ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КУРСОВОЙ ПРОЕКТ

по дисциплине “Сетевое программирование”

на тему

Разработка сетевого приложение. Протокол OSPF.

Выполнил студент Рудцких Владислав Евгеньевич
Ф.И.О.

Группы ИВ-222

Работу принял _____ Г.Н. Третьяков
подпись

Защищена _____ Оценка _____

Новосибирск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

ПРИНЦИП РАБОТЫ 4

ВИДЫ OSPF СООБЩЕНИЙ 8

ЗАКЛЮЧЕНИЕ 10

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 11

ПРИЛОЖЕНИЕ 12

ВВЕДЕНИЕ

Протокол OSPF, наряду с IS-IS, принадлежит к классу протоколов маршрутизации Link State. Принципы этого класса заключается в том, что в памяти маршрутизатора помимо всех оптимальных маршрутов в удаленные сети должна быть полная карта сети, в том числе с действующими связями между другими маршрутизаторами. OSPF изначально создавался как открытый протокол, что сделало его самым распространенным среди протоколов маршрутизации. Его алгоритм позволяет достаточно легко выстраивать стек протоколов для OSPF.

ПРИНЦИП РАБОТЫ

Работа протокола OSPF строится по следующему алгоритму:

1. Маршрутизаторы производят обмен малыми пакетами HELLO.
2. После выполнения обмена между ними устанавливаются соседства. Каждый из маршрутизаторов добавляет в специальную локальную таблицу соседей.
3. Маршрутизаторы выполняют сбор состояний своих связей с соседями (линков). Линки включают id самого маршрутизатора и соседа, сеть и префикс, тип сети и метрику (стоимость линка). После сбора состояний маршрутизатор формирует пакет LSA (Link State Advertisement).
4. LSA рассылается каждому соседу, который передает пакет дальше по сети.
5. После получения пакета LSA каждый маршрутизатор добавляет содержащуюся в нем информацию в локальную таблицу LSDB (Link State Database).
6. В таблице LSDB накапливаются данные обо всех парах маршрутизаторов в пределах сети.
7. На основании накопленных данных выстраивается полная карта сети, которая включает все действующие маршрутизаторы и образованные между ними связи.
8. Используя карту, каждый маршрутизатор выполняет поиск самых коротких маршрутов во все сети и формирует из них таблицу маршрутизации.

Учитывая ресурсоемкий и сложный принцип работы OSPF, от каждого маршрутизатора требуется достаточно высокая производительность и значительный объем оперативной памяти.

В случае разрыва связи с соседом у одного из маршрутизаторов, он отправляет по сети новые пакеты LCA, и повторяется вся процедура формирования таблицы маршрутизации. Чтобы исключить постоянный пересчет в крупных сетях с большим количеством маршрутизаторов, в них применяют разделение на отдельные зоны. В каждой из них выполняются автономные вычисления с передачей между зонами только итогового результата. В любой конфигурации OSPF должна присутствовать корневая зона с индексом 0. Малые сети обычно помещаются в ее пределах, а для больших — требуется формирование дополнительных зон.

Пакет OSPF помещается в пакет IP с мультикастовым адресом получателя. Отправителю же в нем соответствует адрес маршрутизатора. Пакет помещается в мультикастовый фрейм, например, в Ethernet. При формировании списков контроля доступа нужно учитывать, что OSPF инкапсулируется непосредственно в IP, а не в UDP или TCP.

Учитывая, что при построении карты сети маршрутизаторами в качестве узлов выступают другие маршрутизаторы, большое значение имеет наличие у каждого из них уникального имени. Для определения такого имени используется поле Router ID.

Идентификатор записывается в виде IP-адреса IPv4. Неважно, какое он примет значение, главное, чтобы он был уникальным в пределах этой сети. ID маршрутизатора в OSPF можно задать вручную. Если он не задан, то будет присвоен автоматически.

Принцип работы протокола OSPF предусматривает следующий алгоритм назначения ID маршрутизатора:

1. В случае явного задания идентификатора командой router-id, используется назначенный вручную ID.

2. В случае если не было ввода router-id, присваивается больший адрес из настроенных на маршрутизаторе loopback интерфейсов.
3. При отсутствии loopback интерфейсов принимается больший адрес из всех включенных на маршрутизаторе интерфейсов.

Для определения большего адреса используется прямое сравнение по октетам слева направо.

