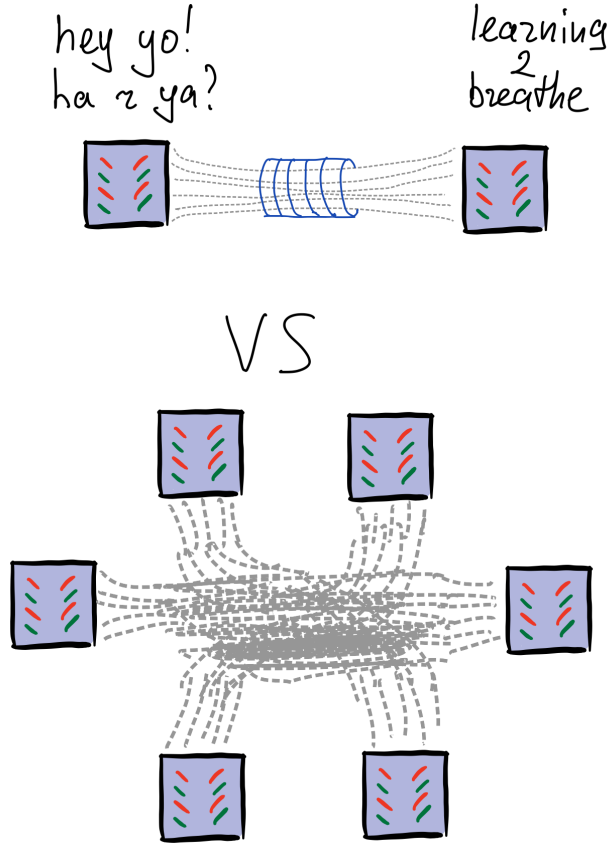
Взаимодействия: число процессов

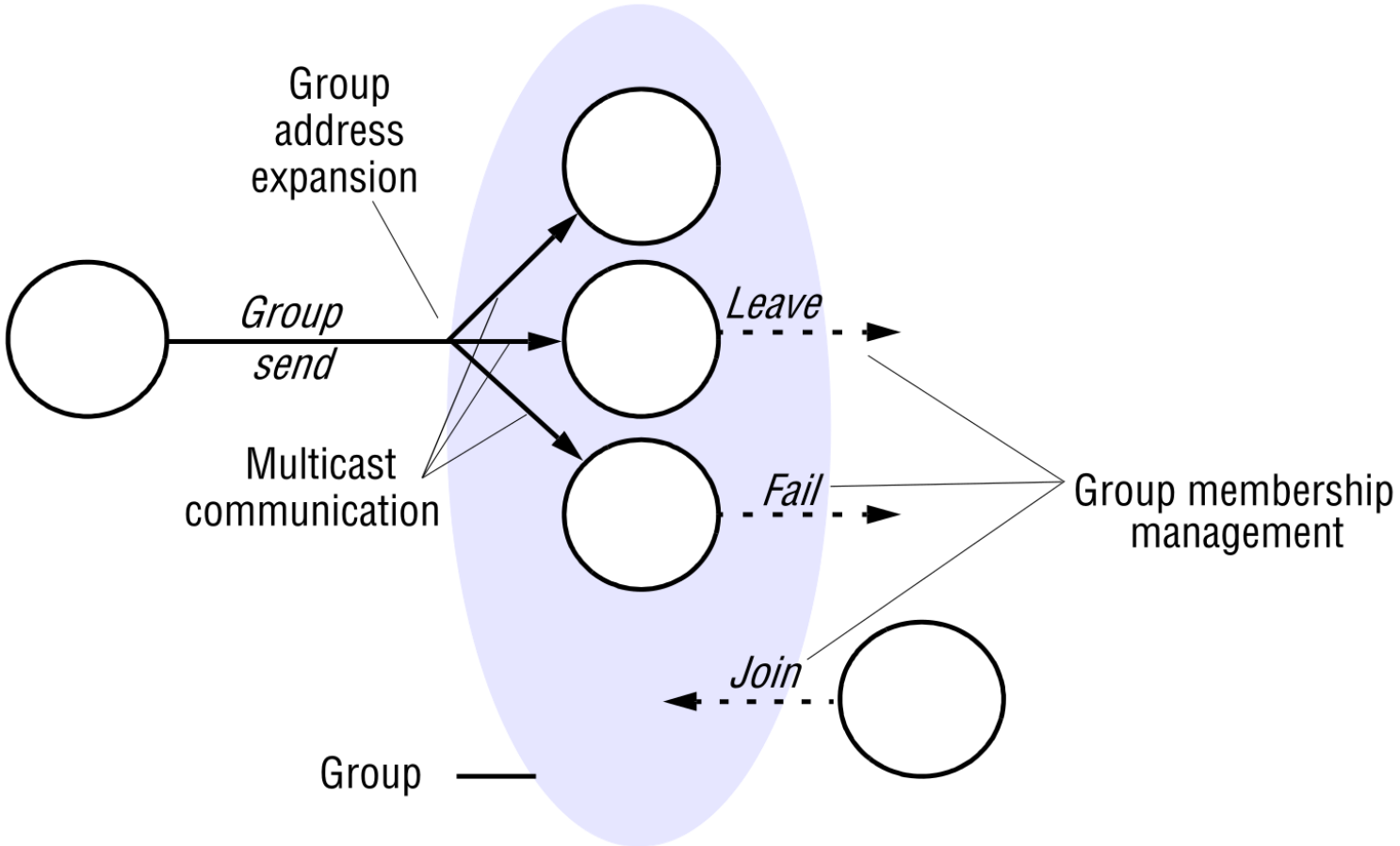
- Парные взаимодействия
  - point-to-point, one-to-one
  - RPC, HTTP
- Групповые взаимодействия
  - one-to-many, many-to-many
  - ???



Применение

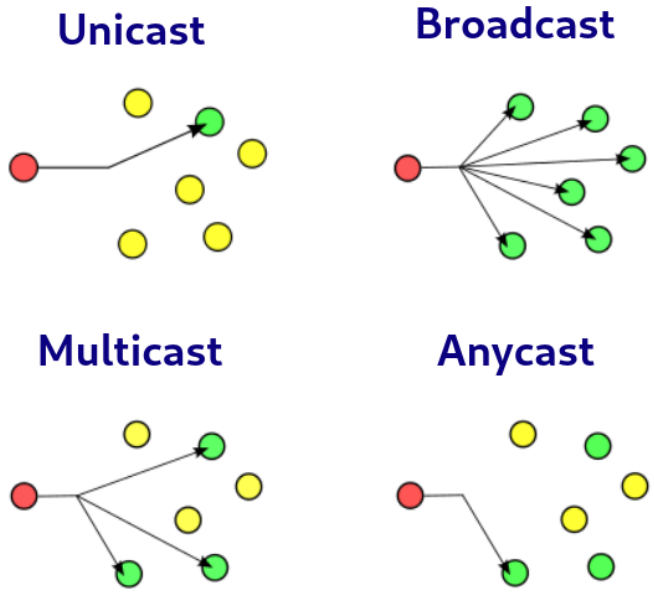
- доставка контента и потоковое вещание
- поиск сервисов и разрешение имен
- синхронизация времени
- рассылка уведомлений о событиях
- поиск данных
- параллельные вычисления
- репликация сервисов или данных

Реализация группового взаимодействия

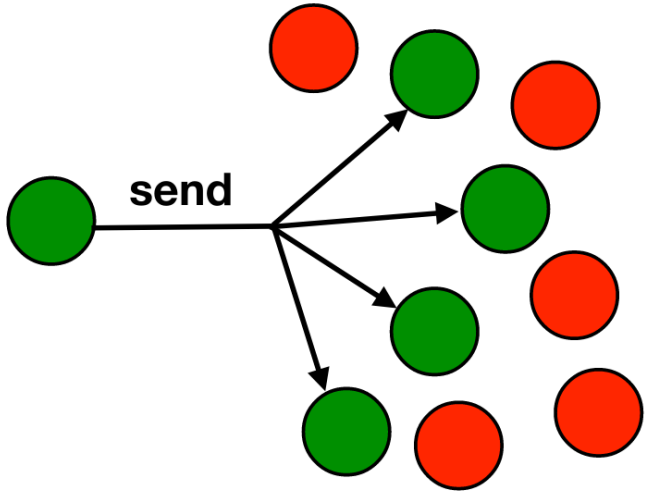


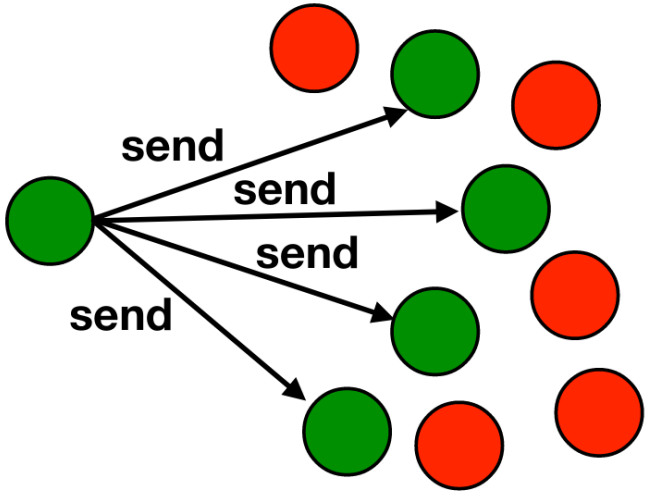
Схемы передачи сообщений

- Unicast
  - одноадресная передача
- Broadcast
  - широковещательная рассылка
- Multicast
  - многоадресная рассылка
  - source-specific multicast (one-to-many)
  - any-source multicast (many-to-many)
- Anycast
  - передача кому угодно

