Распределенные информационно-аналитические системы

Лекция № 8.

Адресат-основанная маршрутизация и алгоритмы маршрутизации по кратчайшему пути

Профессор кафедры КБ-2: д.т.н. Шатовкин Р.Р.

Учебные цели:

Изучить основы научных знаний по адресат-основанной маршрутизации; алгоритму Флойда-Уоршелла; алгоритму кратчайшего пути; алгоритму Netchange.

Учебные вопросы:

1. Адресат-основанная маршрутизация.

2. Алгоритм Флойда-Уоршелла.

3. Алгоритм кратчайшего пути.

4. Алгоритм Netchange.

Процесс (узел в компьютерной сети), в общем случае, не соединен каналом непосредственно с каждым другим процессом. Узел может посылать пакеты информации непосредственно только к подмножеству узлов, называемых ***соседями узла***.

Маршрутизация – термин, используемый для того, чтобы описать решающую процедуру, с помощью которой узел выбирает один (или, иногда, больше) соседей для посылки пакета, продвигающегося к конечному адресату. Цель в проектировании алгоритма маршрутизации – сгенерировать (для каждого узла) процедуру принятия решения для выполнения этой функции и предоставить гарантии для каждого пакета.

Ясно, что некоторая информация относительно топологии сети должна быть сохранена в каждом узле как рабочая основа для (локальной) решающей процедуры. Для этого используются ***таблицы маршрутизации***.

С введением этих таблиц проблема маршрутизации может быть разделена на две части:

1. Вычисление таблицы. Таблицы маршрутизации должны быть вычислены, когда сеть инициализирована и должны быть изменены, если топология сети изменилась.

2. Пересылка пакета. Пакет должен быть послан через сеть, используя таблицы маршрутизации.

Критерии для «хороших» методов маршрутизации следующие:

(1) Корректность. Алгоритм должен доставить каждый пакет, предложенный сети окончательному адресату.

(2) Комплексность. Алгоритм для вычисления таблиц должен использовать несколько сообщений, время и память (хранение) насколько это возможно.

(3) Эффективность. Алгоритм должен послать пакеты через «хорошие» пути, например, пути, которые обусловливают маленькую задержку и гарантируют высокую производительность всей сети. Алгоритм называется ***оптимальным***, если он использует «самые лучшие» пути.

Другие аспекты эффективности – то, как быстро решение маршрутизации может быть реализовано, как быстро пакет может быть подготовлен для передачи, и т.д.

(4) Живучесть. В случае топологического изменения (добавление или удаление канала или узла) алгоритм модифицирует таблицы маршрутизации для выполнения функции маршрутизации в изменяемой сети.

(5) Адаптивность. Алгоритм балансирует загрузку каналов и узлов, адаптируя таблицы, чтобы избежать маршрутов через каналы или узлы, которые перегружены, предпочитая каналы и узлы с меньшей загруженностью в текущий момент времени.

(6) Справедливость. Алгоритм должен обеспечить обслуживание каждому пользователю в равной мере.

Эти критерии иногда конфликтуют, и большинство алгоритмов выполняет хорошо только их некоторую часть.

Как обычно, сеть представляется как граф, где узлы графа – узлы сети, и существует ребро между двумя узлами, если они – соседи (то есть имеется канал связи между ними). Оптимальность алгоритма зависит от того, что называется «самым лучшим» путем в графе. Существует, ***несколько понятий «самый лучший», каждый с собственным классом алгоритмов маршрутизации***:

(1) Минимальное количество переходов. Стоимость использования пути измеряется как число переходов (пройденные каналы или шаги от узла до узла) пути. Минимальный переход, направляющий алгоритм, использует путь с самым маленьким возможным числом переходов.

(2) Самый короткий путь. Каждый канал статически назначен (неотрицательным) весом, и стоимость пути измеряется как сумма весов каналов в пути. Алгоритм с самой короткой дорожкой использует путь с самой низкой возможной стоимостью.

(3) Минимальная задержка. Каждый канал динамически означивается весом в зависимости от трафика в канале. Алгоритм с минимальной задержкой неоднократно перестраивает таблицы таким способом, при котором всегда выбираются пути с минимальной общей задержкой.

В первом учебном вопросе рассмотрим, что, по крайней мере, для маршрутизации с минимальным переходом и с самым коротким путем, можно направить все пакеты, предназначенные d, оптимально через дерево охватов, приложенное к d. Как следствие, отправитель пакета может игнорироваться при расчете маршрутизации.

Во втором и третьем учебном вопросах рассматриваются алгоритмы вычисления таблицы маршрутизации для статической сети с каналами, имеющими вес. Алгоритмы распределенно вычисляют самый короткий путь между каждой парой узлов, и в каждом исходном узле определяют первого соседа на пути к каждому адресату. Недостаток этих алгоритмов в том, что все вычисления должны быть повторены после изменения топологии сети: алгоритмы не масштабируемы.

Алгоритм изменяемой сети, рассматриваемый в четвертом учебном вопросе, не подвержен этому недостатку: он может адаптироваться к потере или восстановлению каналов частичным перевычислением таблиц маршрутизации. Чтобы анализ был простым, он реализован как минимальный переход, то есть число шагов принимается как стоимость пути. Возможно изменить Netchange алгоритм для работы с взвешенными каналами, которые могут теряться или восстанавливаться.

Рассматриваемые алгоритмы маршрутизации используют таблицы маршрутизации (в каждом узле) с записями для каждого возможного адресата. Это может понизить быстродействие больших сетей, состоящих из маленьких узлов.

1. Адресат-основанная маршрутизация

Решение маршрутизации при пересылке пакета обычно основано только на адресате пакета (и содержании таблиц маршрутизации), и не зависит от первоначального отправителя (источника) пакета. Маршрутизация может игнорировать источник и использовать оптимальные пути. Это не зависит от выбора частного критерия оптимальности для путей. Положим, что путь прост, если он содержит каждый узел только один раз, и путь – цикл, если первый узел равняется последнему узлу.

(1) Стоимость посылки пакета через путь P не зависит от фактического использования пути, в частности использование ребер P в соответствии с другими сообщениями. Это предположение позволяет оценивать стоимость использования пути P как функцию пути; таким образом, обозначим стоимость P как $C(P) \in \Re$.

(2) Стоимость конкатенации двух путей равняется сумме стоимостей составных путей, то есть для всякого i = 0, ..., k: $C(<u_0, u_1, ..., u_k>) = C(<u_0, ..., u_i>) + C(<u_i, ..., u_k>)$. Следовательно, стоимости пустого пути $<u_0>$ (путь от u_0 до u_0) удовлетворяет $C(<u_0>) = 0$.

(3) Граф не содержит циклов отрицательной стоимости (этот критерий удовлетворяется критерием самого короткого пути и критерием минимального перехода).

Путь от u до v, называется ***оптимальным***, если не существует никакой другой путь от u до v с более низкой стоимостью. Заметим, что оптимальный путь не всегда единственен; могут существовать различные пути с той же самой (минимальной) стоимостью.

Лемма 1. Пусть u, v ∈ V. Если путь из u в v существует в G, тогда и существует простой путь, который оптимален.

Доказательство. Так как количество простых путей конечное число, то существует простой путь от u до v, назовем его So, с наименьшей стоимостью, то есть для каждого простого пути P' из u в v C(So) ≤ C(P'). Осталось показать, что C(So) – нижняя граница стоимостей всех (не простого) путей

Запишем $V = \{v_1, ..., v_n\}$. Следовательно, удаляя из P узлы, включающие v_1, v_2 и т.д., покажем, что для каждого пути P из u в v существует простой путь P' с C(P') ≤ C(P). Положим $P_o = P$, и построим для i = 1, ..., N путь P_i следующим образом. Если v_i входит в P_{i-1}, тогда $P_i = P_{i-1}$. Иначе, запишем $P_{i-1} = <u_0, ..., u_k>$. Пусть узел u_{j1} будет первым и узел u_{j2} будет последним при вхождении v_i в P_{i-1}.