Открытый протокол маршрутизации не устанавливает отдельных требований к расчету метрики и оценки маршрутов. Его стандарт определяет стоимость каждого линка. В случае прохождения маршрута через несколько линков их стоимость суммируется. Оптимальным признается маршрут с наименьшей стоимостью. При этом принципы подсчета стоимости линка зависят от принципов, примененных конкретным производителем сетевого оборудования.

Например, Cisco применяет два варианта расчета стоимости.

В первом случае стоимость линка рассчитывается как обратная величина от его скорости (1000 — для 1 Мбит, 100 — для 10 Мбит, 10 — для 100 Мбит, 1 — для 1 Гбита и т. д.). Этот вариант подойдет при условии, что все маршрутизаторы будут считать стоимость по данному алгоритму, аэто требует использование только устройств Cisco.

Второй способ предусматривает задание стоимости администратором на основе собственного определения качества линка. Этот вариант используют в тех случаях, когда качество линка определяется не одной только его скоростью. В том числе метрика может быть завышена для линка, на котором чаще других появляются ошибки или осуществляется тарификация трафика. Этот способ применим в сетях, где установлены маршрутизаторы разных производителей.

К основным проблемам OSPF можно отнести работу протокола в сетях с множественным доступом. Распространена топология, при которой множество маршрутизаторов объединяются не через последовательное подключение друг к другу, а через общую сеть. Теоретически OSPF должен выстраивать соседства в пределах общей сети на основе принципа «каждый с каждым». Однако это требует формирования огромных таблиц, работа с которыми сильно перегружает процессор и память.

Решение этой проблемы достигается посредством механизма выбора Designated Router (DR) и Backup Designated Router (BDR), которые представляют собой роли маршрутизаторов. В сети с множественным доступом, к которой подключены более 2 маршрутизаторов, один из них назначается на роль DR, а второй — на роль BDR. При отправке любым маршрутизатором какого-либо пакета, он поступает не всем устройства в сети, а подается на отдельный мультикастовый адрес, доступный только DR и BDR. В свою очередь, DR рассылает пакет всем маршрутизаторам в сети. Такое посредничество значительно снижает нагрузку. BDR выполняет резервную функцию и моментально принимает роль DR при его отключении. После этого среди остальных маршрутизаторов сразу выбирается новый BDR.

ВИДЫ OSPF СООБЩЕНИЙ

Протокол маршрутизации OSPF поддерживает 5 типов сообщений:

1. Hello — периодически направляются для поиска соседей.
2. Database Description DBD — применяются для контроля синхронизации LSDB у соседей.
3. Link state request LSR — запрос у LSA у маршрутизатора, выполняемый принудительно. Применяется в случаях, когда маршрутизатор только включается и ему необходимо определить действующие в сети связи, а также если у него пропала сеть и нужно найти альтернативные маршруты.
4. Link state update LSU — содержит данные о состоянии связей маршрутизатора.
5. 1 – График зависимости коэффициента ускорения от числа процессовНа рисунке 2.1 показано ускорение параллельной программы при разном числе процессов. Из графика видно, что для графов с большим количеством вершин эффективность параллелизации. Для графов с небольшим количеством вершин межпроцессное общение занимает значительную часть времени и не дает такого же прироста. Также стоит учитывать, что увеличение количества процессов на число большее, чем количество строк в матрицы не принесет прироста к производительности.

Hello пакеты отправляются с установленной периодичностью. По умолчанию она составляет 1 раз в 10 секунд для сетей BMA и point-to-point и 1 раз в 40 секунд для сетей NBMA. Также существует понятие Dead-интервала, который по умолчанию равняется 4 Hello-интервалам. Если в течение этого периода маршрутизатор не получает ни одного пакета, то он считает, что сосед отключился. За этим следует пересчет и обновление таблицы маршрутизаторы.