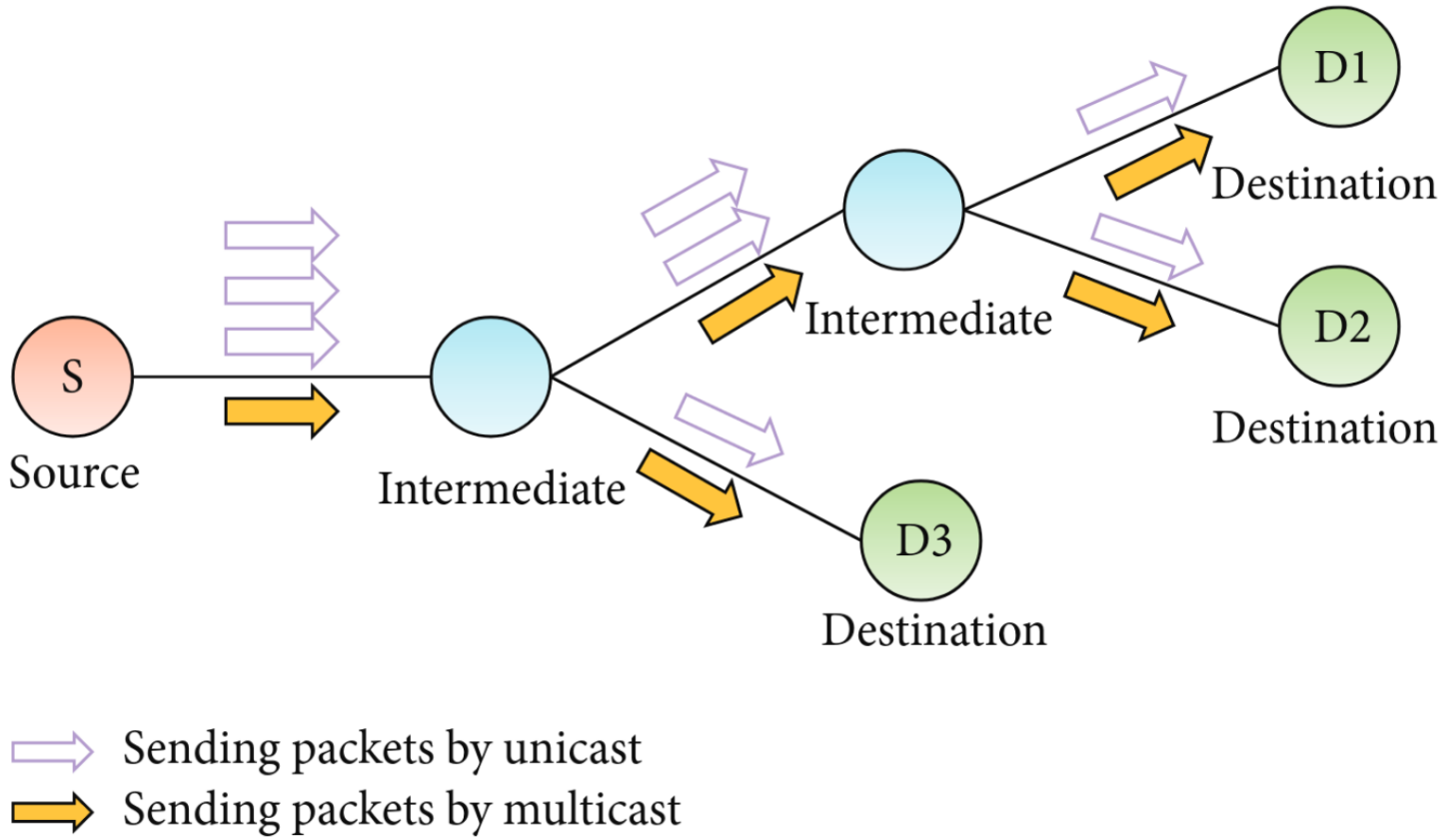


Unicast vs Multicast



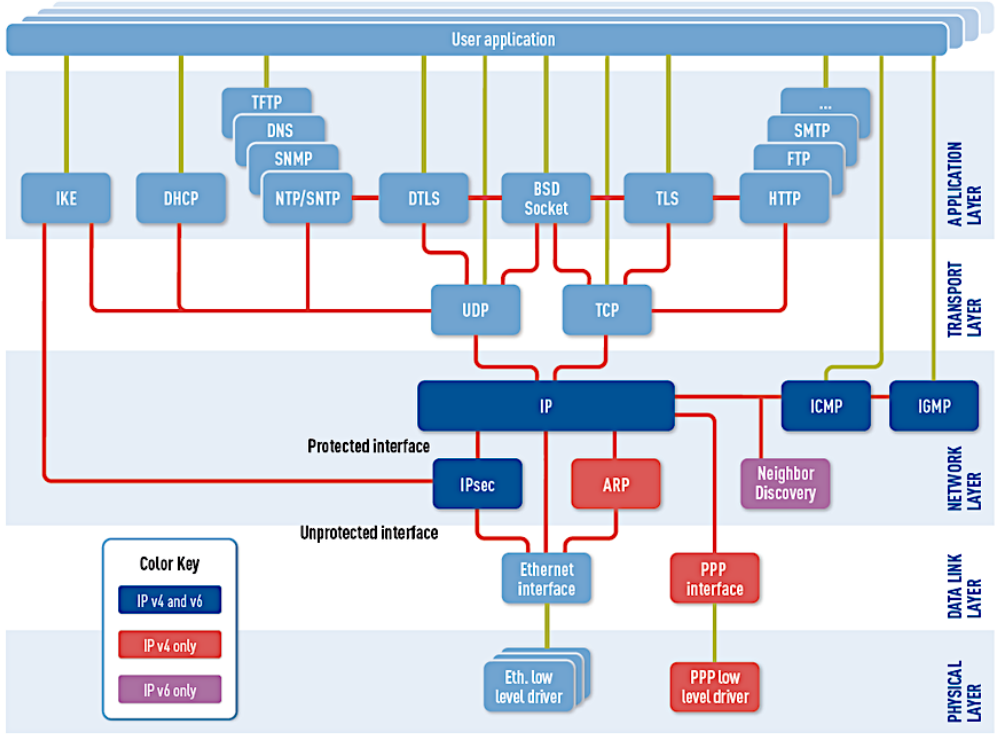


Unicast vs Multicast



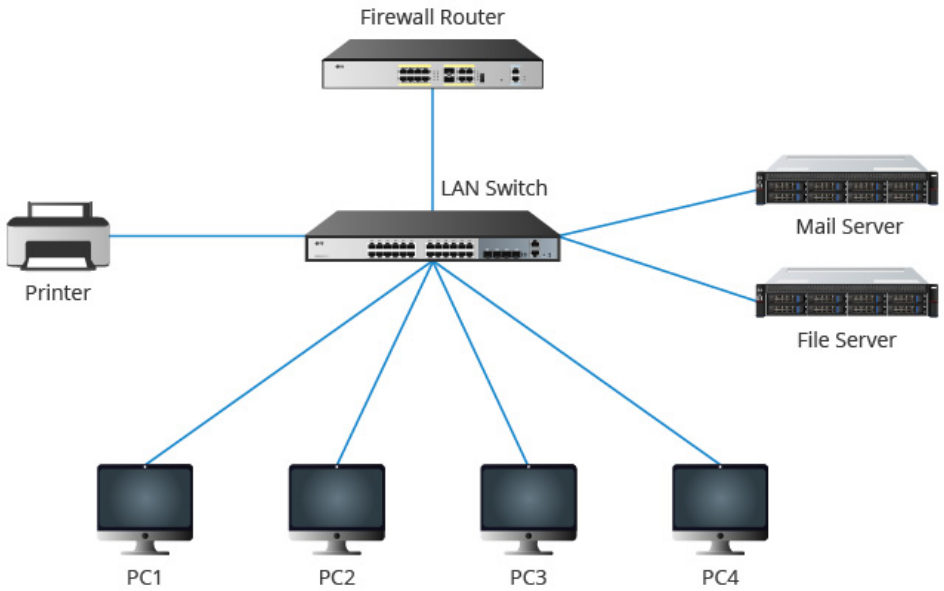
Реализация мультикаста

- На уровне сети
  - канальный уровень (Ethernet)
  - сетевой уровень (IP)
- На уровне приложения