Положим $P_i = <u_0, \ldots, u_{j1}(= u_{j2}), u_{j2} + 1, \ldots, u_k>$ по построению P_i – путь из u к v и содержит все вершины из $\{v_1, ..., v_n\}$ только единожды, следовательно P_N – простой путь из u в v. P_{i-1} состоит из P_i и цикла $Q = u_0, \ldots, u_{j2}$, следовательно, $C(P_{i-1}) = C(P_i) + C(Q)$. Так как не существует узлов с отрицательным весом, то $C(P_i) \le C(P_{i-1})$ и, следовательно, $C(P_N) \le C(P)$.

По выбору So, $C(So) \le C(P_N)$, из которого следует, что $C(So) \le C(P)$. □

Если G содержит циклы отрицательного веса, оптимальный путь не обязательно существует; каждый путь может быть «побежден» другим путем, который пройдет через отрицательный цикл еще раз. Для следующей теоремы, примите, что G связный (для несвязных графов, теорема может применяться к каждому связному компоненту отдельно).

Теорема 1. Для каждого $d \in V$ существует $T_d = (V, E_d)$ такое что $E_d \subseteq E$ и такое что для каждой вершины $v \in V$, путь из v к d в T_d – оптимальный путь от v к d в G.

Доказательство. Пусть $V = \{v_1, ..., v_N\}$. Мы индуктивно построим последовательность деревьев $T_i = (V_i, E_i)$ (для $i = 0, ..., N$) со следующими свойствами

(1) Каждое T_i – поддерево G, то есть $V_i \subset V$, $E_i \subset E$ и T_i – дерево.

(2) Каждое T_i (для $i < N$) поддерево T_{i+1}.

(3) Для всех $i > 0$, $v_i \in V_i$ и $d \in V_i$.

(4) Для всех $w \in V_i$, простой путь от w к d в T_i – оптимальный путь от w к d в G.

Эти свойства подразумевают, что T_N соответствует требованиям для T_d.

Конструируя последовательность деревьев, положим $V_d = \{d\}$ и $E_o = \varnothing$ Дерево T_{i+1} построим следующим образом. Выберем оптимальный простой путь $P = \langle u_0, \ldots, u_k \rangle$ от v_{i+1} к d, и пусть l будет наименьшим индексом таким, что $u_l \in T_i$ (такое l существует, потому что $u_l = d \in T_i$; возможно $l = 0$). Теперь: $V_i = V_i \cup \{u_j : j < l\}$ и $E_{i+1} = E_i \cup \{(u_j, u_{j+1}) : j < 1\}$.

Построение проиллюстрировано представлено на рисунке 1.

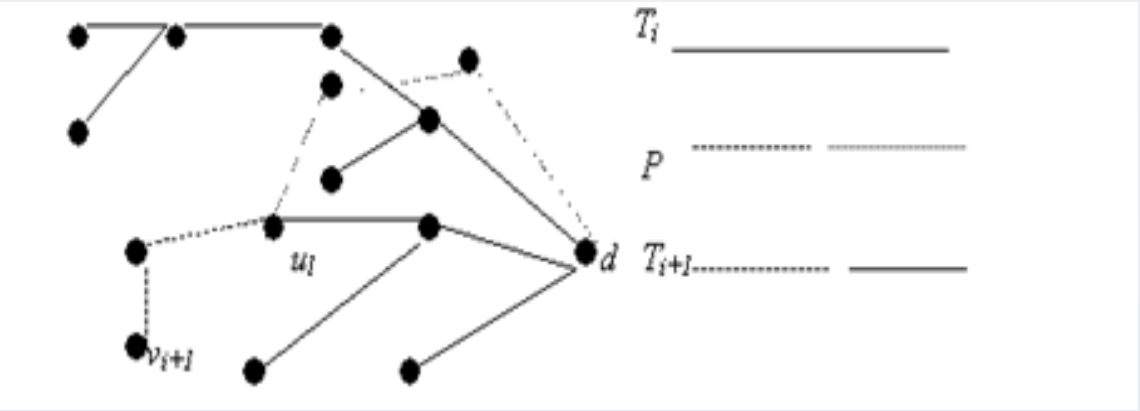

Рисунок 1 – Построение T_{i+1}

Нетрудно заметить, что T_i поддерево T_{i+1} и что $v_{i+1} \in V_{i+1}$. Чтобы увидеть, что T_{i+1} дерево, заметим, что по построению T_{i+1} связный, и число вершин превосходит число ребер на одно.

Осталось показать, что для всех $w \in V_{i+1}$ путь от w к d в T_{i+1} – оптимальный путь от w к d в G.

Для вершин $w \in V_i \subset V_{i+1}$ это следует, потому что T_i поддерево T_{i+1}; путь от w к d в T_{i+1} точно такой же, как путь в T_i, который оптимален.

Теперь пусть $w = u_j$, $j < l$ будет вершиной в $V_{i+1} \setminus V_i$. Запишем Q для пути от u i к d в T_i, тогда в T_{i+1} u_j соединена с d через путь $<u_j, \ldots, u_l>$ соединенный с Q, и осталось показать, что этот путь оптимальный в G.

Во-первых, суффикс $P' = <u_l, \ldots, u_k>$ в P оптимальный путь от u l до d, то есть C(P') = C(Q): оптимальность Q подразумевает что $C(P') \geq C(Q)$, и $C(Q) < C(P')$ подразумевает (добавлением стоимости пути), что путь $<u_0, \ldots, u_l>$ соединен с Q, имеющий меньший путь, чем P, противоречащий оптимальности P. Теперь положим, что R из u j к d имеет меньшую стоимость, чем путь $<u_j, \ldots, u_l>$, соединенный с Q. Тогда, по предыдущим наблюдениям, R имеет меньшую стоимость, чем суффикс $<u_j, \ldots, u_k>$ P, и это предполагает, что путь $<u_0, \ldots, u_j>$ соединенный с R имеет меньшую стоимость, чем P, противоречащий оптимальности P. □

Дерево охвата, приложенное к d, называется ***деревом стока*** для d, и дерево со свойством, данным в Теореме 2, называется ***оптимальным деревом стока***. Существование оптимальных деревьев стока не является компромиссом оптимальности, если только не рассматриваются алгоритмы маршрутизации, для которых механизм пересылки описывается следующим алгоритмом:

Адресат-основанная пересылка (для узла u):

(* Пакет с адресатом d был получен или сгенерирован в узле u *)

if d = u

then доставить «местный» пакет

else послать пакет к table_lookup u (d)

В этом алгоритме, table_lookup u – локальная процедура с одним параметром возвращет соседа u (в соответствии с таблицами маршрутизации). Действительно, поскольку все пакеты для адресата d могут быть направлены оптимально с использованием дерева обхвата, приложенного к d, пересылка оптимальна, если, для всего $u \neq d$, table_lookup u(d) возвращает отца u в дереве охвата T_d.

Когда механизм пересылки имеет эту форму, и никакие дальнейшие изменения топологии не происходят, корректность таблиц маршрутизации может быть удостоверена, используя следующий результат. Таблицы маршрутизации содержат цикл (для адресата d), если существуют узлы $u_1, \ldots, u_k$ такие, что для всех i: $u_i \neq d$, и для всех $i < k$: table_lookup $u(d) = u_{i+1}$ и table_lookup $u(d) = u_{j+1}$. Таблицы являются свободными от циклов, если они не содержат циклов для любого d.

Лемма 2. Механизм пересылки доставляет каждый пакет адресату, тогда и только тогда, когда таблицы маршрутизации свободны от циклов.

Доказательство. Если таблицы содержат цикл для некоторого адресата d, пакет для d никогда не будет доставлен, если источник – узел в цикле.

Примем, что таблицы свободны от циклов и позволяют пакету с адресатом d (и источник u_0) быть посланным через $u_0, u_1, u_2, \ldots$. Если один узел встречается дважды в этой последовательности, скажем $u_i = u_j$, тогда таблицы содержат цикл, а, именно, $<u_i, \ldots, u_j>$, что противоречит предположению, что таблицы являются свободными от циклов. Таким образом, каждый узел входит единожды, то подразумевается, что последовательность конечна и заканчивается, например, в узле u_k ($k < N$). Согласно процедуре пересылки последовательность может заканчиваться только в d, то есть, $u_k = d$, и пакет достиг адресата не больше, чем за $N - 1$ шагов. □

В некоторых алгоритмах маршрутизации случается, что таблицы не свободны от циклов в процессе их вычисления. Когда такой алгоритм используется, пакет может пересекать цикл в процессе вычисления таблиц, но достигает адресата не больше, чем за $N - 1$ шагов после завершения вычисления таблицы, если изменения топологии прекращаются. Если изменения топологии не прекращаются, то есть сеть подчинена бесконечной последовательности изменений топологии, пакеты не обязательно достигают своего адресата, даже если таблицы свободны от циклов во время модификаций.

