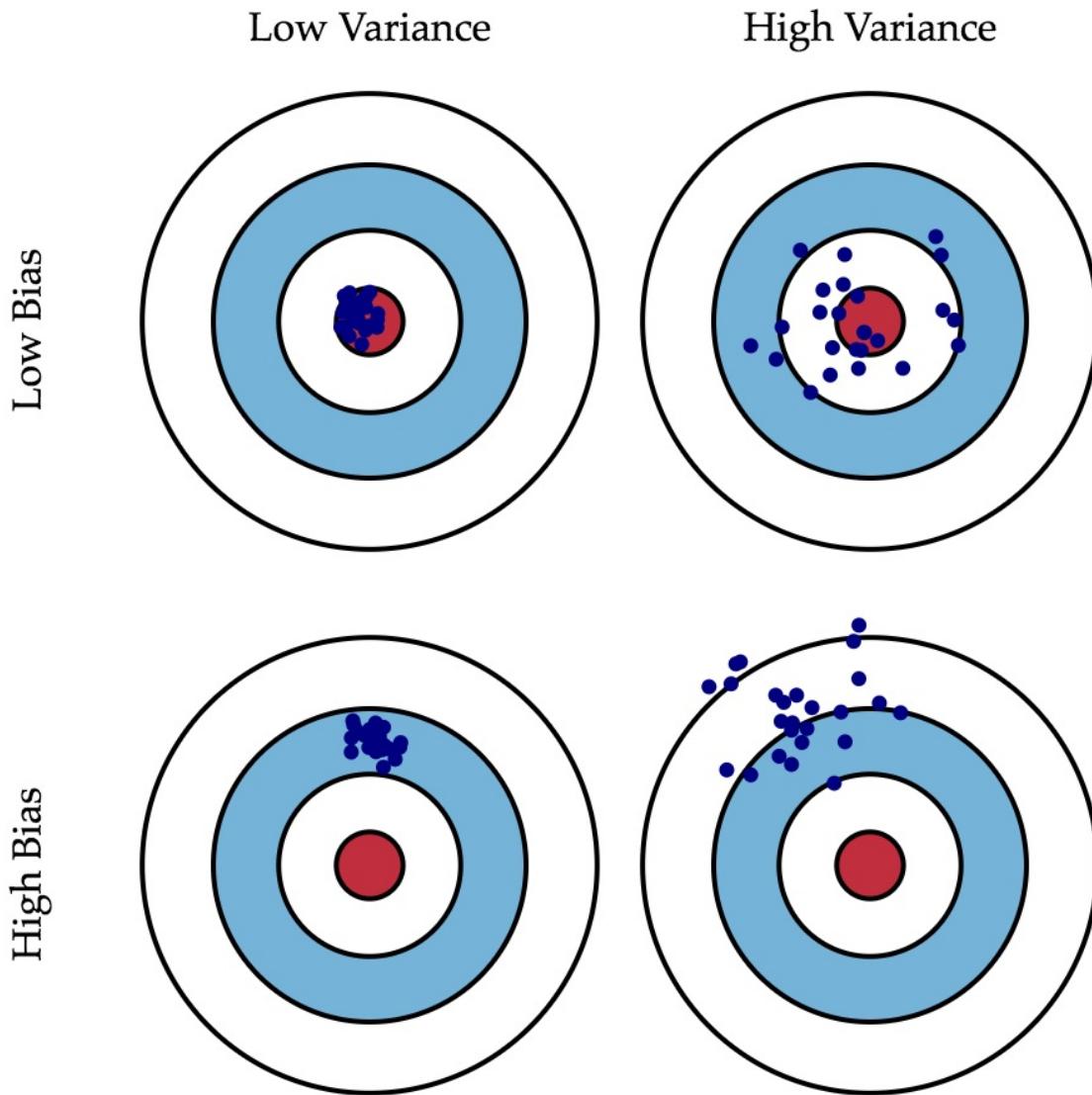
- при  $p = 20$  — оптимальная сложность модели.
- при  $p = 40$  — неустойчивость и переобучение.

14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии.



<http://www.machinelearning.ru/wiki/index.php?title=%D0%9F%D0%B5%D1%80%D0%B5%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%BD%D0%B8%D0%B5>

Еще такая картинка может помочь



Для этого требуется понимать, что мы можем разложить ошибку на bias-variance,

$$\begin{aligned}\text{Err}(x) &= \mathbb{E} \left[ (y - \hat{f}(x))^2 \right] \\ &= \sigma^2 + f^2 + \text{Var}(\hat{f}) + \mathbb{E}[\hat{f}]^2 - 2f\mathbb{E}[\hat{f}] \\ &= (f - \mathbb{E}[\hat{f}])^2 + \text{Var}(\hat{f}) + \sigma^2 \\ &= \text{Bias}(\hat{f})^2 + \text{Var}(\hat{f}) + \sigma^2\end{aligned}$$

- квадрат смещения Bias – средняя ошибка по всевозможным наборам данных;
- дисперсии Var -- вариативность ошибки, то, на сколько ошибка будет отличаться, если обучать модель на разных наборах данных;
- ошибка  $\sigma^2$  является неустранимой

читать подробнее здесь: <https://habr.com/ru/company/ods/blog/323890/>

## Проблема устойчивости решений

14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии.

---

Под **устойчивыми** обучающими алгоритмами понимаются такие, которые дают решение, незначительно изменяющееся при малом изменении обучающей выборки.

Для многомерной линейной регрессии:

(Задача восстановления линейной регрессии - задача обучения по прецедентам при  $Y \rightarrow \mathbb{R}$ , связь задается в виде  $Y = \beta_0 + \beta_1 X_1 + \dots + \beta_n X_n + \varepsilon$ ). При вычислении оценки вектора параметров  $\beta = (\beta_0, \dots, \beta_n)$  в случае многомерной линейной регрессии удобно использовать матрицу  $X$  размера  $m \times (n + 1)$ , которая строится по обучающей выборке.  $j$ -я строка матрицы  $X$  представляет собой вектор значений переменных (признаков) вида  $1, X_1, \dots, X_n$  для объекта  $s_j$  с одной добавленной слева компонентой, содержащей 1 (для  $\beta_0$ ).

Связь  $Y$  с признаками  $X_1, \dots, X_n$  на объектах обучающей выборки может быть описана с помощью матричного уравнения  $y = \beta X^T + \varepsilon$ , где  $\varepsilon = (\varepsilon_1, \dots, \varepsilon_m)$  - вектор ошибок прогнозирования для объектов из выборки. Необходимым условием минимума функционала метода наименьших квадратов (МНК) является выполнение системы из  $n + 1$  уравнений (равенство нулю производных по каждой переменной  $\beta_0, \dots, \beta_n$ ).

В матричной форме эта система может быть записана в виде:  $-2X^T y^T + 2X^T X\beta^T = 0$ . Решение этой системы существует, если  $\det(X^T X)$  не равен 0. При сильной коррелированности одной из переменных  $X_1, \dots, X_n$  на выборке с какой-либо линейной комбинацией других переменных значение  $\det(X^T X)$  оказывается близким к 0. При этом вычисленный вектор оценок  $\beta^T$  может сильно изменяться при относительно небольших чисто случайных изменениях вектора  $y = (y_1, \dots, y_m)$ .

Данное явление называется **мультиколлинеарностью**. Оценивание регрессионных коэффициентов с использованием МНК при наличии мультиколлинеарности оказывается неустойчивым. Также,  $\det(X^T X) = 0$  при  $n + 1 > m$ . Поэтому МНК не может использоваться для оценивания регрессионных коэффициентов, когда число переменных превышает число объектов в обучающей выборке. На практике высокая устойчивость достигается только, когда число объектов в выборках по крайней мере в 3-5 раз превышает число переменных.

## Роль обучающей, валидационной и контрольной выборок при построении распознающей системы

Предположим, что задача прогнозирования решается для некоторого процесса или явления  $F$ . Множество объектов, которые потенциально могут возникать в рамках  $F$ , называется генеральной совокупностью  $\Omega$ .

Поиск алгоритма осуществляется по выборке прецедентов, которая обычно является случайной выборкой объектов из  $\Omega$  с известными значениями  $Y, X_1, \dots, X_n$ . Выборку прецедентов также принято называть **обучающей выборкой**.

Обобщающая способность может оцениваться по случайной выборке объектов из одной и той же генеральной совокупности, соответствующей исследуемому процессу, которую принято называть **контрольной выборкой**. Контрольная выборка не должна содержать объекты из обучающей выборки.

Может быть задано несколько семейств алгоритмов прогнозирования.

- Обучающая выборка используется для выбора алгоритма из каждого семейства
- **Валидационная выборка** используется для выбора того семейства алгоритмов, для которого после обучения распознающая система наилучшая
- Контрольная выборка определяет качество обученной распознающей системы.

Валидационная выборка используется для настройки структуры распознающей системы (какое семейство алгоритмов нам больше подходит, гиперпараметры модели(задаются человеком)).

Контрольная выборка используется для оценки работы обученного классификатора.

Валидационную выборку нельзя использовать в качестве контрольной, потому что валидационная выбирала наилучший классификатор, а потому он покажет заниженную оценку ошибки для валидационной выборки.

14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии.

## Скользящий контроль (кросс-валидация)

**Скользящий контроль** или **кросс-проверка** или **кросс-валидация** (cross-validation, CV) — процедура эмпирического оценивания **обобщающей способности** алгоритмов, **обучаемых по прецедентам**.

Фиксируется некоторое множество разбиений исходной выборки на две подвыборки: *обучающую* и *контрольную*. Для каждого разбиения выполняется настройка *алгоритма* по обучающей подвыборке, затем оценивается его средняя ошибка на объектах контрольной подвыборки. Оценкой скользящего контроля называется средняя по всем разбиениям величина ошибки на контрольных подвыборках.

Если выборка независима, то средняя ошибка скользящего контроля даёт несмещённую оценку вероятности ошибки. Это выгодно отличает её от средней ошибки на обучающей выборке, которая может оказаться смещённой (оптимистически заниженной) оценкой вероятности ошибки, что связано с [явлением переобучения](#).

Скользящий контроль является стандартной методикой тестирования и сравнения алгоритмов классификации, регрессии и прогнозирования.

Рассматривается задача обучения с учителем.

Пусть  $X$  — множество описаний объектов,  $Y$  — множество допустимых ответов

Задана конечная выборка прецедентов  $X^L = (x_i, y_i)_{i=1}^L \subset X \times Y$

Задан **алгоритм обучения** — отображение  $\mu$ , которое произвольной конечной выборке прецедентов  $X^m$  ставит в соответствие функцию (алгоритм)  $a$ :  $X \rightarrow Y$ .

Качество алгоритма  $\mathbf{d}$  оценивается по производительной выборке преодолений  $X^m$  с помощью функции  $m$ .

Качество алгоритма оценивается по произвольной формуле

ектам выборки:

$$Q(a, X^m) = \frac{1}{m} \sum x_i \in X^m \mathcal{L}(a(x_i), y_i),$$

Результаты  $\chi^2$ -теста для  $N$  различий способов определения биомассы

Выборка  $X^L$  разбивается  $N$  различными способами на две непересекающиеся подвыборки:  
 $X^L = X_n^m \cup X_n^k$ , где  $X_n^m$  — обучающая подвыборка длины  $m$ ,  $X_n^k$  — контрольная подвыборка длины  $k = L - m$ .  $n = 1, \dots, N$  — номер разбиения.

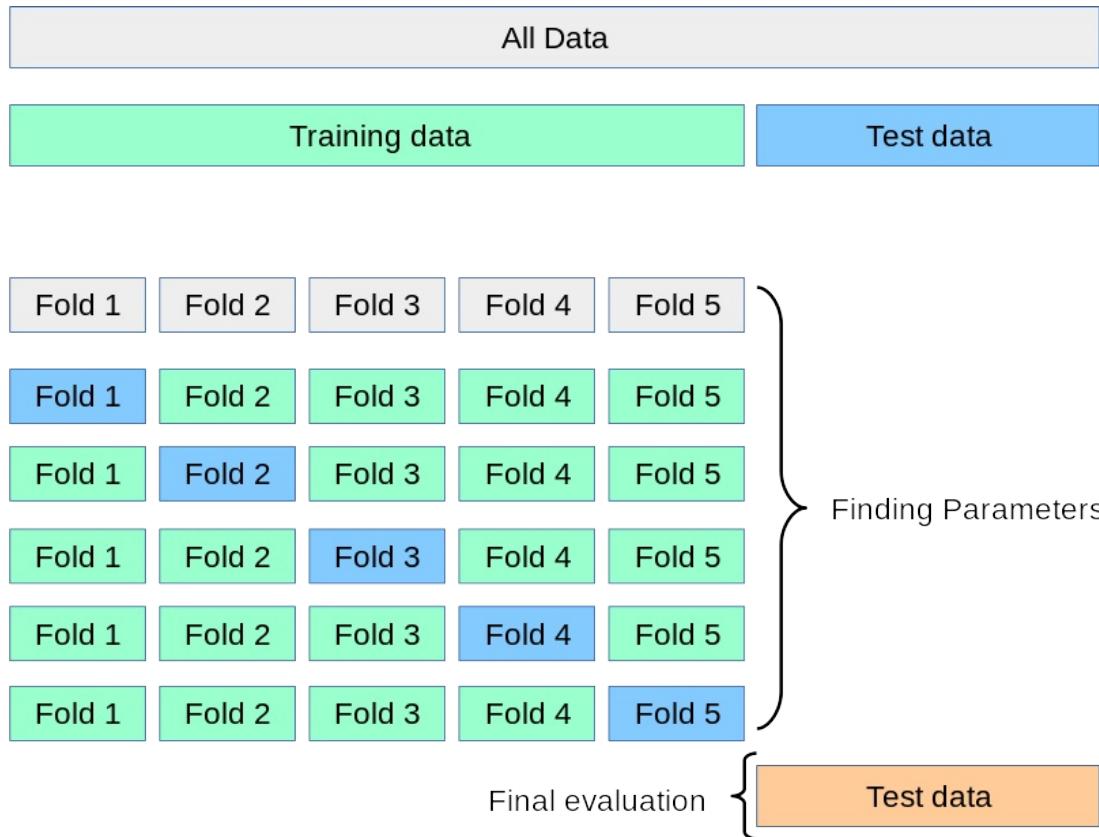
Для каждого разбиения  $n$  строится алгоритм  $a_n = \mu(X_n^m)$  и вычисляется значение функционала качества  $Q_n = Q(a_n, X_n^k)$ . Среднее арифметическое значений  $Q_n$  по всем разбиениям называется оценкой скользящего контроля:

$$CV(\mu, X^L) = \frac{1}{N} \sum_{n=1}^N Q(\mu(X_n^m), X_n^k)$$

Различные варианты скользящего контроля отличаются видами функционала качества и способами разбиения выборки.

k-fold cross validation -- k = 5:

14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии.



## Регуляризация на примере линейной регрессии

Задача линейной регрессии: <https://habr.com/en/company/ods/blog/322076/>

Давайте ограничим пространство гипотез только линейными функциями от  $m + 1$  аргумента, будем считать, что нулевой признак для всех объектов равен единице  $x_0 = 1$

$$\begin{aligned} \forall h \in \mathcal{H}, h(x) &= w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_m x_m \\ &= \sum_{i=0}^m w_i x_i \\ &= x^T w \end{aligned}$$

Эмпирический риск (функция стоимости) принимает форму среднеквадратичной ошибки:

$$\begin{aligned} \mathcal{L}(X, y, w) &= \frac{1}{2n} \sum_{i=1}^n (y_i - x_i^T w)^2 \\ &= \frac{1}{2n} \|y - Xw\|_2^2 \\ &= \frac{1}{2n} (y - Xw)^T (y - Xw) \end{aligned}$$

строки матрицы  $X$  — это признаковые описания наблюдаемых объектов. Один из алгоритмов обучения  $M$  такой модели — это метод наименьших квадратов. Вычислим производную функции стоимости:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial}{\partial w} \frac{1}{2n} (y^T y - 2y^T Xw + w^T X^T Xw) \\ &= \frac{1}{2n} (-2X^T y + 2X^T Xw) \end{aligned}$$

приравняем к нулю и найдем решение в явном виде:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} = 0 &\Leftrightarrow \frac{1}{2n} (-2X^T y + 2X^T Xw) = 0 \\ &\Leftrightarrow -X^T y + X^T Xw = 0 \\ &\Leftrightarrow X^T Xw = X^T y \\ &\Leftrightarrow w = (X^T X)^{-1} X^T y \end{aligned}$$

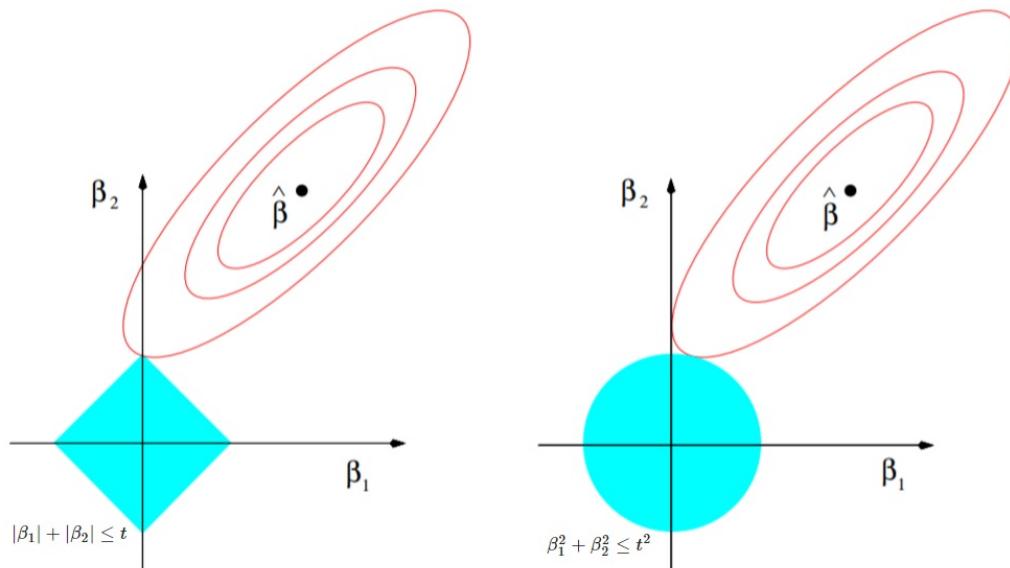
14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии.

Сложный функционал содержит регуляризацию, которая обычно представлена в виде дополнительного регуляризационного слагаемого:  $\min L(X, y, w) + \lambda F(w)$ , где  $L(X, y, w)$  — функция потерь,  $F(w)$  — регуляризационная функция,  $\lambda$  — параметр, задающий степень влияния регуляризации. Регуляризация предназначена для регулирования сложности модели и ее целью является упрощение модели. Это, в частности, помогает бороться с переобучением и позволяет увеличить обобщающую способность модели.

Типичные примеры регуляризационных функций:

1.  $L_1 = \sum |w|$  Известная как LASSO-регуляризация (Least Absolute Shrinkage and Selection Operator), и, как несложно догадаться из названия, она позволяет снижать размерность коэффициентов, обращая некоторые из них в нули. И это весьма удобно, когда исходные данные сильно коррелированы.
2.  $L_2 = \sum w^2$  Иногда ее называют ridge-регуляризацией (гребневая), и она позволяет минимизировать значения коэффициентов модели, а заодно сделать ее более стабильной к незначительным изменениям исходных данных. А еще она хорошо дифференцируется, а значит модель можно рассчитать аналитически.
3.  $L_{EN} = \alpha L_1 + (1-\alpha)L_2$  Совмещая LASSO и ridge, получаем ElasticNet, которая объединяет два мира со всеми их плюсами и минусами.

$w, \beta$  — это одно и тоже (параметры модели)



Продублируем наглядный пример из статьи о [вариациях регрессии](#). Рассмотрим для простоты двумерное пространство независимых переменных. В случае лasso регрессии ограничение на коэффициенты представляет собой ромб ( $|\beta_1| + |\beta_2| \leq t$ ), в случае гребневой регрессии — круг ( $\beta_1^2 + \beta_2^2 \leq t^2$ ). Необходимо минимизировать функцию ошибки, но при этом соблюсти ограничения на коэффициенты. С геометрической точки зрения задача состоит в том, чтобы найти точку касания линии, отражающей функцию ошибки с фигурой, отражающей ограничения на  $\beta$ . Из рисунка интуитивно понятно, что в случае лasso регрессии эта точка с большой вероятностью будет находиться на углах ромба, то есть лежать на оси, тогда как в случае гребневой регрессии такое происходит очень редко. Если точка пересечения лежит на оси, один из коэффициентов будет равен нулю, а значит, значение соответствующей независимой переменной не будет учитываться.

14. Средние и эмпирические операционные характеристики стратегий распознавания (классификаторов, регрессий). Проблема переобучения. Проблема устойчивости решений. Роль обучающей, валидационной и контрольной выборок при построении распознающей системы. Скользящий контроль (кросс-валидация). Регуляризация на примере линейной регрессии.

---

15. Ансамбли классификаторов. Основные этапы работы типичного базового классификатора, возможность коррекции на разных этапах. Бэггинг и случайные подпространства. Бустинг. Случайный лес как композиция основных подходов к построению ансамбля.

**15. Ансамбли классификаторов. Основные этапы работы типичного базового классификатора, возможность коррекции на разных этапах. Бэггинг и случайные подпространства. Бустинг. Случайный лес как композиция основных подходов к построению ансамбля.**

<https://dyakonov.org/2019/04/19/%D0%B0%D0%BD%D1%81%D0%B0%D0%BC%D0%B1%D0%BB%D0%B8%D0%B2-%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%BC-%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B8/>

[http://neerc.ifmo.ru/wiki/index.php?title=%D0%92%D0%B4%D1%8B\\_%D0%B0%D0%BD%D1%81%D0%B0%D0%BC%D0%B1%D0%BB%D0%B5%D0%B9](http://neerc.ifmo.ru/wiki/index.php?title=%D0%92%D0%B4%D1%8B_%D0%B0%D0%BD%D1%81%D0%B0%D0%BC%D0%B1%D0%BB%D0%B5%D0%B9)

<http://www.machinelearning.ru/wiki/images/5/56/Guschin2015Stacking.pdf>

## Ансамбли классификаторов

**Ансамблем (Ensemble, Multiple Classifier System)** называется алгоритм, который состоит из нескольких алгоритмов машинного обучения, а процесс построения ансамбля называется ансамблированием (ensemble learning). Простейший пример ансамбля в регрессии – усреднение нескольких алгоритмов:

$$a(x) = \frac{1}{n} (b_1(x) + \dots + b_n(x)) \quad (1)$$

Алгоритмы из которых состоит ансамбль (в (1) –  $b_t$ ) называются **базовыми алгоритмами (base learners)**.

Если рассматривать значения базовых алгоритмов на объекте как независимые случайные величины с одинаковым матожиданием и одинаковой конечной дисперсией, то понятно, что случайная величина (1) имеет такое же матожидание, но меньшую дисперсию:

$$\xi = \frac{1}{n}(\xi_1 + \dots + \xi_n)$$

$$\mathbf{E} \xi = \frac{1}{n} (\mathbf{E} \xi_1 + \dots + \mathbf{E} \xi_n) = \mathbf{E} \xi_i$$

$$\mathbf{D}\xi = \frac{1}{n^2} (\mathbf{D}\xi_1 + \dots + \mathbf{D}\xi_n) = \frac{\mathbf{D}\xi_i}{n}$$

**Замечание.** Требование равенства матожиданий ответов базовых алгоритмов вполне естественное: если мы берём алгоритмы из несмешённой модели, то их матожидания не только совпадают, но и равны значению истинной метки.

Ансамбль алгоритмов (методов) — метод, который использует несколько обучающих алгоритмов с целью получения лучшей эффективности прогнозирования, чем можно было бы получить от каждого обучающего алгоритма по отдельности.

В задачах классификации простейший пример ансамбля – **комитет большинства**:

15. Ансамбли классификаторов. Основные этапы работы типичного базового классификатора, возможность коррекции на разных этапах. Бэггинг и случайные подпространства. Бустинг. Случайный лес как композиция основных подходов к построению ансамбля.

$$a(x) = \text{mode}(b_1(x), \dots, b_n(x)), \quad (2)$$

где **mode** – мода (значение, которое встречается чаще других среди аргументов функции).

