Зеленский К.И., Ткаченко В.Г.
Одесская национальная академия связи им. А.С. Попова

АЛГОРИТМ МАРШРУТИЗАЦИИ IP-ПАКЕТОВ, ОБЕСПЕЧИВАЮЩИЙ ОПТИМАЛЬНУЮ ЗАГРУЗКУ СЕТИ

В настоящее время широкое распространение получили сети с передачей данных в виде IP-пакетов. Одной из проблем при маршрутизации IP-пакетов является оптимальная загрузка сети [1]. Современные алгоритмы маршрутизации распределяют нагрузку по сети неравномерно. В то время как часть линий перегружена, другие простаивают. Таким образом, сеть не выполняет предъявляемых к ней требований по пропускной способности и качеству обслуживания. При большой загрузке магистральных линий пакеты могут быть задержаны или потеряны.

Из существующих протоколов маршрутизации только некоторые, например OSPF, учитывают загрузку сети [2]. При этом для выбора оптимального маршрута используются алгоритмы, аналогичные алгоритмам поиска кратчайшего маршрута. Под кратчайшим маршрутом понимается такой маршрут, в котором сумма расстояний между узлами сети, по пути следования сообщения, минимальна. Протоколы маршрутизации используют несколько основных алгоритмов поиска пути. Чаще всего это алгоритмы Беллмана-Форда и Флойда-Уоршелла, которые построены на основе алгоритма Дейкстры.

Однако во всех этих алгоритмах на каждом шаге работы алгоритма предполагается просмотр всех вершин графа. Кроме того, представление графа включает в себя матрицу весов, размер которой n^2, где n – количество вершин графа. Если граф достаточно велик, т. е. включает в себя порядка 1000 вершин, то представление графа требует нескольких мегабайтов оперативной памяти. Если этот алгоритм используется одновременно для многих пользователей, то объем используемой памяти растет в линейной зависимости от количества подключений. Кроме того, просмотр на каждом шаге всех вершин сильно замедляет работу алгоритма. Во всех этих алгоритмах предусмотрена только минимизация суммы весов ребер.

Целью данной работы является разработка алгоритма, который позволял бы равномерно распределять нагрузку по сети, выбирая при этом для пакетов путь, который включает в себя линии с минимальной нагрузкой. Также алгоритм минимизирует использование оперативной памяти и сокращает время поиска оптимального маршрута. При этом достигается более равномерное распределение нагрузки по сети и суммарно сеть может пропустить гораздо больший поток информации. Равномерное распределение нагрузки приводит к повышению экономической эффективности при эксплуатации сети. В разработанном алгоритме, кроме минимизации по сумме весов ребер, предусмотрена также минимизация по среднему и по относительному среднему весу ребра в маршруте.

Алгоритм находит кратчайший маршрут, учитывая заданные веса ребер и присваивая в ходе работы определенные веса вершинам графа сети. При этом, в отличие от алгоритма Дейкстры, не нужно просматривать все вершины графа. Алгоритм заканчивается, когда текущей становится конечная вершина маршрута, что упрощает реализацию, уменьшает потребление памяти и увеличивает быстродействие.

В алгоритме Дейкстры в расчетах используется двумерный массив, который при больших объемах исходных данных приводит к очень большому потреблению памяти, что нежелательно [3].

В предлагаемом алгоритме для хранения узлов сети используется двумерный массив размерности $n \cdot 4$, где n – число узлов, а использование четырех ячеек для узла показано в табл. 2.

Можно еще больше уменьшить потребление памяти, не выделяя память сразу под все узлы сети, а выделяя ее динамично по мере необходимости. Предположим, что при работе алгоритма будет просмотрено только 30% узлов, тогда массив сократится еще на две трети. Это хорошо еще и тем, что на каждой итерации цикла будут просматриваться не все вершины сети. Еще одним отличием от алгоритма Дейкстры есть то, что нет необходимости присваивать вершинам бесконечно большие числа. Программно это реализуется записью в ячейку очень большого числа. Минус этого решения в том, что на решение будет израсходовано больше времени, так как на выделение памяти расходуется время. Именно такой способ работы используется в алгоритме.

Есть еще одна модификация алгоритма, в которой оперативная память не используется вообще, а все промежуточные данные записываются на жесткий диск. Однако эта модификация является самой медленной и поэтому не используется.