Ветвящаяся маршрутизация с минимальной задержкой. При маршрутизации через путь с минимальной задержкой стоимость использования пути не может быть оценена как функция этого единственного пути. Кроме того, должен быть принят во внимание трафик канала. Чтобы избежать скопления пакетов (и возникающую в результате этого задержку) на пути, необходимо посылать пакеты, имеющие ту же самую пару «исходный узел – адресат» через различные пути. Трафик для этой пары «распределяется» в один или большее количество узлов промежуточного звена, как представлено на рисунке 2.

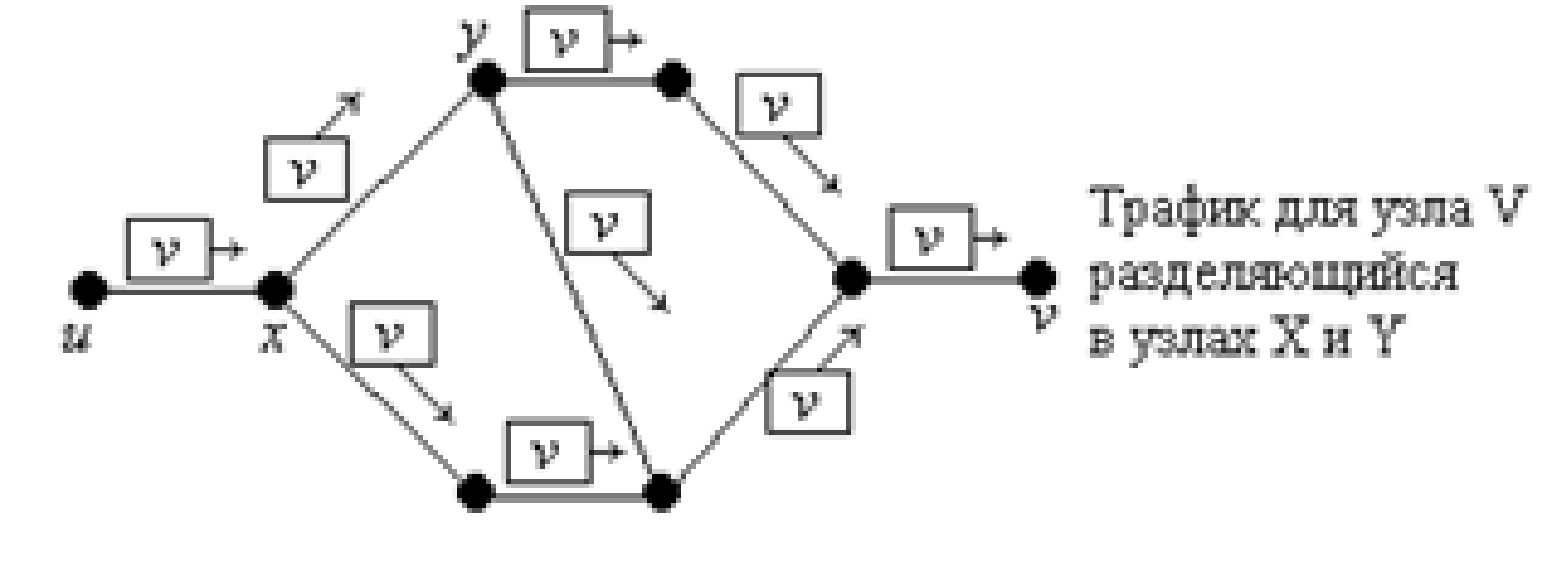

Рисунок 2 – Пример буферизованной маршрутизации

Методы маршрутизации, которые используют различные пути к одному адресату, называются ***много-путевыми*** или ***ветвящимися методами маршрутизации***.

2. Алгоритм Флойда-Уоршелла

Пусть дан взвешенный граф G = (V, E), где вес ребра uv – w_{uv}. Не обязательно допускать, что $w_{uv} = w_{vu}$, но допустим, что граф не содержит циклов с общим отрицательным весом. Вес пути $<u_0, ..., u_k>$ определяется как $\sum_{i=1}^{k} w$. Дистанция от u до v, обозначенная d(u, v), есть наименьший вес любого пути от u к v (∞ – если нет такого пути). Проблема кратчайших путей всех пар – вычисление d(u, v) для каждых u и v.

Для вычисления всех расстояний, алгоритм Флойда-Уоршелла использует понятие S-путей – это пути, в которых все промежуточные вершины принадлежат к подмножеству S из V.

Определение 1. Пусть S – подмножество V. Путь $< u_0, ..., u_k>$ – S-путь, если для любого i, $0 < i < k$, $u_j \in S$. S – расстояние от u до v, обозначенное $d^S(u, v)$, наименьший вес любого S-пути от u до v (∞ – если такого пути нет).

Алгоритм стартует перебором всех ∅-путей и увеличивает вычисления S-путей для больших подмножеств S до тех пор, пока не будут рассмотрены V-пути. Могут быть сделаны следующие наблюдения.

Утверждение 1. Для всех u и S, $d^S(u, u) = 0$. Кроме того, S-пути удовлетворяют следующим правилам для $u \neq v$:

(1) Существует ∅-путь от u к v тогда и только тогда, когда $uv \in E$.

(2) Если $uv \in E$ тогда $d^S(u, v) = w_{uv}$, иначе $d^S(u, v) = \infty$.

(3) Если $S' = S \cup \{w\}$, тогда простой S'-путь от и к v – S-путь от u к v или S-путь от u к w, соединенные S-путем от w к v.

(4) Если $S' = S \cup \{w\}$, тогда $d^{S'}(u, v) = \min(d^S(u, v), d^S(u, w) + d^S(w, v))$.

(5) Путь от u до v существует тогда и только тогда, когда V-путь от u к v существует.

(6) $d(u, v) = d^V(u, v)$.

Доказательство. Для всех u и S $d^S(u, u) \leq 0$ по причине того, что пустой путь (состоящий из 0 ребер) это S-путь от u к u с весом 0. Нет путей, имеющих меньший вес, потому что G не содержит циклов отрицательного веса, таким образом, $d^S(u, u) = 0$.

Для (1): ∅-путь не содержит промежуточных узлов, так ∅-путь от u к v состоит только из канала uv.

Для (2): следует непосредственно из (1).

Для (3): простой S'-путь от u к v содержит узел w единожды, или 0 раз как промежуточный. Если он не содержит w как промежуточную вершину, он – S-путь, иначе он – конкатенация двух S-путей, один к w и один из w.

Для (4): Это можно доказать применив Лемму 1. Получим, что, если S'-путь от u в v существует – это простой S'-путь длиной $d^{S'}(u, v)$ от u к v, такой, что $d^{S'}(u, v) = \min(d^S(u , v), d^S(u, w) + d^S(w, v))$ по (3).

Для (5): каждый S-путь – путь, и наоборот.

Для (6): каждый S-путь – путь, и наоборот, следовательно, оптимальный V-путь также оптимальный путь.

Алгоритм Флойда-Уоршелла:

```
begin (* Инициализация S = ∅и D = ∅-дистанция *)
S := 0;
forall u,v do
if u = v then D[u, v] := 0
else
if uv ∈ E then D[u, v] := w_uv
else D[u. v] := ∞ ;
(* Расширим S «центральными точками» *)
while S ≠ V do
(* Цикл инвариантен: ∀ u, v : D[u, v] = d^S(u, v) *)
begin выбрать w из V \ S ;
(* Выполнить глобальную w-центровку *)
forall u ∈ V do
(* Выполнить локальную w-центровку u *)
forall v ∈ V do
D[u, v] := min (D[u, v], D[u, w] + D[w, v]) ;
S := S ∪{w}
end (* ∀u, v : D[u, v] = d^S(u, v)*)
end
```

Используя Утверждение 1, не сложно разработать алгоритм «динамического программирования» для решения проблемы кратчайших путей всех пар. Алгоритм сначала считает $\varnothing$-пути, и, увеличивая порядок, вычисляет S-пути для больших множеств S (увеличивая S «центральными» кругами), до тех пор, пока все пути не будут рассмотрены.

