Stateless Termination Detection

Gideon Stupp

Sheer Networks, Israel 67021. stupp@sheernetworks.com

Abstract. The switches and routers of large scale networks cannot manage a state per user session. The burden of memory management would overwhelm the network. Therefore, it is important to find distributed network algorithms which hold a state only at the initiating node. Termination detection algorithms are particularly interesting, since they can be used in the implementation of other stateless algorithms.
The importance of stateless termination detection is apparent in mulitcast trees. Multicast trees are commonly used to multicast messages across the network. In many cases the mulitcast message represents a request sent from the root node that must be answered by all the leaves of the tree. In most networks the leaves could send their answer directly to the root. Unfortunately, the root would have no way of knowing when all the leaves answered the request. Broadcast-echo algorithms are often used in this case, but these algorithms require a state in the internal nodes of the mulitcast tree. Nack oriented protocols are also common, particularly in reliable multicast implementations. These algorithms are optimized for continues downstream information from the source to the destinations rather than for transactional request-reply operations.
We present a simple algorithm for termination detection in trees and DAGs which does not require managing a state in the nodes of the graph. The algorithm works even if the graph changes during the execution. For a tree with n nodes, the number of bits added to each message is $O(\log n)$. We also discuss how this algorithm may be used in general graphs.

1 Introduction

Network cores are usually comprised of many switching nodes that are connected to each other by point to point communication links. Users may be connected to some of the switches, and the switching fabric enables communication between any two users by forwarding messages between the switches.

When a user needs to send the same information to many other users, multicast trees are commonly used. A tree is defined over the network where the originating sender is the root and the destinations are the leaves. When the root sends packets they follow the edges of the tree, and duplicate only at the branching points. This way the same packet is never sent twice over the same link. This scheme is used in IP multicast and in ATM point to multipoint connections.

Using mulitcast trees is much more efficient than sending a different message to each of the destinations. However, there is no simple way for the root to know how many nodes received its multicast. This is important if the root must wait

D. Malkhi (Ed.): DISC 2002, LNCS 2508, pp. 163–172, 2002.

for an answer from all the destinations (a distributed query). One solution is to use an echo-broadcast algorithm, where the leaves of the tree send their answer to their parents and each internal node in the tree waits for all the answers from all its children before propagating the result to its parent. While such solutions may be efficient in theory, they are impractical in data communication networks for several reasons:

1. Network cores can not hold a state per user session. The memory management would overwhelm the network switches.
2. The aggregation of the information may be specific to the executing algorithm. It is unlikely that in the near future network cores will support user logic (but see [1]).
3. Backtracking the multicast tree may not be the most efficient way to return data to the root. The structure of the multicast tree may be affected by parameters such as quality of service, or which nodes support multicast. It may be more efficient for the leaves to send a point to point reply directly to the root.

Reliable multicast protocols usually implement some sort of termination detection to verify that all the packets reached their destinations and theoretically could be used for distributed queries. However, these protocols were created to support a large and continuing flow of information from the source to the destinations, where upstream acknowledgments are a major overhead. The information flow in distributed queries is very different since the result of the query must be returned to the source. As acks can be piggybacked on actual information that is flowing upstream, ack implosion is not a problem.

For this paper we model modern data communication networks using a variant of the standard asynchronous message passing model. First, we assume that all nodes can communicate with all other nodes and that the cost of transferring the same message between any two nodes is the same. This property captures the concept of an underlying unsaturated communication infrastructure, where the cost of sending a message through several hops is almost the same as through one hop. Next, we assume that only the initiating node of any algorithm execution can manage a state dedicated for that execution. This property protects the network infrastructure from the affects of any specific execution.

For simplicity we assume that messages are not lost and that nodes do not fail. Since only the root manages a state, a simple timeout mechanism can be used in practice.

