STABILIZATION OF MAXIMUM FLOW TREES

Extended Abstract

Mohamed G. Gouda and Marco Schneider

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

Abstract

We consider networks where edges have positive capacities. The flow of a path in a network is the minimum capacity for an edge in that path. A maximum flow path is one whose flow is greater than or equal to the flow of any path with the same first and last vertices. A maximum flow tree in a network is a rooted spanning tree of the network where every path that ends at the root is a maximum flow path. Maximum flow trees are useful for establishing and maintaining virtual circuits in computer networks. In this paper, we give a distributed and stabilizing algorithm for constructing maximum flow trees in networks.

1. Introduction

Computer networks can be represented as connected, undirected graphs, where vertices represent computers and edges with positive capacities represent channels between computers. The flow of a path in such a network is the minimum capacity for an edge in that path.

Identifying maximum flow paths in networks is useful for establishing and maintaining virtual circuits in these networks. For example, to establish a virtual circuit with some capacity between two vertices in a network, a maximum flow path between the two vertices is first identified, then the required capacity is reserved along the identified path [6]. Also, if an edge on an established circuit happens to fail, a maximum flow path between the two ends of the failed edge is identified, and the identified path is made to replace the failed edge [4].

In this paper, we discuss a distributed algorithm for constructing maximum flow trees in networks. The algorithm is stabilizing in the following sense; see for example [1] and [5]. If the algorithm starts at an arbitrary, possibly illegal, state, then eventually the algorithm reaches a legal state then starts to behave correctly from that point on. Stabilization makes the algorithm adaptable to changes in network topologies and channel capacities.

2. Maximum Flow Trees

A network N is a connected and undirected graph whose set of vertices is V and whose set of edges is E. One vertex r in V is called the root of N. Associated with each edge {v, w} in E is a positive integer c.{v, w}, called the capacity of edge {v, w}.

A path in N is a non-empty sequence of vertices in V such that each pair of consecutive vertices in the sequence is an edge in E. A path is called rooted iff its last vertex is the root r of N. For example, the path that consists of the single vertex r is rooted.

The flow of a rooted path in N is the minimum capacity of an edge in the path. Thus, if a rooted path p in N is $<v_0 ; \ldots ; v_k>$, then

the flow of p =
$\min \{ c.\{v_i, v_{i+1}\} \mid 0 \le i < k \}$

A rooted path p in N is called a maximum flow path iff for every rooted path q that has the same first vertex as p in N,

the flow of p $\geq$ the flow of q

The flow of a vertex v, other than the root r, in N is the flow of a maximum flow path, whose initial vertex is v, in N.

A spanning tree T of N is called a maximum flow tree iff every rooted path in T is a maximum flow path in N.

Theorem 1:
Every network has a maximum flow tree.

Sketch of the Proof:
The proof is by construction. In particular, we show how to construct a maximum flow tree for any network.

Let N be an arbitrary network whose set of vertices is V and whose set of edges is E. It is straightforward to compute the flow F.v of every vertex v, other than the root r, in V. Also, define F.r to be the maximum capacity of an edge in E. Next, we give a construction that takes network N = (V, E) and the computed vertex flows { F.v | v **in** V }, then constructs a maximum flow tree T of N.

i. Let VT be a subset of V, and ET be a subset of E. Initially,
   VT = {r}
   ET = E
   where r is the root of N

ii. **repeat**
   for
   every vertex v **in** VT, and
   every edge {v, w} **in** ET
   do
   if ~(w **in** VT) ^
   F.w = min{F.w , c.{v, w}}
   then VT := VT ∪ {w}
   else ET := ET - {{v, w}}
   rof
   until VT = V

iii. The pair T = (VT, ET) is a maximum flow tree of N.

In the full paper, we show that the resulting pair (VT, ET) is indeed a maximum flow tree of N.
[]

3. Stabilizing Maximum Flow Trees

In this section, we present a distributed algorithm for constructing a maximum flow tree for an arbitrary network N = (V, E). The algorithm is distributed because it consists of several programs, one for each vertex in V. Moreover, the program of each vertex has a small number of constants and variables.

In particular, the program of each vertex v, other than the root r, has only four constants:

An upper bound, B, on the number of vertices in network N.

An upper bound, C, on the capacity of any edge in N.

The set H of neighbors of vertex v in network N.

The capacity c[w] for each edge {v, w} incident at vertex v in N.

The program of each vertex v also has three variables: p.v, d.v, and f.v. When the algorithm terminates, p.v will be the parent of vertex v in the constructed maximum flow tree T, d.v will be the number of edges form vertex v to the root r in tree T, and f.v will be the flow of v.