Теорема 2. Алгоритм Флойда-Уоршелла вычисляет расстояние между всеми парами узлов за $Q(N^3)$ шагов.

Доказательство. Алгоритм начинает с $D[u, v] = 0$, если $u = v$, $D[u, v] = w_{uv}$, если $uv \in E$ и $D[u, v] = \infty$ в другом случае, и $S = 0$. Следуя из положений (1) и (2) Утверждения 1, $\forall$ u, v имеет силу $D[u, v] = d^S(u, v)$. В центральной окружности с центральной вершиной w множество S расширено узлом w, и означивание D[u, v] гарантирует (по положениям (3) и (4) утверждения) что утверждение $\forall$ u, v: $D[u, v] = d^S(u, v)$ сохранено как инвариант цикла. Программа заканчивает работу, когда $S = V$, то есть (по положениям (5) и (6) утверждения и инвариантну цикла) S-расстояние эквивалентно расстоянию.

Главный цикл выполняется N раз, и содержит N^2 операций (которые могут быть выполнены параллельно или последовательно), откуда и следует временная граница, заданная теоремой. □

3. Алгоритм кратчайшего пути

В предыдущем учебном вопросе рассматривался распределенный алгоритм вычисления таблиц маршрутизации – алгоритм Флойда-Уоршелла. Наиболее важным ограничением данного алгоритма является то, что граф не содержит узлов с отрицательным весом. Это ограничение соответствует реальным распределенным системам, где каждый отдельный канал означен положительной оценкой. В этом случае можно дать более строгие ограничения:

A1. Каждый узел в сети имеет положительный вес.

A2. Каждый узел в сети знает обо всех узлах (множество V).

A3. Каждый узел знает какой из узлов его сосед (хранится в $Neigh_u$ для узла u) и веса своих выходящих каналов.

Корректность алгоритма кратчайшего пути будет проще понять, если мы сначала рассмотрим предварительную версию этого алгоритма – «простой алгоритм».

Простой алгоритм. Для реализации этого распределенного алгоритма переменные и операции алгоритма Флойда-Уоршелла распределены по узлам сети. D[u, v] – переменная принадлежащая узлу u; по соглашению, это будет выражено описанием $D_u[v]$.Операция, означивающая $D_u[v]$, должна быть выполнена узлом u, и когда необходимо значение переменной узлу w, это значение должно быть послано u. В алгоритме Флойда-Уоршелла все узлы должны использовать информацию из «центрального» узла (w в теле цикла), который посылает эту информацию ко всем узлам одновременно операцией «распространения». Поэтому простой алгоритм дополнен операцией для поддержки не только длины кратчайших S-путей (как в переменной $D_u[v]$), но и первого канала такого пути (в переменной $Nb_u[v]$).

Простой алгоритм (для узла u):

var S_u : множество вершин;
D_u : массив весов;
Nb_u : массив вершин;
begin $S_u := \varnothing$;
forall $v \in V$ do
if v = u
then begin $D_u[v] := 0$; $Nb_u[v]$:= udef end
else if $v \in Neigh_u$
then begin $D_u[v] := w_{uv}$; $Nb_u[v]$:= v end
else begin $D_u[v] := \infty$; $Nb_u[v]$:= udef end ;
while $S_u \neq V$ do
begin выбрать w из $V \setminus S_u$;
(* Все вершины должны побывать вершиной w *)
if u = = w
then "распространить таблицу D_w "
else "принять таблицу D_w "
forall v Î V do
if $D_u[w] + D_w[v] < D_u[v]$ then
begin $D_u[v] := D_u[w] + D_w[v]$;
$Nb_u[v]$:= Nb[w]
end ;
$S_u := S_u \cup \{w\}$
end
end;

Утверждение что узлы сети имеет положительный вес может использоваться, чтобы показать, что не существует циклов в таблицах маршрутизации.

Лемма 3. Пусть даны S и w, и выполняется:

(1) для всех u: $D_u[w] = d^S(u, w)$ и

(2) если $d^S(u, w) < \infty$ и $u \neq w$, то $Nb_u[w]$ – первый канал кратчайшего S-пути к w.

Тогда направленный граф $T_w = (V_w, E_w)$, где $(u \in V_w \Leftrightarrow D_u[w] < \infty)$ и $(u_x \in E_w \Leftrightarrow (v \neq w \wedge Nb_u[w] = x))$ – дерево с дугами, направленными к w.

Доказательство. Во-первых, заметим, что если $D_u[w] < \infty$ для $u \neq w$, то $Nb_u[w] \neq$ udef и $D_{Nbu}[w] < \infty$. Таким образом для каждого узла $u \in V_w$, $u \neq w$ существует узел x для которого $Nb_u[w] = x$, и $x \in V_w$.

Для каждого узла $u \neq w$ в V_w существует единственное ребро в E_w такое, что число узлов в T_w превышает количество ребер на единицу, и достаточно показать, что T_w не содержит циклов. Так, $u_x \in E_w$ подразумевает, что $d^S(u, w) = w_{ux} + d^S(x, w)$, существование цикла $<u_0, u_1, .. ., u_k>$ в T_w подразумевает, что $d^S(u_0, w) = w_{u0\ u1} + w_{u1\ u2} + \ldots + w_{uk-1\ u0}\ u + d^S(u_0, w)$, то есть $0 = w_{u0\ u1} + w_{u1\ u2} + \ldots + w_{uk-1\ u0}\ u$, что противоречит предположению, что каждый цикл имеет положительный вес. □

Алгоритм Флойда-Уоршелла теперь может быть просто преобразован в простой алгоритм. Каждый узел инициализирует свои собственные переменные и исполняет N итераций основного цикла. Этот алгоритм не является окончательным решением, и он не дан полностью, потому что мы не рассмотрели, как может быть произведено распространение таблиц из центрального узла. Пока это будем использовать как обязательное условие, поскольку операция «распространить таблицу D_w» выполняется узлом w, а операция «принять таблицу D_w» выполняется другими узлами, и каждый узел имеет доступ к таблице D_w.

Некоторое внимание уделим операции «выбрать w из V \ S», чтобы узлы выбирали центры в однообразном порядке. Так как все узлы знают V заранее, то можно предположить, что узлы выбираются в некотором предписанном порядке (например, алфавитный порядок имен узлов).

Корректность простого алгоритма доказана в следующей теореме.

Теорема 3. Простой алгоритм завершит свою работу в каждом узле после N итераций основного цикла. Когда алгоритм завершит свою работу в узле u $D_u[v] = d(u, v)$, и если путь из u в v существует, то $Nb_u[v]$ – первый канал кратчайшего пути из u в v, иначе $Nb_u[v] =$ udef.

Доказательство. Завершение и корректность $D_u[v]$ по завершении работы следует из корректности алгоритма Флойда-Уоршелла (Теорема 2). Утверждение о значении $Nb_u[v]$ справедливо, потому что $Nb_u[v]$ перевычисляется каждый раз, когда означивается $D_u[v]$. □

Усовершенствованный алгоритм. Чтобы сделать распространение в простом алгоритме эффективным, узлу u для каждого $D_u[w] = \infty$ в начале w-централизованного обхода не следует менять свои таблицы в течение всего w-централизованного обхода. Если $D_u[w] = \infty$, то $D_u[w] + D_w[v] < D_u[v]$ не выполняется для каждого узда v. Следовательно, только узлы, принадлежащие T_w (в начале w-централизованного обхода) нуждаются в получении таблиц w, и операция распространения может стать более эффективной при рассылке D_w только через каналы, принадлежащие дереву T_w. Таким образом, w рассылает D_w своим сыновьям в T_w, и каждый узел в T_w, который принимает таблицу (от своего отца в T_w) пересылает ее своим сыновьям в T_w.