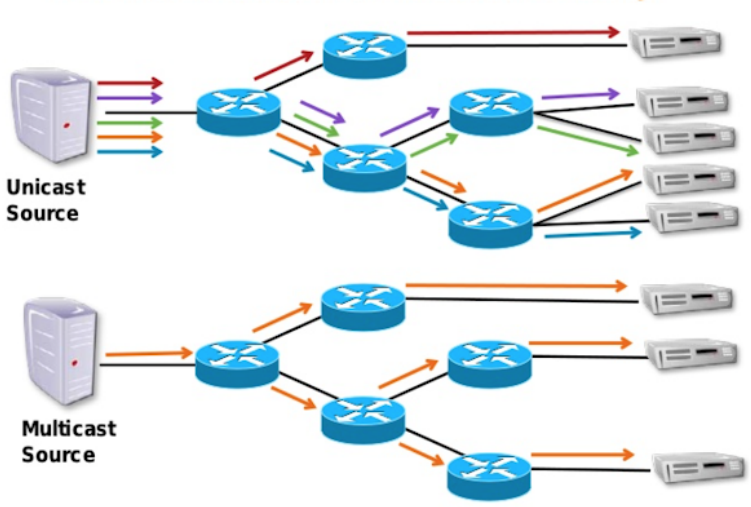


Ethernet

- Выделенный диапазон MAC-адресов
- Рассылка по всем устройствам в сети



IP Multicast



- Позволяет отправить один пакет сразу всем участникам multicast-группы
- Группа идентифицируется с помощью уникального IP-адреса
- Машины в сети могут динамически вступать и выходить из групп
- Для отправки данных не требуется быть участником группы
- Доступ на уровне приложений чаще всего через протокол UDP

Адрес multicast-группы

- IPv4
  - подсеть `224.0.0.0/4 (224.0.0.0-239.255.255.255)`
  - разбита на несколько блоков с разным назначением
  - зарезервированные адреса
    - 224.0.0.1 - all hosts
    - 224.0.0.2 - all routers
    - 224.0.0.22 - IGMP
    - 224.0.0.251 - mDNS
    - 224.0.1.1 - NTP
- IPv6
  - префикс `ff00::/8`

Пример: Отправитель (1)

```
import socket
import struct
import sys

message = b'very important data'
multicast_group = ('224.3.29.71', 10000)

# Create the datagram socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Set a timeout so the socket does not block
# indefinitely when trying to receive data.
sock.settimeout(0.2)

# Set the time-to-live for messages to 1 so they do not
# go past the local network segment.
ttl = struct.pack('b', 1)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)
```

Пример: Отправитель (2)

```
try:
    # Send data to the multicast group
    print('sending {!r}'.format(message))
    sent = sock.sendto(message, multicast_group)

    # Look for responses from all recipients
    while True:
        print('waiting to receive')
        try:
            data, server = sock.recvfrom(16)
        except socket.timeout:
            print('timed out, no more responses')
            break
        else:
            print('received {!r} from {}'.format(data, server))
finally:
    print('closing socket')
    sock.close()
```

Пример: Получатель (1)

```
import socket
import struct
import sys

multicast_group = '224.3.29.71'
server_address = ('', 10000)

# Create the socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind to the server address
sock.bind(server_address)

# Tell the operating system to add the socket to
# the multicast group on all interfaces.
group = socket.inet_aton(multicast_group)
mreq = struct.pack('4sL', group, socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
```

Пример: Получатель (2)

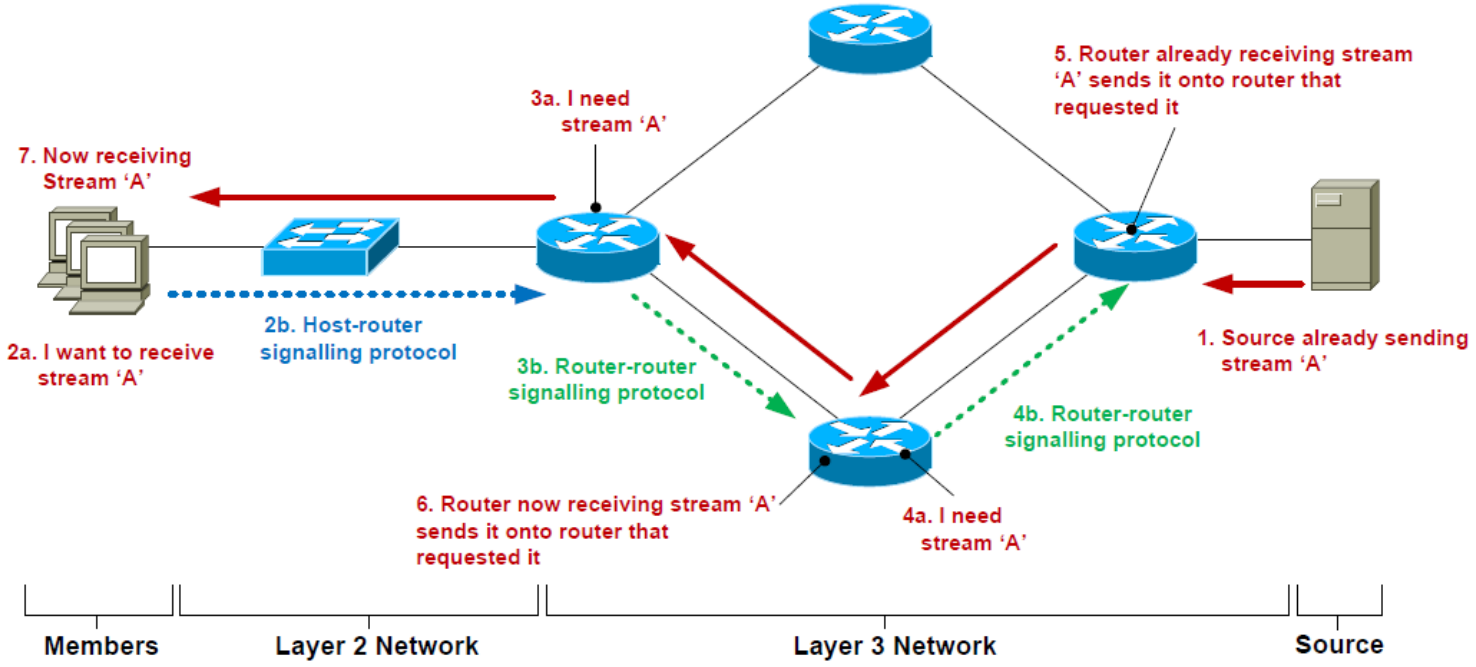
```
# Receive/respond loop
while True:
    print('\nwaiting to receive message')
    data, address = sock.recvfrom(1024)

    print('received {} bytes from {}'.format(len(data), address))
    print(data)

    print('sending acknowledgement to', address)
    sock.sendto(b'ack', address)
```

Гарантии IP Multicast

- Мультикаст на базе UDP
  - контроль целостности
  - доставка не гарантируется
  - сохранение порядка сообщений не гарантируется
- Протокол Pragmatic General Multicast (PGM)
  - IETF experimental protocol
  - надежная доставка и сохранение порядка сообщений
  - использует отрицательные подтверждения (NAKs)

IP Multicast в глобальной сети

- Требуется поддержка с стороны маршрутизаторов
- Распространение данных контролируется с помощью TTL (time to live)
- Основные протоколы: IGMP, PIM



Масштабируемость IP Multicast

- Хорошая масштабируемость по размеру группы
- Ограниченная масштабируемость по числу групп
  - альтернативный подход: explicit multi-unicast (Xcast)