ЗАКЛЮЧЕНИЕ

Протокол OSPF (Open Shortest Path First) является одним из наиболее распространённых и эффективных внутренних протоколов маршрутизации, используемых в современных компьютерных сетях. Благодаря своей способности быстро адаптироваться к изменениям топологии, поддержке масштабируемости и использованию алгоритма Дейкстры для определения кратчайших путей, OSPF обеспечивает стабильную и оптимальную маршрутизацию внутри автономных систем. Его гибкая структура, возможность разделения сети на области и поддержка различных типов интерфейсов делают его предпочтительным выбором для крупных корпоративных и провайдерских сетей. В целом, протокол OSPF играет важную роль в обеспечении надежной и эффективной работы современных сетевых инфраструктур, способствуя быстрому обмену маршрутной информацией и повышая устойчивость сети к сбоям.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

Протокол OSPF // https://ru.wikipedia.org/wiki/OSPF

1. Документация к протоколу OSPF // https://datatracker.ietf.org/doc/html/rfc2328

ПРИЛОЖЕНИЕ

client.cpp

```
#include <iostream>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>


int main(int argc, char **argv)
{
    int sockMain, sockServ;
    struct sockaddr_in serv;
    sockMain = socket(AF_INET, SOCK_STREAM, 0);
    if (sockMain < 0)
    {
        std::cerr << "SOCKET OPEN ERROR" << '\n';
        close(sockMain);
        return 1;
    }
    serv.sin_family = AF_INET;
    serv.sin_addr.s_addr = INADDR_ANY;
    int time = atoi(argv[2]);
    serv.sin_port = htons(atoi(argv[1]));
    socklen_t s_size = sizeof(serv);
    if ((sockServ = connect(sockMain, (struct sockaddr*) &serv, s_size)) == -1)
    {
        std::cerr << "CONNECT ERROR" << '\n';
        close(sockServ);
        close(sockMain);
        return 1;
    }
    std::cout << "CONNECTED TO SERVER" << '\n';
    for(int i = 0; i <= time ; i++)
    {
        std::cout << "send: " << time << '\n';
        if (send(sockMain, &time, 4, 0) < 0)
        {
            std::cerr << "SENDING RROR\n";
            exit(1);
        }
        sleep(time);
        recv(sockMain, &time, sizeof(time), 0);
        std::cout << "received: " << time << '\n';
```

```
    }
    close(sockServ);
    close(sockMain);
    return 0;
}
```

server.cpp

```
#include <iostream>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <signal.h>


void killzom(int sign)
{   int stat;
while(wait3(&stat, WNOHANG, (struct rusage *)0) >= 0);
}


int main (int argc, char **argv)
{
    int sockMain, sockBind, check, sockClient, stat, childs;
    struct sockaddr_in serv;
    sockMain = socket(AF_INET, SOCK_STREAM, 0);
    if (sockMain < 0)
    {
        std::cerr << "SOCKET ERROR" << '\n';
        return 1;
    }
    childs = 0;
    socklen_t size = sizeof(struct sockaddr);
    serv.sin_family = AF_INET;
    serv.sin_addr.s_addr = INADDR_ANY;
    serv.sin_port = 0;
    signal(SIGCHLD, killzom);
    sockBind = bind(sockMain, (struct sockaddr *) &serv, sizeof(serv));
    if (sockBind == -1)
    {
        std::cerr << "BIND ERROR" << '\n';
        return 1;
    }
    if (getsockname(sockMain, (struct sockaddr *) &serv, &size))
```

```
{
    std::cerr << "GET SOCKET NAME ERROR" << '\n';
    return 1;
}
std::cout << "PORT: " << ntohs(serv.sin_port) << '\n';
while (1)
{
    if ((check = listen(sockMain, 5)) == -1)
    {
        std::cerr << "LISTEN ERROR" << '\n';
        return 1;
    }
    if ((sockClient = accept(sockMain, (struct sockaddr *) 0, 0)) == -1)
    {
        std::cerr << "ACCEPT ERROR" << '\n';
        return 1;
    }
    std::cout << "| connected |" << '\n';
    childs++;
    pid_t im = fork();
    switch (im)
    {
        case -1:
            std::cerr << "FORK ERROR" << '\n';
            wait(&stat);
            return -1;
            break;
        case 0:
            while (1) {
                int buf = 0;
                ssize_t tmp = 0;
                tmp = recv(sockClient, &buf, 4, 0);
                std::cout << "received: " << buf << '\n';
                if (tmp == 0)
                {
                    close(sockClient);
                    exit(0);
                }
                send(sockClient, &buf, 4, 0);
            }
            break;
        default:
        if(childs >= 5)
        {
            wait(&stat);
            close(sockMain);
            close(sockClient);
            exit(0);
```

```
            }
            break;
        }
    }
    return 0;
}
```