Большинство приёмов в прикладном ансамблировании направлено на то, чтобы ансамбль был «достаточно разнообразным»\*\*, тогда ошибки отдельных алгоритмов на отдельных объектах будут компенсироваться корректной работой других алгоритмов. По сути, при построении ансамбля:

- повышают качество базовых алгоритмов,
- повышают разнообразие (diversity) базовых алгоритмов.

## Основные этапы работы типичного базового классификатора, возможность коррекции на разных этапах

В целом я хз, че тут говорить, но вот например я сгенерировал такой текст:

По сути внутри базового классификатора лежит какая-то из моделей -- линейная регрессия, решающее дерево, например.

В случае бэггинга каждому базовому классификатору соответствует какая-то выборка из обучающего датасета, на ней каждый классификатор обучается и подстраивает свои параметры. Общая композиция из этих "недообученных" классификаторов затем одним из методов принимает решение, к какому классу принадлежит тот или иной объект.

В случае бустинга итоговый результат представляется в виде взвешенной суммы из корректирующих друг друга классификаторов.

В то время как бустинг алгоритмически не ограничен, большинство алгоритмов бустинга состоит из итеративного обучения слабых классификаторов с целью сборки их в сильный классификатор. Когда они добавляются, им обычно приписываются некоторым образом веса, которые, обычно, связаны с точностью обучения. После того, как слабый классификатор добавлен, веса пересчитываются, что известно как [«пересчёт весовых коэффициентов»](#)[en]. Неверно классифицированные входные данные получают больший вес, а правильно классифицированные экземпляры теряют вес[nb 1]. Тем самым последующее слабое обучение фокусируется больше на примерах, где предыдущие слабые обучения дали ошибочную классификацию.

## Бэггинг и случайные подпространства

Пусть имеется выборка  $X$  размера  $N$ . Количество классификаторов  $M$

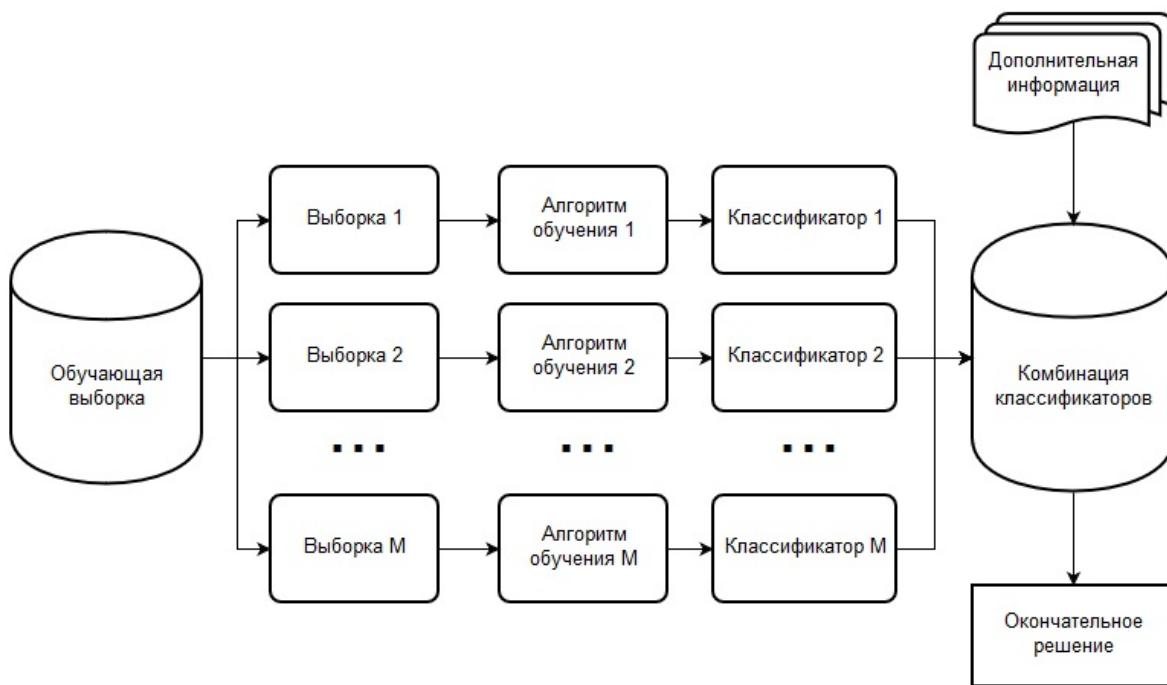
Алгоритм использует метод бутстрэпа (англ. *bootstrap*):

Из всего множества объектов равновероятно выберем  $N$  объектов с возвращением. Это значит, что после выбора каждого из объектов мы будем возвращать его в множество для выбора. Отметим, что из-за возвращения некоторые объекты могут повторяться в выбранном множестве.

Обозначим новую выборку через  $X_1$   
. Повторяя процедуру  $M$  раз, сгенерируем  $M$  подвыборок  $X_1 \dots X_M$   
. Теперь мы имеем достаточно большое число выборок и можем оценивать различные статистики исходного распределения.

Шаги алгоритма бэггинг:

- Генерируется с помощью бутстрэпа  $M$  выборок размера  $N$  для каждого классификатора.
- Производится независимое обучения каждого элементарного классификатора (каждого алгоритма, определенного на своем подпространстве).
- Производится классификация основной выборки на каждом из подпространств (также независимо).
- Принимается окончательное решение о принадлежности объекта одному из классов. Это можно сделать несколькими разными способами, подробнее описано ниже.



В методе случайных подпространств (random subspace method, RSM) базовые алгоритмы обучаются на различных подмножествах признакового описания, которые также выделяются случайным образом. Этот метод предпочтителен в задачах с большим числом признаков и относительно небольшим числом объектов, а также при наличии избыточных неинформативных признаков. В этих случаях алгоритмы, построенные по части признакового описания, могут обладать лучшей обобщающей способностью по сравнению с алгоритмами, построенными по всем признакам. Широко известным алгоритмом, использующим одновременно метод случайных подпространств и бэггинг, является случайный лес.

Разбиение объектов в вершине случайного леса ищется среди случайного подмножества признаков, а обучение каждого дерева в композиции происходит на выборке, полученной с помощью операции бутстрапа.

## Бустинг

[https://neerc.ifmo.ru/wiki/index.php?title=%D0%91%D1%83%D1%81%D1%82%D0%B8%D0%BD%D0%B3,\\_AdaBoost](https://neerc.ifmo.ru/wiki/index.php?title=%D0%91%D1%83%D1%81%D1%82%D0%B8%D0%BD%D0%B3,_AdaBoost)

Бустинг основан на вопросе, поднятом Кернсом и Вэлиантом (1988, 1989)[3][4]: «Может ли набор слабых обучающих алгоритмов создать сильный обучающий алгоритм?». Слабый обучающий алгоритм определяется как **классификатор**, который слабо коррелирует с правильной классификацией (может помечать примеры лучше, чем случайное угадывание). В отличие от слабого алгоритма, сильный обучающий алгоритм является классификатором, хорошо коррелирующим с верной классификацией.

Положительный ответ Роберта Шапири в статье 1990 года[5] на вопрос Кернса и Вэлианта имел большое значение для теории машинного обучения и **статистики**, и привёл к созданию широкого спектра алгоритмов бустинга[6].

**Гипотеза о бустинге** относилась к процессу настройки алгоритма слабого обучения для получения строгого обучения. Неформально, спрашивается, вытекает ли из существования эффективного алгоритма обучения, выходом которого служит гипотеза, эффективность которой лишь слегка лучше случайного гадания (то есть слабое обучение), существование эффективного алгоритма, который даёт гипотезу произвольной точности (то есть сильное обучение)[3]. Алгоритмы, которые получают быстро такую гипотезу, становятся известны просто как «бустинг».

**Бустинг.** Пошаговое наращивание ансамбля алгоритмов. Алгоритм, который присоединяется к ансамблю на шаге  $k$  обучается по выборке, которая формируется из объектов исходной обучающей выборки.

15. Ансамбли классификаторов. Основные этапы работы типичного базового классификатора, возможность коррекции на разных этапах. Бэггинг и случайные подпространства. Бустинг. Случайный лес как композиция основных подходов к построению ансамбля.

Классификаторы ансамбля строятся последовательно и на каждой итерации происходит коррекция (перевзвешивание) наблюдений обучающей выборки (на первой итерации веса всех наблюдений равны). Коррекция осуществляется таким образом, чтобы соответствующий классификатор делал меньше ошибок на тех наблюдениях, на которых часто делали ошибки классификаторы, построенные на предыдущих итерациях алгоритма. Кроме того, каждому классификатору приписывается некоторый вес исходя из количества допущенных им ошибок.

Например, один из первых алгоритмов бустинга Boost1 использовал ансамбль из 3-х моделей, первая из которых обучалась на всем наборе данных, вторая – на выборке примеров, в половине из которых первая дала правильные ответы, а третья – на примерах, где «ответы» первых двух разошлись. Т.е. происходит последовательная обработка примеров цепочкой классификаторов, причем так, что задача для каждого последующего становится труднее. Результат определяется путем простого голосования: пример относится к тому классу, который выдан большинством моделей ансамбля.

Развитием данного подхода явилась разработка более совершенного семейства алгоритмов бустинга AdaBoost, который может использовать произвольное число классификаторов и производить обучение на одном наборе примеров, поочередно применяя их на различных шагах.

## Случайный лес как композиция основных подходов к построению ансамбля.

<https://dyakonov.org/2016/11/14/%d1%81%d0%bb%d1%83%d1%87%d0%b0%d0%b9%d0%bd%d1%8b%d0%b9-%d0%bb%d0%b5%d1%81-random-forest/>

**Решающие деревья** – классификация объекта с помощью ответов на иерархически организованную систему вопросов. Вопрос, задаваемый на последующем иерархическом уровне, зависит от ответа, полученного на предыдущем уровне.



**RF (random forest)** — это множество решающих деревьев (ансамбль). В задаче регрессии их ответы усредняются, в задаче классификации принимается решение голосованием по большинству. Все деревья строятся независимо по следующей схеме:

- Выбирается подвыборка обучающей выборки размера samplesize (м.б. с возвращением) – по ней строится дерево (для каждого дерева — своя подвыборка). (**бэггинг**)
- Для построения каждого расщепления в дереве просматриваем max\_features случайных признаков (для каждого нового расщепления — свои случайные признаки). (**случайные подпространства**)
- Выбираем наилучшие признак и расщепление по нему (по заранее заданному критерию). Дерево строится, как правило, до исчерпания выборки (пока в листьях не останутся представители только одного класса), но в

современных реализациях есть параметры, которые ограничивают высоту дерева, число объектов в листьях и число объектов в подвыборке, при котором проводится расщепление.

**Алгоритм случайногo лесa** сочетает в себе две идеи: метод бэггинга и метод случайных подпространств. Для полученных данных алгоритм создает множество деревьев принятия решений и потом усредняет результат их предсказаний (бэггинг). Случайность: если создать много одинаковых деревьев, то результат их усреднения будет обладать точностью одного дерева. На практике: из всего набора входных данных случайнym образом выбирается некоторое количество столбцов и строк и строится первое дерево принятия решений. Такая процедура повторяется множество раз, на выходе получается множество деревьев => результат – класс, к которому отнесли большинство деревьев.

16. Задача кластеризации как фундаментальная задача интеллектуального анализа данных, сопоставление с операцией группирования и задачей классификации. Различные постановки: разбиение, стохастическая, нечёткая, иерархическая, упорядочивание, однокластерная (последовательная). Примеры методов кластеризации для разных постановок.

---

## 16. Задача кластеризации как фундаментальная задача интеллектуального анализа данных, сопоставление с операцией группирования и задачей классификации. Различные постановки: разбиение, стохастическая, нечёткая, иерархическая, упорядочивание, однокластерная (последовательная). Примеры методов кластеризации для разных постановок.

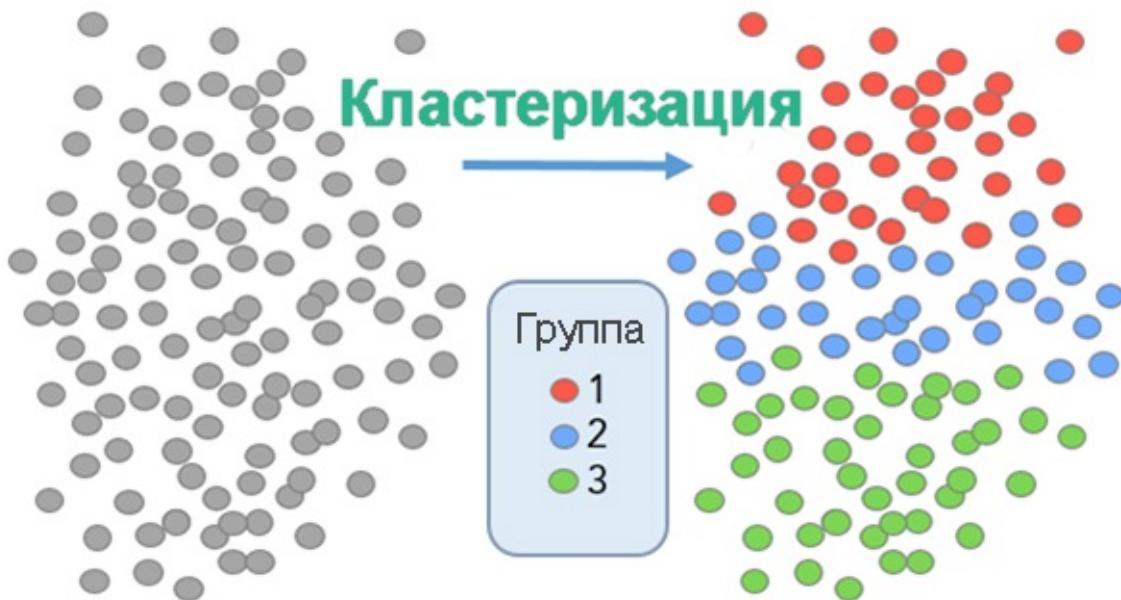
Предварительно отмечу, что этой темы не было у нас в курсе вообще!!!

### Задача кластеризации как фундаментальная задача интеллектуального анализа данных, сопоставление с операцией группирования и задачей классификации

<http://www.machinelearning.ru/wiki/index.php?title=%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F>

<https://habr.com/ru/post/101338/>

**Кластерный анализ** (Data clustering) — задача разбиения заданной **выборки** объектов (ситуаций) на непересекающиеся подмножества, называемые **кластерами**, так, чтобы каждый кластер состоял из схожих объектов, а объекты разных кластеров существенно отличались.



Задача кластеризации относится к широкому классу задач **обучения без учителя**.

16. Задача кластеризации как фундаментальная задача интеллектуального анализа данных, сопоставление с операцией группирования и задачей классификации. Различные постановки: разбиение, стохастическая, нечёткая, иерархическая, упорядочивание, однокластерная (последовательная). Примеры методов кластеризации для разных постановок.

Пусть  $X$  — множество объектов,  $Y$  — множество номеров (имён, меток) кластеров. Задана функция расстояния между объектами  $\rho(x, x')$ . Имеется конечная обучающая выборка объектов  $X^m = \{x_1, \dots, x_m\} \subset X$ . Требуется разбить выборку на непересекающиеся подмножества, называемые *кластерами*, так, чтобы каждый кластер состоял из объектов, близких по метрике  $\rho$ , а объекты разных кластеров существенно отличались. При этом каждому объекту  $x_i \in X^m$  приписывается номер кластера  $y_i$ .

Примеры функций расстояния:

- *Евклидово расстояние* Наиболее распространенная функция расстояния. Представляет собой геометрическим

$$\rho(x, x') = \sqrt{\sum_i^n (x_i - x'_i)^2}$$

расстоянием в многомерном пространстве:

- *Квадрат евклидова расстояния* Применяется для придания большего веса более отдаленным друг от друга

$$\rho(x, x') = \sum_i^n (x_i - x'_i)^2$$

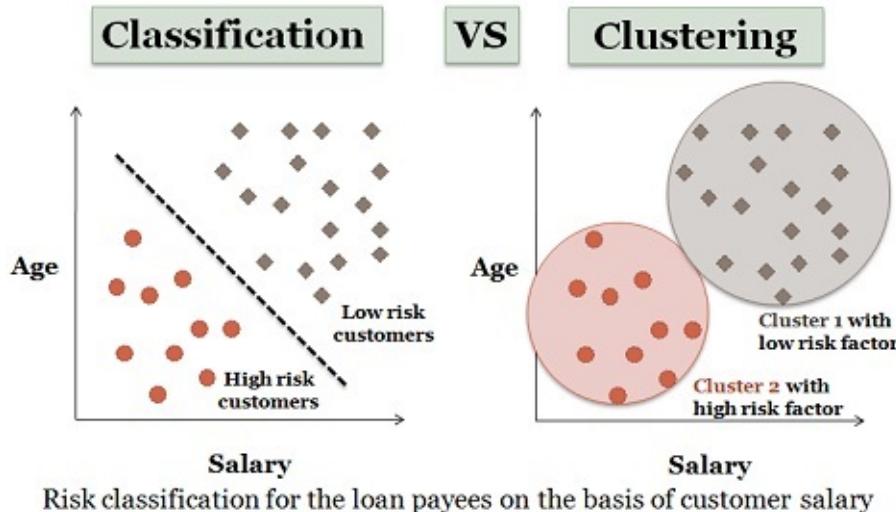
объектам. Это расстояние вычисляется следующим образом:

- *Расстояние городских кварталов (манхэттенское расстояние)* Это расстояние является средним разностей по координатам. В большинстве случаев эта мера расстояния приводит к таким же результатам, как и для обычного расстояния Евклида. Однако для этой меры влияние отдельных больших разностей (выбросов) уменьшается (т.к. они

$$\rho(x, x') = \sum_i^n |x_i - x'_i|$$

не возводятся в квадрат). Формула для расчета манхэттенского расстояния:

Алгоритм кластеризации — это функция  $a: X \rightarrow Y$ , которая любому объекту  $x \in X$  ставит в соответствие номер кластера  $y \in Y$ . Множество  $Y$  в некоторых случаях известно заранее, однако чаще ставится задача определить оптимальное число кластеров, с точки зрения того или иного критерия качества кластеризации.



Кластеризация (*обучение без учителя*) отличается от классификации (*обучения с учителем*) тем, что метки исходных объектов  $y_i$  изначально не заданы, и даже может быть неизвестно само множество  $Y$ .

Решение задачи кластеризации принципиально неоднозначно, и тому есть несколько причин:

- Не существует однозначно наилучшего критерия качества кластеризации. Известен целый ряд эвристических критериев, а также ряд алгоритмов, не имеющих чётко выраженного критерия, но осуществляющих достаточно разумную кластеризацию «по построению». Все они могут давать разные результаты.
- Число кластеров, как правило, неизвестно заранее и устанавливается в соответствии с некоторым субъективным

16. Задача кластеризации как фундаментальная задача интеллектуального анализа данных, сопоставление с операцией группирования и задачей классификации. Различные постановки: разбиение, стохастическая, нечёткая, иерархическая, упорядочивание, однокластерная (последовательная). Примеры методов кластеризации для разных постановок.

---

- **Признаковое описание** объектов. Каждый объект описывается набором своих характеристик, называемых *признаками*. Признаки могут быть числовыми, порядковыми и категориальными.
- **Матрица расстояний** между объектами. Каждый объект описывается расстояниями до всех остальных объектов обучающей выборки.

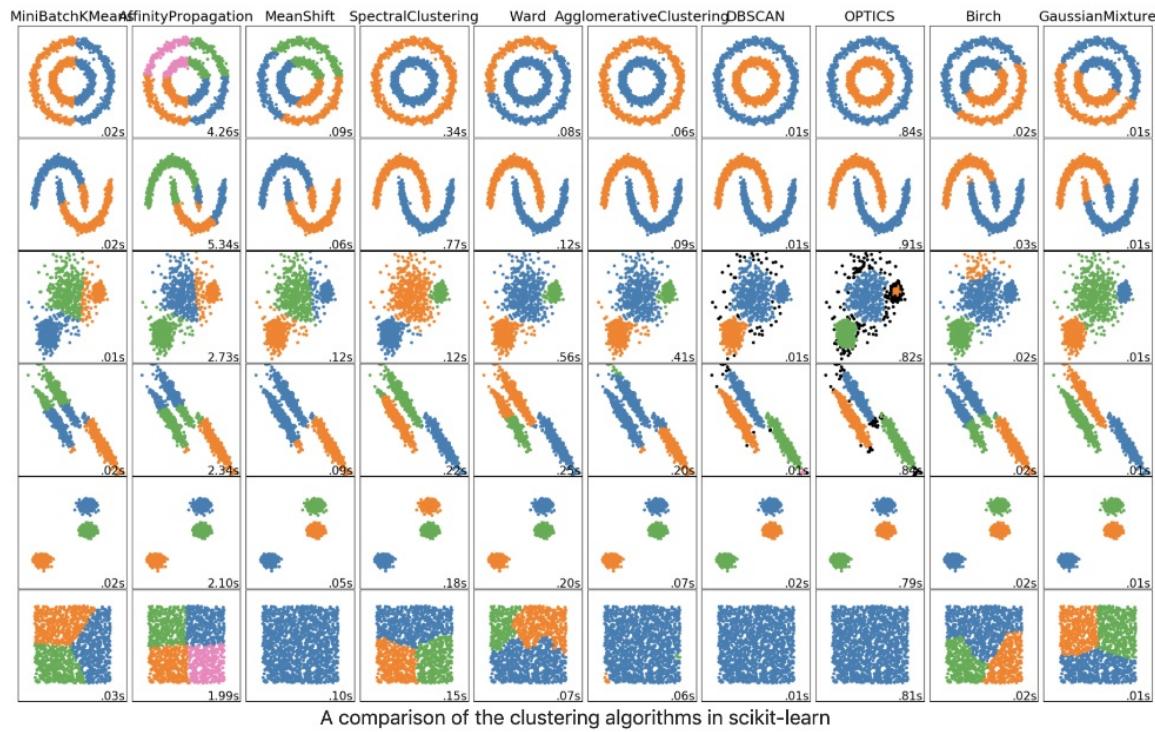
#### Цели кластеризации

- **Понимание данных** путём выявления кластерной структуры. Разбиение выборки на группы схожих объектов позволяет упростить дальнейшую обработку данных и принятия решений, применяя к каждому кластеру свой метод анализа (стратегия «[разделяй и властвуй](#)»).
- **Сжатие данных**. Если исходная выборка избыточно большая, то можно сократить её, оставив по одному наиболее типичному представителю от каждого кластера.
- **Обнаружение новизны** (novelty detection). Выделяются нетипичные объекты, которые не удается присоединить ни к одному из кластеров.

### **Различные постановки: разбиение, стохастическая, нечёткая, иерархическая, упорядочивание, однокластерная (последовательная). Примеры методов кластеризации для разных постановок.**

- Разбиение -- непосредственно разбиение объектов на конкретные кластеры, алгоритмы: k-means, алгоритмы иерархической кластеризации. (наверное)
- Стохастическая -- хз что это, не могу нигде найти. Предположу, что выдвигается гипотеза о том, что каждый кластер представляет из себя некоторое распределение, а значит, все разделяемое пространство объектов -- смесь таких распределений. Значит, задача сводится к разделению смеси распределений. Алгоритм: Gaussian Mixture Models.
- Нечеткая кластеризация -- Нечёткую кластеризацию от просто кластеризации отличает то, что объекты, которые подвергаются кластеризации, относятся к конкретному кластеру с некой принадлежностью, а не однозначно, как это бывает при обычной кластеризации. Например, при нечёткой кластеризации объект А относится к кластеру K1 с вероятностью 0.9, к кластеру K2 — с вероятностью 0.04 и к кластеру K3 — с вероятностью 0.06. При обычной же (чёткой) кластеризации объект А был бы отнесен к кластеру K1. Алгоритм: c-means
- Иерархические алгоритмы (также называемые алгоритмами таксономии) строят не одно разбиение выборки на непересекающиеся кластеры, а систему вложенных разбиений. Т.о. на выходе мы получаем дерево кластеров, корнем которого является вся выборка, а листьями — наиболее мелкие кластера. Нисходящие алгоритмы работают по принципу «сверху-вниз»: в начале все объекты помещаются в один кластер, который затем разбивается на все более мелкие кластеры. Более распространены восходящие алгоритмы, которые в начале работы помещают каждый объект в отдельный кластер, а затем объединяют кластеры во все более крупные, пока все объекты выборки не будут содержаться в одном кластере. Таким образом строится система вложенных разбиений.
- Упорядочивание -- не знаю, вероятно, выдача меток объектам, при этом кластеры не образуют никаких вложенных структур. Алгоритмы: K-Means
- Однокластерная -- (выдуманное определение) в данной постановке размечаются объекты, принадлежащие к одному и тому же кластеру, остальные считаются не принадлежащими к нему, каждый кластер ищется последовательно, одним из примеров алгоритмов, который использует данные принципы, алгоритм плотностной кластеризации DBScan

16. Задача кластеризации как фундаментальная задача интеллектуального анализа данных, сопоставление с операцией группирования и задачей классификации. Различные постановки: разбиение, стохастическая, нечёткая, иерархическая, упорядочивание, однокластерная (последовательная). Примеры методов кластеризации для разных постановок.