Мультикаст на уровне приложения

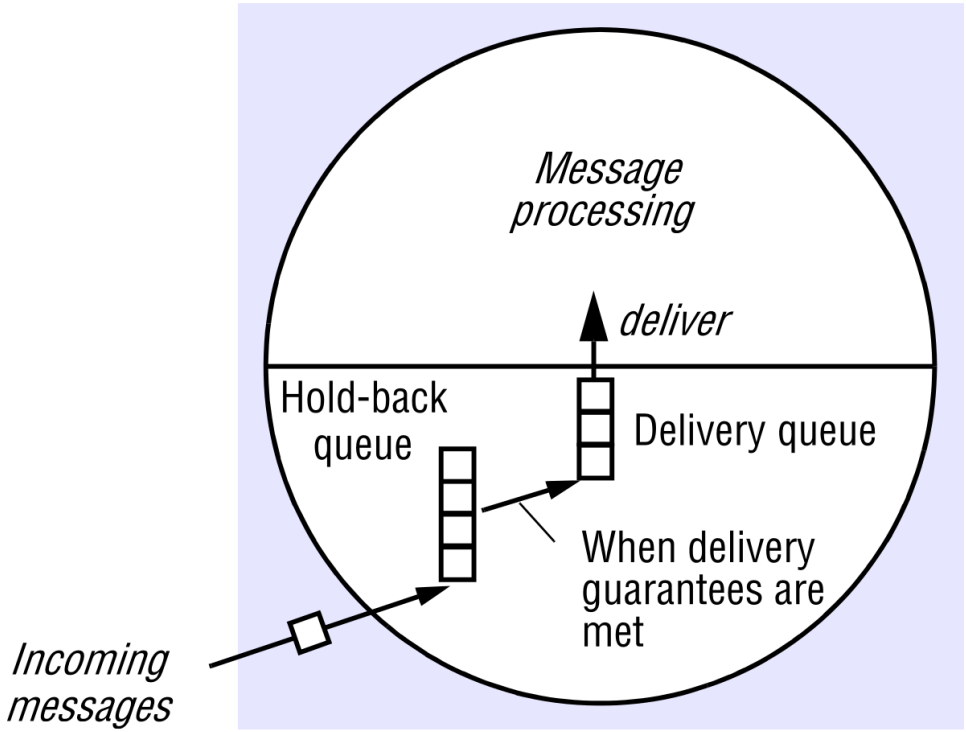
- Отсутствует поддержка со стороны сети
- Недостаточно предоставляемых возможностей и гарантий

Важные аспекты реализации

- Адресация участников группы
- Надежность доставки
- Порядок доставки
- Семантика доставки
- Семантика ответа
- Структура группы

Интерфейс

- *join(group)*
- *leave(group)*
- *multicast(group, message)*
  - внутри сообщения указываются sender и group
- *receive(group) -> message*
- обратный вызов *deliver(message)*
- ...



Предположения

Далее рассмотрим несколько реализаций, использующих следующие предположения:

- все процессы знают адреса друг друга
- каналы между процессами надежные (см. следующий слайд)
- процессы могут отказывать только путем полной остановки (crash-stop)
- группы закрытые и непересекающиеся
- состав участников групп зафиксирован

Надежная доставка (one-to-one)

- **Validity**: каждое сообщение будет доставлено
  - если корректный процесс p отправляет сообщение m корректному процессу q, то q в конце концов доставит m
- **No Duplication**: отсутствуют повторы сообщений
  - никакое сообщение не доставляется процессом более одного раза
- **No Creation**: сообщения доставляются без искажений
  - если некоторый процесс q доставил сообщение m от процесса p, то m было ранее отправлено от p к q
- **Integrity**: No Duplication + No Creation

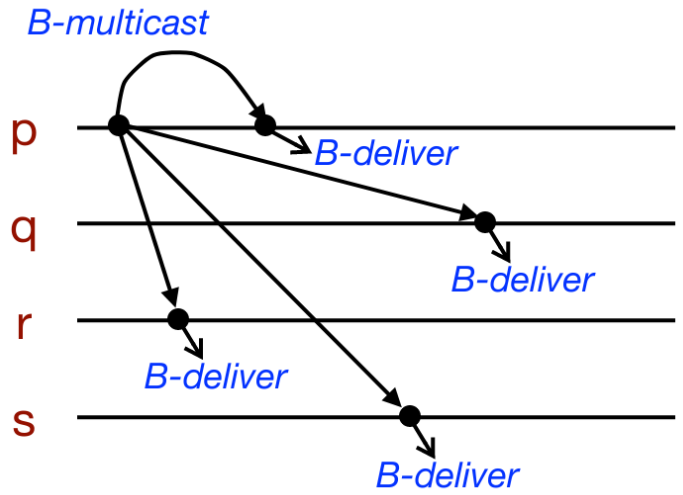
Basic Multicast: Свойства

- **Validity**: если корректный процесс рассылает сообщение m, то каждый корректный процесс в конце концов доставит m
- **No Duplication**: корректный процесс p доставляет сообщение m не более одного раза
- **No Creation**: если корректный процесс p доставил сообщение m с отправителем s, то m было ранее разослано s

Basic Multicast: Реализация

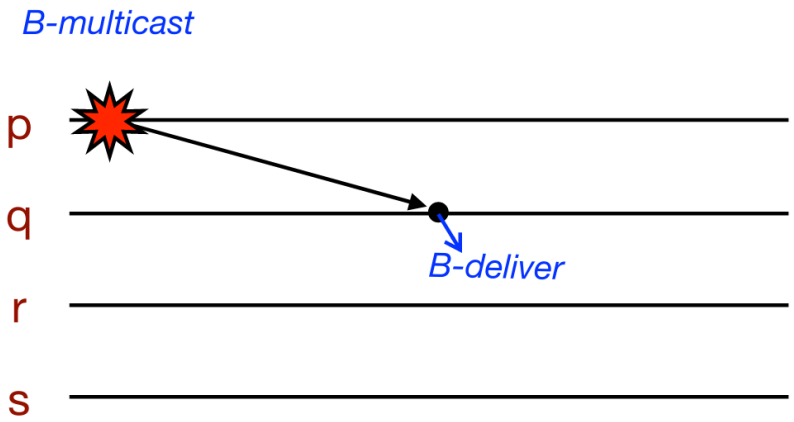
To *B-multicast(g, m)*:
for each process p ∈ g, *send(p, m)*

On *receive(m)* at p:
B-deliver(m) at p



- Использует надежный канал (one-to-one доставку) в виде операции *send()*
- Выполнение свойств следует из свойств надежного канала
- Подвержена Ack-Implosion Problem

Отказ отправителя



- При отказе отправителя часть процессов может получить сообщение, а часть нет
- Отсутствует согласие (agreement) между процессами относительно доставки сообщения

Reliable Multicast: Свойства

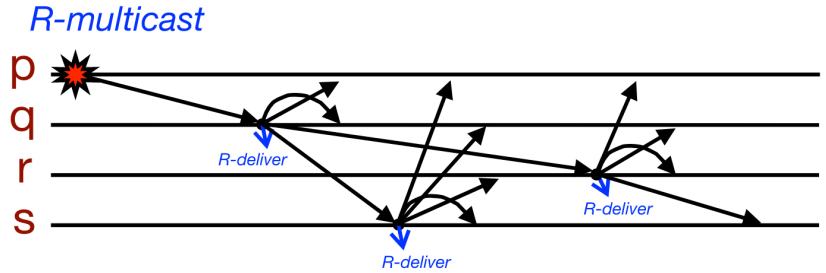
- **No Duplication**: корректный процесс p доставляет сообщение m не более одного раза
- **No Creation**: если корректный процесс p доставил сообщение m с отправителем s, то m было ранее разослано s
- **Validity**: если корректный процесс рассылает сообщение m, то он в конце концов доставит m
- **Agreement**: если некоторый корректный процесс доставил сообщение m, то все остальные корректные процессы в группе в конце концов доставят m

Reliable Multicast: Реализация

On initialization
Received := {};

For process p to R-multicast message m to group g
B-multicast(*g*, *m*); // $p \in g$ is included as a destination

On B-deliver(*m*) *at process q with g = group*(*m*)
if ($m \notin$ *Received*)
then
Received := *Received* $\cup$ {*m*};
if ($q \neq p$) *then B-multicast*(*g*, *m*); *end if*
R-deliver m;
end if



- Опирается на реализацию Basic Multicast
- Упражнение: показать, что выполняются все указанные свойства
- Низкая эффективность - *O(N^2)* сообщений

Reliable Multicast: Другая реализация

- Используем IP multicast и подтверждения
  - подтверждения отправляются вместе с рассылаемыми сообщениями (piggyback)
  - отдельное сообщение в случае обнаружения пропуска сообщения (negative ack)
- Каждый процесс p хранит локально
  - S_p^g - sequence number группы g, в начале 0
  - R_q^g - номер последнего доставленного им сообщения от q в g
- Отправка сообщения
  - к сообщению добавляются значение S_p^g и подтверждения $\langle q, R_q^g \rangle$
  - сообщение со добавкой рассылается через IP multicast
  - значение S_p^g увеличивается на 1

Reliable Multicast: Другая реализация (2)