Algorithms that do not hold a state in the nodes sometimes move the state to the message. For example, a trivial solution for detecting termination in a multicast tree would be to have each message keep a history of the nodes it visited and the number of children each node had. The leaves return the messages to the root, which can then reconstruct the entire tree. Such solutions are inefficient in the size of the messages and in the amount of memory used in the root. In this paper we suggest an algorithm for termination detection in trees and DAGs which uses $O(\log n)$ bits in each message (n = number of nodes in the tree) and $O(n)$ bits in the root. The algorithm is based on a new counter that is

carried by the messages, much like a hop count. The new counter, which we call the *bifurcation count (b-count)*, gives an estimate on the amount of duplications created by a single branch of the multicast tree.

Definition 1. *Let $T = (V, E)$ be a tree. For any node v, denote by v_c the number of children of v and by $A(v)$ the set of ancestors of v (i.e., the set of nodes on the path from the root to v). The* bifurcation count (b-count) *of node v, v_b, is defined to be the logarithm of the multiplication of the number of children of all the ancestors of v with respect to base two. Formally:*

$$v_b = \begin{cases} 0 & A(v) = \emptyset \\ \log(\prod_{u \in A(v)} u_c) & A(v) \neq \emptyset \end{cases} \quad (1)$$

One of the simplest uses of b-count is to limit the amount of messages created by mulitcast using a TTL. However, it can also be used more sophisticatedly. In our algorithms the b-count of messages that reach the leaves is sent back to the root and the root uses this information to estimate the expected number of acknowledge messages.

This work was done as part of the development of a large scale distributed object oriented network management system (Sheer MetroCentral$^{\text{TM}}$). In this system, hundreds of thousands of autonomous objects communicate by sending asynchronous messages to each other. Many queries in the system are implemented by accessing one object, which delegates the request to one or more other objects, and so on. Thus, multiple concurrent sessions of multicast requests naturally occur. The algorithm presented in this paper was designed to replace a standard broadcast-echo implementation. In OO programming it is common to delegate from one object to just one other object without branching. Thus, in many cases there are a lot of internal nodes in the multicast tree and only a few leaves, which makes backtracking extremely inefficient.

Related Work: Computation trees have long been used to detect termination, either for diffusing algorithms [2] or for decentralized computations [3]. These algorithms are usually used for detecting termination in multicast trees, but require bidirectional links and more importantly, a state in the internal nodes of the multicast tree. Wave based solutions [4], both synchronous and asynchronous [5, 6] are sometimes used instead of computational trees, to optimize the average message complexity, yet they too manage a state in each node.

A considerable amount of research was done in reliable multicast algorithms, which use termination detection to ensure the delivery of packets ([7,8]). These algorithms are typically optimized for transferring streams of information from the root of the tree to the destinations, rather than for supporting request-reply sessions.

In Sect. 2 we present an algorithm for termination detection in binary multicast trees. In Sect. 3 we make a small adjustment to make it practical for general trees and directed acyclic graphs (DAGs).

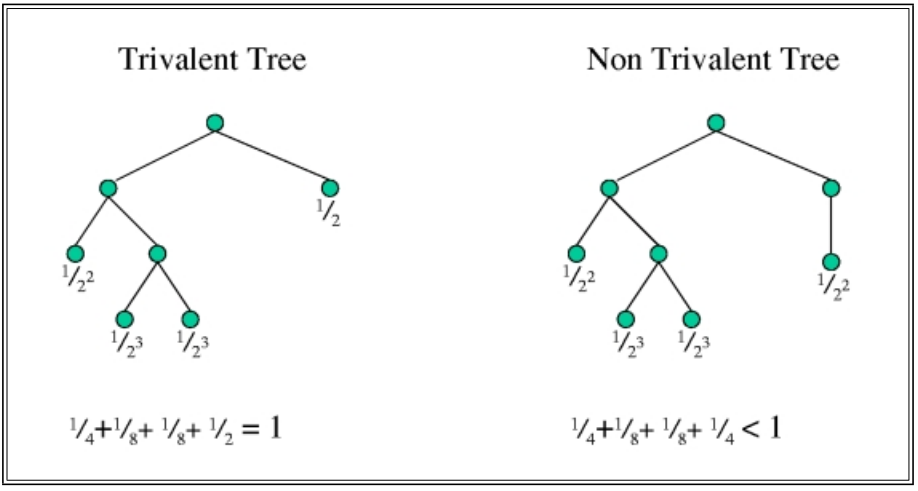

Fig. 1. Sum of weights of the leaves of a trivalent tree.