A comparison of the clustering algorithms in scikit-learn

На картинке демонстрация результатов работы различных алгоритмов кластеризации. Как видно, для разных данных различные алгоритмы по-разному работают.

## 17. Дискреционное управление доступом. Модели HRU и Take-Grant. Задача проверки безопасности системы защиты от НСД.

### Модели дискреционного доступа

Политика дискреционного доступа охватывает самую многочисленную совокупность моделей разграничения доступа, реализованных в большинстве защищенных КС, и исторически является первой, проработанной в теоретическом и практическом плане.

Модели дискреционного доступа непосредственно основываются и развиваются субъектно-объектную модель КС как совокупность некоторых множеств взаимодействующих элементов (субъектов, объектов и т. д.).

Множество (область) безопасных доступов в моделях дискреционного доступа определяется дискретным набором троек "Пользователь (субъект)-поток (операция)-объект".

Права доступа предоставляются («прописываются» в специальных информационных объектах-структурах, ассоциированных с монитором безопасности), отдельно каждому пользователю к тем объектам, которые ему необходимы для работы в КС.

При запросе субъекта на доступ к объекту монитор безопасности, обращаясь к ассоциированным с ним информационным объектам, в которых «прописана» политика разграничения доступа, определяет «легальность» запрашиваемого доступа и разрешает/отвергает доступ

**Механизмы реализации дискреционного разграничения доступа** Различаются:

- в зависимости от принципов и механизмов программно-информационной структуры объекта(объектов), ассоциированных с монитором безопасности, в которых хранятся «прописанные» права доступа (тройки доступа)
- в зависимости от принципа управления правами доступа, т.е. в зависимости от того – кто и как заполняет/изменяет ячейки матрицы доступа (принудительный и добровольный принцип управления доступом).

Проблема безопасности в КС рассматривается с точки зрения анализа и исследования условий, правил, порядка и т. п. разрешений запросов на доступ, при которых система, изначально находясь в безопасном состоянии, за конечное число переходов перейдет также в безопасное состояние.

**модель на основе матрицы доступа:** - матрица определяющая права субъекта по отношению к объекту

- централизованная или децентрализованная:
  - при централизованной: матрица доступа создается как отдельный самостоятельный объект с особым порядком размещения и доступа к нему, она может достигать очень больших величин, и, кроме того, подвержено динамическому изменению. В большинстве систем при централизованном подходе строки матрицы доступа характеризуют не субъектов, а непосредственно самих пользователей и их группы, зарегистрированные для работы в системе.
  - при децентрализованной: матрица доступа как отдельный объект не создается, а представляется или так называемыми "списками доступа", распределенными по объектам системы, или так называемыми "списками возможностей", распределенными по субъектам доступа.
- с понятием владения объектами (тогда владелец сам задаёт права доступа для других) или без понятия владения (тогда админ всем управляет)

### Модели HRU (Харрисона-Руззо-Ульмана)

одна из [формальных моделей управления доступом](#) субъектов (пользователей) к объектам, реализованная с помощью [матрицы доступов](#) -- матрица с полным описанием пользовательских прав к файлам. Изменения в эту матрицу вводятся с помощью специальных команд.

Введем некоторые обозначения:  $S$  - множество субъектов;  $O$  - множество объектов;  $R = (r_1, r_2, \dots, r_n)$  - множество прав доступа; Строки матрицы доступа  $M$  соответствуют субъектам, столбцы объектам. Текущее состояние системы  $Q$  однозначно записывается тройкой  $Q=(S,O,M)$

Для того, чтобы был возможен переход из  $Q=(S,O,M)$  в  $Q'=(S',O',M')$  нужно ввести некоторые элементарные операции. Для этой цели в данной модели существует шесть операторов op:

- *Enter r into M(s,o)* - ввести право r в ячейку  $M(s,o)$ ;
- *Delete r from M(s,o)* - удалить право r из ячейки  $M(s,o)$ ;
- *Create subject s* - создать субъект s (т.е. новую строку матрицы M);
- *Create object o* - создать объект o (т.е. новый столбец матрицы M);
- *Destroy subject s* - уничтожить субъект s;
- *Destroy object o* - уничтожить объект o;

Условия выполнения и новое состояние системы записаны в виде таблицы

Оператор	Условия выполнения	Новое состояние системы
$op = enter r into M[s, o]$	$s \in S, o \in O$	$S' = S, O' = O, A'[s, o] = A[s, o] \cup \{r\},$ $if (s', o') \neq (s, o) \Rightarrow A'[s', o'] = A[s', o']$
$op = delete r from M[s, o]$	$s \in S, o \in O$	$S' = S, O' = O, A'[s, o] = A[s, o] \setminus \{r\},$ $if (s', o') \neq (s, o) \Rightarrow A'[s', o'] = A[s', o']$
$op = create subject s'$	$s' \notin S$	$S' = S \cup \{s'\}, O' = O,$ $if (s, o) \in S \times O \Rightarrow A'[s, o] = A[s, o],$ $if o \in O' \Rightarrow A'[s', o] = \emptyset$
$op = create object o'$	$o' \notin O$	$S' = S, O' = O \cup \{o'\},$ $if (s, o) \in S \times O \Rightarrow A'[s, o] = A[s, o],$ $if s \in S' \Rightarrow A'[s, o'] = \emptyset$
$op = destroy subject s$	$s' \in S$	$S' = S \setminus \{s'\}, O' = O,$ $if (s, o) \in S' \times O' \Rightarrow A'[s, o] = A[s, o],$
$op = destroy object o$	$o' \in O$	$S' = S, O' = O \setminus \{o'\},$ $if (s, o) \in S' \times O' \Rightarrow A'[s, o] = A[s, o],$

**Системы в модели HRU** Любая система в модели HRU характеризуется матрицей доступа  $M$ , конечным количеством прав  $R = (r_1, r_2, \dots, r_n)$ , объектов  $O = (o_1, o_2, \dots, o_m)$ , субъектов  $S = (s_1, s_2, \dots, s_l)$  и операций  $A = (a_1, a_2, \dots, a_k)$ . Система является монооперационной, если каждая команда  $a_i$  данной системы выполняет лишь одну элементарную операцию op.

**Критерий безопасности системы** Для заданной системы исходное состояние  $Q_0 = (S_0, O_0, M_0)$  называется безопасным относительно права r, если не существует такой последовательности команд, которая изменила бы заданное начальное состояние системы так, что право r записалось бы в ячейку  $M(s,o)$ , в которой оно отсутствовало в начальном состоянии  $Q_0$ . Если это условие не выполнено, то произошла утечка информации

### Теоремы:

- Существует алгоритм, проверяющий исходное состояние  $Q_0$  монооперационной системы на безопасность относительно права r.
- Задача определения безопасности исходного состояния  $Q_0$  системы общего вида для данного права r является неразрешимой. (Чтобы доказать данную теорему, достаточно свести задачу определения безопасности к задаче остановки машины Тьюринга, которая является заведомо неразрешимой.)

### Преимущества HRU

1. Простота и наглядность, так как для данной модели не требуется сложных алгоритмов.
2. Эффективность в управлении, так как возможно управление правами пользователей с точностью до операции.
3. Сильный критерий безопасности.

### Недостатки HRU

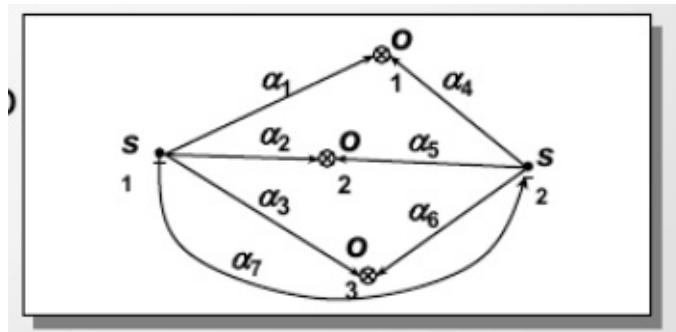
1. Не существует алгоритма проверки на безопасность для произвольной системы.
2. Уязвимость к атаке с помощью троянского коня, так как в данной модели не существует контроля за потоками информации между субъектами.

### Модель Take-Grant

Система защиты представляет совокупность следующих множеств:

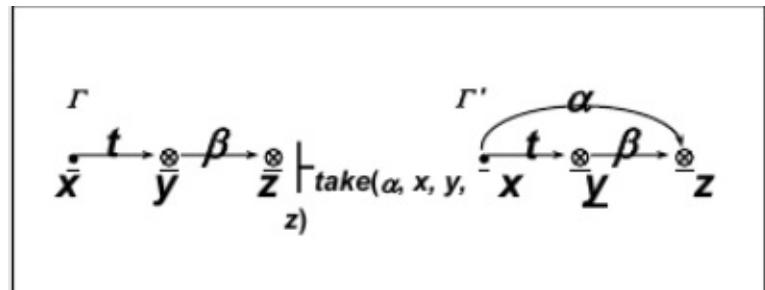
- множество исходных объектов  $O = (o_1, o_2, \dots, o_m)$
- множество исходных субъектов  $S = (s_1, s_2, \dots, s_l)$ , при этом  $S \subset O$
- множество прав, которые м.б. даны субъектам по отношению к объектам  $(r_1, r_2, \dots, r_n) \cup \{t, g\}$ 
  - право take ( $t$  – право брать права доступа у какого-либо объекта по отношению к другому объекту)
  - право grant ( $g$  – право предоставлять права доступа к определенному объекту другому субъекту)
- множеством E установленных прав доступа  $(x, y, \alpha)$  субъекта x к объекту y с правом  $\alpha$  из конечного набора прав.

При этом состояние системы представляется графом доступов  $\Gamma(O, S, E)$

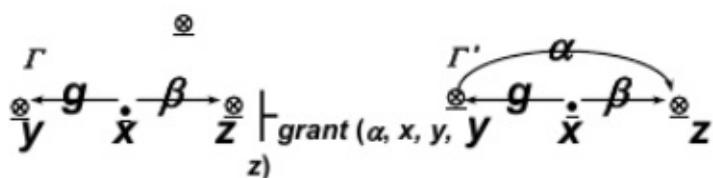


Состояние системы (Графа доступов) изменяется под воздействием элементарных команд 4-х видов:

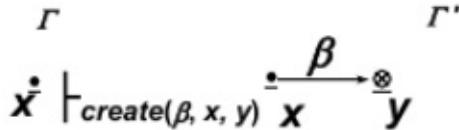
- Команда "Брать" –  $\text{take}(\alpha, x, y, z)$ : субъект x берет права доступа  $\alpha \subseteq \beta$  на объект z у объекта y



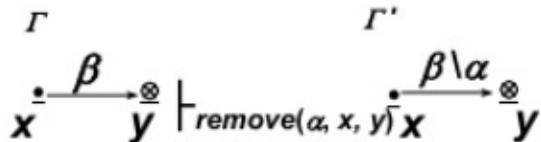
- Команда "Давать" –  $\text{grant}(\alpha, x, y, z)$ : субъект x дает объекту y право  $\alpha \subseteq \beta$  на доступ к объекту z



- Команда "Создать" –  $\text{create}(\beta, x, y)$ : субъект x создает объект y с правами доступа на него  $\beta \subseteq R$  ( $y$  – новый объект,  $O' = O \cup \{y\}$ ), в т. ч. с правами t, или g, или  $\{t, g\}$ .



- Команда "Изъять" –  $remove(\alpha, x, y)$ : субъект x удаляет права доступа  $\alpha \subseteq \beta$  на объект y



**Безопасность системы** рассматривается с точки зрения возможности получения каким-либо субъектом прав доступа к определенному объекту (в начальном состоянии  $\Gamma_0(O_0, S_0, E_0)$  такие права отсутствуют) при определенной кооперации субъектов путем последовательного изменения состояния системы на основе выполнения элементарных команд.

Рассматриваются две ситуации – условия санкционированного, т.е. законного получения прав доступа, и условия «похищения» прав доступа.

**Обозначения:**  $\vdash_c$  – переход графа  $\Gamma$  в новое состояние  $\Gamma'$  по команде c ;

**Определения:**

- Для исходного состояния системы  $\Gamma_0(O_0, S_0, E_0)$  и прав доступа  $\alpha \subseteq R$  предикат "возможен доступ( $\alpha, x, y, \Gamma_0$ )" является истинным тогда и только тогда, когда существуют графы доступов системы  $\Gamma_1, \Gamma_2, \dots, \Gamma_N$  такие, что:  $\Gamma_0 \vdash_c 1 \Gamma_1 \vdash_c 2 \dots \vdash_c N \Gamma_N$ , и  $(x, y, \alpha) \in E_n$ , где  $c1, c2, \dots, cN$  – команды переходов
- Вершины графа доступов являются tg-связными (соединены tg-путем), если в графе между ними существует такой путь, что каждая дуга этого пути выражает право t или g (без учета направления дуг)
- Для исходного состояния системы  $\Gamma_0(O_0, S_0, E_0)$  и прав доступа  $\alpha \subseteq R$  предикат "возможно похищение ( $\alpha, x, y, \Gamma_0$ )" является истинным тогда и только тогда, когда существуют графы доступов системы  $\Gamma_1, \Gamma_2, \dots, \Gamma_N$  т такие, что:  $\Gamma_0 \vdash_c 1 \Gamma_1 \vdash_c 2 \dots \vdash_c N \Gamma_N$ , и  $(x, y, \alpha) \in E_n$ , где  $c1, c2, \dots, cN$  – команды переходов; при этом, если  $\exists (s, y, \alpha) \in E_0$ , то  $\forall z \in S_j, j = 0, 1, \dots, N$ , выполняется:  $c1 = grant(\alpha, s, z, y)$ .

**Теоремы:**

- В графе доступов  $\Gamma_0(O_0, S_0, E_0)$ , содержащем только вершины-субъекты, предикат "возможен доступ( $\alpha, x, y, \Gamma_0$ )" истинен тогда и только тогда, когда выполняются следующие условия:
  - существуют субъекты  $s_1, \dots, s_m$  такие, что  $(s_i, y, \gamma_i) \in E_0$  для  $i=1, \dots, m$  и  $\alpha = \gamma_1 \cup \dots \cup \gamma_m$ .
  - субъект x соединен в графе  $\Gamma_0$  tg-путем с каждым субъектом  $s_i$  для  $i=1, \dots, m$
- Еще одна теорема на слайде

**Определение** Островом в произвольном графе доступов  $\Gamma(O, S, E)$  называется его максимальный **tg-связный** подграф, состоящий только из вершин субъектов.

**Определение**. Мостом в графе доступов  $\Gamma(O, S, E)$  называется **tg-путь**, концами которого являются вершины-субъекты; при этом словарная запись **tg-пути** должна иметь вид

$\vec{t}^*, \vec{t}^*, \vec{t}^* \vec{g} \vec{t}^*, \vec{t}^* \vec{g} \vec{t}^*$

где символ \* означает многократное (в том числе нулевое) повторение.

**Определение** Начальным пролетом моста в графе доступов  $\Gamma(O, S, E)$  называется **tg-путь**, началом которого является вершина-субъект; при этом словарная запись **tg-пути**:

$\vec{t}^* \vec{g}$

**Определение** Конечным пролетом моста в графе доступов  $\Gamma(O, S, E)$  называется **tg-путь**, началом которого является вершина-субъект; при этом словарная запись **tg-пути**:

$\vec{t}^*$

### Теорема.

В произвольном графе доступов  $\Gamma_0(O_0, S_0, E_0)$  предикат "**возможен доступ**( $\alpha, x, y, \Gamma_0$ )" истинен тогда и только тогда, когда выполняются условия:

1. существуют объекты  $s_1, \dots, s_m$  такие, что  $(s_i, y, \gamma_i) \in E_0$  для  $i=1, \dots, m$  и  $\alpha = \gamma_1 \cup \dots \cup \gamma_m$ .
2. существуют вершины-субъекты  $x_1', \dots, x_m'$  и  $s_1', \dots, s_m'$  такие, что:
  - $x = x_i'$  или  $x_i'$  соединен с  $x$  начальным пролетом моста для  $i=1, \dots, m$ ;
  - $s_i = s_i'$  или  $s_i'$  соединен с  $s_i$  конечным пролетом моста для  $i=1, \dots, m$ .
  - для каждой пары  $(x_i', s_i')$ ,  $i=1, \dots, m$ , существуют острова  $I_{i1}, \dots, I_{iu_i}$ ,  $u_i \geq 1$ , такие, что  $x_i' \in I_{i1}$ ,  $s_i' \in I_{iu_i}$ , и мосты между островами  $I_{ii}$  и  $I_{i+1}$ .

- В произвольном графе доступов  $\Gamma_0(O_0, S_0, E_0)$  предикат "возможно похищение ( $\alpha, x, y, \Gamma_0$ )" истинен тогда и только тогда, когда выполняются условия:

- $(x, y, \alpha) \notin E_0$
  - существуют субъекты  $s_1, \dots, s_m$  такие, что  $(s_i, y, \gamma_i) \in E_0$  для  $i=1, \dots, m$  и  $\alpha = \gamma_1 \cup \dots \cup \gamma_m$ .
  - являются истинными предикаты "возможен доступ( $t, x, s_i, \Gamma_0$ )" для  $i=1, \dots, m$ .

### Достоинства дискреционных моделей

- Хорошая гранулированность защиты (позволяют управлять доступом с точностью до отдельной операции над отдельным объектом)
- Простота реализации

### Недостатки дискреционных моделей

- Слабые защитные характеристики из-за невозможности для реальных систем выполнять все ограничения безопасности
- Проблема "троянских коней"
- Сложности в управлении доступом из-за большого количества назначений прав доступа

### Задача проверки системы от несанкционированного доступа

**Защита информации от несанкционированного доступа (НСД)** - деятельность, направленная на предотвращение получения защищаемой информации заинтересованным субъектом с нарушением установленных правовыми документами или собственником, владельцем информации прав или правил доступа к защищаемой информации.

**Несанкционированный доступ к информации (НСД)** - доступ к информации, нарушающий правила разграничения доступа с использованием штатных средств, предоставляемых средствами вычислительной техники или

автоматизированными системами. **Организация защиты информации** - содержание и порядок действий, направленных на обеспечение защиты информации. **Система защиты информации** - совокупность органов и (или) исполнителей, используемой ими техники защиты информации, а также объектов защиты, организованная и функционирующая по правилам, установленным соответствующими правовыми, организационно -распорядительными и нормативными документами в области защиты информации **Средство защиты информации** - техническое, программное средство, вещество и (или) материал, предназначенные или используемые для защиты информации **Средство контроля эффективности защиты информации** - техническое, программное средство, вещество и (или) материал, предназначенные или используемые для контроля эффективности защиты информации **Потоком информации** между объектом  $O_m$  и объектом  $O_j$  называется произвольная операция над объектом  $O_j$ , реализуемая в субъекте  $S_i$  и зависящая от  $O_m$ .  $Stream(S_i O_m) \rightarrow O_j$  \*\*Монитор порождения субъектов (МПС) — **субъект, активизирующийся при любом порождении субъектов**. Монитор безопасности субъектов (МБС)\* — субъект, который разрешает порождение потоков только для фиксированного подмножества пар активизирующих субъектов и объектов-источников.

Потоки информации в рамках субъектно-объектной модели документа либо легальны (множество потоков L), либо нелегальны (множество потоков N), т.е. нарушают целостность или конфиденциальность.

Требования к МБО - монитору безопасности:

- полнота и непрерывность (он работает всегда и его не обойти)
- изолированность (монитор должен быть защищён)
- верифицируемость (нужно уметь проверить корректность монитора)

## 18. Методы аутентификации в сети. Протокол аутентификации Kerberos.

### Основные определения

**Идентификация** — это процесс распознавания элемента системы, обычно с помощью заранее определённого идентификатора или другой уникальной информации — каждый субъект или объект системы должен быть однозначно идентифицируем.

**Аутентификация** — это проверка подлинности идентификации пользователя, процесса, устройства или другого компонента системы (обычно осуществляется перед разрешением доступа).

Идентификация и аутентификация — взаимосвязанные процессы распознавания и проверки подлинности субъектов (пользователей). Именно от них зависит решение системы, можно ли разрешить доступ к ресурсам системы конкретному пользователю или процессу.

После того как субъект *идентифицирован и аутентифицирован*, выполняется его *авторизация*.

### Методы идентификации на основе секрета

- **Простая аутентификация** (на основе использования паролей);
- **Строгая аутентификация** (на основе использования криптографических методов и средств);
- Процессы (протоколы) аутентификации, обладающие свойством доказательства с **нулевым знанием**.

Слабость простой аутентификации именно в самом пароле и человеческом факторе. Далее рассматриваем строгую аутентификацию.

### Строгая аутентификация

Идея строгой аутентификации, реализуемая в криптографических протоколах, заключается в следующем. Проверяемая (доказывающая) сторона доказывает свою подлинность проверяющей стороне, демонстрируя знание какого-либо секрета, который, например, может быть предварительно распределен безопасным способом между сторонами аутентификационного обмена. Важно, что доказывающая сторона демонстрирует только знание секрета, но сам он в ходе аутентификационного обмена не раскрывается.

В соответствии с рекомендациями стандарта X.509 различают процедуры строгой аутентификации следующих типов:

- односторонняя аутентификация;
- двусторонняя аутентификация;
- трехсторонняя аутентификация.

В зависимости от используемых криптографических алгоритмов протоколы строгой аутентификации можно разделить на следующие группы:

- протоколы аутентификации на основе симметричных алгоритмов;
- протоколы аутентификации на основе хэш-функций;
- протоколы аутентификации на основе асимметричных алгоритмов;

- протоколы аутентификации на основе цифровой подписи.

Так как билет про Kerberos, далее рассматриваются протоколы аутентификации на основе симметричных алгоритмов. На всякий случай вспомним:

**Симметричное шифрование** — способ шифрования, в котором для шифрования и расшифровывания применяется один и тот же криптографический ключ.

Простой пример как можно сделать двустороннюю аутентификацию на симметричном шифровании:

(1)  $A \leftarrow B : R_B$

(2)  $A \rightarrow B : E_K(R_A, R_B, B)$

(3)  $A \leftarrow B : E_K(R_A, R_B)$

Шаг 0: А и В хотят взаимно аутентифицироваться, симметричный ключ  $E_K$  есть у обоих участников

Шаг 1: Участник В посыпает участнику А случайное число

Шаг 2: Участник А отвечает зашифрованной  $E_K$  последовательностью из принятого числа, еще одного случайного числа и идентификатора В.

Шаг 3: Участник В отвечает зашифрованной последовательностью из двух ранее посланных случайных чисел.

## KDC

Помимо аутентификации нужно еще генерировать ключ сессии для шифрования пользовательских данных, которые мы хотим послать (а шифровать все одним ключом не по феншу), поэтому для обеспечения аутентификации и распределения ключа сессии в сети используется двухуровневая иерархия ключей симметричного шифрования, которая используется двумя сторонами или основана на использовании доверенного центра (3-й стороны) распределения ключей (KDC).

Рассмотрим вариант с KDC.

Каждый участник разделяет секретный ключ, называемый также мастер-ключом, с KDC. KDC отвечает за создание ключей, называемых ключами сессии, и за распределение этих ключей с использованием мастер-ключей. Ключи сессии применяются для шифрования только данной сессии между двумя участниками.

Существует много протоколов, реализующих эту идею, нам будет важен протокол Нидхэма-Шредера, взятый за основу для Kerberos. Опять же, у протокола много вариантов, рассмотрим самый базовый.