- Получение сообщения с номером S от p
  - если $S = R_p^g + 1$, то сообщение доставляется и R_p^g увеличивается на 1
  - если $S \leq R_p^g$, то сообщение было получено ранее и отбрасывается
  - если $S > R_p^g + 1$, то сообщение помещается в hold-back queue
  - если $S > R_p^g + 1$ или $R > R_q^g$ для подтверждения $\langle q, R \rangle$ из сообщения, то какие-то сообщения еще не получены и возможно потеряны при рассылке
  - процесс запрашивает недостающие сообщения от их отправителей или других процессов, который получали эти сообщения, путем отправки negative acknowledgement
- Особенности
  - требуется постоянная (бесконечная) рассылка сообщений
  - необходимо (вечное) хранение доставленных сообщений на всех процесссах
  - попутно получили сохранение порядка сообщений

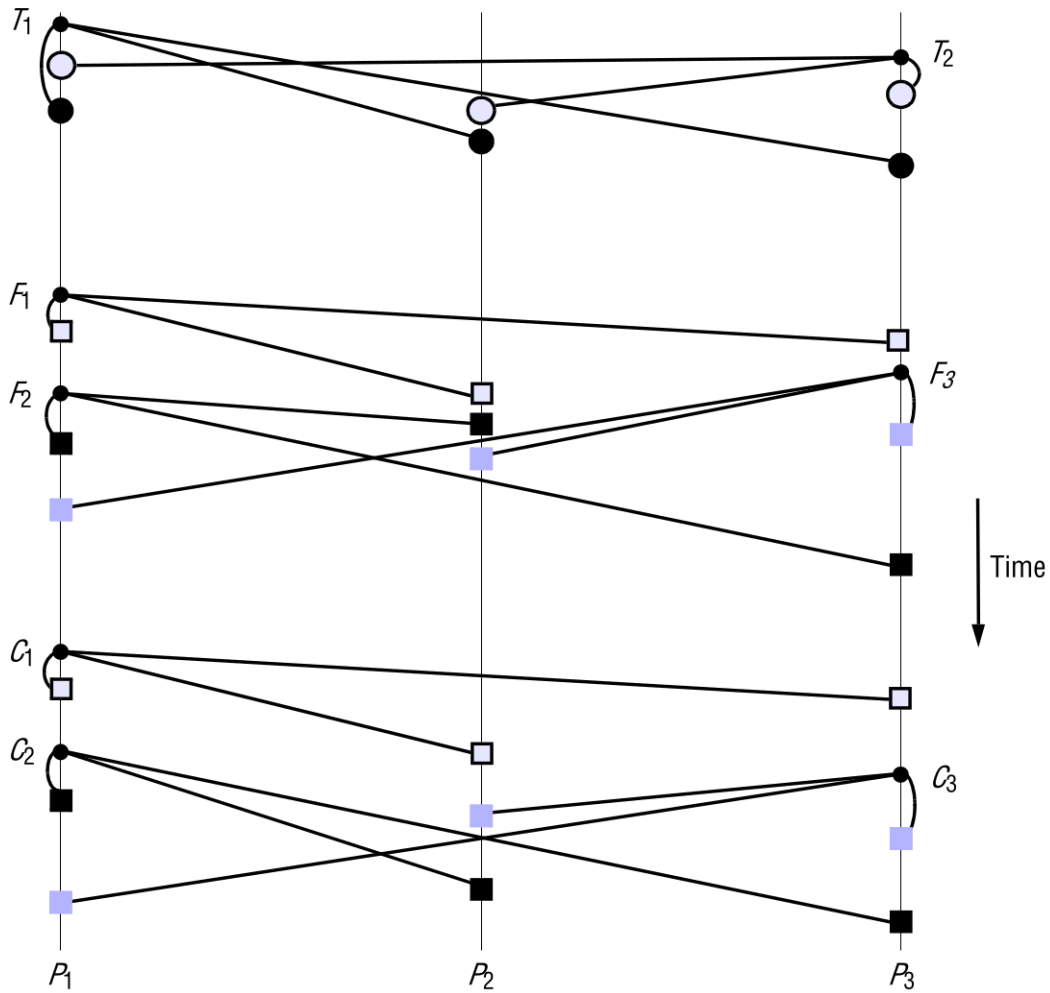
Uniform Agreement

- Расширение свойства agreement с только корректных до всех процессов, включая отказавшие
  - Если некоторый процесс доставил сообщение m, то все корректные процессы в группе в конце концов доставят m
- В каких случаях это требуется?
- Удовлетворяют ли этому свойству наши реализации Reliable Multicast?

Гарантии на порядок доставки сообщений

- **Arbitrary Order**: сообщения доставляются в произвольном порядке, разные участники группы могут наблюдать различный порядок
- **FIFO Order**: если корректный процесс сначала отправил m а потом m', то любой корректный процесс, который доставил m', доставит m до m'
- **Causal Order**: если отправка m произошла до (happened-before) отправки m', то любой корректный процесс, который доставил m', доставит m до m'
  - включает в себя FIFO порядок
- **Total Order**: если некоторый корректный процесс доставил m до m', то любой другой корректный процесс, который доставил m', доставит m до m'
  - полный порядок, не гарантирует FIFO или Causal порядки
  - возможные комбинации: FIFO-Total, Causal-Total

Примеры различных порядков



Какие свойства требуются?

Приложение, позволяющее пользователям обсуждать различные темы

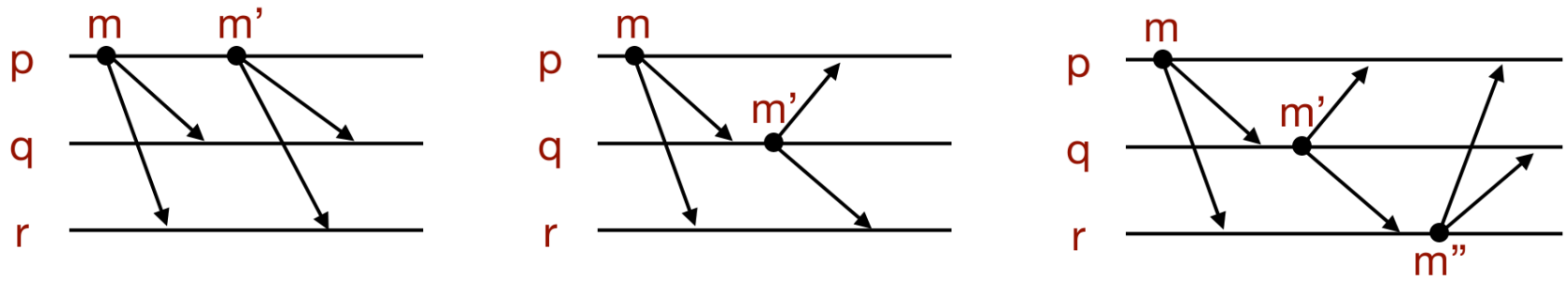
- Пользователи (клиенты) распределены
- Обмен сообщениями реализуется с помощью мультикаста

Bulletin board: *os.interesting*		
Item	From	Subject
23	A.Hanlon	Mach
24	G.Joseph	Microkernels
25	A.Hanlon	Re: Microkernels
26	T.L'Heureux	RPC performance
27	M.Walker	Re: Mach
end		

FIFO Order: Реализация

- Основана на использовании sequence numbers
- Каждый процесс p хранит локально
  - S_p^g - сколько сообщений p отправил в группу
  - R_q^g - номер последнего сообщения от q в g, которое доставил p
- При отправке процесс добавляет к сообщению S_p^g и затем увеличивает S_p^g на 1
- При получении сообщения с номером S от процесса q
  - если $S = R_q^g + 1$, то сообщение доставляется
  - если $S > R_q^g + 1$, то сообщение добавляется в hold-back queue
- Для рассылки достаточно использовать B-Multicast
  - если использовать R-Multicast, то получим надежный FIFO-мультикаст

Causal Order: Реализация



- Каждый процесс p поддерживает локально вектор размера N
  - j-я компонента вектора равно числу сообщений, которые p доставил от j
- Векторы рассылаются вместе с ообщениями и используются для упорядочивания сообщений
- Вариант *векторных часов*, рассматриваемых далее в курсе
- Для рассылки можно использовать B-multicast или R-multicast

Causal Order: Реализация

Algorithm for group member p_i $(i = 1, 2\ldots, N)$

On initialization

$V_i^g[j] := 0$ $(j = 1, 2\ldots, N)$;

To CO-multicast message m to group g

$V_i^g[i] := V_i^g[i] + 1$;

B-multicast$(g, <V_i^g, m>)$;

On B-deliver$(<V_j^g, m>)$ *from* p_j $(j \neq i)$, *with* $g = group(m)$

place $<V_j^g, m>$ in hold-back queue;

wait until $V_j^g[j] = V_i^g[j] + 1$ and $V_j^g[k] \leq V_i^g[k]$ $(k \neq j)$;

CO-deliver m; // after removing it from the hold-back queue

$V_i^g[j] := V_i^g[j] + 1$;