В начале w-централизованного обхода узел u с $D_u [w] < \infty$ знает кто его отец (в T_w) , но не знает кто его сыновья. Поэтому каждый узел v должен послать сообщение к каждому своему соседу u, спрашивая u является ли v сыном u в T_w.

Узел может участвовать в пересылке таблицы w, когда известно, что его соседи являются его сыновьями в T_w. Алгоритм использует три типа сообщений:

(1) <ys, w> сообщение (ys – обозначение для «your son») u посылает к x; в начале w-централизованного обхода, если x отец u в T_w.

(2) <nys, w> сообщение (nys – обозначение для «not your son») u посылает x в начале w-централизованного обхода, если x не отец u в T_w.

(3) <dtab, w, D> сообщение посылается в течение w-централизованного обхода через каждое ребро T_w, чтобы переслать значение D_w к каждому узлу, который должен использовать это значение.

Полный алгоритм (для узла u):

var S_u : множество узлов ;
D_u : массив весов;
Nb_u : массив узлов ;
begin
$S_u := \varnothing$;
forall v ∈ V do
if v = u
then begin $D_u[v] := O$; $Nb_u[v] :=$ udef end
else if v ∈ $Neigh_u$
then begin $D_u[v] := w_{uv}$; $Nb_u[v] := v$ end
else begin $D_u[v] := \infty$; $Nb_u[v] :=$ udef end ;
while $S_u \neq V$ do
begin выбрать w из $V \setminus S_u$;
(* Построение дерева Tw *)
forall x ∈ $Neigh_u$ do
if $Nb_u[w] = x$ then send <ys, w> to x
else send <nys, w> to x ;
$num\_rec_u := O$; (* u должен получить $|Neigh_u|$ сообщений *)
while $num\_rec_u < |Neigh_u|$ do
begin получить <ys, w> или <nys, w> сообщение ;
$num\_rec_u := num\_rec_u + 1$
end ;
if $D_u[w] < \infty$ then (* участвует в централизованном обходе*)
begin if u ¹ w
then получить <dtab, w, D> от $Nb_u[w]$;
forall x ∈ $Neigh_u$ do
if <ys, w> было послано от x
then послать <dtab, w, D>) к x ;
forall v ∈ V do (* локальный w-центр *)
if $D_u[w] + D[v] < D_u[v]$ then
begin $D_u[v] := D_u[w] + D[v]$:
$Nb_u[v] := Nb_u[w]$
end
end ;
$S_u := S_u \cup \{w\}$
end
end

Полагая, что вес (ребра или пути) вместе с именем узла можно представить W битами, сложность алгоритма показана следующей теоремой.

Теорема 4. Полный алгоритм вычисляет для каждых u и v дистанцию от u к v, и, если эта дистанция конечная, выбирается первый канал. Алгоритм обменивается 0(N) сообщениями на канал, 0(N*|E|) сообщений всего, $O(N^2W)$ бит на канал, $O(N^3W)$ бит всего, и требуется 0(NW) бит хранения на узел.

Доказательство. Полный алгоритм основан на простом алгоритме, который корректен.

Каждый канал переносит два (<ys, w> или <nys, w>) сообщения (одно в каждом направлении) и не более одного <dtab, w, D> сообщения в w-централизованном обходе, который включает не более 3N сообщений на канал. <ys, w> или <nys, w> сообщение содержит O(W) бит, а <dtab, w, D > сообщение содержит O(NW) бит, что и является границей для числа бит на канал. Не более N^2 <dtab, w, D> сообщений и 2N – |E| (<ys,w> и <nys,w>) сообщений обмена, и того всего $O(N^2 – NW + 2N – |E| – W) = O(N^3W)$ бит. Таблицы D_u и Nb_u, хранящиеся в узле u, требуют 0(NW) бит. □

В течение w-центализованного обхода узлу разрешено принимать и обрабатывать сообщения только данного обхода, то есть те, которые переносят параметр w. Если каналы удовлетворяют дисциплине FIFO, тогда сообщения <ys,w > и <nys, w> прибывают первыми, по одному через каждый канал, и затем сообщение <dtab, w, D> от $Nb_u[w]$ (если узел в V_w). Таким образом возможно опустить параметр w во всех сообщениях, если каналы удовлетворяют дисциплине FIFO. Если каналы не удовлетворяют дисциплине FIFO возможно, что сообщение с параметром w' придет пока узел ожидает сообщения для обхода w, тогда w' становится центром вместо w. В этом случае параметр используется, чтобы различить сообщения для каждого централизованного обхода, и локальная буферизация (в канале и узле) должна использоваться для отсрочки выполнения w'-сообщения.

С учетом того, что узел u_2 есть потомок u_1, если u_2 принадлежит поддереву u_1, возможна дальнейшая оптимизация алгоритма.

Лемма 4. Пусть $u_1 \neq w$, и пусть u_2 потомок u_1 в T_w в начале w-централизованного обхода, если u_2 изменит свое расстояние до v во время w-централизованного обхода, тогда и u_1 изменит свое расстояние до v в этом же обходе.

Доказательство. Так как u_2 потомок u_1 в T_w:

$d^S(u_2, w) = d^S(u_2, u_1) + d^S(u^1, w).$ (1)

Так как $u_1 \in S$:

$d^S(u_2, v) \leq d^S(u_2, u_1) + d^S(u_1, v).$ (2)

Узел u_2 изменит $D_{u2}[v]$ в данном обходе тогда и только тогда, когда

$d^S(u_2, w) + d^S(w, v) < d^S(u_2, v).$ (3)

Применяя (2), и затем (1), и вычитая $d^S(u_2, u_1)$, мы получим

$d^S(u_1, w) + d^S(w, v) < d^S(u_1, v),$ (4)

значит u_1 изменит $D_{u1}[v]$ в этом обходе. □

В соответствии с этой леммой, полный алгоритм может быть модифицирован следующим образом. После получения таблицы D_w, (сообщение <dtab, w, D>) узел u вначале выполняет локальные w-централизованные операции, и затем рассылает таблицы своим сыновьям в T_w. Когда пересылка таблицы закончилась достаточно переслать те ссылки D[v], для которых $D_u[v]$ изменилась в течение локальной w-централизованной операции. С этой модификацией таблицы маршрутизации не содержат циклов не только между централизованными обходами, но также в течение централизованных обходов.

Анализ алгоритма. Представленный алгоритм является примером того, как распределенный алгоритм может быть получен непосредственным образом из последовательного алгоритма. Переменные последовательного алгоритма распределены по процессам, и любое означивание переменной x (в последовательном алгоритме) выполняется процессом, владеющим x. Всякий раз, когда означивающее выражение содержит ссылки на переменные из других процессов, связь между процессами потребуется для передачи значения и синхронизации процессов. Специфические свойства последовательного алгоритма могут быть использованы для минимизации числа соединений.

Алгоритм кратчайшего пути прост для понимания, имеет низкую вычислительную сложность и маршрутизирует через оптимальные пути; его главный недостаток в плохой живучести. Когда топология сети меняется, все вычисления должны производится заново.

Во-первых, однообразный выбор всеми узлами следующего центрального узла (w) требует, чтобы множество участвующих узлов было известно заранее. Так как это в основном не известно априори, то вычисление этого множества должно предшествовать исполнению алгоритма кратчайших путей.

Во-вторых, алгоритм кратчайшего пути основан на повторяющемся применении $d(u, v) \leq d(u, w) + d(w, v)$. Оценивание правой стороны (u) требует информацию о d(w, v), и эта информация часто является удаленной, то есть не доступна ни в u, ни в любом из его соседей. Зависимость от удаленных данных делает необходимым транспортирование информации к удаленным узлам.

Как альтернатива, определенное ниже равенство для d(u, v) может использоваться в алгоритмах для выбора кратчайшего пути:

$d(u, v) = 0$ если $u = v$; (5)

$d(u, v) = w_{uw} + d(w,v)$ если $u \neq v$.

Два свойства этого равенства делают алгоритмы, основанные на нем, отличными от алгоритма кратчайшего пути:

(1) Локальность данных. Во время оценивания правой стороны равенства (5), узлу u необходима только информация доступная локально (именно, w_{uw}) или в соседях (именно, d(w, v)). Транспортирование данных между удаленными узлами избегается.