The program of each vertex v has two actions. Each action is as follows.

<guard.w> --> <statement.w>

where <guard.w> is a boolean expression over the variables of v and the variables of a neighbor w of v, and <statement.w> is a sequence of assignment statements that assign values to variables of v based on the current values of variables of v and its neighbor w.

An action is executed only when there is a neighbor w of v which makes the boolean expression <guard.w> true. The action is executed by executing the sequence of assignment statements in <statement.w>.

The program for any vertex v, other than the root r, in network N is as follows.

```
const
   B : positive integer,
   /* B is an upper bound on |V| */

   C : positive integer,
   /* C is an upper bound on the */
   /* capacity of an edge in E   */

   H : set {w | w is a neighbor of v in N },
   /* H is set of neighbors of v in N */

   c  : array [H] of 0..C
   /* c[w] = capacity of edge {v, w} */

var
   p.v :  H,    /* parent of v in T */
   d.v :  0..B, /* distance of v from r */
   f.v :  0..C  /* flow of v  in T*/

par
   w :  H

action
   p.v = w    ^
   ( d.v ≠ min {d.w + 1, B} ∨
     f.v ≠ min {f.w, c[w]}
   ) -->
   p.v := w;
   d.v := min {d.w+1, B};
   f.v  := min {f.w, c[w]}

[] p.v ≠ w     ^
   d.w < B-1  ^
   ( (d.w + 1 - min {f.w, c[w]*B}) <
     (d.v - f.v*B)
   ) -->
   p.v := w;
   d.v := min {d.w+1, B};
   f.v  := min {f.w, c[w]}
end
```

The program for the root vertex r has only two constants, B and C, two variables, d.r and f.r, and one action. This program is as follows.

```
const
   B : positive integer,
   /* B is an upper bound on |V| */

   C : positive integer,
   /* C is an upper bound on the */
```

```
    /* capacity of an edge in E     */
var
    d.r :  0..B,  /* distance of v from r */
    f.r :  0..C  /*"flow of r" in N       */
action
    d.r ≠ 0  ∨  f.r ≠ C  –>  d.r := 0 ; f.r := C
end
```

4. Proof of Correctness

In the full paper, we prove the correctness of the above distributed algorithm. (The proof is similar to those given in [1], [2], and [3].) In particular, we prove that the distributed algorithm satisfies the two properties of closure and convergence. Before we can state these properties, we need to define the concepts of algorithm states, fixed points, and computations.

A state of the algorithm is defined by one value for each variable in each program in the algorithm.

A state of the algorithm is called a fixed point if at that state no action at any program in the algorithm can be executed.

A computation of the algorithm is a maximal sequence of algorithm states

s.0 ; s.1 ; s.2 ; ...

such that the following two conditions are satisfied. First, at each state s.i, that is not the last state, in the sequence, at least one action in a program in the algorithm can be executed, and execution of this action starting at state s.i yields state s.(i+1). Second, the last state in the sequence, if any, is a fixed point.

We can now define the two properties of closure and convergence. (Proving that the algorithm satisfies these properties is given in full paper.)

Closure: At any fixed point of the algorithm, the p.v variables define a rooted spanning tree whose root is vertex r, and each f.v variable defines the flow of vertex v.

Convergence: Any computation that starts at an arbitrary state of the algorithm is finite; i. e. is guaranteed to reach a fixed point of the algorithm.

References

[1] A. Arora and M. G. Gouda, "Distributed Reset", Proc. of the tenth Conference on Foundations of Software technology and Theoretical Computer Science, Lecture Notes on Computer Science 472, Springer-Verlag, 1990.

[2] A. Arora, M. G. Gouda, and T. Herman, "Composite Routing Protocols", Proc. of the Second IEEE Symposium on Parallel and Distributed Processing, 1990.

[3] N. S. Chen, F. P. Yu, and S.T. Huang, "A Self-Stabilizing Algorithm for Constructing Spanning Trees", Information Processing Letters, Vol. 39, pp. 147 - 151, 1991.

[4] C. E. Chow, J. D. Bickell, and S. Syed, "Performance Analysis of Fast Distributed Link Restoration Algorithms", Accepted in the International Journal of Digital and Analog Communications Systems, 1994.

[5] M. Schneider, "Self-Stabilization", ACM Computing Surveys, Vol. 25, No. 1, March 1993.

[6] M. Schneider, Ph. D. Dissertation, The University of Texas at Austin, in preparation, 1994.