В алгоритме Дейкстры для гарантии того, что найденный маршрут действительно минимальный, необходимо проверить все вершины и присвоить всем им статус проверенных [4]. В описанном алгоритме, на каждом шаге выбирается минимальный вес ребра, что избавляет от необходимости проверять все вершины, так как маршрут при достижении конечного пункта будет минимальным.

В рассматриваемом алгоритме нагрузка на линию является весом ребра графа. Алгоритм для определения оптимального пути может вычислять три типа весов вершин. Простейший случай, когда вес вершины равен сумме весов ребер от начальной вершины до данной вершины. Второй случай – когда вес вершины равен среднему весу ребер от начальной вершины до рассматриваемой вершины. Третий случай – относительная средняя нагрузка. Вес вершины равен среднему относительному весу ребер. При этом под относительным весом ребра понимается отношение текущего веса ребра к максимальному весу ребра графа. Во втором и третьем случаях в ходе выполнения алгоритма необходимо хранить количество ребер от начальной до рассматриваемой вершины.

При работе алгоритм использует одну таблицу базы данных (табл. 1). Это таблица ”Участки линий связи“, в которую записываем номера линий связи (ребер), начальную и конечную вершины каждого ребра, нагрузку и максимальную пропускную способность этой линии.

Таблица 1 – Структура таблицы ”Участки линий связи“

1	2	3	4	5
Номер участка (ребра)	Начальный узел	Конечный узел	Нагрузка на линию	Максимальная пропускная способность

Шаг 1. Инициализация и индексирование

1. Выделяются начальная и конечная вершины. Запоминаются их номера.
2. Создается двумерный рабочий массив, количество строк которого равно количеству узлов, рассмотренных в ходе работы алгоритма, а количество столбцов равняется 4.
3. В массив заносится начальная вершина. Номер узла для нее устанавливается равным единице, поле со значением предыдущего узла равняется нулю, номер ребра равняется нулю, третье поле также устанавливается в нуль, четвертое поле устанавливается в единицу. Структура массива показана в табл. 2. В случае использования весов вершин второго и третьего типа в массив необходимо добавить поле пять. В нем будет храниться количество ребер от начальной вершины до рассматриваемой.

Таблица 2 – Структура массива, используемого для сохранения параметров вершины при первом типе веса вершины

1	2	3	4
Номер узла	Предыдущий узел, необходимый для определения маршрута в конце алгоритма	Суммарное расстояние ребер от начального узла к текущему	Выбирался ли узел в качестве текущего

Индексируем файл с таблицей ”Участки линий связи“ по номеру начального узла. Это значит, что ребра, которые, начинаются из этого узла, будут записаны в списке первыми, а следовательно при поиске сопредельных вершин, который выполняется на следующем шаге, будет израсходовано минимальное количество времени.

В табл. 3 показана часть таблицы ”Участки линий связи“, отсортированная по возрастанию, начиная с узла с номером 6000.

Таблица 3 – Заполненная таблица ”Участки линий связи“

Номер	Начальный узел	Конечный узел	Вес	Максимальная пропускная способность
1	6000	6001	3	10
2	6000	6002	4	12
3	6000	6003	2	32
4	6001	6025	9	33

Шаг 2. Перебор узлов, смежных с текущим узлом

Первый цикл

Организуем цикл, в котором перебираем все вершины, смежные с текущей вершиной. Номера смежных вершин считываются из проиндексированной таблицы базы данных.

1. Если вершины нет в массиве и разница между текущим весом ребра и максимальной нагрузкой этого ребра больше, чем необходимая пропускная способность для данного сообщения, заносим ее в массив.
2. Если вершина есть в массиве, сравниваем старое значение, записанное в поле 3 записи массива, с новым значением (определяем, какой путь минимальный, из нескольких путей, существующих к этому узлу).
3. Если новое значение 3-го поля меньше и различие между текущим весом ребра и максимальной нагрузкой этого ребра больше, чем необходимая пропускная способность для данного сообщения, то в записи, соответствующей рассмотренной вершине, обновляем поле 2 и поле 3.
4. Новое значение третьего поля массива, в случае весов вершин первого типа (сумма весов ребер от начальной к рассмотренной вершине), считаем так: вес ребра от текущей вершины к рассмотренной суммируем с весом ребер от начальной вершины до текущей вершины.

