Minimizing the traffic in the information distribution

Frédéric Havet
Projet Mascotte, INRIA Sophia-Antipolis,
20004 route des Lucioles BP 93,
06902 Sophia-Antipolis Cedex, France
e-mail:fhavet@sophia.inria.fr

and

Marc Wennink
BT exaCT Research,
B29 136, Adastral Park,
Martlesham Heath, IP5 3RE, United Kingdom
e-mail:marc.wennink@bt.com

February 9, 2001

Abstract: In this paper, we introduce the Push Tree problem which contains elements from both the Steiner Tree and the Shortest Path problem. The Push Tree problem deals with the trade-offs between the push and pull mechanisms used in information distribution and retrieval. We present some initial complexity results and analyse several heuristics. Moreover, we discuss what lessons can be learned from the static and deterministic Push Tree problem for more realistic scenarios characterised by high uncertainty and changing information request and update patterns.

1 Problem formulation and observations

Let $G = (V(G), E(G))$ be an undirected graph, and let $l(e) = l(u, v) > 0$ be the length of any edge $e = (u, v) \in E(G)$. Moreover, let $l(H) = \sum_{e \in E(H)} l(e)$ be the length of any subgraph H of G. Then, the Push Tree problem can be formulated as follows.

Given a graph G, a source node $s \in V(G)$, *an* update rate $\mu \geq 0$, *a* request set $R \subseteq V(G)$, *and* request rates $r(v) \geq 0, \forall v \in R$. *Find a subgraph PT of G, with $s \in V(PT)$, and paths P_v from v to any node in $V(PT)$, $\forall v \in R$, such that*

$$\mu l(PT) + \sum_{v \in R} r(v) l(P_v)$$

is minimal.

The source node s contains a piece of information that changes μ times per time unit. After every change, the updated information is *pushed* from s to all nodes in $V(PT)$ using the edges in $E(PT)$. Multicast is used, which means that only one update message is sent over every edge. We measure the size of the traffic associated with a message as the number of bits times the distance covered. The total amount of update traffic per time unit is therefore equal to $\mu l(PT)$.

Requests for the information stored at s occur $r(v)$ times per time unit at node $v \in R$. Each request results in a request message to be sent over the *pull* path P_v from v to a node in PT which contains an updated copy of the information. This node then sends a reply message to v using the same path, but now in reverse direction. The total amount of request traffic per time unit is therefore equal to $\sum_{v \in R} r(v) l(P_v)$.

This results in the total traffic being equal to $\mu l(PT) + \sum_{v \in R} r(v) l(P_v)$.

It is easy to see that, in the optimal solution, PT is a tree. Moreover, for a given tree PT, the optimal path P_v for a request node v is a shortest path from v to any node in PT. Moreover there exists an optimal solution to the push tree problem such that the graph T formed by the push tree and all the pull paths is a tree. Finding such a shortest paths is easy. Therefore, the main difficulty is to find the optimal push tree PT.

Suppose that μ is *sufficiently* small (e.g., $\mu < r(v), \forall v \in R$). Then all nodes in R should be included in the push tree, and the optimal push tree is the minimum tree connecting all nodes in $R \cup \{s\}$. This tree is known as the *Steiner Tree* of $R \cup \{s\}$ in G. For general graphs, finding the Steiner Tree is known to be an NP-hard problem and therefore so is the Push Tree problem. On the other hand, if μ is sufficiently large (e.g., $\mu > \sum_{v \in R} r(v)$), then the optimal push tree consists of the single node s. The push tree problem then is equivalent to finding shortest paths from every node in R to s. In a graph with n nodes and m edges, this can be achieved in $O(m + n \log n)$ time.

The Push Tree problem can be solved in linear time if G itself is a tree. Indeed, let $T = (V, E)$ be a tree with a source $s \in V$ and a request set $R \subset V$. Removing any edge $e \in E$ from T result in the tree breaking down into two components, V_e^1 that contains s, and V_e^2. Let $\lambda(e) = \sum_{v \in V_e^2} r(v)$. Then e is in the push tree if and only if $\lambda(e) > \mu$. Thus finding the push tree only requires $O(|E|)$ time. Note that the lengths of the edges are irrelevant for determining the optimal push tree. The corresponding traffic, however, depends on these lengths. We denote by $w(e) = l(e). \min\{\lambda(e), \mu\}$

the traffic that passes through edge e. The total traffic size then is $w(T) = \sum_{e \in E} w(e)$. We denote by w^* the amount of traffic in the optimal solution. If the request function that we are considering needs to be identified, we use the notations $w_r(T)$ and w_r^*.

2 Heuristics

We study a particular class of heuristics for the general Push Tree problem which uses a two step approach. In the first step, a *routing tree* T is constructed and it connects the source node and all the request nodes. In the second step, the optimal push tree in T is obtained by solving the problem in the routing tree. Each heuristic in this class is thus characterised by the used method for obtaining the routing tree.

This class of heuristics is attractive for two reasons. The first is its simplicity, which is combined with a low time complexity for many choices of the tree construction method. The second reason relates to the practical situations in which the Push Tree problem occurs. In many cases, it is not practical to completely change the push tree whenever a change in the update rate or any of the request rates is observed. It may, however, be feasible to maintain a stable infrastructure over which adaptations to changing update and request patterns can be applied efficiently and in an intrinsically distributed manner. A tree may form a suitable basis for such an infrastructure.

Since the Steiner Tree problem is a particular case of the Push Tree problem, we construct the routing tree with heuristics for the Steiner Tree problem for the source and request nodes. With such classical heuristics, we obtain a routing tree T such that $w(T) \leq 2\frac{\mu}{r_m} w^*$, where r_m is the smallest non-zero request rate. We also show that the ratio $2\frac{\mu}{r_m}$ is tight.

3 Robustness

The above heuristics are attractive because they generate an infrastructure, while still allowing rapid adaptations to changing traffic patterns. We study the sensitivity of solutions obtained in this way to changes in update and request rates. Suppose that we have obtained a routing tree T that results in an optimal or near-optimal (with approximation ratio ρ) push tree for a particular request function r. How good is this routing tree if the request function is modified?

We discuss two different scenarios. In the first one, the set of request nodes R remains unchanged, and all request rates change almost uniformly, that is the new request function r' satisfies $(1-\xi)r \leq r' \leq (1+\varepsilon)r$. We show $w_{r'}(T) \leq \rho\frac{1+\varepsilon}{1-\xi} w_{r'}^*$.

In the second scenario, R changes. This corresponds to $\xi = 1$ or $\varepsilon = \infty$. And indeed, for any $\rho \geq 1$ and M we can find an example such that $w_r(T) \leq \rho w_r^*$ and $w_r(T) \geq M w_{r'}^*$ when r' is obtained from r by adding or removing a single request node. However, if r' is obtained from r by adding no more than $\kappa\mu$ requests, we can show that $w_{r'}(T) \leq (\rho + \kappa) w_{r'}^*$. For the case of removing k request nodes, we show $w_r'(T) \leq (3 - \frac{r_m}{\mu}) w_{r'}^*$, if $k = 1$ and T is optimal for r, and we conjecture $w_r'(T) \leq (2k + 1 - \frac{r_m}{\mu}) w_{r'}^*$, for any k. Unfortunately, for any $\rho > 1$ and M, one can find a ρ-approximating tree T for the request function r in a graph such that if we remove the request at a particular vertex $w_{r'}(T) \leq M w_{r'}^*$. However such a tree T, can not be obtained by one of the above heuristics.