(2) Независимость пункта назначения. Расстояния до v (именно, d(w, v), где w сосед u) только нуждаются в вычислении расстояния от u в v. Таким образом, вычисление всех расстояний до фиксированного пункта назначения v_0 может происходить независимо от вычисления расстояния до других узлов, и также, может бать сделано обособленно.

4. Алгоритм Netchange

Алгоритм Таджибнаписа Netchange вычисляет таблицы маршрутизации, которые удовлетворяют мере «минимальное количество шагов». Алгоритм содержит дополнительную информацию, которая позволяет таблицам только частично перевычисляться после отказа или восстановления канала. Алгоритм основан на следующих предположениях.

N1. Узлы знают размер сети (N).

N2. Каналы удовлетворяют дисциплине FIFO.

N3. Узлы уведомляют об отказах и восстановлениях смежных к ним каналов.

N4. Цена пути – количество каналов в пути.

Алгоритм может управлять отказами и восстановлениями или добавлениями каналов, но, положим, что узел уведомляет, когда смежный с ним канал отказывает или восстанавливается. Отказ и восстановление узлов не рассматривается: отказ узла может рассматриваться его соседями как отказ соединяющего канала. Алгоритм содержит в каждом узле u таблицу $Nb_u[v]$, дающую для каждого пункт назначения v соседа u через которого u пересылает пакеты для v. Требования алгоритмов следующие:

R1. Если топология сети остается постоянной после конечного числа топологических изменений, тогда алгоритм завершается после конечного числа шагов.

R2. Когда алгоритм завершает свою работу таблицы $Nb_u[v]$ удовлетворяют

(a) если v = u то $Nb_u[v]$ = local;

(b) если путь из u в v ≠ u существует, то $Nb_u[v]$ = w, где w первый сосед u в кратчайшем пути из u в v;

(c) если нет пути из u в v, тогда Nb_u [v] = udef.

Описание алгоритма. Алгоритм Netchange представлен следующим образом.

Алгоритм Netchange (часть I, для узла u):

var $Neigh_u$: множество узлов ; (* Соседи u *)
D_u : массив 0.. N ; (* $D_u[v]$ – оценки d(u, v) *)
Nb_u : массив узлов ; (* $Nb_u[v]$ – предпочтительный сосед для v *)
$ndis_u$: массив 0.. N ; (*$ndis_u$ [w, v] – оценки d (w, v) *)

Инициализация:

begin forall w ∈ $Neigh_u$, v ∈ V do $ndis_u$ [w, v] := N ,
forall v ∈ V do
begin $D_u[v]$:= N ;
$Nb_u[v]$:= udef
end ;
$D_u[u]$:= 0 ; $Nb_u[u]$:= local ;
forall w ∈ $Neigh_u$ do послать <mydist, u, 0> к w
end
процедура Recompute (v):
begin if v = u
then begin $D_u[v]$:= 0 ; $Nb_u[v]$:= local end
else begin (* оценка расстояния до v *)
d := 1 + min{ $ndis_u$ [w, v] : w ∈ $Neigh_u$ } ;
if d < N then
begin $D_u[v]$:= d ;
$Nb_u[v]$:= w with 1 + $ndis_u[w, v]$ = d
end
else begin $D_u[v]$:= N ; $Nb_u[v]$:= udef end
end;
if $D_u[v]$ изменилась then
forall x Î $Neigh_u$ do послать <mydist, v, $D_u[v]$> к x
end

Алгоритм Netchange (часть 2, для узла u):

Обработка сообщения <mydist ,v, d> от соседа w :

{ <mydist ,v, d> через очередь Q_{wv} }

begin получить <mydist ,v, d> от w;

$ndis_u[w,v]$:= d ; Recompute (v)

end

Произошел отказ канала uw:

begin получить < fail, w> ; $Neigh_u$:= $Neigh_u$ \ {w} ,

forall v ∈ V do Recompute (v)

end

Произошло восстановление канала uw:

begin получить < repair, w> , $Neigh_u$:= $Neigh_u$ È {w} ;

forall v ∈ V do

begin $ndis_u$ [w, v] := N;

послать < mydist, v, $D_u[v]$> to w

end

end

Выбор соседа, через которого пакеты для v будут посылаться, основан на оценке расстояния от каждого узла до v. Предпочитаемый сосед – всегда сосед с минимальной оценкой расстояния. Узел u содержит оценку d(u, v) в $D_u[v]$ и оценки d(w, v) в $ndis_u$ [w, v] для каждого соседа u. Оценка $D_u[v]$ вычисляется из оценок $ndis_u[w, v]$, и оценки $ndis_u[w,v]$ получены посредством коммуникаций с соседями.

Вычисление оценок $D_u[v]$ происходит следующим образом. Если u = v, тогда d(u, v) = 0, таким образом $D_u[v]$ становится 0 в этом случае. Если u ≠ v, кратчайший путь от u в v (если такой путь существует) состоит из канала от u до соседа, присоединенного к кратчайшему пути, и из соседа до v, и следовательно d(u, v) = 1 + min d(w, v).

Исходя из w ∈ $Neigh_u$, узел u ≠ v оценивает d(u, v) применением этой формулы к оценочным значениям d(w, v), найденным в таблицах $ndis_u$ [w, v]. Так как всего N узлов, путь с минимальным количеством шагов имеет длину не более чем N – 1. Узел может подозревать, что такой путь не существует, если оцененное расстояние равно N или больше.

Алгоритм требует, чтобы узел имел оценки расстояний до v своих соседей. Их они получают от этих узлов послав им сообщение <mydist, ., .> следующим образом. Если узел u вычисляет значение d как оценку своего расстояния до v ($D_u[v] = d$), то эта информация посылается всем соседям в сообщении < mydist , v , d >. На получение сообщения <mydist, v, d> от соседа w, u означивает $ndis_u [w, v]$ значением d. В результате изменения $ndis_u[w, v]$ оценка u расстояния d(u, v) может измениться, и, следовательно, оценка перевычисляется каждый раз при изменении таблицы $ndis_u$ Если оценка на самом деле изменилась то, на d', например, происходит соединение с соседями, используя сообщение <mydist, v, d'> .

Алгоритм реагирует на отказы и восстановления каналов изменением локальных таблиц, посылая сообщение <mydist, ..., ...>, если оценка расстояния изменилась. Мы предположим, что уведомление, которое узлы получают о падении или подъеме канала (предположение N3) представлено в виде сообщений <fail, . > и <repair, . >. Канал между узлами u_1 и u_2 смоделирован двумя очередями, $Q_{u1\ u2}$ для сообщений от u_1 к u_2 и $Q_{u2\ u1}$ для сообщений из u_2 в u_1. Когда канал отказывает, эти очереди удаляются из конфигурации (фактически вызывается потеря всех сообщений в обоих очередях), и узлы на обоих концах канала получают сообщение <fail, .> . Если канал между u_1 и u_2 отказывает, u_1 получает сообщение <fail, u_2> и u_2 получает сообщение <fail, u_1> . Когда канал восстанавливается (или добавляется новый канал в сети) две пустые очереди добавляются в конфигурацию, и два узла соединяются через канал, получая сообщение < repair, . > . Если канал между u_1 и u_2 восстановлен, u_1 получает сообщение < repair, u_2 > и u_2 получает сообщение <repair, u_1> .

Реакция алгоритма на отказы и восстановления выглядит следующим образом. Когда канал между u и w отказывает, w удаляется из $Neigh_u$ и наоборот. Оценка расстояния перевычисляется для каждого пункта назначения и, конечно, рассылается всем существующим соседям, если оно изменилось. Это случай, если лучший маршрут был через отказавший канал и нет другого соседа w' с $ndis_u [w', v] = ndis_u [w, v]$. Когда канал восстановлен (или добавлен новый канал), то w добавляется в $Neigh_u$, но u имеет теперь неоцененное расстояние d(w, v) (и наоборот). Новый сосед w немедленно информирует относительно $D_u[v]$ для всех пунктов назначения v (посылая сообщения <mydist, v, $D_u[v$ >). До тех пор пока u получает подобное сообщения от w, u использует N как оценку для d(w,v), то есть он устанавливает $ndis_u [w, v]$ в N.