Total Order: Реализация

- Основная идея: назначить каждому сообщению уникальный номер
  - sequence numbers на уровне всей группы
  - каждый процесс может локально упорядочить сообщения
- Возможные подходы
  - централизованный - выделенный процесс (seqencer)
  - распределенный - процессы согласуют номера друг с другом

Total Order: Реализация с Sequencer

1. Algorithm for group member p

On initialization: $r_g := 0$;

To TO-multicast message m to group g
    B-multicast($g \cup \{sequencer(g)\}$, <*m*, *i*>);

On B-deliver(<*m*, *i*>) *with* $g = group(m)$
    Place <*m*, *i*> in hold-back queue;

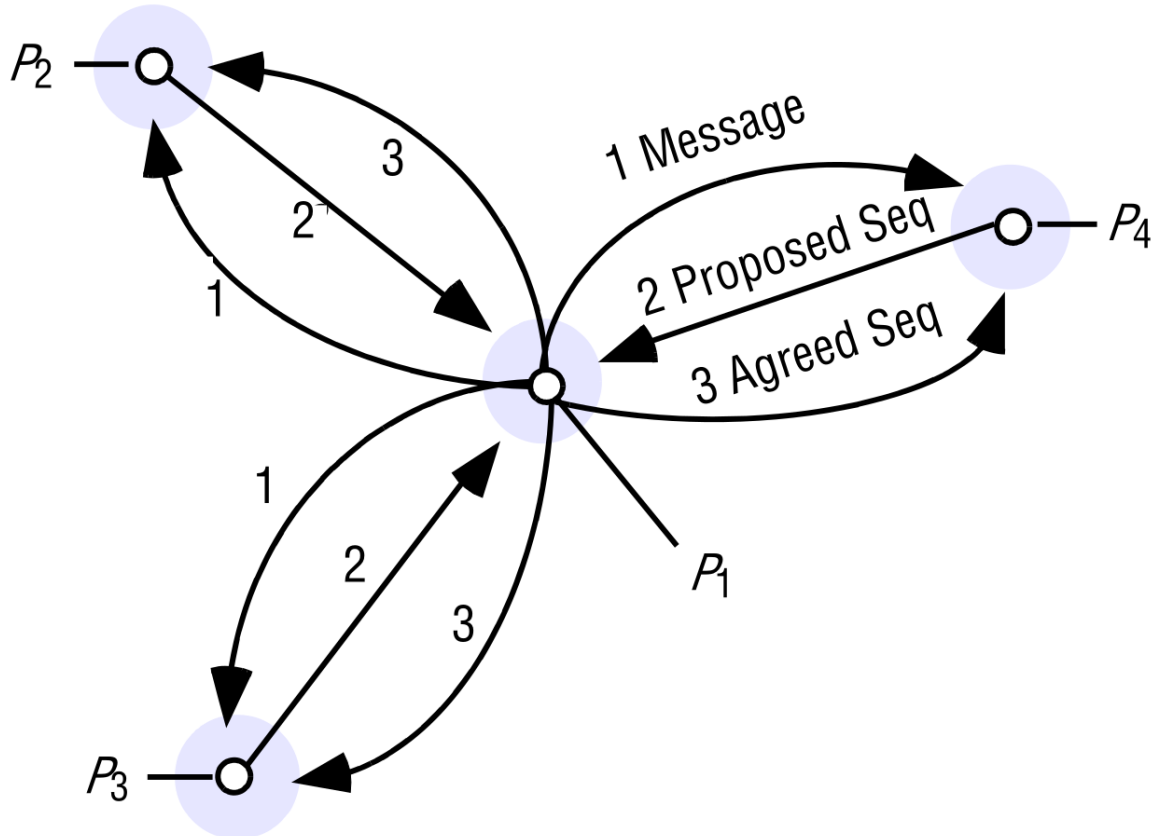
On B-deliver(m_{order} = <"order", *i*, *S*>) *with* $g = group(m_{order})$
    wait until <*m*, *i*> in hold-back queue and $S = r_g$;
    TO-deliver m; // (after deleting it from the hold-back queue)
    $r_g := S + 1$;

2. Algorithm for sequencer of g

On initialization: $s_g := 0$;

On B-deliver(<*m*, *i*>) *with* $g = group(m)$
    B-multicast(*g*, <"order", *i*, s_g >);
    $s_g := s_g + 1$;

Total Order: Распределенная реализация



Рассмотренные реализации мультикаста

- Basic
- Reliable
- (Reliable) FIFO
- (Reliable) Causal
- Total Ordered

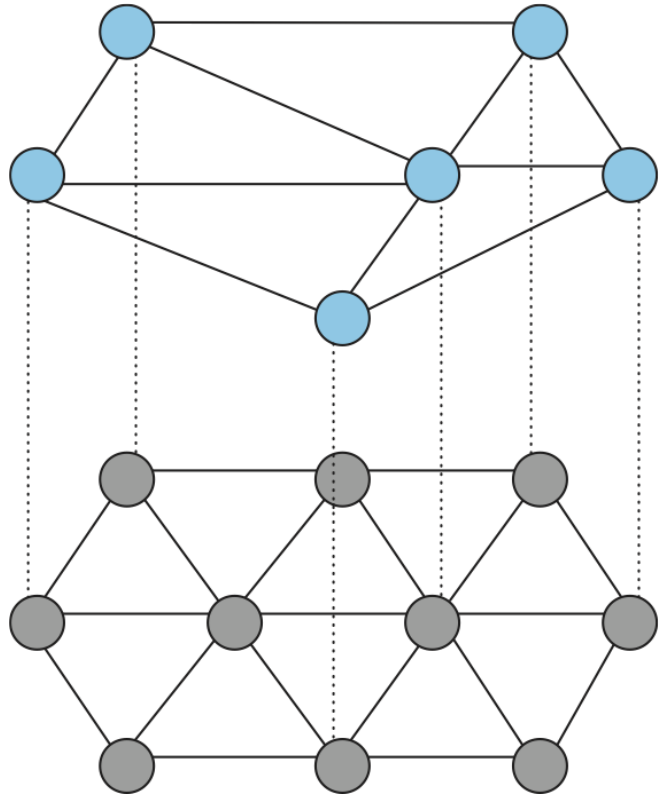
- Что насчёт Reliable + Total Ordered = Atomic Multicast?
  - Эквивалентен задаче консенсуса, рассматриваемой позже в курсе

Рассмотренные реализации мультикаста

- Акцент на гарантиях надежности и порядка
- Упрощающие предположения
  - отказы только crash-stop
  - все процессы знают друг друга
  - состав групп зафиксирован
  - группы не пересекаются
- Как обеспечить масштабируемость?
  - участников очень много
  - они могут находится в разных частях Интернета и не знать друг о друге
  - классические подходы не работают или создают большую нагрузку на сеть

Оверлейная сеть (overlay network)

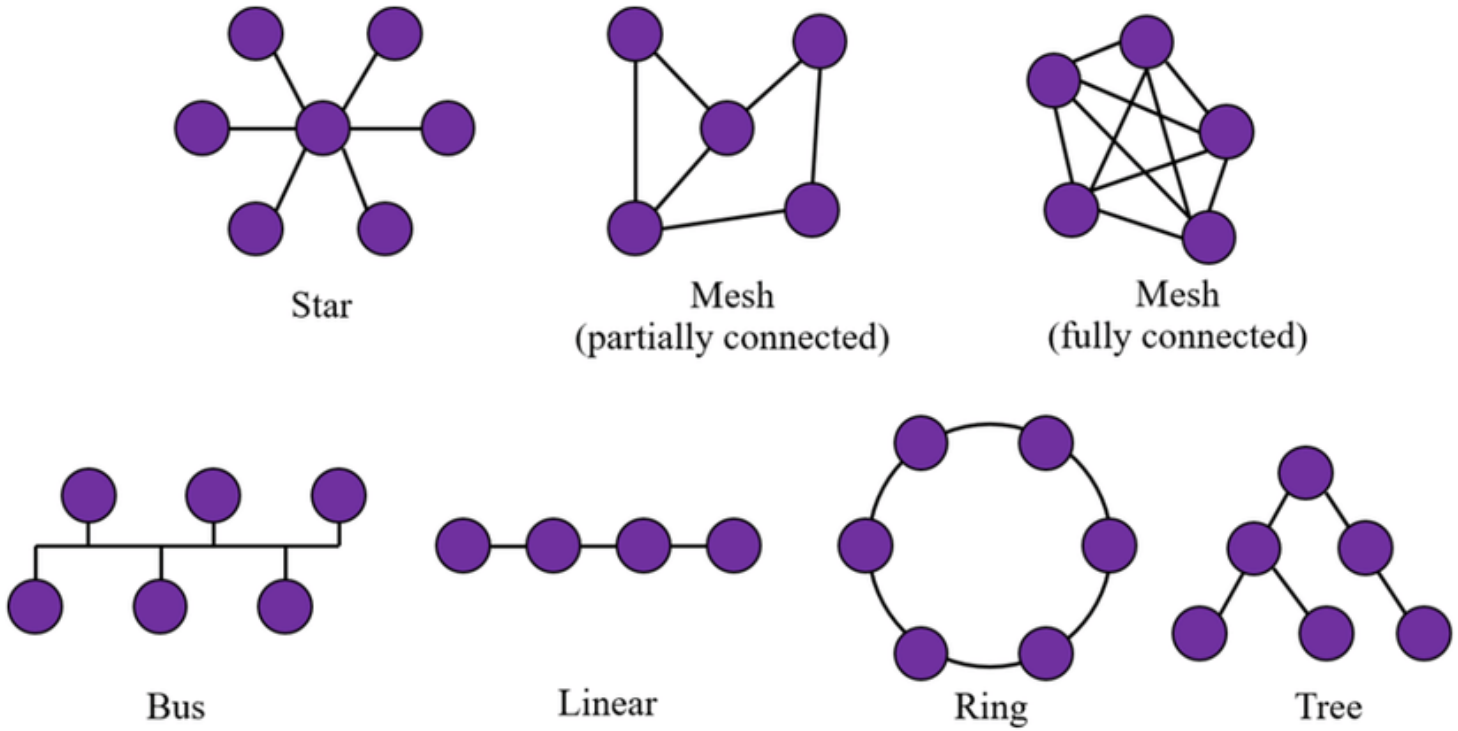
- "Виртуальная" сеть поверх физической сети
- Реализует набор сервисов
  - специфичных для приложения
  - более эффективных, чем доступные в обычной сети
  - недоступных в обычной сети
- Основные элементы
  - Топология
  - Адресация узлов
  - Протоколы
  - Алгоритмы маршрутизации