1.  $A \rightarrow KDC : ID_a || ID_b || N_a$
2.  $KDC \rightarrow A : E_{K_a}[K_s || ID_b || N_a]$   
 $E_{K_b}[K_s || ID_a]$
3.  $A \rightarrow B : E_{K_b}[K_s || ID_a]$
4.  $B \rightarrow A : E_{K_s}[N_b]$
5.  $A \rightarrow B : E_{K_s}[f(N_b)]$

Шаг 0: Предполагается, что секретные мастер-ключи  $K_a$  и  $K_b$  разделяют соответственно А и KDC, В и KDC. Целью протокола является аутентификация и безопасное распределение ключа сессии  $K_s$  между А и В

Шаг 1: А запрашивает у KDC ключ сессии для установления защищенного соединения с В. Сообщение включает идентификаторы А и В и уникальный идентификатор (номер) данной транзакции, который обозначен как  $N_a$  — он должен быть уникальным для каждого запроса — суть идентификатор сессии, не дает ей повториться. Сообщение зашифровано мастер-ключом  $K_a$ .

Шаг 2: Все и так понятно,  $K_s$  это ключ сессии, приходит 2 сообщения к стороне А.

Шаг 3: А передает второе сообщение от KDC на В.

Шаг 4 и 5: По сути проверка сессионного ключа:  $N_b$  случайное число,  $f$  известная функция (например квадрат числа).

## Kerberos

- Протокол аутентификации пользователей на основе доверенной третьей стороны (KDC)
- Работает по умолчанию при доменной аутентификации в Windows (можно настроить и для linux с AD контроллером на винде, причем вы не только сможете логиниться с linux машины на сервисы с доменной аутентификацией, но и сами ее проводить, шок да и только), но если живете в чисто linux мире вредил вам это понадобиться, скорее всего ограничитесь x509 сертами.
- Использует в качестве транспорта протокол TCP. Ну как бы очевидно.
- Сама система работает прозрачно для пользователя, он просто вводит пароль

## Протокол с билетом

Сервер аутентификации Kerberos использует развитие двухуровневой системы аутентификации на основе применения концепции “билетов/мандатов”.

1.  $A \rightarrow B : ID_a || N_a$
2.  $B \rightarrow KDC : ID_b || N_b || E_{Kb} [ID_a || N_a || T_b]$
3.  $KDC \rightarrow A : E_{Ka} [ID_b || N_a || K_s || T_b] || E_{Kb} [ID_a || K_s || T_b] || N_b$
4.  $A \rightarrow B : E_{Kb} [ID_a || K_s || T_b] || E_{Ks} [N_b]$

Шаг 0: Предполагается, что секретные мастер-ключи  $K_a$  и  $K_b$  разделяют соответственно А и KDC, В и KDC. Целью протокола является аутентификация и безопасное распределение ключа сессии  $K_s$  между А и В

Шаг 1: А инициализирует аутентификационный обмен созданием идентификатора  $N_a$  и посылкой его и своего идентификатора к В в незашифрованном виде. Этот  $N_a$  вернется к А в зашифрованном сообщении, включающем ключ сессии, гарантируя А, что ключ сессии не старый.

Шаг 2: В сообщает KDC, что необходим ключ сессии. Это сообщение к KDC включает идентификатор В и  $N_b$ . Данный  $N_b$  вернется к В в зашифрованном сообщении, которое включает ключ сессии, гарантируя В, что ключ сессии не устарел. Последний блок используется для указания KDC, когда заканчивается время жизни  $T_b$  данного ключа сессии и специфицирует намеченного получателя и содержит  $N_a$ , полученный от А.

Шаг 3: Да, да это одно сообщение, но можно и два, ничего не изменит. KDC получил  $N_a$  и  $N_b$  от А и В и посыпает А блок, зашифрованный секретным ключом, который В разделяет с KDC. Блок служит **билетом**, который может быть использован А для последующих аутентификаций. KDC также посыпает А блок, зашифрованный секретным ключом, разделяемым А и KDC, что доказывает:

- В получил начальное сообщение А ( $ID_b$ );
- в нем содержится допустимая отметка времени и нет повтора ( $N_a$ ).

То есть запомним **билет** это:  $E_{Kb} [ID_a || K_s || T_b]$

Шаг 4: А посыпает полученный билет В вместе с  $N_b$ , зашифрованным ключом сессии. Этот билет обеспечивает В ключом сессии, который тот использует для дешифрования и проверки  $N_b$ . Тот факт, что  $N_b$  расшифрован ключом сессии, доказывает, что сообщение пришло от А и не является повтором.

Вот и все, это основная идея, осталось повторить 2 раза.

Данный протокол аутентифицирует А и В и распределяет ключ сессии. Более того, протокол предоставляет в распоряжение А билет, который может использоваться для его последующей аутентификации, исключая необходимость повторных контактов с аутентификационным сервером.

## Мандаты на мандаты в Kerberos

В протоколе аутентификации Kerberos используются два уровня билетов (мандат на право получения билета и собственно билет) и следовательно имеем 2 сервера:

- Kerberos — сервера аутентификации\центра распространения ключей (KDC\AS), который выдает мандат (“главный билет”)
- TGS сервер, для выдачи “билетов” (вторичных мандатов) для доступа к сетевым службам.

По сути, это разделение функций аутентификации и авторизации.

Процедура получения мандата, за счет использования меток времени вместо случайных номеров, сокращена по сравнению с описанной выше:

1.  $A \rightarrow KDC : ID_a || ID_{tgs} || T_a$
2.  $KDC \rightarrow A : E_{K_a}[ID_{tgs} || K_{a-tgs} || T_{tgs} || (\delta T_{tgs}) || (M_{tgs})]$ , где M (**мандат**):

$$(M_{tgs}) = E_{K_{tgs}}[ID_a || K_{a-tgs} || AD_a || ID_{tgs} || T_{tgs} || (\delta T_{tgs})]$$

В отличие от описанной выше схемы  $N_x$  заменены на временные метки  $T_x$  и срок ( $\delta T_x$ ) для объекта x. Добавлен параметр сетевой адрес А -  $AD_a$  и В заменен на TGS (соответственно сеансовый ключ стал обозначаться  $K_{a-tgs}$ , а общий секрет TGS и KDC —  $K_{tgs}$ ).

Важно, что сообщение (2) шифруется общим секретом А и KDC (разделяемым ключом или его генерацией на основе пароля пользователя), а следующие сообщения шифруются уже полученным сеансовым ключом.

После этого на основе этого мандата можно обращаться в TGS за вторичными мандатами (билетами), на каждый новый сервис (например В). Билет (мандат) на получение сервиса имеет ту же структуру, что и мандат на получение мандата:

$$(\text{Билет В}) = E_{K_b}[ID_a || K_{s-b} || AD_a || ID_b || T_b || (\delta T_b)]$$

Таким образом, пользователь сначала запрашивает мандат на получение мандата (Мандат TGS ) у сервера аутентификации AS.

Этот мандат сохраняется модулем клиента на рабочей станции пользователя. Каждый раз, когда пользователю требуется новый сервис, клиент обращается к TGS и использует этот мандат, чтобы идентифицировать себя.

В ответ TGS выдает билет (вторичный мандат) на получение конкретного сервиса. Клиент сохраняет билет на получение сервиса и использует его для идентификации пользователя сервером всякий раз, когда запрашивается данный сервис.

Теперь рассмотрим как происходит получение итогового мандата (билета):

1.  $A \rightarrow TGS : ID_b || M_{tgs} || U_{A1}$
2.  $TGS \rightarrow A : E_{K_{a-tgs}}[K_{a-b} || ID_b || Ts_4 || \delta Ts_2 || M_b]$

$$\text{Мандат tgs: } M_{tgs} = E_{K_{tgs}}[ID_a || K_{a-tgs} || AD_a || ID_{tgs} || T_{tgs} || (\delta T_{tgs})]$$

$$\text{Мандат b: } M_b = E_{K_b}[K_{a-b} || ID_a || AD_a || ID_b || Ts_4 || \delta Ts_4]$$

$$\text{Удостоверение A1: } U_{A1} = E_{K_{a-tgs}}[ID_a || AD_a || Ts_3]$$

А передает удостоверение, включающее идентификатор и адрес пользователя А, а также метку даты-времени. В отличие от мандата, который предполагает многократное использование, удостоверение предлагается использовать только один раз и срок его действия весьма ограничен. TGS может сравнить содержащиеся в удостоверении имя и сетевой адрес с соответствующими элементами мандата и с сетевым адресом поступившего сообщения. Мандат никого не идентифицирует, а обеспечивает способ защищенного распределения ключей. Задачи аутентификации клиента выполняет удостоверение.

1.  $A \rightarrow B : M_b || U_{A2}$

2.  $B \rightarrow A : E_{K_{a-b}}[Ts_5 + 1]$

Удостоверение A2:  $U_{A2} = E_{K_{a-b}}[ID_a || AD_a || Ts_5]$

Если требуется взаимная аутентификация, сервер может ответить так, как предлагается в сообщении (6). Сервер возвращает значение метки даты-времени из удостоверения, увеличенное на 1 и зашифрованное на сеансовом ключе. Система A может расшифровать это сообщение и проверить увеличенное значение метки даты-времени. Поскольку сообщение было зашифровано сеансовым ключом, А убеждается в том, что это сообщение могло быть создано только сервером В и не является воспроизведением старого ответа.

Итого все вместе:

- Обмен службы аутентификации: получение мандата на получение мандата:
  - $A \rightarrow KDC : ID_a || ID_{tgs} || T_a$
  - $KDC \rightarrow A : E_{K_a}[ID_{tgs} || K_{a-tgs} || T_{tgs} || (\delta T_{tgs}) || (M_{tgs})]$
- Обмен службы выдачи мандатов: получение мандата (билета) на получение сервиса
  - $A \rightarrow TGS : ID_b || M_{tgs} || U_{A1}$
  - $TGS \rightarrow A : E_{K_{a-tgs}}[K_{a-b} || ID_b || Ts_4 || \delta Ts_2 || M_b]$
- Обмен аутентификации клиента/сервера: получение сервиса
  - $A \rightarrow B : M_b || U_{A2}$
  - $B \rightarrow A : E_{K_{a-b}}[Ts_5 + 1]$

Ура

Заметим, что в Kerberos еще есть понятие области (домена, realm). И возможна также междоменная аутентификация. Сделать это можно разделив секретный ключ с интересующей областью. По факту появится 3-й уровень мандатов.

## Kerberos 5

Вышесказанное справедливо для Kerberos 4, в 5 версии был сделан ряд улучшений:

- версия 4 требует применения DES. В версии 5 к шифрованному тексту присоединяется идентификатор типа шифрования, поэтому может использоваться любая схема шифрования. Для ключей шифрования указываются длины, что позволяет использовать одни и те же ключи в различных алгоритмах и указывать варианты одного алгоритма для применения.
- сетевые адреса сопровождаются метками типа и длины, что дает возможность использовать любые типы сетевых адресов.
- включают явное указание времени начала и окончания действия мандата, что позволяет указывать любые сроки действия мандатов.
- оптимизация аутентификации в удаленной области.

Конечно поменялся и сам алгоритм, в мандаты добавились области, границы времени, флаги и на этом моменте стоит остановиться и при желании посмотреть слайды.

## Недостатки

**Версия 4** Kerberos имеет технические недостатки:

1. Двойное шифрование. Обратите внимание (сообщения 2 и 4), мандаты, выдаваемые клиентам, шифруются дважды — один раз секретным ключом сервера назначения, а затем снова, секретным ключом, известным клиенту. Второе шифрование не является необходимым, и поэтому вызывает излишнее потребление вычислительных ресурсов.
2. DES
3. Сеансовые ключи. Каждый мандат включает сеансовый ключ, используемый клиентом для шифрования удостоверений, посылаемых службе, связываемой с данным мандатом. Кроме того, этот сеансовый ключ может впоследствии использоваться клиентом и сервером для защиты сообщений, пересылаемых в ходе сеанса. Однако,

ввиду того, что один и тот же мандат может использоваться повторно для получения соответствующего сервиса от конкретного сервера, существует риск, что нарушитель может предъявить клиенту или серверу воспроизведенные сообщения старого сеанса. В версии 5 для клиента и сервера существует возможность договориться о сеансовом подключении, который действует только в одном соединении. При новом доступе клиент должен будет использовать новый сеансовый подключ.

**Версии 4 и 5** Kerberos имеет технические недостатки:

1. Атаки на пароль. Обе версии уязвимы в отношении атак на пароль. Сообщение клиенту от системы AS включает данные, зашифрованные с помощью ключа, построенного на основе пароля клиента. Нарушитель может перехватить это сообщение и попытаться дешифровать его, используя разные пароли. Если в результате попыток расшифрования получится сообщение правильного вида, то нарушитель узнает пароль клиента и сможет впоследствии использовать этот пароль для того, чтобы получать удостоверения аутентификации от Kerberos. Версия 5 предлагает механизм, называемый предварительной аутентификацией, призванный затруднить атаки на пароль, но не исключает их возможность полностью.

## **19. Пасивные и активные сетевые атаки (снифинг, спуфинг, МИТМ, имперсонация).**

### **Типичная атака**

Стадия 1: Внешняя разведка

Стадия 2: Внутренняя разведка

Стадия 3: Атака

Стадия 4: Скрытие следов

Стадия 5: Получение прибыли

### **Пассивные атаки**

#### **Прослушивание (снифинг)**

Перехват трафика может осуществляться:

- обычным «прослушиванием» сетевого интерфейса. Метод эффективен при использовании в сегменте концентраторов вместо коммутаторов, в противном случае данный метод малоэффективен, поскольку на снiffer попадают лишь отдельные кадры;
- подключением снifferа в разрыв канала;
- ответвлением (программным или аппаратным) трафика и направлением его копии на снiffer;
- через атаку на канальном или сетевом уровне, приводящую к перенаправлению трафика жертвы или всего трафика сегмента на снiffer с последующим возвращением трафика в надлежащий адрес.

#### **Противодействие**

Ограничить область прослушивания в сети Ethernet можно разбиением сети на сегменты с помощью коммутаторов. В этом случае злоумышленник, не прибегая к активным действиям, может перехватить только кадры, получаемые или отправляемые узлами сегмента, к которому он подключен.

Простое прослушивание также не позволяет злоумышленнику модифицировать передаваемые между двумя другими узлами данные. Для решения этих задач злоумышленник должен перейти к активным действиям, чтобы внедрить себя в тракт передачи данных в качестве промежуточного узла.

**Единственным надежным способом борьбы с прослушиванием сегмента Ethernet является шифрование данных.**

Еще в слайдах описан метод AntiSniff — метод выявления пакетных снifferов основанный на предположении, что программа прослушивания на хосте злоумышленника выполняет обратные DNS-преобразования для IP-адресов подслушанных датаграмм, но это как мне кажется смешно. Однако у AntiSniff есть интересный метод выявление нод в монитор mode, посыпом шквала сообщений с несуществующими в текущем сегменте Mac-адресами.

#### **Сканирование сети**

Сканирование сети имеет своей целью выявление активных компьютеров сети. Почему то в этом пункте перечислены только ping и traceroute (и внезапно DNS).

## Сканирование TCP портов

Мне правда нужно про это писать?

Сканируем TCP порты через хэндшейк.

1. Методы открытого сканирования: непосредственный инициатор однозначно определяется объектом сканирования по IP-адресу запросов.

2. Методы "невидимого" анонимного сканирования. Непосредственный инициатор не определяется объектом сканирования (однозначно определяется только "промежуточный" источник сканирующих запросов), таким образом, гарантируется анонимность инициатора сканирования.

Есть более хитрые способы чем ломиться с хэндшейком.

Сканирование TCP FIN — в этом методе на сканируемый порт посыпается сегмент с установленным битом FIN. Хост должен ответить RST-сегментом, если FIN-сегмент адресован закрытому порту.

Сканирование с использованием IP-фрагментации — не является само по себе новым видом сканирования. Данный метод предназначен для усложнения задачи обнаружения факта сканирования и может применяться совместно с упомянутыми ранее методами. Суть его состоит в разбиении TCP SYN- или TCP FIN-запроса на несколько маленьких IP-фрагментов (минимальный размер поля данных в IP-фрагменте 8 байт, следовательно, TCP SYN-запрос, имеющий минимальный размер 20 байт, можно разбить на три фрагмента). Первый фрагмент не содержит флагов, на основании которых данные пакеты могут быть заблокированы. Тем не менее, если межсетевой экран выполняет дефрагментацию пакетов, то данный метод является неэффективным.

## UDP сканирование

Сканирование UDP-сервисов представляет собой достаточно сложную процедуру. Это связано с тем, что протокол UDP является протоколом без установления соединения, а, следовательно, отсутствуют предварительные шаги по созданию виртуального канала.

Для выполнения сканирования UDP-портов можно воспользоваться тем фактом, что большинство хостов в ответ на получение пакет, направленного на закрытый UDP-порт отвечают ICMP пакетом ICMP\_PORT\_UNREACH. Таким образом, можно обнаружить закрытые порты.

Исключив закрытые порты можно получить список открытых портов. Данный вид сканирования является медленным, так как в соответствии с RFC 1812 хост может значительно ограничить количество ответов содержащих ICMP-сообщения об ошибках.

## Другие сканирования

По различному поведению в разных сценариях можно выяснить и ОС системы и версию ПО и тд.

## Активные атаки

Спуфинг — ситуация, в которой один человек или программа успешно маскируется под другую путём фальсификации данных и позволяет получить незаконные преимущества.

## Ложный ARP-сервер

Это называется ARP **спуфинг** — отвечаем на арг запросы, что хост или маршрутизатор это мы, получаем пакеты.

## Ложный DNS-сервер

## Перехват DNS запроса

Для реализации атаки путем перехвата DNS-запроса злоумышленнику необходимо перехватить запрос, извлечь из него номер UDP-порта хоста отправителя, двухбайтовое значение ID-идентификатора DNS-запроса и искомое имя, а затем послать ложный DNS-ответ на извлеченный из DNS-запроса UDP порт, где в качестве искомого IP-адреса указать настоящий IP-адрес ложного DNS-сервера. Такой вариант атаки в дальнейшем позволит полностью перехватить трафик между атакуемым хостом и сервером и активно воздействовать на него. Необходимым условием осуществления данного варианта атаки является возможность перехвата DNS-запроса, а это возможно только в том случае, если атакующий находится, либо на пути следования запроса к DNS-серверу, либо в одном сегменте с DNS-сервером.

## Направленный штурм ложных DNS-ответов

Другой вариант осуществления удаленной DNS-атаки - внедрение в сеть Internet ложного сервера путем создания направленного штурма ложных DNS-ответов на атакуемый хост. В этом случае злоумышленник осуществляет постоянную передачу на атакуемый хост заранее подготовленного ложного DNS-ответа от имени настоящего DNS-сервера без предыдущего приема DNS-запроса.

Но IP-адрес отправителя ответа должен совпадать с IP-адресом DNSсервера, а имя в DNS-ответе - с именем в DNS-запросе; кроме того, DNS-ответ следует направить на тот же UDP-порт, с которого было послано сообщение (в данном случае это первая проблема для взломщика), и поле идентификатора запроса (ID) в заголовке DNS-ответа должно содержать то же значение, что и в переданном запросе (а это вторая проблема).

Таким образом, реализация данной удаленной атаки, использующей пробелы в безопасности службы DNS, позволяет из любой точки сети Internet нарушить маршрутизацию между двумя заданными объектами (хостами). Такая атака осуществляется межсегментно по отношению к цели атаки и угрожает безопасности любого хоста Internet, использующего обычную службу DNS.

## Имперсонация

Предположим, что узел А обменивается IP-датаграммами с узлом В, при этом узлы идентифицируют друг друга по IP-адресам, указываемым в датаграммах. Предположим далее, что узел В имеет особые привилегии при взаимодействии с А: то есть А предоставляет В некоторый сервис, недоступный для других хостов Internet. Злоумышленник на узле X, желающий получить такой сервис, должен имитировать узел В — такие действия называются имперсонацией узла В узлом X.

Пусть узел находится в сети, не имеющей никакого отношения к узлам А и В и не лежащей между ними (А и В могут находиться как в одной, так и в разных сетях). Легко видеть, что имперсонация UDP-сообщений без обратной связи является тривиальной - злоумышленник должен только сфабриковать датаграмму, адресованную от узла В узлу А, и отправить ее по назначению.

Схема атаки с имперсонацией TCP-соединения без обратной связи

1. Злоумышленник выводит из строя узел В.
2. Злоумышленник делает несколько пробных попыток установить соединения с узлом А с целью получить от А последовательность значений ISN(A). Сразу после поступления SYN-сегмента от А злоумышленник разрывает наполовину установленное соединение посылкой сегмента с флагом RST. Проанализировав полученные значения ISN(A), злоумышленник определяет закон формирования этих значений.
3. Злоумышленник отправляет в А SYN сегмент от имени В.
4. Узел А отвечает узлу В своим SYN сегментом, подтверждающим получение SYN-сегмента от В, и указывает значение ISN(A) для этого соединения. Злоумышленник не видит этого сегмента.
5. На основе ранее полученных данных злоумышленник предсказывает значение ISN(A) и отправляет в А сегмент от имени В, содержащий подтверждение ISN(A)+1 и данные для прикладного процесса. Получив этот сегмент, узел А считает соединение с В установленным и передает поступившие данные прикладному процессу. Цель атаки достигнута. Данные могут быть, например, командой, которую узел А выполняет, потому что она поступила от

доверенного узла В.

## **МИТМ**

Про него в слайдах не говорится, но когда мы пропускаем через себя чужой трафик и выступай своего рода проксей (тайно для участников) это и есть МИТМ. Чтобы всех этих проблем не было используйте TLS современных версий с рекомендациями от Mozilla и не забывайте обновлять ОС, Nginx и openssl).

## 20. Коммуникационные протоколы. Ошибки, возникающие при передаче сообщений. Задача надежного обмена сообщениями. Симметричные протокол скользящего (раздвижного) окна: устройство протокола и обоснование его корректности. Протокол альтернирующего бита.

### Короткая версия (см подробную версию ниже)

Основное назначение коммуникационного протокола - передача данных, то есть получение информации от одного узла сети и доставка ее по назначению другому узлу сети. При передаче данных возможны ошибки (потеря, дублирование, искажение). Эти ошибки нужно обнаруживать и исправлять. Для этого в протоколе ведется учет состояния информации. Для использования состояния информации применяется управление соединением - инициализация и аннулирование состояния информации. Инициализация называется установлением соединения, а аннулирование - завершением соединения.

#### Симметричный протокол раздвижного окна (коммуникационный протокол)

Предназначен для обмена данными между двумя узлами сети, которые имеют прямое соединение (например, кабель). Асинхронный протокол, относящийся к уровню управления передачей данных (второй уровень модели OSI). Не будет рассматриваться управление соединением для этого протокола. Предполагается, что физическое соединение обычно функционирует непрерывно в течение долгого времени, а не устанавливается и завершается периодически. При физическом соединении сообщения не могут обгонять друг друга и дублироваться, поэтому рассматриваются только ошибки потери сообщения. Содержание сообщения, передаваемого по каналу связи может быть повреждено. Предполагается, что процесс-получатель способен обнаруживать искажения сообщений, например, при помощи счетчиков четности или кодирования с исправлением ошибок.

Постановка задачи:

Процессам  $p$  и  $q$  нужно передать данные  $in_p$  и  $in_q$  друг другу и записать полученные данные в  $out_p$  и  $out_q$ . В канале связи возможны помехи, приводящие к потере сообщений.

Общая идея алгоритма:

- Входные данные одного процесса служат для подтверждения получения сообщений от другого процесса.
- Сообщение - пакеты - наборы вида  $< pack, w, i >$ , где  $w$  - информационное слово,  $i$  - порядковый номер пакета
- Пакет  $< pack, w, i >$ , отправленный процессом  $p$ , передает слово  $w = in_p[i]$  процессу  $q$  и подтверждает успешное получение ряда пакетов, отправленных процессом  $q$ .
- Процесс  $p$  может опережать процесс  $q$  на некоторое заданное число пакетов  $l_p$ , если мы постановим, что отправление пакета  $< pack, w, i >$  процессом  $p$  подтверждает получение слов с номерами  $0, 1, \dots, (i - l_p)$  от процесса  $q$ .
- Константы опережения  $l_p$  и  $l_q$  известны процессам  $p$  и  $q$ .