$$V_j = V_i + W_{i,j},$$

где i – номер текущей вершины, j – номер рассматриваемой вершины, V_i – вес текущей вершины, V_j – вес рассматриваемой вершины, $W_{i,j}$ – вес ребра от текущей вершины до рассматриваемой. При использовании весов вершин второго типа новое значение веса вершины рассчитывается по формуле

$$V_j = (V_i k_i + W_{i,j})/(k_i+1),$$

где k_i – количество ребер от начальной вершины до текущей. При использовании весов вершин третьего типа новое значение веса вершины рассчитывается по формуле

$$V_j = (V_i k_i + W_{i,j}/M_{i,j})/(k_i+1),$$

где $M_{i,j}$ – максимальная нагрузка на линии от текущей вершины до рассматриваемой. Во втором и третьем случаях в пятое поле массива необходимо записывать количество ребер от начальной вершины до рассматриваемой, равное

$$k_j = k_i + 1.$$

5. Поле 2 становится равным номеру текущей вершины.

Шаг 3. Поиск новой вершины с минимальным весом ребер

Второй цикл

1. В цикле пересматриваем все элементы массива, в которых в поле 4 записан нуль. Если в поле 4 записан нуль, то это значит, что эта вершина еще не была избрана текущей.
2. В результатах поиска ищем ту вершину, у которой поле 3 минимально (вес вершины минимален).
3. Эта вершина становится текущей. В поле 4 записывается единица.
4. Переход к шагу 4. На этом шаге проверяем, закончена или нет работа алгоритма.

Шаг 4. Условие выхода из алгоритма

1. Если текущей стала вершина, номер которой совпадает с номером конечной, переход к шагу 5. Найденный маршрут является минимальным. На каждом шаге выбирался участок пути (ребро) с минимальным весом или с минимальной суммой ребер к начальной вершине.
2. Если номер вершин не совпадает, – переход к шагу 2.

Шаг 5. Построение найденного маршрута

Определение найденного маршрута

1. Устанавливаем текущей конечную вершину.
2. Организуем цикл, в котором движемся от конечной к начальной вершине.
3. Считываем значение, записанное в колонку 2 записи текущей вершины, и устанавливаем вершину с этим номером текущей.
4. Записываем номер текущей вершины в массив. Память в массиве выделяется динамически, по мере надобности.
5. Условие выхода из цикла – номер текущей вершины совпадает с номером начальной вершины.
6. Переход на следующую итерацию цикла.

После окончания цикла в массив будут записаны номера вершин, через которые проходит маршрут. При этом маршрут будет записан, начиная с конечной вершины.

Основное преимущество предлагаемого метода проявляется при применении его к сети с большим количеством узлов.

Если взять сеть с количеством узлов 10000, то при размере типа данных integer – 4 байта, массив $10000 \cdot 10000$ ячеек займет в памяти 381,5 мегабайта, а массив $10000 \cdot 4$ займет всего 0,152 мегабайта ($4 \cdot 4 \cdot 10000 = 160000$ байт приблизительно равняется 0,152 мегабайта) Таким образом, выигрыш в количестве используемой памяти будет составлять около 2000 раз.

В заключение отметим, что в ходе проведенной работы разработан алгоритм и его программная реализация на языке Delphi, которая позволяет находить маршрут, используя один из трех видов минимизации весов ребер, что обеспечивает гибкость для решения различных задач. У каждого из трех видов минимизации есть свои достоинства. При минимизации по сумме весов ребер, будет найден путь, который будет проходить по наименее загруженным линиям, при этом не сильно отклоняясь от кратчайшего пути по расстоянию. При минимизации по среднему значению весов ребер в первую очередь будут использоваться линии местного значения и разгружаться магистральные линии для мощных потоков трафика, что позволит использовать их более полно. При минимизации по относительному среднему значению весов ребер нагрузка будет наиболее равномерно распределена по всем линиям независимо от их пропускной способности.

Литература

1. *Олифер В.Г., Олифер Н.А.* Компьютерные сети. Принципы, технологии, протоколы. – С.Пб.: Питер, 2002. – 672 с.
2. *Хизер Остерлох*. Маршрутизация в IP-сетях. Принципы, протоколы, настройки. – М.: ДиаСофтЮП, 2002. – 512 с.
3. *Ахо А., Хопкрофт Дж., Ульман Дж*. Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979. – 536 с.
4. *Харари Ф*. Теория графов.– М.: Мир, 1973. – 300 с.