Применение оверлейных сетей

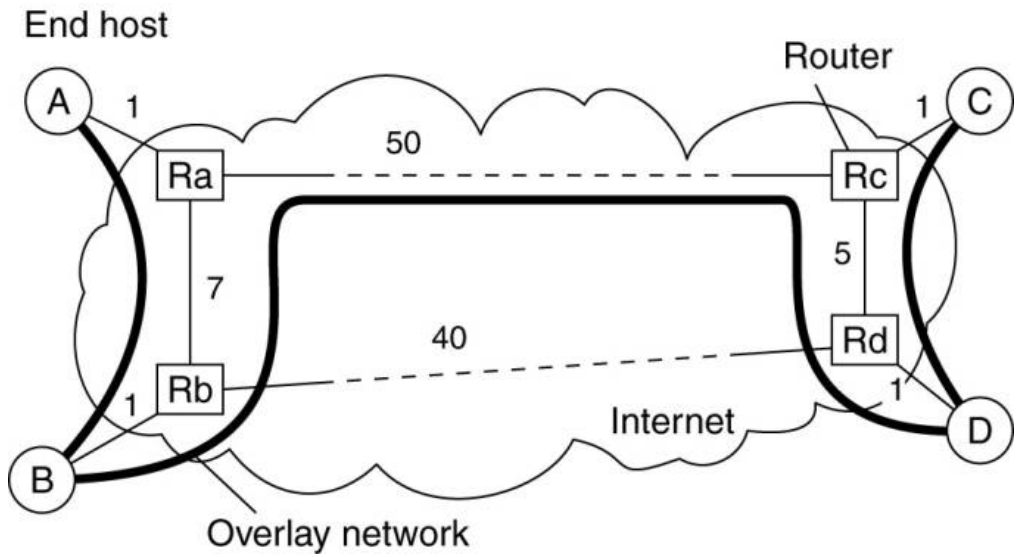
- Мультикаст
- Доставка контента, VoIP, потоковое видео
- Улучшенная маршрутизация в Интернете
- Именование и поиск (peer-to-peer сети)
- Беспроводные и самоорганизующиеся сети
- Обспечение безопасности (VPN)

Возможные топологии

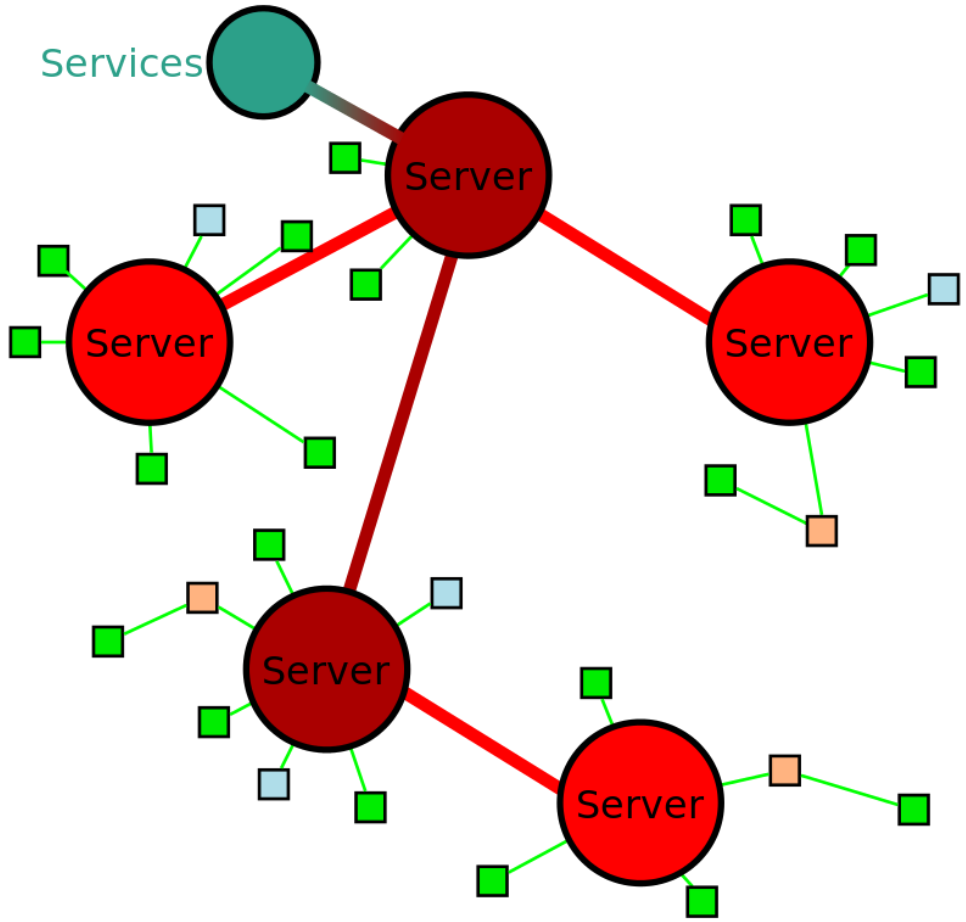


Характеристики оверлейных сетей

- Link stress
  - как часто пакет пересекает канал
- Stretch
  - отношение задержек между парой узлов в оверлейной и физической сетях
- Tree cost
  - сумма весов ребер дерева (например, задержек)