Таким образом в протоколе соблюдаются два принципа:

- Процесс  $p$  может отправить слово  $in_p[i]$  только после того, как будут занесены в память все слова, начиная с  $out_p[0]$  и заканчивая  $out_p[i - l_p]$ , то есть когда будет выполняться неравенство  $i < s_p + l_p$ , где  $s_p = \min j : out_p[j] = undef$
- Как только  $p$  получает пакет  $< pack, w, i >$  отпадает необходимость в повторной передаче слов, начиная с  $in_p[0]$  и оканчивая  $in_p[i - l_q]$ .

Алгоритм состоит из следующих трех действий:

20. Коммуникационные протоколы. Ошибки, возникающие при передаче сообщений. Задача надежного обмена сообщениями. Симметричные протокол скользящего (раздвижного) окна: устройство протокола и обоснование его корректности. Протокол альтернирующего бита.

---

- Действие  $S_p$  осуществляет отправление  $i$ -го входного слова процесса  $p$ .
- Действие  $R_p$  осуществляет прием слова процессом  $p$ .
- Действие  $L_p$  моделирует потерю пакета, адресатом которого является процесс  $p$ .

**Требования корректности.** Нужно доказать, что протокол работает правильно, т.е. каждое слово из входного массива  $in_p$  процесса  $p$  будет рано или поздно записано в выходной массив  $out_q$  процесса  $q$ , и наоборот. Более строго это выражается двумя требованиями.

1. Безопасная доставка сообщений. В каждой достижимой конфигурации протокола выполняются соотношения  $out_p[0..s_p - 1] = in_q[0..s_p - 1]$  и  $out_q[0..s_q - 1] = in_p[0..s_q - 1]$ .
2. Неизбежная доставка сообщений. Для каждого целого числа  $k \geq 0$ , в ходе выполнения протокола будет достигнута конфигурация, в которой  $s_p \geq k$ ,  $s_q \geq k$ .

Симметричный протокол удовлетворяет этим требованиям по теоремам.

### Теорема.

Симметричный протокол раздвижного окна удовлетворяет требованию безопасной доставки сообщений, т.е. в каждой достижимой конфигурации протокола выполняются соотношения  $out_p[0..s_p - 1] = in_q[0..s_p - 1]$  и  $out_q[0..s_q - 1] = in_p[0..s_q - 1]$

### Теорема

Симметричный протокол раздвижного окна удовлетворяет требованию неизбежной доставки сообщений, т.е. для каждого целого числа  $k \geq 0$ , в ходе любого выполнения протокола будет достигнута конфигурация, в которой  $s_p, s_q \geq k$ .

### Протокол альтернирующего бита

Особо интересный вариант протокола раздвижного окна возникает в том случае, когда  $l_p = 1$  и  $l_q = 0$  (или наоборот). В качестве начальных значений переменных  $a_p$  и  $a_q$  в этом случае выбирается не 0, а числа  $-l_p$  и  $-l_q$ . Такой вариант алгоритма раздвижного окна называется протоколом чередующихся (альтернирующих) битов; он предназначен для односторонней передачи данных.

## Подробная версия

Основное назначение коммуникационного протокола - передача данных, то есть получение информации от одного узла сети и доставка ее по назначению другому узлу сети. При передаче данных возможны ошибки (потеря, дублирование, искажение). Эти ошибки нужно обнаруживать и исправлять. Для этого в протоколе ведется учет состояния информации. Для использования состояния информации применяется управление соединением - инициализация и аннулирование состояния информации. Инициализация называется установлением соединения, а аннулирование - завершением соединения.

### Симметричный протокол раздвижного окна (коммуникационный протокол)

Предназначен для обмена данными между двумя узлами сети, которые имеют прямое соединение (например, кабель). Асинхронный протокол, относящийся к уровню управления передачей данных (второй уровень модели OSI). Не будет рассматриваться управление соединением для этого протокола. Предполагается, что физическое соединение обычно функционирует непрерывно в течение долгого времени, а не устанавливается и завершается периодически. При физическом соединении сообщения не могут обгонять друг друга и дублироваться, поэтому рассматриваются только ошибки потери сообщения. Содержание сообщения, передаваемого по каналу связи может быть повреждено.

Предполагается, что процесс-получатель способен обнаруживать искажения сообщений, например, при помощи счетчиков четности или кодирования с исправлением ошибок.

Постановка задачи:

20. Коммуникационные протоколы. Ошибки, возникающие при передаче сообщений. Задача надежного обмена сообщениями. Симметричные протокол скользящего (раздвижного) окна: устройство протокола и обоснование его корректности. Протокол альтернирующего бита.

Процессам  $p$  и  $q$  нужно передать данные  $in_p$  и  $in_q$  друг другу и записать полученные данные в  $out_p$  и  $out_q$ . В канале связи возможны помехи, приводящие к потери сообщений.

Общая идея алгоритма:

- Входные данные одного процесса служат для подтверждения получения сообщений от другого процесса.
- Сообщение - пакеты - наборы вида  $\langle pack, w, i \rangle$ , где  $w$  - информационное слово,  $i$  - порядковый номер пакета
- Пакет  $\langle pack, w, i \rangle$ , отправленный процессом  $p$ , передает слово  $w = in_p[i]$  процессу  $q$  и подтверждает успешное получение ряда пакетов, отправленных процессом  $q$ .
- Процесс  $p$  может опережать процесс  $q$  на некоторое заданное число пакетов  $l_p$ , если мы постановим, что отправление пакета  $\langle pack, w, i \rangle$  процессом  $p$  подтверждает получение слов с номерами  $0, 1, \dots, (i - l_p)$  от процесса  $q$ .
- Константы опережения  $l_p$  и  $l_q$  известны процессам  $p$  и  $q$ .

Таким образом в протоколе соблюдаются два принципа:

- Процесс  $p$  может отправить слово  $in_p[i]$  только после того, как будут занесены в память все слова, начиная с  $out_p[0]$  и заканчивая  $out_p[i - l_p]$ , то есть когда будет выполняться неравенство  $i < s_p + l_p$ , где  $s_p = \min j : out_p[j] = undef$
- Как только  $p$  получает пакет  $\langle pack, w, i \rangle$  отпадает необходимость в повторной передаче слов, начиная с  $in_p[0]$  и оканчивая  $in_p[i - l_q]$ .

Действия алгоритма:

### Симметричный протокол раздвижного окна

```
var  $s_p, a_p$  : integer
     $in_p$  : array of word
     $out_p$  : array of word
init 0, 0 ;
(* Data to be sent *) ;
init undef, undef, ... ;
```

$S_p$ :  $\{ a_p \leq i < s_p + l_p \}$  begin send  $\langle pack, in_p[i], i \rangle$  to  $q$  end

```
Rp:  $\{ \langle pack, w, i \rangle \in Q_p \}$ 
begin receive  $\langle pack, w, i \rangle$  ;
if  $out_p[i] = undef$  then
    begin  $out_p[i] := w$  ;  $a_p := \max(a_p, i - l_q + 1)$  ;
         $s_p := \min\{j | out_p[j] = undef\}$ 
    end
(* else игнорировать повторное получение пакета *)
end
```

```
Lp:  $\{ \langle pack, w, i \rangle \in Q_p \}$ 
begin  $Q_p := Q_p \setminus \{\langle pack, w, i \rangle\}$  end
```

Действие  $S_p$  осуществляет отправление  $i$ -го входного слова процесса  $p$ .

Действие  $R_p$  осуществляет прием слова процессом  $p$ .

Действие  $L_p$  моделирует потерю пакета, адресатом которого является процесс  $p$ .

$S_p$ :

20. Коммуникационные протоколы. Ошибки, возникающие при передаче сообщений. Задача надежного обмена сообщениями. Симметричные протокол скользящего (раздвижного) окна: устройство протокола и обоснование его корректности. Протокол альтернирующего бита.

---

Данные для отправления выбираются из раздвижного окна  $a_p < i < s_p + l_p$ . Предполагается, что  $a_p$  наименьший номер в массиве  $in_p$ , получение которого еще не подтвердил процесс  $q$ ,  $s_p$  - наименьший номер того элемента в массиве  $out_p$ , в который еще не записаны полученные данные.

$R_p$ :

Получив сообщение процесс в начале проверяет не было ли идентичное сообщение получено ранее (возникает повторное получение сообщения). Если это не так, то слово, содержащееся в сообщении, записывается в выходной массив, при этом значения переменных  $a_p$  и  $s_p$  изменяются.

$L_p$ :

Моделирование потери сообщения проводится путем удаления произвольного сообщения из множества сообщений  $Q_p$ , пребывающих на этапе пересылки от процесса  $q$  к процессу  $p$ .

Что плохого может случиться?

1. Створки окон обоих процессов могут “захлопнуться”, и процессы будут обречены (безуспешно) ожидать сообщений друг от друга (блокировка , deadlock );
2. Створки окон могут “застыть”, и процессы будут обречены передавать одни и те же сообщения (активный тупик , livelock );
3. Данные могут быть потеряны при передаче, и процесс не заметит этого;
4. Процесс может “забыть” передать данные;
5. Створки окна могут раздвигаться, отдаляясь друг от друга неограниченно широко.

**Требования корректности.** Нужно доказать, что протокол работает правильно, т.е. каждое слово из входного массива  $in_p$  процесса  $p$  будет рано или поздно записано в выходной массив  $out_q$  процесса  $q$  , и наоборот. Более строго это выражается двумя требованиями.

1. Безопасная доставка сообщений. В каждой достижимой конфигурации протокола выполняются соотношения  $out_p[0..s_p - 1] = in_q[0..s_p - 1]$  и  $out_q[0..s_q - 1] = in_p[0..s_q - 1]$  .
2. Неизбежная доставка сообщений. Для каждого целого числа  $k \geq 0$  , в ходе выполнения протокола будет достигнута конфигурация, в которой  $s_p \geq k$ ,  $s_q \geq k$  .

Многие свойства распределенных алгоритмов, нуждающиеся в проверке, относятся к одному из двух типов: условие безопасности и условие живости.

Условие безопасности требует, чтобы каждая достижимая конфигурация в любом выполнении системы обладала определенным свойством. Условие живости требует, чтобы хотя бы одна достижимая конфигурация в любом выполнении системы обладала определенным свойством.

Ограничения накладываемые, чтобы протокол обладал свойством живости:

- В качестве  $l_p$  и  $l_q$  можно взять любые неотрицательные константы, удовлетворяющие неравенству  $l_p + l_q > 0$  .
- Выдвигаются два требования справедливости:
  1. F1. Если бесконечно часто возникает возможность отправки пакета, то этот пакет будет отправляться бесконечно часто.
  2. F2. Если один и тот же пакет отправляется бесконечно часто, то и принимается он также бесконечно часто.

### Теорема.

Симметричный протокол раздвижного окна удовлетворяет требованию безопасной доставки сообщений, т.е. в каждой достижимой конфигурации протокола выполняются соотношения  $out_p[0..s_p - 1] = in_q[0..s_p - 1]$  и  $out_q[0..s_q - 1] = in_p[0..s_q - 1]$

Доказательство:

Следует из Теоремы 3.2 (о свойстве инвариантов) и Теоремы 3.4. Из условий

20. Коммуникационные протоколы. Ошибки, возникающие при передаче сообщений. Задача надежного обмена сообщениями. Симметричные протокол скользящего (раздвижного) окна: устройство протокола и обоснование его корректности. Протокол альтернирующего бита.

---

(0p):  $\forall i < s_p : out_p[i] = undef$  и

(2p):  $\forall i : out_p[i] = undef \Rightarrow out_p[i] = in_q[i] \wedge (a_p > i - l_q)$  следует выполнимость равенства  $out_p[0..s_p - 1] = in_q[0..s_p - 1]$ , а из условий (0q) и (2q) следует выполнимость равенства  $out_q[0..s_q - 1] = in_p[0..s_q - 1]$ .

Вспомогательные теоремы (без доказательства)

Теорема 3.2. Если  $Q$  является инвариантом системы переходов  $S$ , и для каждой конфигурации  $\gamma \in C$  выполняется  $Q(\gamma) \Rightarrow P(\gamma)$ , то для любого выполнения системы  $S$  утверждение  $P$  будет истинно в каждой конфигурации выполнения.

Теорема 3.4. Утверждение Р является инвариантом алгоритма раздвижного окна.

$$\begin{aligned} P \equiv & \forall i < s_p : out_p[i] = undef \\ & \wedge \forall i < s_q : out_q[i] = undef \\ & \wedge \forall i < pack, w, i \in Q_p \Rightarrow w = in_q[i] \wedge (i < s_q + l_q) \\ & \wedge \forall i < pack, w, i \in Q_q \Rightarrow w = in_p[i] \wedge (i < s_p + l_p) \\ & \wedge \forall i : out_p[i] = undef \Rightarrow out_p[i] = in_q[i] \wedge (a_p > i - l_q) \\ & \wedge \forall i : out_q[i] = undef \Rightarrow out_q[i] = in_p[i] \wedge (a_q > i - l_p) \\ & a_p \leq s_q \\ & a_q \leq s_p \end{aligned}$$

### Теорема

Симметричный протокол раздвижного окна удовлетворяет требованию неизбежной доставки сообщений, т.е. для каждого целого числа  $k \geq 0$ , в ходе любого выполнения протокола будет достигнуты конфигурация, в которой  $spk, sqk$ .

Доказательство:

Предположим, что есть вычисление  $C$ , в котором значения хотя бы одной из переменных  $s_p$  и  $s_q$  увеличиваются лишь конечное число раз. Тогда, согласно неравенству Леммы 3.1.  $s_p - l_q \leq s_q \leq s_p + l_p$ , значение другой переменной также не может увеличиваться бесконечно часто. Пусть  $\sigma_p$  и  $\sigma_q$  - наибольшие значения переменных  $s_p$  и  $s_q$ . Тогда, согласно Лемме 3.2., либо отправление пакета  $< pack, in_p[\sigma_q], \sigma_q >$  процессом  $p$ , либо отправление пакета  $< pack, in_q[\sigma_p], \sigma_p >$  процессом  $q$  допустимо бесконечно долго после того, как переменные  $s_p$ ,  $s_q$ ,  $a_p$  и  $a_q$  примут свои окончательные значения. Тогда, согласно допущению F1, один из этих пакетов (например,  $< pack, in_q[\sigma_p], \sigma_p >$ ) отправляется бесконечно часто, и, согласно допущению F2, он должен приниматься также бесконечно часто. Получение процессом  $p$  пакета  $< pack, in_q[\sigma_p], \sigma_p >$  приводит к тому, что значение  $s_p$  (равное  $\sigma_p$ ) увеличивается. Это противоречит выбору значения  $\sigma_p$ .

Леммы будут приведены без доказательств

Лемма 3.1. В любой достижимой конфигурации выполняются неравенства  $s_p - l_q \leq a_p \leq s_q \leq a_q + l_p \leq s_p + l_p$ . (Створки окон процессов  $p$  и  $q$  “разъезжаются” не слишком далеко друг от друга.)

Лемма 3.2. В любой достижимой конфигурации допустимо хотя бы одно из двух действий: отправление пакета  $< pack, in_p[s_q], s_q >$  процессом  $p$  или отправление пакета  $< pack, in_q[s_p], s_p >$  процессом  $q$ .

### Протокол альтернирующего бита

Особо интересный вариант протокола раздвижного окна возникает в том случае, когда  $l_p = 1$  и  $l_q = 0$  (или наоборот). В качестве начальных значений переменных  $a_p$  и  $a_q$  в этом случае выбирается не 0, а числа  $-l_p$  и  $-l_q$ . Такой вариант алгоритма раздвижного окна называется протоколом чередующихся (альтернирующих) битов; он предназначен для односторонней передачи данных.

20. Коммуникационные протоколы. Ошибки, возникающие при передаче сообщений. Задача надежного обмена сообщениями. Симметричные протокол скользящего (раздвижного) окна: устройство протокола и обоснование его корректности. Протокол альтернирующего бита.

---

## 21. Задача маршрутизации. Алгоритм Флойда-Уоршалла построения кратчайших путей в графе. Алгоритм маршрутизации Туэга: описание алгоритма, обоснование его корректности и оценка сложности по числу обменов сообщениями.[1, стр. 113-128]

**Маршрутизация** - это процедура принятия решения о том, кому (иногда не единственному) следует передать пакет, чтобы он в конце концов был доставлен по назначению.

Цель, которая ставится при проектировании алгоритма маршрутизации, состоит в том, чтобы снабдить каждый узел процедурой, которая сможет выполнять эту функцию и гарантировать доставку каждого пакета по назначению. Для этого на каждом узле должна храниться некоторая информация о топологии сети. Эта информация предоставляется в виде **таблицы маршрутизации**.

Задача маршрутизации:

- Вычисление таблиц. Таблицы маршрутизации должны быть вычислены при инициализации сети и должны обновляться при изменении топологии сети.
- Продвижение пакета. Когда пакет пересыпается по сети, то его передвижение осуществляется на основе таблиц маршрутизации.

Критерии оценки качества методов маршрутизации учитывают след. показатели:

- Корректность
- Эффективность
- Сложность
- Устойчивость
- Адаптивность
- Справедливость

Существует несколько вариантов понятия *лучший путь*:

- минимальное количество звеньев
- минимальная стоимость
- минимальная задержка

Выбор маршрута обычно проводится только с учетом узла-адресата.

### Алгоритм Флойда-Уоршалла

Алгоритм нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе. Работает корректно, если в графе нет циклов отрицательной величины. Алгоритм работает за  $O(n^3)$  времени.

Предположим, что имеется взвешенный граф  $G = (V, E)$ , в котором каждому ребру  $uv$  приписан вес  $\omega_{uv}$ . Мы будем считать, что в графе отсутствуют циклы отрицательного веса. **Вес пути**  $< u_0, \dots, u_k >$  - это число, равное  $\sum_{i=0}^{k-1} w_{u_i u_{i+1}}$ . **Расстояние** между вершинами  $u$  и  $v$  - это наименьший вес  $d(u, v)$  пути, соединяющего вершины  $u$  и  $v$  (если таких путей нет, то  $d(u, v) = \infty$ ). Задача построения кратчайших путей для всех пар вершин состоит в том, чтобы вычислить расстояние  $d(u, v)$  для каждой пары вершин  $u$  и  $v$ .

Для вычисления всех расстояний в алгоритме Флойда-Уоршалла используется понятие S-путей, в которых все промежуточные вершины принадлежат подмножеству S множества вершин V. **Определение S-расстояния** Пусть S некоторое подмножество множества вершин V. Путь  $< u_0, \dots, u_k >$  называется S-путем, если  $u_i \in S$  для каждого  $i, 0 <$

21. Задача маршрутизации. Алгоритм Флойда–Уоршалла построения кратчайших путей в графе. Алгоритм маршрутизации Туэга: описание алгоритма, обоснование его корректности и оценка сложности по числу обменов сообщениями.

$i < k$ . S-расстоянием между вершинами  $u$  и  $v$ , которое обозначается  $d^S(u, v)$ , называется наименьший вес S-пути между  $u$  и  $v$  (если таких путей нет, то  $d^S(u, v) = \infty$ ). Работа алгоритма начинается с построения всех  $\emptyset$ -путей, а затем множество S-путей наращивается для все более обширных подмножеств S, до тех пор пока не будут рассмотрены все V-пути.

**Теорема 5.2. (об S-путях)** (думаю, не обязательно писать в билете) Для всех вершин  $u$  и подмножеств S выполняется равенство  $d^S(u, u) = 0$ . Если  $u = v$ , то S-пути подчиняются следующим правилам.

1.  $\emptyset$ -путь из вершины  $u$  в вершину  $v$  существует в том и только том случае, когда  $uv \in E$ .
2. Если  $uv \in E$ , то  $d^\emptyset(u, v) = \omega_{uv}$ ; иначе  $d^\emptyset(u, v) = \infty$ .
3. Если  $S' = S \cup \{w\}$ , то простой  $S'$ -путь из  $u$  в  $v$  – это либо S-путь из  $u$  в  $v$ , либо соединение двух S-путей, один из которых ведет из  $u$  в  $w$ , а другой из  $w$  в  $v$ .
4. Если  $S' = S \cup \{w\}$ , то  $d^{S'}(u, v) = \min(d^S(u, v), d^S(u, w) + d^S(w, v))$ .
5. Вершины  $u$  и  $v$  соединены путем в том и только том случае, когда между вершинами  $u$  и  $v$  существует V-путь.
6.  $d(u, v) = d^V(u, v)$

**Алгоритм:** Вначале множество S – пустое. Потом на каждом шаге алгоритма берем новую вершину из V. Для всех пар вершин из V пересчитываем расстояния d. Новая вершина добавляется в S. Так пока S не равно V.

Код:

## Алгоритм Флойда–Уоршалла

```
begin(* Вначале S равно  $\emptyset$ , а в D заданы  $\emptyset$ -расстояния *)
  S :=  $\emptyset$  ;
  forall u, v do
    if  $u = v$  then  $D[u, v] := 0$ 
    else if  $uv \in E$  then  $D[u, v] := \omega_{uv}$  else  $D[u, v] := \infty$  ;
    (* Добавить к S опорную вершину *)
  while S ≠ V do
    (* Инвариант цикла:  $\forall u, v : D[u, v] = d^S(u, v)$  *)
    begin pick w from  $V \setminus S$  ;
    (* Глобальная обработка опорной вершины w *)
      forall u ∈ V do
        (* Локальная обработку опорной вершины w для u *)
          forall v ∈ V do
             $D[u, v] := \min(D[u, v], D[u, w] + D[w, v])$  ;
            S := S ∪ {w}
        end (*  $\forall v : D[u, v] = d(u, v)$  *)
      end
    end
```

## Алгоритм маршрутизации Туэга

В основу алгоритма Туэга (Toueg) положен алгоритм Флойда–Уоршалла.

Допущения:

1. Каждый цикл имеет положительный вес
2. Каждый процесс сети располагает информацией об отличительных признаках всех узлов системы (множества

21. Задача маршрутизации. Алгоритм Флойда-Уоршалла построения кратчайших путей в графе. Алгоритм маршрутизации Туэга: описание алгоритма, обоснование его корректности и оценка сложности по числу обменов сообщениями.

---

вершин V)

3. Каждый процесс знает, кто его соседи (для каждой вершины  $u$  эти сведения содержатся в массиве  $neigh_u$ ) и вес соединяющих их каналов

Переменные и операции алгоритма Флойда-Уоршалла разносятся по разным узлам сети. Переменная  $D[u, v]$  отдается процессу  $u$ ; для удобства обозначения мы будем записывать  $u$  на месте индекса следующим образом  $D_u[v]$ . Операции, которые присваивают значения переменной  $D_u[v]$  должны выполняться в узле  $u$ , и когда значение некоторой переменной, относящейся к узлу  $w$ , необходимо для этой операции, это значение должно быть послано процессу  $w$ . В алгоритме Флойда-Уоршалла все вершины должны использовать информацию от опорной вершины, которая отправляет эту информацию одновременно всем вершинам посредством широковещательной рассылки. Нужно ввести специальную операцию, для того чтобы не только вычислять длины кратчайших S-путей, но также и первый канал в каждом таком пути (для этого мы будем использовать переменные  $Nb_u[v]$ ).

### Лемма 5.3. (об отсутствии циклов)

Пусть заданы множество вершин S и вершина w. Предположим также, что

1. для всех вершин  $u$  верно равенство  $D_u[w] = d^S(u, w)$ ,
2. если  $d^S(u, w) < \infty$  и  $u = w$ , то значением переменной  $Nb_u[w]$  является имя соседа вершины  $u$  на кратчайшем S-пути, ведущем к вершине  $w$ .

Тогда ориентированный граф  $T_w = (V_w, E_w)$ , где  $V_w = \{u : D_u[w] < \infty\}$  и  $E_w = \{ux : u = w \wedge Nb_u[w] = x\}$ , является деревом, в корне которого расположена вершина  $w$ .