Инварианты алгоритма Netchange. Утверждение P(u, w, v) констатирует, что если u закончил обработку сообщения <mydist, v, .> от w, то оценка u расстояния d(w,v) эквивалентна оценке w расстояния d(w, v). Пусть предикат up(u, w) истинен тогда и только тогда, когда (двунаправленный) канал между u и w существует и действует. Утверждение L(u, v) констатирует, что оценка u расстояния d(u, v) всегда согласована с локальными данными u, и $Nb_u[v]$, таким образом, означен. Докажем, что эти утверждения являются инвариантами.

Инварианты P(u, w, v) и L(u, v):

$P(u, w, V) \equiv up(u, w) \Leftrightarrow w \in Neigh_u$ (6)

$\wedge\ up(u, w) \wedge Q_{wu}$ содержит сообщение <mydis t, v, d> $\Rightarrow$последнее такое сообщение удовлетворят $d = D_u[v]$ (7)

$\wedge\ up(u, w) \wedge Q_{wu}$ не содержит сообщение <mydis t, v, d> $\Rightarrow ndis_u[w, v] = D_w[v]$ (8)

$L(u, v) \equiv u = v \Rightarrow (D_u[v] = 0 \wedge Nb_u[v] = local)$ (9)

$\wedge\ (u \neq v) \wedge \exists w \in Neigh_u : ndis_u[w, v] < N - 1) \Rightarrow (D_u[v] = 1 + \min ndis_u[w, v] = 1 + ndis_u[Nbu[v], v])$ (10)

$w \in Neigh_u$

$\wedge\ (u \neq v \wedge \forall w \in Neigh_u : ndis_u[w, v] \geq N - 1) \Rightarrow (D_u[v] = N \wedge Nb_u[v] = udef)$ (11)

Выполнение алгоритма заканчивается, когда нет больше сообщений в любом канале. Эти конфигурации не терминальны для всей системы, так как вычисления в системе могут продолжится позже, начавшись отказом или восстановлением канала (на которые алгоритм должен среагировать). Мы пошлем сообщение конфигурационной стабильности, и определим предикат stable как

stable = $\forall$ u, w : up(u, w) $\Rightarrow$ Q_{wu} не содержит сообщений <mydist, ., .>.

Это предполагает, что переменные $Neigh_u$ корректно отражают существующие рабочие коммуникационные каналы, то есть, что (6) существует изначально. Для доказательства инвариантности утверждений мы должны рассмотреть три типа переходов:

(1) Получение сообщения <mydist, .,.> . Первое выполнение результирующего кодового фрагмента является автоматическим и рассматривается отдельным переходом. Обратим внимание, что в данном переходе принимается сообщение и возможно множество сообщений отправляется.

(2) Отказ канала и обработка сообщения <fail . .> узлами на обоих концах канала.

(3) Восстановление канала и обработка сообщения <repair. .> двумя соединенными узлами.

Лемма 5. Для всех u_0, w_0 и v_0, $P(u_0, w_0, v_0)$ – инвариант.

Доказательство. Изначально, то есть после выполнения инициализационной процедуры каждым узлом, (6) содержится предположением. Если изначально мы имеем Ø $up(u_0, w_0)$, (7) и (8) тривиально содержатся. Если изначально мы имеем $up(u_0, w_0)$, тогда $ndis_{u0}[w_0, v_0] = N$. Если $w_0 = v_0$, то $D_{w0}[w_0] = 0$, но сообщение <mydist, v_0, 0> в Q_{w0u0}, таким образом (7) и (8) истинны. Если $w_0 \neq v_0$, то $D_{w0}[v_0] = N$ и нет сообщений в очереди, что также говорит, что (7) и (8) справедливы. Мы рассмотрим три типа констатированных переходов упомянутых выше:

Тип (1). Предположим что u получает сообщение <mydist, v, d > от w. Следовательно нет топологических изменений и нет изменений в множестве Neigh, следовательно (1) остается истинно. Если $v \neq v_0$, то это сообщение не меняет ничего в $P(u_0, w_0, v_0)$. Если $v = v_0$, $u = u_0$, и $w = w_0$, то значение $ndis_u[w_0, v_0]$ может изменится. Однако, если сообщение <mydist , v_0, .> остается в канале, значение этого сообщения продолжает удовлетворять (7), так как (7) справедливо, то и (8) также справедливо, потому что посылка ложна. Если полученное сообщение было последним этого типа в канале, то $d = D_{w0}[v_0]$ согласно (7), которое подразумевает что заключение (8) становится истинным, и (8) справедливо. Посылка (7) становится ложной, таким образом (7) справедливо. Если $v = v_0$, $u = w_0$ (и u_0 сосед u), то заключение (7) или (8) может быть ложно, если значение $D_{w0}[v_0]$ изменилось в следствие выполнения Recompute(v) в w_0. В этом случае, однако, сообщение <mydist, v_0, .> с новым значением посылается к u_0, которое подразумевает что посылка (8) нарушена, и заключение (7) становится истинным, таким образом (7) и (8) справедливы. Это происходит и в случае, когда сообщение <mydist, v_0, .> добавляется в Q_{w0u0}, и это всегда удовлетворяет $d = D_{w0}[v_0]$. Если $v = v_0$ и $u \neq u_0$, w_0 ничего не изменяет в $P(u_0, w_0, v_0)$.

Тип (2). Предположим, что канал uw отказал. Если $u = u_0$ и $w = w_0$ этот отказ нарушил посылку, (7) и (8) в этом случае справедливы. (6) в безопасности, потому что w_0 удалился из $Neigh_{u0}$. Напротив, нечто произойдет, если $u = w_0$ и $w = u_0$. Если $u = w_0$, но $w \neq u_0$, заключение (7) или (8) может быть нарушено, так как значение $D_{w0}[v_0]$ изменилось. В этом случае пересылка сообщения <mydist, v_0, .> узлом w_0 опять нарушит посылку (8) и сделает заключение (7) истинным, следовательно (7) и (8) в безопасности. Во всех других случаях нет изменений в $P(u_0, w_0, v_0)$.

Тип (3). Предположим, добавление канала uw. Если $u = u_0$ и $w = w_0$, то $up(v_0, w_0)$ истинно, но добавлением w_0 в $Neigh_{u0}$ (и обратно) это защищает (6). Посылка <mydist, v_0, $D_{w0}[v_0]$> узлом w_0 делает заключение (7) истинным и посылку (8) ложной, таким образом $P(u_0, w_0, v_0)$ обезопасен. Во всех других случаях нет изменений в $P(u_0, w_0, v_0)$.

Лемма 6. Для каждого u_q и v_0, $L(u_0, v_0)$ –инвариант.

Доказательство. Изначально $D_{u0}[u_0] = 0$ и $Nb_u[u_0] = local$. Для $v_0 \neq u_0$, изначально $ndis_u[w, v_0] = N$ для всех $w \in Neigh_u$ и $D_{u0}[v_0] = N$ и $Nb_{u0}[v_0] = udef$.

Тип (1). Положим, что u получил сообщение <mydist, v, d> от w. Если $u \neq u_0$ или $v \neq v_0$ нет переменных упомянутых изменениях $L(u_0, v_0)$. Если $u = u_0$ и $v = v_0$, значение $ndis_u[w, vo]$ меняется, но $D_{u0}[v_0]$ и $Nb_{u0}[v_0]$ перевычисляется точно так, как удовлетворяется $L(v_0, v_0)$.

Тип (2). Положим, что канал uw отказал. Если $u = u_0$ или $w = u_0$, то $Neigh_{u0}$ изменился, но опять $D_{u0}[v_0]$ и $Nb_{u0}[v_0]$ перевычисляются точно так, как удовлетворяется $L(u_0, v_0)$.

Тип (3). Положим, что добавлен канал uw. Если $u = u_0$, то изменился $Neigh_{u0}$ добавлением w, но так как u устанавливает $ndis_{u0}[w, v_0]$ в N, это сохраняет $L(u_0, v_0)$.

Корректность алгоритма Netchange. Должны быть доказаны два требования корректности.