Дерево: Internet Relay Chat



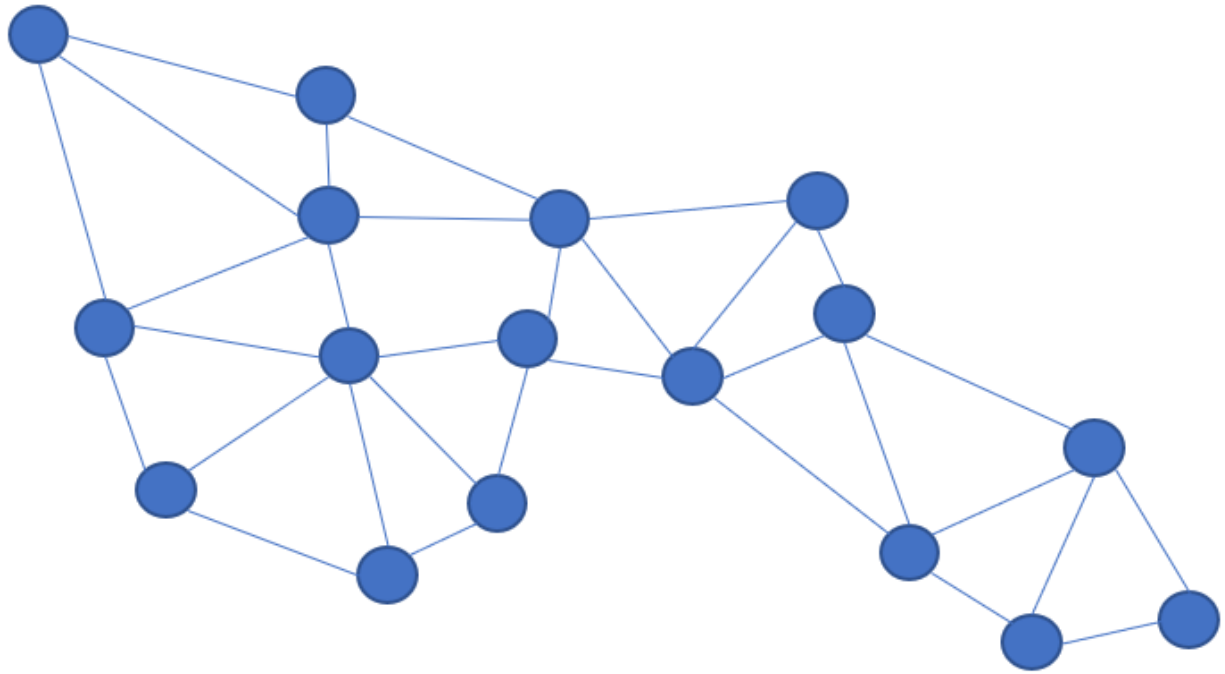
Дерево: Особенности

- Построение эффективного остовного дерева
  - Корнем является источник мультикаста
  - За основу можно взять существующую mesh-сеть
  - Или динамически определять "близость" узлов
- Добавление нового узла
  - Выбор родителя для нового узла
  - Баланс между минимизацией длин путей и нагрузкой на узлы
  - Может потребоваться переконфигурация дерева
- Починка дерева в случае отказа
- Примеры: switch-trees (литература), PlumTree (семинар)

Поддержка нескольких групп?

- Использование общей оверлейной сети
- Использование отдельной оверлейной сети для каждой группы

Mesh-сеть: Flooding и Pull

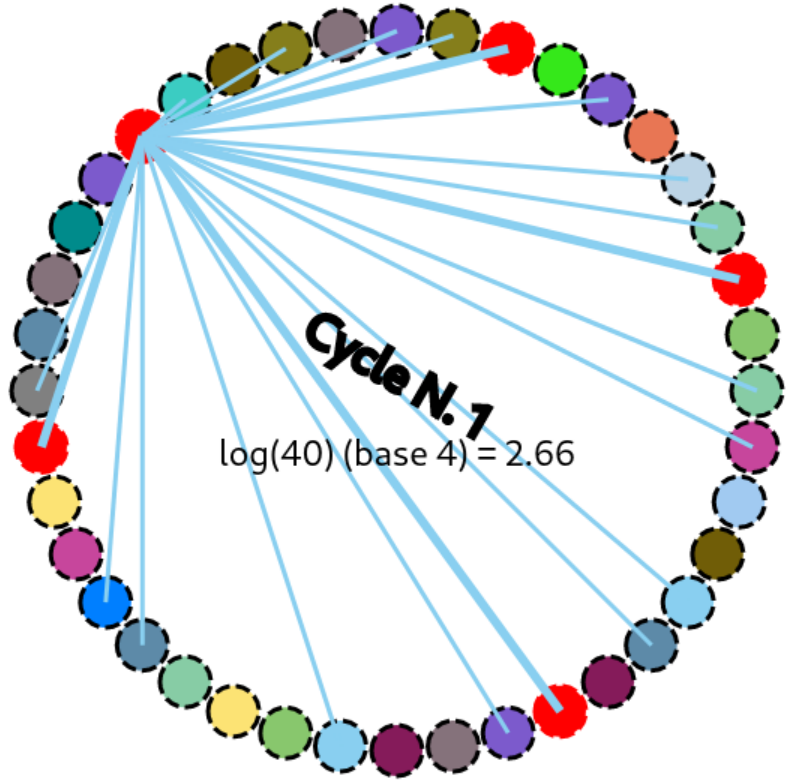


Mesh-сеть: Особенности

- Лучшая устройчивость к отказам, чем дерево
- Лучше приспособлены к динамическому составу участников (churn rate)
- Сложнее организовать эффективную рассылку
- Может требоваться буферизация полученных данных (pull)

Gossip

- Подход к распространению информации на основе локальных связей
- Аналогии с распространением слухов или болезней (epidemic protocols)
- Возможные состояния узла: infected, susceptible, removed
- В каждом раунде узел взаимодействует с одним или несколькими соседями (fanout)
- Для распространения данных на все узлы требуется *O(logN)* раундов

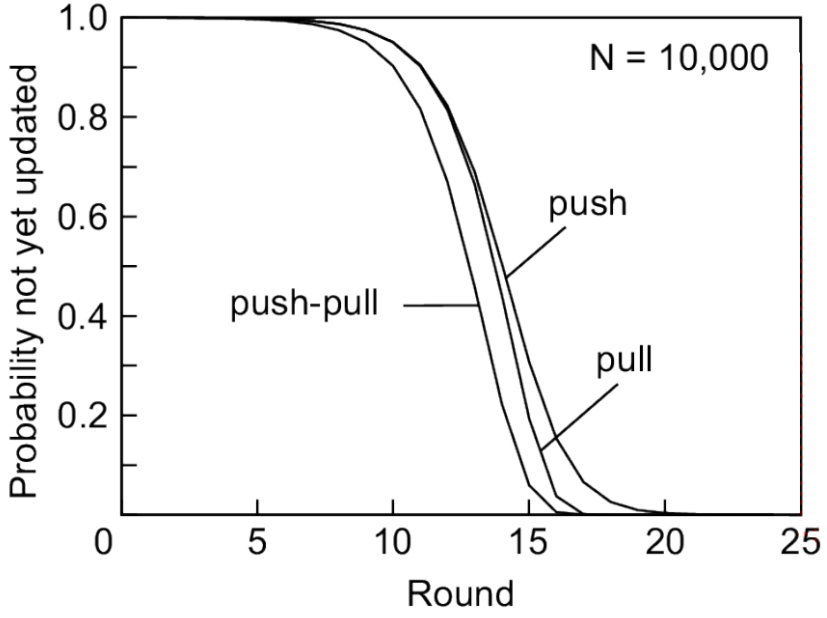


Демонстрация

https://flopezluis.github.io/gossip-simulator/

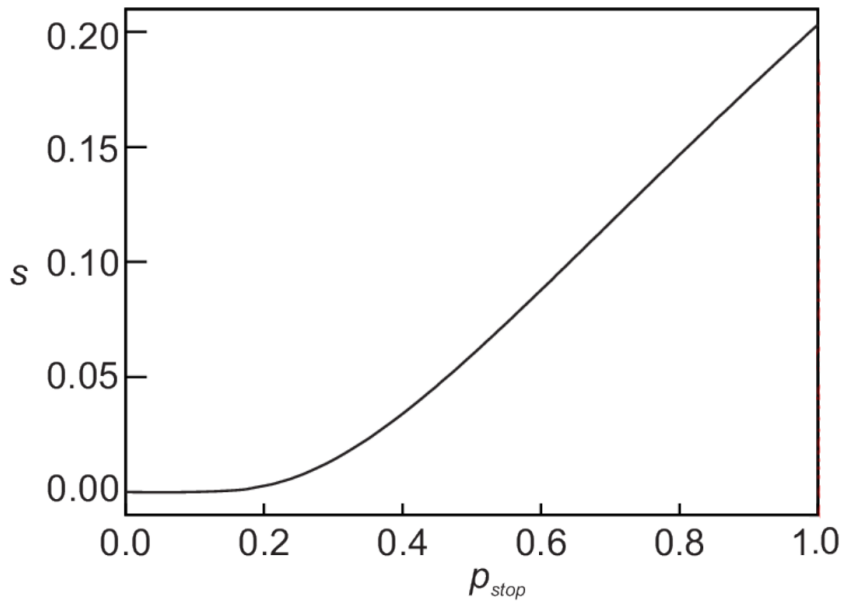
Анти-энтропия

- Узел *P* выбирает случайным образом другой узел *Q*
  - Push: *P* отправляет *Q* известную ему информацию (обновления)
  - Pull: *P* запрашивает у *Q* известную тому информацию
  - Push-Pull: *P* и *Q* обмениваются известной им информацией



Распространение слухов

- Если сосед уже имеет информацию, то узел перестает распространять ее с вероятностью *p_stop*
- Не гарантирует распространение информации до всех узлов



Gossip: Особенности

- Хорошая масштабируемость
- Адаптивность к отказам и изменениям состава
- Учёт топологии физической сети
- Выбор соседних узлов (peer sampling)

Литература

- Coulouris G.F. et al. Distributed Systems: Concepts and Design. Pearson, 2011 (разделы 4.4, 4.5, 6.2, 15.4)
- van Steen M., Tanenbaum A.S. Distributed Systems: Principles and Paradigms. Pearson, 2017. (раздел 4.4)

Литература (дополнительно)

- Peterson L., Davie B. Computer Networks: A Systems Approach (разделы 4.3, 9.4)
- Сети для самых маленьких. Часть девятая. Мультикаст.