Туэг заметил, что если  $D_u[w] = \infty$  в начале этапа обработки опорной вершины  $w$ , то по окончании этого этапа таблица маршрутизации узла  $u$  не изменяется, т.к. неравенство  $D_u[w] + D_w[v] < D_u[v]$  неверно для каждой вершины  $v$ . Поэтому таблицу маршрутизации  $D_w$  нужно доставить только в те узлы, которые принадлежат дереву  $T_w$  (в том виде, в котором оно построено к началу этапа обработки опорной вершины), и широковещательную рассылку можно провести эффективно, отправляя таблицу  $D_w$  только по тем каналам, которые входят в состав дерева  $T_w$ . Узел  $w$  отправляет таблицу  $D_w$  своим сыновним вершинам в дереве  $T_w$ , и каждый узел в дереве  $T_w$ , получив эту таблицу от родительской вершины в дереве  $T_w$ , передает ее своим сыновним вершинам в дереве  $T_w$ .

В начале этапа обработки опорной вершины  $w$  каждый узел  $u$ , для которого  $D_u[w] < \infty$ , знает, какая вершина является его родителем в дереве  $T_w$ , но не знает, какие вершины являются его сыновьями. Поэтому каждый узел  $v$  должен отправить сообщение каждому своему соседу  $u$ , сообщив процессу  $u$ , является ли  $v$  сыновней вершиной для  $u$  в дереве  $T_w$ . Любой узел может принять участие в распространении таблицы опорной вершины  $w$ , как только он получит известия о том, какие из соседей являются его сыновними вершинами в дереве  $T_w$ .

В алгоритме используются сообщения трех типов:

1. Сообщения  $<ys, w>$  (ys от your son) отправляются от узла  $u$  к узлу  $x$  в начале этапа обработки опорной вершины  $w$ , если  $x$  является родительской вершиной для  $u$  в дереве  $T_w$ .
2. Сообщение  $<nys, w>$  (nys от not your son) отправляется от узла  $u$  к узлу  $x$  в начале этапа обработки опорной вершины  $w$ , если  $x$  не является родительской вершиной для  $u$  в дереве  $T_w$ .
3. Сообщение  $<dtab, w, D>$  отправляется по ходу обработки опорной вершины  $w$  по каждому ребру дерева  $T_w$ , чтобы доставить таблицу  $D_w$  в каждую вершину, которая должна будет воспользоваться этим значением.

Пусть  $W$  число битов для записи веса пути.

**Теорема 5.5. (корректности и сложности алгоритма Туэга)** Для каждой пары вершин  $u$  и  $v$  алгоритм Туэга вычисляет расстояние между  $u$  и  $v$ . Если это расстояние конечно, то он также определяет первый канал в кратчайшем пути. По ходу работы алгоритма по каждому каналу проходит  $O(N)$  сообщений,  $O(N^2 W)$  битов информации. Таким образом, суммарно по ходу работы алгоритма передается  $O(N \cdot |E|)$  сообщений и  $O(N^3 \cdot W)$  битов информации. Кроме того в каждом узле используется память, объем которой составляет  $O(N \cdot W)$  битов.

**Доказательство** На каждом этапе обработки опорной вершины  $w$  по каждому каналу связи проходят два сообщения вида  $<ys, w>$  или  $<nys, w>$  (по одному сообщению в каждом направлении) и не более одного сообщения вида  $<dtab, w, D>$ , и значит по каждому каналу проходит не более  $3N$  сообщений. Сообщения вида  $<ys, w>$  или  $<nys, w>$  содержат  $O(W)$  битов, а сообщение вида  $<dtab, w, D>$  содержит  $O(NW)$  битов, и отсюда следует верхняя оценка битовой сложности обмена информацией по каждому каналу связи. За время работы алгоритма передается не более  $N^2$  сообщений вида  $<dtab, w, D>$  и  $2N \cdot |E|$  сообщений вида  $<ys, w>$  и  $<nys, w>$ ; таким образом всего по сети передается  $O(N^2 \cdot NW + 2N \cdot |E| \cdot W) = O(N^3 W)$  битов информации. Для хранения таблиц  $D_u$  и  $Nb_u$  каждому процессу  $u$  требуется  $O(NW)$  битов памяти.

Завершаемость и частичная корректность алгоритма следуют из корректности алгоритма Флойда-Уоршалла. Отсюда следует и справедливость утверждения о том, что значением переменной  $Nb_u[v]$  является наименование первого канала в кратчайшем пути из  $u$  в  $v$  или  $Nb_u[v] = udef$ , поскольку значение  $Nb_u[v]$  изменяется всякий раз, когда переменной  $D_u[v]$  присваивается новое значение.

(На самом деле для доказания корректности в презентациях вводится еще и упрощенный алгоритм Туэга, но это уже совсем долго)

(Не обязательно)

**Достоинства:**

- Прост, имеет небольшую сложность, и строит оптимальные маршруты.

**Недостатки:**

- Плохая устойчивость («робастость»): при изменении топологии сети все вычисления нужно проводить заново.
- Согласованный выбор очередной опорной вершины ( $w$ ) всеми узлами сети предполагает, что множество участвующих в алгоритме процессов заранее известно.
- В алгоритме Туэга часто применяется неравенство треугольника  $d(u, v) < d(u, w) + d(w, v)$ . Для вычисления правой части этого неравенства (в узле  $u$ ) требуется «глобальная» информация о  $d(w, v)$ , которой не обладает ни процесс  $u$ , ни его соседи. Зависимость от удаленных данных вынуждает нас организовать доставку этой информации удаленным вершинам.

## 22. Общие принципы дедуктивной верификации программ. Операционная семантика императивных программ. Формальная постановка задачи верификации программ. Логика Хоара: правила вывода и свойства. Автоматизация проверки правильности программ.

Общие принципы дедуктивной верификации:

1. Программа  $\pi$  вычисляет отношение  $R_\pi$  между данными, подаваемыми на вход и получаемыми на выходе
2. Текст программы  $\pi$  — это формальное описание отношения  $R_\pi$
3. Спецификация программы  $\Phi$  — это формальное описание отношения  $R_\Phi$  между данными программы
  - Отношение, описываемое спецификацией, — это требования, которым должно удовлетворять отношение, вычисляемое программой
4. Формальная верификация программы  $\pi$  относительно спецификации  $\Phi$  — это строгое доказательство того, что программа  $\pi$  удовлетворяет требованиям  $\Phi$ , то есть доказательство включения  $R_\pi \subseteq R_\Phi$

Чтобы уметь формально верифицировать программы, нужно:

1. Страно описать, какие записи мы считаем программами (синтаксис программ) и как программы преобразуют входные данные в выходные (семантику программ)
2. Выбрать формальный язык описания требований к программам
3. Предложить метод проверки того, удовлетворяет ли заданная программа предъявленным к ней требованиям

Синтаксис императивных программ (сигнатуры  $\sigma$ ) задаётся следующей формой Бэкуса-Наура:

$\pi ::= \text{instr} \mid \text{instr}; \pi$

$\text{instr} ::= \emptyset \mid x := t \mid \text{if } C \text{ then } \pi \text{ else } \pi \text{ fi} \mid \text{while } C \text{ do } \pi \text{ od}$

Где

- $\pi$  — программа
- $\text{instr}$  — инструкция (пустая инструкция, присваивание, ветвление, цикл)
- $x$  — переменная
- $t$  — терм
- $C$  — условие: формула, не содержащая кванторов

**Операционная семантика:**

- Вычисление программы — это последовательное изменение состояния вычисления
- Программой задаётся отношение переходов, описывающее, какое состояние вычисления будет получено следующим из произвольного текущего состояния
- Значение программы — это функция преобразования входных данных в выходные, определяемая на основе транзитивного замыкания отношения переходов
- Хорошо подходит для описания значения императивных программ

**Операционная семантика для императивных программ:**

**Состояние управления** — это произвольная программа

**Состояние данных (оценка переменных)** — это подстановка, отображающая каждую переменную программы в терм, не содержащий переменных

**Состояние вычисления** — это пара  $\langle \pi, \theta \rangle$ , где  $\pi$  — состояние управления, а  $\theta$  — состояние данных

**Отношение переходов** → на множестве состояний вычисления для программы  $\pi$  в интерпретации  $I$  определяется так:

- если  $\langle x := t, \theta \rangle \rightarrow \langle \emptyset, \{x/t\}\theta \rangle$
- если  $I \models C\theta$ , то:  $\langle \text{if } C \text{ then } \pi_1 \text{ else } \pi_2, \theta \rangle \rightarrow \langle \pi_1, \theta \rangle$
- если  $I \not\models C\theta$ , то:  $\langle \text{if } C \text{ then } \pi_1 \text{ else } \pi_2, \theta \rangle \rightarrow \langle \pi_2, \theta \rangle$
- если  $I \models C\theta$ , то:  $\langle \text{while } C \text{ do } \pi \text{ od}, \theta \rangle \rightarrow \langle \pi; \text{while } C \text{ do } \pi \text{ od}, \theta \rangle$
- если  $I \not\models C\theta$ , то:  $\langle \text{while } C \text{ do } \pi \text{ od}, \theta \rangle \rightarrow \langle \emptyset, \theta \rangle$
- если  $\langle \pi_1, \theta \rangle \rightarrow \langle \pi'_1, \eta \rangle$ , то  $\langle \pi_1 ; \pi_2, \theta \rangle \rightarrow \langle \pi'_1 ; \pi_2, \eta \rangle$
- $\langle \emptyset; \pi, \theta \rangle \rightarrow \langle \pi, \theta \rangle$

**Трасса** программы  $\pi$  на оценке  $\theta$  в интерпретации  $I$  — это последовательность состояний вычисления вида  $\langle \pi, \theta \rangle \rightarrow \langle \pi_1, \theta_1 \rangle \rightarrow \langle \pi_2, \theta_2 \rangle \rightarrow \dots$

**Вычисление программы**  $\pi$  на оценке  $\theta$  в интерпретации  $I$  — это максимальная по длине трасса  $\pi$  на  $\theta$  в  $I$

**Результат конечного вычисления** программы  $\pi$  на оценке  $\theta$  в интерпретации  $I$  — это оценка данных последнего состояния вычисления  $\pi$  на  $\theta$  в  $I$ , иными словами, если  $\langle \pi, \theta \rangle \rightarrow^* \langle \emptyset, \eta \rangle$ , то  $\eta$  — результат вычисления  $\pi$  на  $\theta$  в  $I$   
 $(\rightarrow^* — \text{транзитивное замыкание отношения} \rightarrow)$

#### Формальная постановка задачи верификации

Предусловие  $\phi$  и постусловие  $\psi$  — это формулы логики предикатов, а  $\pi$  — императивная программа

Требование корректности  $\pi$  относительно  $\phi, \psi$  записывается в виде триплета Хоара (или тройки Хоара):  $\{\phi\}\pi\{\psi\}$

Триплет Хоара  $\{\phi\}\pi\{\psi\}$  истинен в интерпретации  $I$  ( $I \models \{\phi\}\pi\{\psi\}$ ), если для любых оценок  $\theta, \eta$  верно: если  $I \models \phi\theta$  и  $\langle \pi, \theta \rangle \rightarrow^* \langle \emptyset, \eta \rangle$ , то  $I \models \psi\eta$

Программа  $\pi$  частично корректна в интерпретации  $I$  относительно предусловия  $\phi$  и постусловия  $\psi$ , если  $I \models \{\phi\}\pi\{\psi\}$

#### Правила вывода

$$\text{SKIP: } \frac{}{\text{true}}$$

$$\text{AS: } \frac{\{\varphi\} \{x/t\} \ x := t \ \{\varphi\}}{\text{true}} \\ (\text{переменная } x \text{ свободна} \\ \text{для терма } t \text{ в формуле } \varphi)$$

$$\text{INF: } \frac{\{\varphi\} \pi \{\psi\}}{\varphi \rightarrow \varphi', \{\varphi'\} \pi \{\psi'\}, \psi' \rightarrow \psi}$$

$$\text{COMP: } \frac{\{\varphi\} \pi_1; \pi_2 \{\psi\}}{\{\varphi\} \pi_1 \{\chi\}, \{\chi\} \pi_2 \{\psi\}}$$

$$\text{IF: } \frac{\{\varphi\} \text{ if } C \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi } \{\psi\}}{\{\varphi \& C\} \pi_1 \{\psi\}, \{\varphi \& \neg C\} \pi_2 \{\psi\}}$$

$$\text{WHILE: } \frac{\{\varphi\} \text{ while } C \text{ do } \pi \text{ od } \{\varphi \& \neg C\}}{\{\varphi \& C\} \pi \{\varphi\}}$$

#### Теорема корректности правил вывода Хоара

Для любой интерпретации  $I$  и любого правила вывода логики Хоара (SKIP, AS, INF, COMP, IF, WHILE)

$$\frac{\Phi}{\Psi_1}, \quad \frac{\Phi}{\varphi}, \quad \frac{\Phi}{\Psi_1, \Psi_2}, \quad \frac{\Phi}{\varphi, \Psi_1, \psi}$$

если  $I \models \Psi_1, I \models \Psi_2, I \models \phi$  и  $I \models \psi$ , то  $I \models \Phi$

**Аннотация** — это запись вида  $\{\phi\}$ , где  $\phi$  — произвольная формула

**Аннотированная программа** — это программа, в которой до и после каждой инструкции могут располагаться последовательности аннотаций

Аннотация может расцениваться как

- предусловие инструкции, следующей за аннотацией
- постусловие инструкции, предшествующей аннотации
- составная часть триплета, располагающегося в выводе (некоторого исходного триплета)

**Теорема.** Если существует аннотированная программа  $\pi$ , обосновывающая истинность триплета  $\{\phi\}\pi\{\psi\}$  в интерпретации  $I$ , то  $I \models \{\phi\}\pi\{\psi\}$

Проблема корректности программ в общем случае неразрешима

**Слабейшее предусловие** для программы  $\pi$  и постусловия  $\psi$  в интерпретации  $I$  — это формула  $wpr(\pi, \psi, I)$ , для которой выполнены следующие условия:

1.  $I \models \{wpr(\pi, \psi, I)\} \pi \{\psi\}$
2. Для любой формулы  $\chi$ , такой что  $I \models \{\chi\}\pi\{\psi\}$ , верно  $I \models \chi \rightarrow wpr(\pi, \psi, I)$

**Теорема.**  $I \models \{\phi\}\pi\{\psi\} \Leftrightarrow I \models \phi \rightarrow wpr(\pi, \psi, I)$

Для полной автоматизации проверки корректности программ достаточно иметь два алгоритма:

1. Алгоритм проверки истинности формул логики предикатов
2. Алгоритм вычисления слабейшего предусловия для произвольных программ и постуловий

Первый алгоритм зависит от выбора интерпретации программ, и в общем случае может не существовать.

Со вторым алгоритмом всё немного лучше:

**Теорема.**

- $wpr(x:=t, \psi) = \psi\{x/t\}$
- $wpr(\pi_1 ; \pi_2, \psi) = wpr(\pi_1, wpr(\pi_2, \psi))$
- $wpr(\text{if } C \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \psi) = C \& wpr(\pi_1, \psi) \vee \neg C \& wpr(\pi_2, \psi)$

Для применения правила WHILE (а также для аннотации цикла, и для вычисления слабейшего предусловия цикла) необходимо предварительно найти подходящую формулу  $\phi$  — инвариант цикла

## 23. Темпоральная логика деревьев вычислений CTL.

### Синтаксис и семантика CTL. Примеры спецификаций моделей в терминах формул CTL. Темпоральная логика линейного времени PLTL. Синтаксис и семантика PLTL. Свойства живости и безопасности. Ограничения справедливости. Задача верификации моделей (model-checking).

Модель Кripке  $M$  над множеством атомарных высказываний  $AP$  это четверка  $M = (S, S_0, R, L)$ :

- $S$  - конечное множество состояний
- $S_0 \subseteq S$  - множество начальных состояний
- $R \subseteq S \times S$  - отношение переходов, которое должно быть тотальным ( для каждого состояния  $s \in S$  должно существовать такое состояние  $s' \in S$ , что имеет место  $R(s, s')$ )
- $L : S \rightarrow 2^{AP}$  - функция разметки, которая помечает каждое состояние множеством атомарных высказываний, истинных в этом состоянии.

**Трасса** - последовательность событий (вычисление системы)

**Свойство системы** - множество трасс

Свойство вычислений  $P$  называется **свойством безопасности**, если оно удовлетворяет следующему требованию: какова бы ни была трасса  $\alpha$ ,  $\alpha \in (2^{AP})^\omega \setminus P$ , существует такой ее конечный префикс  $\beta$ , что для любой трассы  $\alpha'$  выполняется соотношение  $\beta\alpha' \notin P$ .

Содержательный смысл: "отсутствие безопасности - это когда если что-то плохое случилось, то этого уже не исправить"

Свойство вычислений  $P$  называется **свойством живости**, если оно удовлетворяет следующему требованию: для любого конечного слова  $\beta$ ,  $\beta \in (2^{AP})^*$ , существует такая трасса  $\alpha$ ,  $\alpha \in (2^{AP})^\omega$ , для которой выполняется соотношение  $\beta\alpha \in P$ .

Содержательный смысл: "что бы ни случилось в начале, потом всегда можно достичь своей цели"

Различают два типа **ограничений справедливости** ( условий, которым должны удовлетворять пути в моделях Кripке, чтобы эти пути соответствовали вычислениям, построенным по принципу чередования - важно для асинхронно работающих систем, чтобы только один процесс не выполнялся бесконечно часто).

- Слабая справедливость : если путь всегда проходит через состояния, в которых может быть выполнено действие  $act$  , то действие  $act$  должно быть выполнено бесконечно часто.
- Сильная справедливость : если путь бесконечно часто проходит через состояния, в которых может быть выполнено действие  $act$  , то действие  $act$  должно быть выполнено бесконечно часто.

Темпоральные логики предназначены для описания свойств вычислений реагирующих систем, т.е. множеств трасс в размеченных системах переходов (моделях Кripке).

**CTL\***

**Синтаксис**

В CTL имеются формулы двух типов: **формулы состояния** (способные обращаться в истину в некотором состоянии) и **формулы пути** (способные быть истинными на протяжении некоторого пути). Формулами CTL называются все формулы состояний, полученные по следующим правилам.

Формулы состояния:

- $p \in AP$ ,  $p$  - формула состояния
- $f_1, f_2$  - формулы состояния, то  $\neg f_1, f_1 \wedge f_2, f_1 \vee f_2$  - формулы состояния
- $f$  - формула пути, то  $\mathbf{E}f$  и  $\mathbf{A}f$  - формулы состояния

Формулы пути:

- $f$  - формула состояния, то  $f$  - формула пути
- $g_1, g_2$  - формулы пути, то  $\neg g_1, g_1 \wedge g_2, g_1 \vee g_2, \mathbf{X}g_1, \mathbf{F}g_1, g_1 \mathbf{U}g_2, g_1 \mathbf{R}g_2$  - формулы пути

### Семантика

Формул состояния:

- $M, s \models p \Leftrightarrow p \in L(s)$
- $M, s \models \neg f_1 \Leftrightarrow M, s \not\models f_1$
- $M, s \models f_1 \vee f_2 \Leftrightarrow M, s \models f_1$  или  $M, s \models f_2$
- $M, s \models f_1 \wedge f_2 \Leftrightarrow M, s \models f_1$  и  $M, s \models f_2$
- $M, s \models \mathbf{E}f \Leftrightarrow$  в  $M$  есть такой путь  $\pi$  из состояния  $s$ , что  $M, \pi \models f$  (формула выполняется на протяжении пути)
- $M, s \models \mathbf{A}f \Leftrightarrow$  в  $M$  для любого пути  $\pi$  из состояния  $s$ , что  $M, \pi \models f$

Формул пути:

- $M, \pi \models g_1 \Leftrightarrow$  для первого состояния  $s$  на пути  $\pi$  в  $M$  верно  $M, s \models g_1$
- $M, \pi \models \neg g_1 \Leftrightarrow M, \pi \not\models g_1$
- $M, \pi \models g_1 \vee g_2 \Leftrightarrow M, \pi \models g_1$  или  $M, \pi \models g_2$
- $M, \pi \models g_1 \wedge g_2 \Leftrightarrow M, \pi \models g_1$  и  $M, \pi \models g_2$
- $M, \pi \models \mathbf{X}g_1 \Leftrightarrow M, \pi^1 \models g_1$  ( $\pi^i$  - это суффикс пути  $\pi$  начинающийся из состояния  $s_i$ )
- $M, \pi \models \mathbf{F}g_1 \Leftrightarrow$  существует такое  $k \geq 0$ , что  $M, \pi^k \models g_1$
- $M, \pi \models \mathbf{G}g_1 \Leftrightarrow$  для любого  $k \geq 0$  верно  $M, \pi^k \models g_1$
- $M, \pi \models g_1 \mathbf{U}g_2 \Leftrightarrow$  существует такое  $k \geq 0$ , что верно  $M, \pi^k \models g_2$  и для каждого  $0 \leq j < k$  верно  $M, \pi^j \models g_1$
- $M, \pi \models g_1 \mathbf{R}g_2 \Leftrightarrow$  каково бы ни было  $j \geq 0$ , если для каждого  $i < j$  верно  $M, \pi^i \not\models g_1$ , то  $M, \pi^j \models g_2$

CTL и LTL являются подмножествами CTL\*.

### CTL

В логике ветвящегося времени темпоральные операторы находятся непосредственно под действием кванторов по тем путям, которые исходят из заданного состояния. То есть каждый темпоральный оператор **X**, **F**, **G**, **U**, **R** должен следовать непосредственно за квантором пути **A**, **E**.

### Примеры спецификаций CTL:

- **EF** (Start  $\wedge \neg$  Ready): можно достичь такого состояния, в котором условие Start выполняется, а Ready - нет;
- **AG** (Req  $\rightarrow$  AF Ack): когда бы ни был получен запрос, он рано или поздно будет подтвержден;
- **AG** (AF DeviceEnabled): условие DeviceEnabled выполняется бесконечно часто на каждом пути вычисления;
- **AG** (EF Restart): из любого достижимого состояния достижимо состояние Restart.

### LTL

В логике линейного времени операторы предназначены для описания событий на протяжении единственного пути вычисления. Состоит из всех формул вида **Af**, где  $f$  - формула пути, в которой все формулы состояния - это атомарные высказывания.

### Примеры спецификаций LTL:

- **A** (FG enabled  $\rightarrow$  GF fired)

### Задача model-checking

Пусть задана модель Кripке  $M = (S, S_0, R, L)$ , и формула темпоральной логики  $\phi$ , которая выражает некоторую желаемую спецификацию. Требуется найти в множестве  $S$  подмножество  $S_\phi$  всех состояний  $s$ , в которых выполняется  $\phi$ , т.е. множество  $S_\phi = \{s \in S | M, s \models \phi\}$ , и проверить выполнимость включения  $S_0 \subseteq S_\phi$ .

## 24. Временные автоматы как формальные модели распределенных систем реального времени. Вычисления временных автоматов. Примеры использования временных автоматов для моделирования встроенных систем. Зеноновские вычисления. Синтаксис и семантика TimedCTL. Задача верификации моделей программ реального времени. Программно-инструментальное средство верификации моделей программ реального времени UPPAAL.

**Система реального времени** (CPB) — это система, поведение которой существенно зависит не только от того, в каком порядке изменяются состояния компонентов системы, но и от того, за какое время происходит изменение состояний.