Теорема 5. Когда достигнута стабильная конфигурация, таблицы $Nb_u[v]$ удовлетворяют:

(1) если u = v, то $Nb_u[v]$ = local;

(2) если путь от u до v ≠ u существует, то $Nb_u[v]$ = w, где w первый сосед u на кратчайшем пути от u до v;

(3) если пути от u до v не существует, то $Nb_u[v]$ = udef.

Доказательство. Когда алгоритм прекращает работу, предикат stable добавляется к P(u, w, v) для всех u, v, и w, и это подразумевает, что для всех u, v, и w:

$up(u, w) \Rightarrow ndis_u[w, v] = Dw[v]$. (12)

Применив также L(u, v) для всех u и v мы получим

$D_u[v] = 0$, если u ≠ v;

$D_u[v] = 1 + \min D_w[v]$, если $u \neq v \wedge \exists w \in Neigh_u: D_w[v] < N - 1$;

$D_u[v] = N$ $w \in Neigh_u$, если $u \neq v \wedge \forall w \in Neigh_u: D_w[v] \geq N - 1$, (13)

которого достаточно для доказательства, что $D_u[v] = d(u, v)$ – если u и v в некотором связном компоненте сети; и $D_u[v] = N$ – если u и v в различных связных компонентах.

Во-первых, покажем индукцией по d(u, v), что если u и v в некотором связном компоненте, то $D_u[v] \leq d(u, v)$.

Случай d(u, v) = 0: подразумевает u = v, и, следовательно, $D_u[v] = 0$.

Случай d(u, v) = k +1: это подразумевает, что существует узел $w \in Neigh_u$ с d(w, v) = k. По индукции $D_u[v] \leq k$, которое согласно (13) подразумевает, что $D_u[v] \leq k + 1$.

Теперь покажем индуктивно по $D_u[v]$, что если $D_u[v] < N$, то существует путь между u и v и $d(u, v) \leq D_u[v]$.

Случай $D_u[v] = 0$: формула (13) подразумевает, что $D_u[v] = 0$ только для u = v, что дает пустой путь между u и v, и d(u, v) = 0.

Случай $D_u[v] = k + 1 < N$: формула (13) подразумевает, что существует узел $w \in Neigh_u$ с $D_w[v] = k$. По индукции существует путь между w и v и $d(w, v) \leq k$, что подразумевает существование пути между u и v и $d(u, v) < k+ 1$.

Следовательно, если u и v в некотором связном компоненте, то $D_u[v] = d(u, v)$, иначе $D_u[v] = N$. Это, формула (12) и ∀ u, v: L(u, v) и доказывает теорему о $Nb_u[v]$. □

Докажем, что стабильная ситуация в конечном счете достигается, если топологические изменения прекращаются, норм-функция в отношении stable определена. Определим, для конфигурации алгоритма g, t_i = (число сообщений <mydist, ., i>) + (число упорядоченных пар u, v таких что $D_u[v] = i$) и функцию f: f(g) = (t_0, t_1, ..., t_N), где f(g) – кортеж из (N + l) натуральных чисел, в некотором лексиграфическом порядке.

Лемма 7. Обработка сообщения <mydist, .,. > уменьшает f.

Доказательство. Пусть узел u с $D_u[v] = d_1$ получил сообщение <mydist, v, d_2>, и после перевычислил новое значение $D_u[v]$ – d. Алгоритм подразумевает что $d < d_i + 1$.

Случай $d < d_1$: Пусть $d = d_1 + 1$, что подразумевает, что t_{d2} уменьшилось на 1 (и t_{d1} следовательно), и только t_d с $d > d_1$ увеличилось. Это подразумевает что значение f уменьшилось.

Случай $d = d_1$: Не новое сообщение <mydist, ., .> посланное u, и влияет только на f так, что t_{d2} уменьшилось на 1, таким образом, значение f уменьшилось.

Случай $d > d_1$: Пусть t_{d1} уменьшилось на 1 (и t_{d2} следовательно), и только t_d с $d > d_1$ увеличилось. Это подразумевает что значение f уменьшилось. □

Теорема 6. Если топология сети остается неизменной после конечного числа топологических изменений, то алгоритм достигнет стабильной конфигурации за конечное число шагов.

Доказательство. Если сетевая топология остается постоянной только обрабатывая сообщения <mydist .,.,.>, которые имеют место, и, по предыдущей лемме, значение f уменьшается с каждым таким переходом. Это следует из хорошей обоснованности области f в которой может быть только конечное число таких переходов; следовательно алгоритм достигает стабильной конфигурации после конечного числа шагов. □

Анализ алгоритма. Формальные результаты правильности алгоритма, гарантирующие сходимость для исправления таблиц за конечное время после последнего топологического изменения, не очень хорошо показывают фактическое поведение алгоритма. Предикат stable может быть ложным практически долгое время (а именно, если топологические изменения часты) и когда stable ложен, ничто не известно о таблицы маршрутизации. Они могут содержать циклы или даже давать ошибочную информацию относительно достижимости узла назначения. Алгоритм поэтому может только применяться, где топологические изменения настолько редки, что время сходимости алгоритма является малым по сравнению с средним временем между топологических изменений. Тем более, потому что stable – глобальное свойство, и устойчивые конфигурации алгоритма неразличимы от неустойчивых для узлов. Это означает, что узел никогда не знает, отражают ли таблицы маршрутизации правильно топологию сети, и не может отсрочить отправления пакетов данных, пока устойчивая конфигурация не достигнута.

Асинхронная обработка уведомлений. Ранее мы предположили, что уведомления о топологических изменениях обрабатываются автоматически вместе с изменением в единой транзакции. Обработка происходит на обоих сторонах удаленного или добавленного канал одновременно. Лэмпорт произвел анализ мелких деталей и учел задержки в обработке этих уведомлений. Коммуникационный канал от w до u смоделирован объединением трех очередей:

(1) OQ_{wu} – выходная очередь w,

(2) TQ_{wu} – очередь передаваемых сообщений (и пакетов данных)

(3) IQ_{wu} – входная очередь u.

При нормальном функционировании канала, w посылает сообщение к u добавлением его в OQ_{wu}, сообщения двигаются от OQ_{wu} к TQ_{wu} и от TQ_{wu} к IQ_{wu}, и u получает их удаляя из IQ_{wu}. Когда канал отказывает сообщения в TQ_{wu} выбрасываются и сообщения в OQ_{wu} соответственно выбрасываются раньше чем добавились к TQ_{wu}. Сообщение <fail, w> становится в конец IQ_{wu}, и когда нормальное функционирование восстановилось сообщение <repair ,w> также становится в конец IQ_{wu}, предикаты P(u, w, v) принимают слегка более сложную форму, но алгоритм остается тот же самый.

Маршрутизация по кратчайшему пути. Возможно означить вес каждого канала и модифицировать алгоритм так чтобы вычислять кратчайший путь вместо пути с минимальным количеством шагов. Процедура Recompute алгоритма Netchange использовать вес канала uw в вычислении оценки длины кратчайший путь через w если константу 1 заменить на w_{uw}. Константа N в алгоритме должна быть заменена верхней границей диаметра сети.

Довольно просто показать, что, когда модифицированный алгоритм достигнет стабильной конфигурации, таблицы маршрутизации будут корректны и содержать оптимальный путь (все циклы в сети должны иметь положительный вес). Доказательство что алгоритм действительно достигает такого состояния требует более сложной нормфункции.

Даже возможно расширить алгоритм для работы с изменяющимися весами каналов; реакция узла u на изменение веса канала – перевычисление $D_u[v]$ для v. Алгоритм был бы практическим, однако, только в ситуации, когда среднее время изменений стоимостей каналов больше времени сходимости что не реально. В этих ситуациях должна алгоритм должен предпочесть гарантию свободы от циклов в течение сходимости.

Выводы

В ходе лекции рассмотрены следующие вопросы:

- адресат-основанная маршрутизация;

- алгоритм Флойда-Уоршелла;

- алгоритм кратчайшего пути;

- алгоритм Netchange.

Задание на самостоятельную работу

1. Конспект лекций.

Вид и тема следующего занятия

Практическое занятие №8. Dependency Injection (ч. 2)