**Элементарными временными ограничениями** над множеством часов  $C$  называются выражения вида:  $\text{true}$ ,  $x < k$ ,  $x \leq k$ ,  $x - y < k$ ,  $x - y \leq k$ , где  $x, y \in C$ ,  $k \in N_0$

$ETC(C)$  — множество всех элементарных временных ограничений над  $C$

Синтаксис **временных ограничений** над  $C$  задаётся формой Бэкуса-Наура  $g ::= (\text{etc}) \mid (g \& g) \mid (\neg g)$ , где  $g$  — временное ограничение и  $\text{etc} \in ETC(C)$

$TC(C)$  — множество всех временных ограничений над  $C$

Временное ограничение **инвариантно**, если в нём не содержатся подвыражения  $x - y < k$ ,  $x - y \leq k$  и связка  $\neg$

$IC(C)$  — множество всех инвариантных временных ограничений над  $C$

**Оценкой часов** множества  $C$  называется отображение вида  $v : C \rightarrow R_{\geq 0}$

**Выполнимость временного ограничения**  $tc$  на оценке часов  $v$  ( $v \models tc$ ) определяется следующим образом ( $\bowtie = <, \leq$ ):

- $v \models \text{true}$
- $v \models x \bowtie k \Leftrightarrow v(x) \bowtie k$
- $v \models x - y \bowtie k \Leftrightarrow v(x) - v(y) \bowtie k$
- $v \models tc_1 \& tc_2 \Leftrightarrow v \models tc_1 \text{ и } v \models tc_2$
- $v \models \neg tc_1 \Leftrightarrow v \not\models tc_1$

**Временной автомат** над множеством атомарных высказываний AP — это система  $(L, l_0, \xi, C, I, T)$ , где  $L$  — конечное множество состояний  $l_0$  — начальное состояние  $l_0 \in L$ ,  $\xi : L \rightarrow 2^{AP}$  — разметка состояний событиями  $C$  — конечное множество часов  $I : L \rightarrow IC(C)$  — разметка состояний инвариантами  $T \subseteq L \times TC(C) \times 2^C \times L$  отношение переходов —  $(l_i, g, X, l_2)$  — переход из состояния  $l_1$  в состояние  $l_2$  с предусловием  $g$  и множеством сбрасываемых часов  $X$  [ $l_1 \xrightarrow{g, X} l_2$ ]

**Вычислительная конфигурация** временного автомата  $A = (L, l_0, \xi, C, I, T)$  имеет вид  $\sigma = (l, v)$ , где  $l \in L$  и  $v : C \rightarrow R_{\geq 0}$  ( $v$  — это оценка часов)

Конфигурации (дискретно) преобразуются автомтомом  $A$  в процессе вычисления двумя способами:

1. Продвижение времени ( $\sigma \mapsto \sigma'$ ) [ $d \in R_{\geq 0}$ ]

i.  $\sigma' = \sigma + d$

ii.  $v + d \models I(l)$  — новая оценка часов удовлетворяет ограничениям состояния  $l$  при изменении времени

24. Временные автоматы как формальные модели распределенных систем реального времени. Вычисления временных автоматов. Примеры использования временных автоматов для моделирования встроенных систем. Зеноновские вычисления. Синтаксис и семантика Timed CTL. Задача верификации моделей программ реального времени. Программноинструментальное средство верификации моделей программ реального времени UPPAAL.

---

## 2. Выполнение перехода ( $\sigma \hookrightarrow \sigma'$ ) [ $l \xrightarrow{g,X} l' \in T$ ]

- i.  $\sigma' = \sigma[X][l/l']$  сброс часов
- ii.  $v \models g$  оценка часов удовлетворяет ограничению перехода
- iii.  $v[X] \models I(l')$  в новом состоянии выполняются оценки часов

**Трассой** автомата, исходящей из конфигурации  $\sigma$  (или, коротко,  $\sigma$ -трассой), называется последовательность конфигураций вида  $\sigma \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \dots$

**Частичным вычислением автомата** называется трасса, исходящая из начальной конфигурации.

Конфигурация  $\sigma$  называется **тупиковой**, если не существует конфигурации  $\sigma'$ , такой что  $\sigma \rightarrow \sigma'$ .

Трасса называется **полной**, если она бесконечна или оканчивается тупиковой конфигурацией.

**Вычислением** автомата называется полная трасса, исходящая из начальной конфигурации.

**Примеры использования:**

- Оценка времени реакции встроенной системы на определенное событие (Например после возникновения ошибки в системе должна загореться сигнальная лампочка в течение 3 секунд, и переключится в режим обработки ошибки в течение 10 секунд)
- Проверка свойств взаимодействия (Например, что в системе управления транспортом невозможна такая ситуация, что светофоры горят зеленым для пересекающихся дорог)

Полная трасса конвергентна, если её длительность конечна, и дивергентна, если её длительность бесконечна.

Вычислением Зенона (или **зеноновским вычислением**) называется конвергентное вычисление, в котором выполнение перехода ( $\hookrightarrow$ ) встречается бесконечно часто.

Логика ветвящегося реального времени (Timed CTL; TCTL) — аналог CTL, адаптированный к особенностям поведения CPB.

Минимальный синтаксис TCTL-формул над множеством атомарных высказываний AP и множеством часов C задаётся формой Бэкуса-Наура

$\phi ::= a \mid (etc) \mid (\phi \& \phi) \mid (\neg \phi) \mid (\mathbf{E}(\phi \mathbf{U} \phi)) \mid (\mathbf{A}(\phi \mathbf{U} \phi))$ , где  $\phi$  — TCTL-формула,  $a \in AP$  и  $etc \in ETC(C)$

Содержательная трактовка кванторов **E**, **A** и оператора **U** схожа с их трактовкой в CTL но адаптирована к особенностям выполнения CPB:

- **EΦ**: существует дивергентное выполнение CPB, для которого верно  $\Phi$
- **AΦ**: для любого дивергентного выполнения CPB верно  $\Phi$
- $\phi \mathbf{U} \psi$ : в реальном будущем станет верным  $\psi$ , а до тех пор будет верно  $\phi$

Рассмотрим временной автомат  $A = (L, l_0, \xi, C, I, T)$  над AP, его конфигурацию  $\sigma = (l, \nu)$  и TCTL-формулу  $\phi$  над C и AP

Формула  $\phi$  выполняется в конфигурации  $\sigma$  автомата  $A$  ( $A, \sigma \models \phi$ ) в следующих случаях:

- $A, \sigma \models a$ , где  $a \in AP \Leftrightarrow a \in \xi(l)$
- $A, \sigma \models etc$ , где  $etc \in ETC(C) \Leftrightarrow \nu \models etc$
- $A, \sigma \models \psi \& \chi \Leftrightarrow A, \sigma \models \psi$  и  $A, \sigma \models \chi$
- $A, \sigma \models \neg \psi \Leftrightarrow A, \sigma \not\models \psi$
- $A, \sigma \models \mathbf{E}\Phi \Leftrightarrow$  существует дивергентная  $\sigma$ -трасса  $\tau$  автомата  $A$ , такая что  $A, \tau \models \Phi$
- $A, \sigma \models \mathbf{A}\Phi \Leftrightarrow$  для любой дивергентной  $\sigma$ -трассы  $\tau$  автомата  $A$  верно  $A, \tau \models \Phi$

- $A, \tau \models \psi \mathbf{U} \chi$ , где  $\tau = (\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \dots)$  — дивергентная трасса  $\Leftrightarrow A, \sigma_0 \models \chi$  или существуют номер  $k, k \geq 1$ , и конфигурация  $\sigma$ , порождаемая на  $k$ -м шаге трассы  $\tau$ , такие что

- $A, \sigma \models \chi$  и
- для всех конфигураций  $\delta$ , порождаемых трассой  $\sigma_0 \rightarrow \sigma_1 \rightarrow \dots \rightarrow \sigma_k \rightarrow \sigma$ , верно  $A, \delta \models \psi \vee \chi$

TCTL-формула  $\phi$  **выполняется** на автомата  $A$  ( $A \models \phi$ ), если она выполняется в начальной конфигурации этого автомата

**Задача model checking для TCTL** Для заданного корректного временного автомата  $A$  и заданной TCTL-формулы  $\phi$  проверить справедливость соотношения  $A \models \phi$

**UPPAAL** – это средство верификации систем реального времени.

Структура:

- Модуль описания – входная модель (расширенная модель сети временных автоматов), возможность описания параметризованных шаблонов, наличие данных различной степени локальности
- Модуль симуляции – генерация и визуализация трассы временного автомата
- Модуль верификации – проверка темпоральных свойств, предоставление трассы-контрпримера

Там, где  $\hookrightarrow$  про зеноновские вычисления – это переходы без продвижения по времени

Примеры придуманы из воздуха (в курсе только примеры с птичками, голодными и не очень)

## 25. Дискретные цепи Маркова. Метод вложенных цепей Маркова при исследовании систем массового обслуживания.

Пусть  $X_1, X_2, \dots, X_n, \dots$  - последовательность случайных величин, принимающих значения  $1, \dots, k, \dots$  (счетное множество). Процесс  $X_t$  будет марковской цепью при условии того, что следующее значение в цепи зависит только от предыдущего:  $P(X_{n+1} = i_{n+1} | X_n = i_n, \dots, X_1 = i_1) = P(X_{n+1} = i_{n+1} | X_n = i_n)$

Важным параметром характеристики цепи Маркова является вероятность перехода за один шаг  $P_{i_n j_{n+1}, n}$ , и если она не зависит от  $n$ , то цепь Маркова называется однородной. Далее рассматриваются только такие цепи.

Пусть вероятность перехода из состояния  $i$  в состояние  $j$ :  $P(X_{n+1} = j | X_n = i) = P_{ij}$ . Тогда матрица вероятностей перехода за один шаг обозначается  $(P_{ij})$  - матрица вероятностей перехода за один шаг.

Вероятность перехода из состояния  $i$  в состояние  $j$  за  $n$  шагов обозначается следующим образом  $P(X_{k+n} = j | X_k = i) = P_{ij}^{(n)}$ .

$P_i^0 = P(X_1 = i)$  - задает начальное распределение вероятностей.

Для определения вероятности перехода за любое число шагов используется уравнение Колмогорова-Чепмена:  $P_{ij}^{n+m} = \sum_{k=1}^{\infty} P_{ik}^{(n)} P_{k,j}^{(m)}$ .

**Некоторые свойства цепей Маркова:**

**Достижимость состояния** - состояние  $j$  достижимо из  $i \Leftrightarrow \exists n : P_{ij}^{(n)} > 0$

**Несущественность состояния** - состояние  $i$  называется несущественным, если  $\exists j : j$  из  $i$  достижимо, но  $i$  из  $j$  недостижимо

**Поглощающее состояние** - состояние  $i$  называется поглощающим, если  $P_{ii} = 1$

Далее будем рассматривать цепи Маркова без несущественных состояний и без поглощающих состояний

**Сообщающиеся состояния** -  $i$  и  $j$ , сообщающиеся  $i \leftrightarrow j$ , если  $i$  достижимо из  $j$  и наоборот

**Неразложимость цепи** - цепь Маркова неразложима, если все состояния образуют один класс сообщающихся состояний

**Период состояния**  $i : d(i) = \text{НОД}(n : P_{ii}^{(n)} > 0, n \geq 1)$

**Непериодическое состояние** -  $d(i) = 1$

**Утверждение:** если  $i \leftrightarrow j$ , то  $d(i) = d(j)$ . Таким образом для неразложимой цепи можно говорить о периоде цепи Маркова

**Вероятность первого возвращения за  $n$  шагов**  $f_{ii}^{(n)} = P(X_n = i, X_{n-1} = i, \dots, X_1 = i | X_0 = i)$

**Вероятность возвращения за конечное число шагов**  $f_{ii} = \sum_{n=1}^{\infty} f_{ii}^{(n)}$

**Возвратность состояния** - состояние  $i$  возвратна, если  $f_{ii} = 1$

**Утверждение:** если  $i$  возвратно и  $i \leftrightarrow j$ , то  $j$  возвратно. Таким образом для неразложимой цепи можно говорить о возвратности цепи

Цепь Маркова называется эргодической, если  $\pi_j = \lim_{n \rightarrow \infty} P_{ij}^{(n)}$ .

**Теорема.** Рассмотрим неразложимую, непериодическую возвратную цепь Маркова. Тогда  $\exists \lim_{n \rightarrow \infty} P_{ii}^{(n)} = \frac{1}{\sum_{n=1}^{\infty} n f_{ii}^{(n)}}$ , и  $\lim_{n \rightarrow \infty} P_{ji}^{(n)} = \lim_{n \rightarrow \infty} P_{ii}^{(n)}$

$$\lim_{n \rightarrow \infty} P = \lim_{n \rightarrow \infty} P$$

Важным понятием является **стационарное распределение** цепи Маркова  $\pi = (\pi_0, \pi_1, \dots)$ ,  $\pi_k \geq 0, k \geq 0$ ,  $\sum_{k \geq 0} \pi_k = 1$ ,

которое удовлетворяет соотношениям  $\pi_j = \sum_k \pi_k P_{kj}$

**Теорема.** Существует и единственно стационарное распределение при условии неразложимости, непериодичности и возвратности цепи Маркова.

### Система массового обслуживания

В случайные моменты времени в системе появляются запросы на обслуживание. Система массового обслуживания имеет приборы для обслуживания заявок и дисциплины обслуживания принятых заявок.

**Входящий поток** это  $\{z_i\}$ ,  $i \geq 1$  - последовательность неотрицательных случайных величин, задающая последовательность интервалов между поступлениями заявок на обслуживание

Если  $z_1, z_2, \dots$  - независимые одинаково распределенные случайные величины, то такой поток называется **рекурентным** и  $P(z_1, \dots, z_n) = P(z_1) \dots P(z_n) = P(z)^n = A(x)$ .

**Пуассоновский поток** - поток, где  $A(x) = 1 - e^{-ax}$

**Интенсивность потока**  $a$  - среднее число заявок в единицу времени

### Метод вложенных цепей

$L(t)$  - число заявок в системе в момент времени  $t$ . Если не все потоки поступления и обработки заявок имеют показательное распределение (а значит процесс не марковский в общем), то стоит рассматривать систему только в те моменты времени, когда процесс будет марковским. Например  $d$  моменты завершения обработки заявки (если время обработки не показательное) и рассматривать  $L_n = L(t_n + 0)$  - в момент завершения  $n$ -го кванта времени.

Далее методами теории цепей Маркова исследуют стационарное распределение вложенной цепи и затем по этому распределению восстанавливают стационарное распределение исходного процесса.

## 26. Процессы гибели и рождения. Исследование марковских систем обслуживания с помощью теории процессов гибели и рождения.

Случайный процесс  $X_t$  называется процессом гибели и рождения. Если:

1. Множество его значений содержитя в множестве целых неотрицательных чисел
2. Время пребывания в состоянии  $i$  подчинено показательному распределению с параметром  $\lambda_i + \mu_i > 0$
3. При этом процесс переходит в состояние  $i + 1$  с вероятностью  $\frac{\lambda_i}{\lambda_i + \mu_i}$ , а в  $i - 1$  с вероятностью  $\frac{\mu_i}{\lambda_i + \mu_i}$

Процесс, где  $L(t)$  - число требований в системе в момент времени  $t$ , во многих марковских системах массового обслуживания является частным случаем процесса гибели и рождения. В систему могут поступать и уходить заявки в течение случайных промежутков времени.

В случае непрерывной марковской цепи  $X_t, t \in [0, +\infty)$  для нее необходимо выполнения условия:  $\forall n, \forall t_1 < t_2 < \dots < t_n P(X_{t_n} = i_n | X_{t_{n-1}} = i_{n-1}, \dots, X_{t_1} = i_1) = P(X_{t_n} = i_n | X_{t_{n-1}} = i_{n-1})$ . То есть вероятность перехода зависит только от предыдущего состояния, в котором находился процесс. Введем матрицу вероятностей перехода за фиксированное время  $t$  (переходная функция)  $P_{ij}(t) = P(X_{t+s} = j | X_s = i)$  - вероятность перехода из состояния  $i$  в состояние  $j$  за время  $t$ .

Прямые уравнения Колмогорова для вероятностей перехода в случае процесса гибели и рождения имеют вид (слева производная):

$$P'_{i0}(t) = -\lambda_0 P_{i0}(t) + \mu_1 P_{i1}(t)$$

$$P'_{ij}(t) = -(\lambda_j + \mu_j) P_{ij}(t) + \lambda_{j-1} P_{ij-1}(t) + \mu_{j+1} P_{ij+1}(t), j \geq 1$$

$$P_{ij}(0) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Решение этой бесконечной системы дифференциальных уравнений в явном виде существует только при стационарном распределении вероятностей процесса  $\lim_{t \rightarrow \infty} P_{ij}(t) = \pi_j$ . В таком случае система преобразуется в следующий вид (производная константы равна нулю):

$$-\lambda_0 \pi_0 + \mu_1 \pi_1 = 0$$

$$-(\lambda_j + \mu_j) \pi_j + \lambda_{j-1} \pi_{j-1} + \mu_{j+1} \pi_{j+1} = 0 \Leftrightarrow -\lambda_j \pi_j + \mu_{j+1} \pi_{j+1} = -\lambda_{j-1} \pi_{j-1} + \mu_j \pi_j$$

Далее возможно выделить рекуррентное соотношение

$$\lambda_{j-1} \pi_{j-1} = \mu_j \pi_j$$

$$\pi_j = -\frac{\lambda_{j-1}}{\mu_j} \pi_{j-1}$$

Обозначим  $\rho_j = \frac{\lambda_{j-1}}{\mu_j}$ ,  $\rho_0 = 1$ , тогда при  $\sum_{j=1}^{\infty} \prod_{i=1}^j \rho_i < +\infty$  - существует общее решение системы:

$$\pi_0 = \frac{1}{\sum_{i=1}^{\infty} \prod_{i=1}^j \rho^i}, \pi_j = \left( \prod_{i=1}^j \rho_i \right) \pi_0$$

Таким образом найдены вероятности нахождения системы во всех состояниях при длительной работе системы ( $t \rightarrow \infty$ ).

## 27. Понятие антагонистической игры. Верхнее и нижнее значения конечных и бесконечных антагонистических игр. Седловая точка. Необходимые и достаточные условия существования седловой точки. Теорема Фон Неймана о существовании седловой точки у вогнуто-выпуклых функций.

В антагонистической игре принимают участие два игрока: первый (1) и второй (2). Игрок 1 выбирает стратегию  $x$  из множества стратегий  $X$ . Игрок 2 выбирает стратегию  $y$  из множества  $Y$ . Задана функция выигрыша  $F(x, y)$  первого игрока, определенная на  $X \times Y$ . Выигрыш первого игрока является проигрышем для второго. Цель первого игрока состоит в увеличении выигрыша  $F(x, y)$ , а цель второго - в уменьшении  $F(x, y)$ .

Антагонистическая игра задается набором  $\Gamma = \langle X, Y, F(x, y) \rangle$

Антагонистическая игра  $\Gamma$  называется матричной, если множества стратегий игроков конечны:  $X = \{1, \dots, m\}$ ,  $Y = \{1, \dots, n\}$ . Стратегия первого игрока обозначается  $i$ , стратегия второго игрока обозначается  $j$ , выигрыш первого игрока  $F(i, j) = a_{ij}$ . Матрица  $A = (a_{ij})_{m \times n}$  называется матрицей игры. Первый игрок выбирает номер строки  $i$ , второй - номер столбца  $j$ .

### Нижнее значение игры

$I = \max_{i=1,n} \min_{j=1,m} F(i, j)$  - конечная игра, наилучший гарантированный результат для первого игрока в отсутствии знания о выборе стратегии вторым игроком

$I = \sup_{x \in X} \inf_{y \in Y} F(x, y) = \inf_{y \in Y} F(x_0, y)$  - бесконечная игра,  $x_0$  - оптимальная стратегия первого игрока

### Верхнее значение игры

$I = \min_{j=1,m} \max_{i=1,n} F(i, j)$  - конечномерный случай, наилучший гарантированный результат для второго игрока в отсутствии знания о выборе стратегии первым игроком

$I = \inf_{y \in Y} \sup_{x \in X} F(x, y) = \sup_{x \in X} F(x, y_0)$  - бесконечная игра,  $y_0$  - оптимальная стратегия второго игрока

### Седловая точка

$(x_0, y_0)$  - седловая точка, если

$F(x, y_0) \leq F(x_0, y_0) \leq F(x_0, y)$  для  $\forall x \in X, y \in Y$  - бесконечная игра

$F(i, j_0) \leq F(i_0, j_0) \leq F(i_0, j)$  для  $\forall i = 1, m, j = 1, n$  - конечная игра

**Теорема 1:** Седловая точка существует тогда и только тогда, когда  $I = I$

**Теорема 2:** Если  $\exists i_0, j_0, v = const : F(i, j_0) \leq v \leq F(i_0, j), \forall i, j$ , то  $v = F(i_0, j_0)$  и  $(i_0, j_0)$  - седловая точка

**Теорема 3:** Существует седловая точка, тогда и только тогда, когда существуют и равны  $\max_{x \in X} \inf_{y \in Y} F(x, y) = \inf_{y \in Y} \sup_{x \in X} F(x, y) = V$  ( $V$  равно значению  $F(x, y)$  в седловой точке)

### Теорема Фон-Неймана

Пусть  $F(x, y)$  - определена и непрерывна на множестве  $X \times Y$ ,  $X \subset E^m$ ,  $Y \subset E^n$  - выпуклые компактные множества в евклидовых пространствах и  $F(x, y)$  - выпукла по  $y$ , вогнута по  $x$ , тогда  $F(x, y)$  имеет на  $X \times Y$  седловую точку.



## 28. Понятие потока в сети. Задача о максимальном потоке. Алгоритмы Форда-Фалкерсона и Карзанова. Теорема о максимальном потоке и минимальном разрезе. Сведение задачи составления допустимого расписания с прерываниями для многопроцессорной системы при заданных директивных интервалах к задаче о максимальном потоке в сети.

Сеть состоит из ориентированного графа  $G = (V = \{1, \dots, n\}, A = \{(i, j)\}, i, j \in V)$  с двумя выделенными вершинами  $s$  - источник и  $t$  - сток. Каждая дуга обладает пропускной способностью  $u_{ij}$ .  $(i, j) \in A : u_{ij} > 0$

**Поток:**  $f_{ij}, (i, j) \in A$  и его свойства:

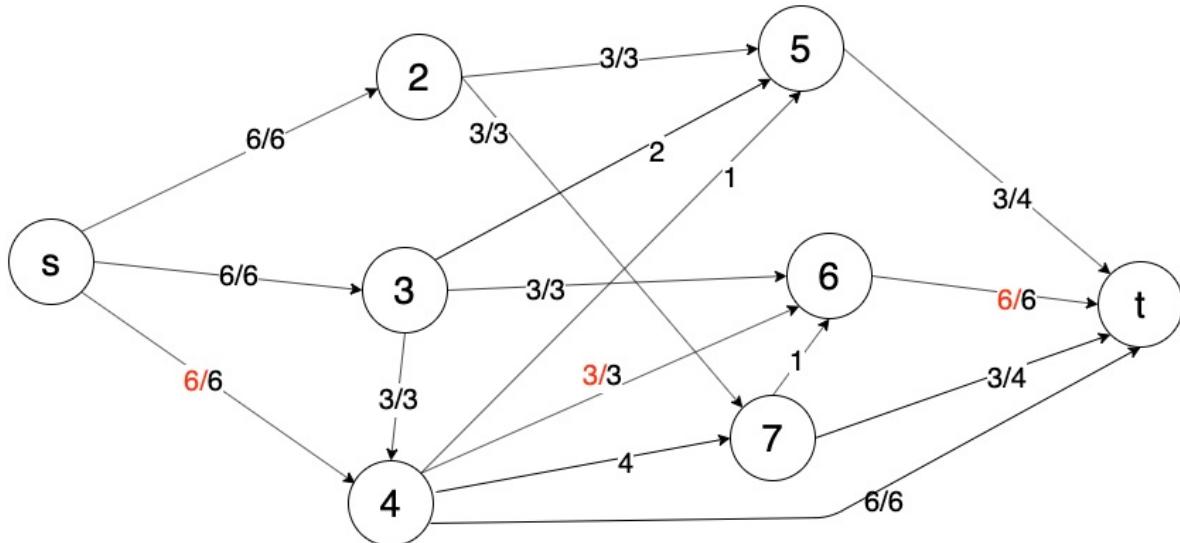
1.  $f_{ij} \geq 0$
2.  $f_{ij} \leq u_{ij} \forall (i, j) \in A$
3.  $\sum_{j:(j,i_0) \in A} f_{j,i_0} - \sum_{j:(i_0,j) \in A} f_{i_0,j} = 0, \forall i_0 \in V, i_0 = s, t$  - сохранение потока

$\sum_{(s,j) \in A} f_{sj} - \sum_{(j,s) \in A} f_{js} = M(f)$  - величина потока (Суммарный поток выходящий из истока минус суммарный поток входящий в исток)

### Задача о максимальном потоке

Необходимо найти поток  $f'_{i,j}$ , такой что  $M(f') = \max_{f_{ij}} M(f)$ , то есть величина потока максимальна.

### Пример сети с построенным максимальным потоком



### Алгоритм Форда-Фалкерсона

Пусть  $\pi$  - это путь

по прямой дуге:  $\delta_{ij} = u_{ij} - f_{ij} > 0$   $f_{ij} = f_{ij} + \delta(\pi)$

По обратной дуге:  $\delta_{kj} = f_{kj} > 0$   $f_{kj} = f_{kj} - \delta(\pi)$

28. Понятие потока в сети. Задача о максимальном потоке. Алгоритмы Форда-Фалкерсона и Карзанова. Теорема о максимальном потоке и минимальном разрезе. Сведение задачи составления допустимого расписания с прерываниями для многопроцессорной системы при заданных директивных интервалах к задаче о максимальном потоке в сети.

$$\delta(\pi) = \min_{(i,j) \in \pi} \delta_{ij}$$

**Увеличивающий путь** - это путь  $\pi$  (последовательность дуг из истока в сток), для которого величина  $\delta(\pi) > 0$ , которая определяется следующим образом: для каждой дуги  $(i, j)$  входящей в путь  $\pi$  вычисляется величина  $\delta_{ij} = \begin{cases} u_{ij} - f_{ij} > 0 & \text{прямая дуга} \\ f_{ji} > 0 & \text{обратная дуга} \end{cases}$  и  $\delta(\pi) = \min_{(i,j) \in \pi} \delta_{ij}$ .

неформально это путь из истока в сток, вдоль которого можно увеличить поток на некоторую величину.

Про обратную дугу - можно забирать поток назад

### Алгоритм

1.  $f_{ij}$  - начальный поток ( $f_{ij} = 0$ )
2. Увеличить путь  $\pi$  на  $\delta(\pi)$  [итеративно]
3. Если нет увеличивающего пути - остановка. [нашли максимальный поток]

Пусть  $u_{ij} \in \mathbb{N}$  - закончится, так как как минимум на 1 каждый раз увеличивается, поэтому алгоритм не зацикливается.

### Алгоритм Карзанова

$G(f) = (V, A(f))$  - остаточная сеть

$A(f)$ :

1.  $(i, j) \in A, f_{ij} < u_{ij} (i, j) \rightarrow A(f) v_{ij} = u_{ij} - f_{ij}$
2.  $(i, j) \in A, f_{ij} > 0 (j, i) \rightarrow A(f) v_{ji} = f_{ij}$

**Слоистая сеть**  $G^*(f) = (V^*, A^*(f))$  включает в себя множество всех кратчайших путей (по числу дуг) из источника в сток по слоям:

Нулевой слой  $V_0 = \{s\}$

Первый слой  $V_1 = \{i : (s, i) \in A(f)\}$

Второй слой  $V_2 = \{j : (j, i) \in A(f), i \in V_1, j \notin V_1 \cup V_0\}$  и так далее, на последнем слое может быть не только сток, в промежуточных может быть тупиковые, поэтому надо убирать висячие узлы как только дошли до стока и инцидентные им дуги.

**Тупиковый поток** - поток, относительно которого нет прямого увеличивающего пути (содержит только прямые дуги) из источника в сток.

### Алгоритм

1. Начальный нулевой поток в  $G$
2. Построить  $G(f)$  - остаточную сеть
3. Если нет прямого пути из  $s$  в  $t$ , то стоп  $f$  - максимальный поток
4. Построить  $G^*(f)$  - слоистую сеть
5. Построить тупиковый поток в  $G^*(f)$  -  $g_{ij}$
6. Изменить потоки:  $(i, j) \in A^*(f)$ - прямая  $f_{ij} = f_{ij} + g_{ij}$ , для обратных дуг  $(i, j) \in A^*(f)$   $f_{ij} = f_{ij} - g_{ij}$
7. Перейти на 2 шаг

**Разрез**:  $V = V_c \cup V_{\bar{c}}$ ,  $V_c \cap V_{\bar{c}} = \emptyset$ ,  $s \in V_c$ ,  $t \in V_{\bar{c}}$

$u(V_c, V_{\bar{c}}) = \sum_{i \in V_c, j \in V_{\bar{c}}} u_{ij}$  - пропускная способность разреза

Разрез с минимальной пропускной способностью называется **минимальным разрезом**.

**Теорема о максимальном потоке и минимальном разрезе.** Величина максимального потока в сети равна величине минимального разреза в сети.

28. Понятие потока в сети. Задача о максимальном потоке. Алгоритмы Форда-Фалкерсона и Карзанова. Теорема о максимальном потоке и минимальном разрезе. Сведение задачи составления допустимого расписания с прерываниями для многопроцессорной системы при заданных директивных интервалах к задаче о максимальном потоке в сети.

### Задача составления допустимого расписания с прерываниями для многопроцессорной системы:

Пусть имеется  $m$  - процессоров. Задано  $N = \{1, \dots, n\}$  - число работ и для каждой работы заданы  $t_i$  - длительность,  $(b_i, f_i]$  - директивный интервал  $i$ -ой работы ( $b_i < f_i$  и  $t_i \leq f_i - b_i$ ). Во время выполнения работ допускаются прерывания и переключения. Требуется ответить на вопрос существует ли допустимое расписание и как его построить.

#### Сведение задачи к задачи поиска максимального потока в сети:

Обозначим  $y_0 < y_1 < y_2 < \dots < y_p$  - все упорядоченные значения  $b_i, f_i$

Составим отрезки  $I_j = (y_{j-1}, y_j]$ ,  $j = 1, p$

Сеть будет состоять из узлов  $s, t, I_j, w_i$

$\Delta_j = y_j - y_{j-1}$  - длительность интервала  $I_j$

Добавляем дуги  $(s, I_j)$  с пропускной способностью  $m\Delta_j$

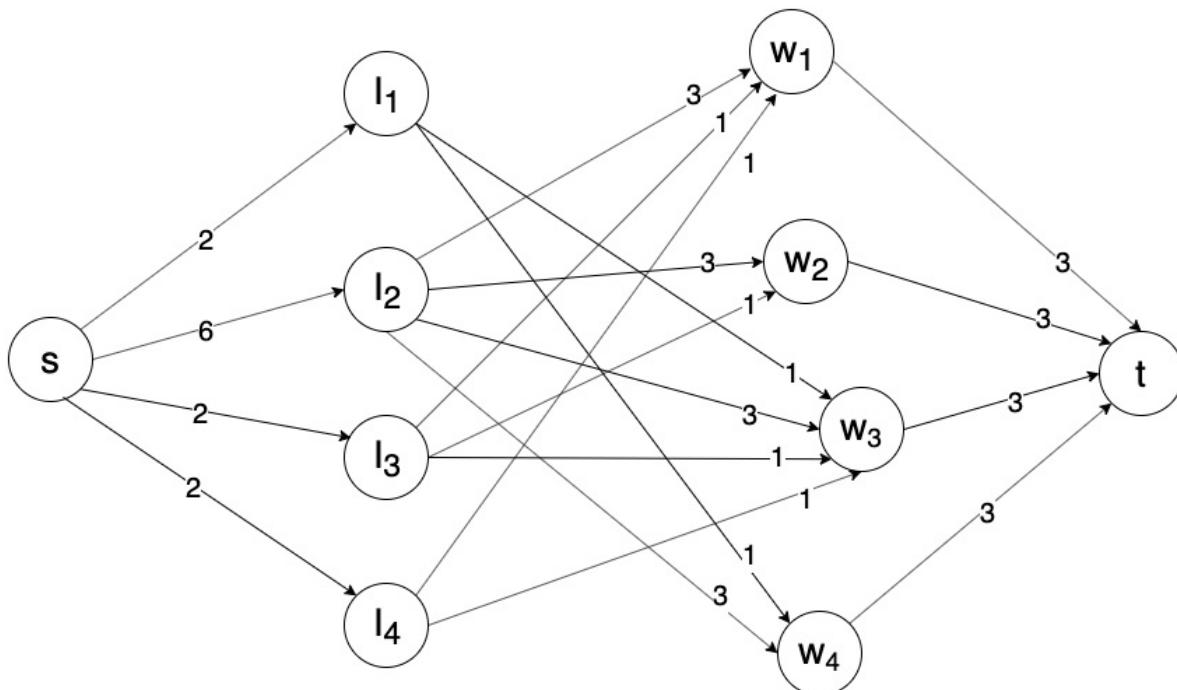
Добавляем дуги  $(I_j, w_i)$ , если  $I_j \subset (b_i, f_i]$  пропускной способностью  $\Delta_j$

Добавляем дуги  $(w_i, s)$  пропускной способностью  $t_i$

Пример сети для задачи при  $m = 2$

Раб.	$b_i$	$f_i$	$t_i$
1	1	6	3
2	1	5	3
3	0	6	3
4	0	4	3

$I_1 = [0, 1], I_2 = [1, 4], I_3 = [4, 5], I_4 = [5, 6]$



**Теорема.** Допустимое расписание существует, тогда и только тогда, когда Максимальный поток в G насыщает все выходные дуги.

#### Как можно искать увеличивающий путь

Есть состояния у каждого узла : Н, ПН, ПП - непомеченный, помеченный непросмотренный, помеченный просмотренный

Метим следующие для рассмотрения вершины по которым теоретически можно получить увеличивающий путь

1. S [-], S - ПН
2. Если нет ПН, то останавливаемся (нет увеличивающих путей)
3.  $i - \text{ПН}, \forall j - \text{Н}, (i, j) \in A, f_{ij} < u_{ij}$  - метим  $[i]$ ,  $j - \text{ПН}$

$i - \text{ПН}, \forall j - \text{Н}, (i, j) \in A, f_{ij} > 0$  - метим  $[i]$ ,  $j - \text{ПН}$

далее  $i - \text{ПП}$

1. Если  $t - \text{Н}$ , то переходим на шаг 2

Если  $t - \Pi$  - построить увеличивающий путь от истока к стоку

#### Как строить тупиковый поток

$i \in V^*(f), i = s, t$

$a(i) = \min\left\{\sum_{(j,i) \in A^*(f)} v_{ji}, \sum_{(j,i) \in A^*(f)} v_{ij}\right\}$  - пропускная способность узла

$a(s) = \sum_{(s,j) \in A^*(f)} v_{sj}$

$a(s) = \sum_{(j,s) \in A^*(f)} v_{jt}$

$i_0$  - самый слабый узел  $a(i_0) = \min_{i \in V^*(f)} a(i)$

Выбираем его

$a_{i_0}$  - будем проталкивать поток от  $i_0$  до стока и до истока (можно по разным дугам, но разрешена только одна недонасыщенная)

После этого вершина  $i_0$  исключается из сети и все полностью насыщенные дуги (если дуги были не полностью насыщены и остались в сети то им изменяем пропускную способность)

и так сначала, пока не останется тупиковый поток

## 29. Псевдополиномиальные алгоритмы решения задач: разбиение, рюкзак, расписание для многопроцессорной системы (число процессоров фиксировано).

Пусть есть задача распознавания свойств  $\Pi$ .  $D_\Pi$  - множество индивидуальных задач.

Алгоритм  $A$  решения задачи  $\Pi$  называется псевдополиномиальным, если его вычислительная сложность ограничена сверху полиномом от функции длины и функции максимума (максимальное число в данной индивидуальной задаче) :  
 $T_A(I) \leq p(l(I), M(I))$  ( $I \in D_\Pi$ )

### Разбиение

Пусть даны  $n$  чисел :  $N = \{a_1, \dots, a_n\}$ . Такие, что  $B = \sum_{i \in N} a_i$  - четное число и  $0 < a_i < \frac{B}{2}$  |  $i = 1, n$ . Необходимо проверить, возможно ли разбиение множества  $N$  на два непересекающихся подмножества  $N$  и  $N \setminus N$ , таких, что  $\sum_{i \in N} a_i = \sum_{j \in N \setminus N} a_j = \frac{B}{2}$

Алгоритм:

Необходимо составить таблицу  $t(i, j)$  размера  $n \times \frac{B}{2}$ .

$$\text{Значение ячейки } t(i, j) = \begin{cases} 1 & \exists \tilde{N} \subseteq N' : \sum_{k \in \tilde{N}} a_k = j, \text{ при этом } N' = \{a_1, \dots, a_i\} \\ 0 & \text{иначе} \end{cases}$$

Построение таблицы производится по строкам, начиная с первой:

- В первой строке  $i = 1$  единица ставится столбцу  $j = a_1$  :  $t(1, a_1) = 1$
- Для следующих строк  $i > 1$ :
  1. Переносим все единицы с предыдущей строки:  $t(i, j) = 1$ , если  $t(i-1, j) = 1$
  2. Добавляем единицы, полученные сдвигом существующих на  $a_i$ : если  $t(i-1, j) = 1$  и  $j + a_i \leq \frac{B}{2}$ , то  $t(i, j + a_i) = 1$
  3. Добавляем единицу в столбец с номером  $a_i$ :  $t(i, a_i) = 1$

Останавливаемся, когда получаем первую единицу в столбце с номером  $\frac{B}{2}$  - значит можно построить разбиение. Если нет, то и нельзя построить разбиение.

Сложность алгоритма равна  $O(nB)$  - по размеру таблицы, которую надо заполнить.

### Рюкзак

Дано  $n$  предметов с весами  $w_1, \dots, w_n$  и стоимостями  $p_1, \dots, p_n$ .  $N = \{1, \dots, n\}$ . Задан вес  $W$  и стоимость  $P$ .

Необходимо проверить, возможно ли найти такое множество предметов  $N$ , что сумма их весов не превышает заданный вес  $\sum_{i \in N} w_i \leq W$  и суммарная стоимость не меньше заданной стоимости  $\sum_{i \in N} p_i \geq P$ .

Алгоритм:

Необходимо составить таблицу  $t(i, j)$  размера  $n \times W$ .

Построение таблицы производится по строкам, начиная с первой:

- В первой строке  $i = 1$  ячейка со столбцом  $j = w_1$  заполняется стоимостью  $p_1$  :  $t(1, w_1) = p_1$
- Для следующих строк  $i > 1$ :
  1. Переносим все значения с предыдущей строки:  $t(i, j) = t(i-1, j)$   $j = 1, W$
  2. Выбираем максимальную стоимость между 2 вариантам: добавить  $i$ -ую вещь в наборы или не добавлять (уже есть набор с таким же весом и большей стоимостью) : если  $t(i, j) = 0$ , то  $t(i, j + w_i) = \max\{t(i-1, j) +$

$$p_i, \quad t(i-1, j + w_i) \}$$

3. Добавляем максимальную стоимость для множества с весом  $w_i$ :  $t(i, w_i) = \max\{p_i, \quad t(i-1, w_i)\}$

Останавливаемся, когда получаем значение стоимости в ячейке, больше, чем  $P$ , - значит можно найти необходимое множество предметов. Если не получили, то и нельзя найти необходимое множество предметов.

Теперь перейдем к поиску самого набора предметов, в случае, если необходимое множество предметов найти можно:

1. Изначально множество  $N'$  пусто
2. Выбираем минимальную строку  $i_1$ , такую, что  $\exists j_1 : t(i_1, j_1) \geq P$ . Добавляем  $i_1$  к  $N'$ . Запоминаем  $j_1$
3. Уменьшаем  $j_1$  на величину веса предмета  $i_1$ :  $j_1 := j_1 - w_{i_1}$
4. Если  $j_1 = 0$ , то закончили. Иначе ищем минимальную строку  $i_2$ , такую, что  $t(i_2, j_1) = t(i_1, j_1)$ . Добавляем  $i_2$  в множество  $N'$ .  $i_1 := i_2$  и переход к шагу 3.

Сложность алгоритма равна  $O(nW)$  - по размеру таблицы, которую надо заполнить.

### Расписание для многопроцессорной системы

Пусть задано  $N = \{1, \dots, n\}$  работ с заданными длительностями выполнения  $t_i \leq T$ .  $T$  - единый директивный срок для всех работ. Имеется  $m$  процессоров на которых запрещены прерывания. Необходимо проверить - возможно ли построение расписания выполнения работ, которое уложится в директивный интервал.

Алгоритм:

Алгоритм заключается в построении точек в  $m$ -мерной решетке и проверки принадлежности точек  $m$ -мерному кубу со стороной  $T$ .

1.  $l = 1$  - строим первые  $m$  точек  $m$ -мерной решетки:  $(t_1, 0, \dots, 0), (0, t_1, \dots, 0), \dots, (0, 0, \dots, t_1)$ . Изначально все эти точки активны.
  2. Исключаем из списка активных точек те, которые не принадлежат определенному выше кубу. Если  $l = n$ , то решение найдено - из числа активных точек берем решение. Если активных не осталось, то расписание не может построить, удовлетворяющее директивному интервалу  $T$ .
  3. На шаге  $l := l + 1$  для каждой активной точки  $a = (a_1, a_2, \dots, a_m)$  строим множество новых активных точек вида  $(a_1 + t_l, a_2, \dots, a_m), (a_1, a_2 + t_l, \dots, a_m), \dots, (a_1, a_2, \dots, a_m + t_l)$ , при этом  $a$  исключаем из списка активных.
- Переходим к шагу 2.

Сложность  $O(T^m)$  - псевдополиномиальный только при фиксированном числе процессоров  $m$ .

Т<sup>m</sup> - число точек рассматриваемого куба, которые проверяем

## 30. Метод ветвей и границ на примере минимаксной задачи теории расписаний. Приближенные алгоритмы решения NP-трудных задач: упаковка в контейнеры, рюкзак, коммивояжер, расписание для многопроцессорной системы, вершинное покрытие. Оценки их сложности и погрешности.

Метод ветвей и границ относится к точным алгоритмам. Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ). Пусть есть оптимальная задача  $\min_{x \in X} f(x)$ . Ветвление разделяет множество  $X = X_1 \cup X_2$ . Вычисляются оценки на этих множествах (нижние и верхние)  $F_{1l} \leq \min_{x \in X_1} f(x) \leq F_{1h}$  и  $F_{2l} \leq \min_{x \in X_2} f(x) \leq F_{2h}$ . Если  $F_{1h} \leq F_{2l}$ , то можно не рассматривать  $X_2$  и осуществлять ветвление дальше только на  $X_1$ .

**Минимаксная задача построения расписания для многопроцессорной системы:** имеется вычислительная система  $P = P_1, \dots, P_m$ , состоящая из  $m$  процессоров. Дано  $n$  работ  $T = T_1, \dots, T_n$ , для каждой из которых известна длительность  $t_{ij}$  на процессоре  $P_j$ . Требуется распределить задания по процессорам так, чтобы общее время выполнения всего множества работ было минимальным.

Ветвление: на первом уровне определяется размещение 1 работы ---  $m$  вариантов, далее рекурсивно на уровне  $k$  определяется процессор для работы  $k$ .

Оценки:

- Нижняя (на уровне  $k$ , то есть известно размещение первых  $k$  работ):
    - $L_1 = \max_{j=1,m} T_j$ , где  $T_j$  - текущее время выполнения размещенных работ на процессоре  $j$ . Общая длительность расписания будет не меньше уже существующей.
    - $L_2 = \max_{i=k+1,n} \min_{j=1,m} (T_j + t_{ij})$  - длительность расписания увеличится еще на одну размещенную работу (самым быстрым способом)
    - $L_3 = \left( \sum_{j=1}^m T_j + \sum_{i=k+1}^n \min_{j=1,m} t_{ij} \right) \frac{1}{m}$  - учет размещение всех работ, не размещенных к данному ветвлению
- Итоговая нижняя оценка  $L = \max(L_1, L_2, L_3)$

- Верхняя (на уровне  $k$ , то есть известно размещение первых  $k$  работ): неназначенные работы распределяются жадным алгоритмом и  $H = \max_{j=1,m} T_j^*$ , где  $T_j^*$  - длительность работ на процессорах с учетом распределения остальных работ жадным алгоритмом.

Приближенным алгоритмом для задачи П называется полиномиальный по времени алгоритм, возвращающий для каждого входа  $I \in D_{\Pi}$  какое-то решение  $x(I) \in sol(I)$ . Обозначим через  $f(x_A(I))$  стоимость решения, найденного алгоритмом А на входе  $I$ . Пусть  $f(x^*(I)) = \min_{x(i) \in sol(I)} f(x(I))$  - оптимальная стоимость решения.

Часто рассматриваются следующие **виды погрешностей** для индивидуальных задач  $I$ :

$$r_A^1(I) = \frac{f(x_A(I))}{f(x^*(I))} - \text{во сколько раз больше решение}$$

$$r_A^2(I) = f(x_A(I)) - f(x^*(I)) - \text{разность}$$

$$r_A^3(I) = \frac{f(x_A(I)) - f(x^*(I))}{f(x^*(I))} - \text{относительная погрешность}$$

$$r_A^3(I) = r_A^1(I) - 1$$

**Погрешности** для алгоритма определяются следующим образом:

30. Метод ветвей и границ на примере минимаксной задачи теории расписаний Приближенные алгоритмы решения NP-трудных задач: упаковка в контейнеры, рюкзак, коммивояжер, расписание для многопроцессорной системы, вершинное покрытие. Оценки их сложности и погрешности.

$$r_A^1 = \sup_{I \in D_\Pi} r_A^1(I) \text{ - погрешность алгоритма в целом}$$

$$r_A^2 = \sup_{I \in D_\Pi} r_A^2(I) \text{ - - погрешность алгоритма в целом}$$

### Упаковка

Упаковка  $N$  объектов (каждый своего объёма  $v_i$ ) в конечное число контейнеров (объём контейнеров фиксирован  $V$ ) таким способом, чтобы число использованных контейнеров было наименьшим.

Алгоритм:

Первый подходящий: объекты помещаются в первый подходящий контейнер, если не помещаются, то создается новый.

Погрешность:  $r_A^1 \leq 2$ , так как все контейнеры, максимум кроме одного будут заполнены на половину, значит  $\sum_{i \in N} v_i \geq \frac{V}{2} f(x_A(I))$ , следовательно  $f(x_A(I)) < 2 \cdot \frac{\sum_{i \in N} v_i}{V}$ , но  $\frac{\sum_{i \in N} v_i}{V} \leq f(x^*(I))$

Сложность  $O(n^2)$

Если объемы упорядочить по невозрастанию, то  $r_A^1 \leq \frac{11}{9}$

### Рюкзак

Из заданного множества предметов со свойствами стоимость и вес, требуется отобрать некое число предметов таким образом, чтобы получить максимальную суммарную стоимость при одновременном соблюдении ограничения на суммарный вес.

Алгоритм:

Жадный алгоритм: необходимо отсортировать вещи по их удельной ценности (то есть отношению цены предмета к его весу), и поместить в рюкзак предметы с наибольшей удельной ценностью.

Итоговая сложность  $O(N \log(N))$

Погрешность:  $r_A^1 \leq 2$

### Коммивояжер

Пройти все вершины неориентированного графа  $G = V, A$  с весами  $d_{ij}, (i, j) \in A$  ровно по одному разу и вернуться в исходную вершину. Необходимо найти такой обход графа с минимальной суммой весов пройденных ребер.

Алгоритм:

1. Ищем минимальное оставное дерево (МОД)  $f(x_{\text{МОД}}(I)) < f(x^*(I))$ 
  - i.  $V' = \emptyset, A' = \emptyset$
  - ii. Добавляем вершину  $i = 1$  в  $V'$
  - iii. Ищем ребро  $(i_0, j_0)$ , такое что  $d_{i_0 j_0} = \min_{i \in V', j \in V \setminus V', (i, j) \in A} d_{ij}$ . Вершина  $j_0$  добавляется в  $V'$ , а ребро  $(i_0, j_0)$  в  $A'$
2. Используем двойной обход по МОД ( $f(x_2(I)) < 2f(x^*(I))$ )
3. Строим маршрут коммивояжера ( $f(x_A(I)) < 2f(x^*(I))$ )

Погрешность:  $r_A^1 \leq 2$

Сложность:  $O(n^3)$

### Вершинное покрытие

Выбрать в неориентированном графе  $G = (V, E)$  минимальное (по количеству вершин) множество вершин  $S$  так, чтобы оно покрывало все ребра графа. То есть так, чтобы у каждого из ребер графа хотя бы один из концов принадлежал  $S$ .

Алгоритм:

1.  $V' = \emptyset$

/

30. Метод ветвей и границ на примере минимаксной задачи теории расписаний Приближенные алгоритмы решения NP-трудных задач: упаковка в контейнеры, рюкзак, коммивояжер, расписание для многопроцессорной системы, вершинное покрытие. Оценки их сложности и погрешности.

---

2. Выбираем ребро  $(i, j) \in E$ , добавляем вершины  $i, j$  в множество  $V'$
3. Удаляем все инцидентные этим вершинам ребра  $A$ ,  $E := E \setminus A$ . Если еще есть ребра, то перейти к шагу 2.

Погрешность  $r_A^1 \leq 2$ , так как, если  $|V'| = f(x_A(I)) = 2K$ , то  $f(x^*(I)) \geq K$

Сложность  $O(n^2)$