2 Binary Trees

Algorithm 1 presents the termination detection code in the root and the internal nodes of a binary multicast tree. The Algorithm is based on a well known property of trivalent trees which is described in Theorem 1. The theorem states that if branches are not missing from a binary tree (i.e., it is a trivalent tree) then a weight can be distributed between the leaves of the tree, according to their depth, such that the total weight is always 1 (See Figure 1).

Definition 2. *A* trivalent *tree (also known as a 3-cayley tree or a boron tree [9]) is a binary tree where the number of children of every node is either 2 or 0.*

For trivalent trees, the bifurcation count of node v is simply the depth of v.

Let $T = (V, E)$ be a weighted binary tree. Denote by $w(v)$ the weight of node v and by $S(T) = \sum_{v \in V} w(v)$ the sum of the weights of all the nodes. Denote by $d(v)$ the depth of node v.

Theorem 1. *Let T be a weighted trivalent tree where the weight of any interior node is* 0 *and the weight of any leaf node v is $\frac{1}{2^{d(v)}}$. Then $S(T) = 1$.*

Proof. We prove the theorem by induction on the height, $H(T)$, of tree T with root r.

$H(T) = 1$**:** Since r has no children it is a leaf with weight $\frac{1}{2^0} = 1$. Thus $S(T) = w(r) = 1$.

$H(T) = k + 1$**:** Denote by T_1 the subtree rooted at the left child of r, by T_2 the subtree rooted at the right child of r and notice that both T_1 and T_2 must exist since T is trivalent. Adding 1 to the depth of all the leaves in T_1 (T_2) halves $S(T_1)$ ($S(T_2)$). It follows that $S(T) = \frac{S(T_1)}{2} + \frac{S(T_2)}{2}$. Since

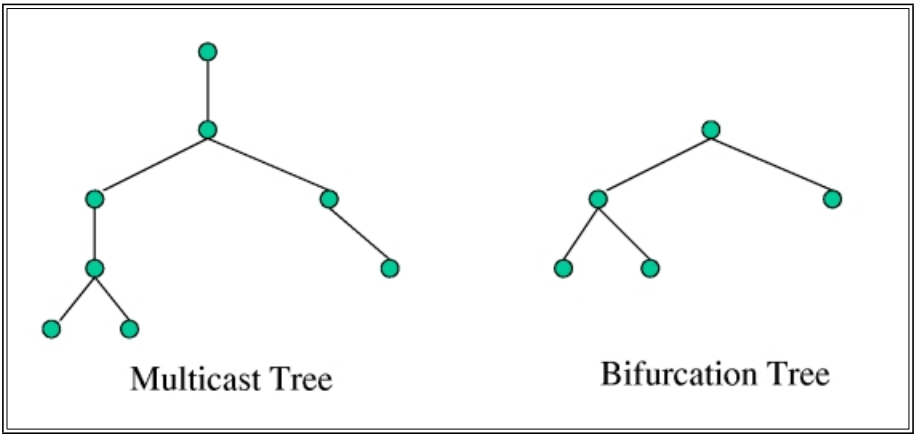

Fig. 2. A binary tree and its trivalent bifurcation tree.

T is trivalent both T_1 and T_2 are trivalent so by the induction hypothesis $S(T) = \frac{1}{2} + \frac{1}{2} = 1$ and the theorem holds.

□

Corollary 1 states that partial sums of $S(T)$ are always smaller than 1. This follows from the fact that all the weights of the leaves of T are positive.

Definition 3. *Tree T' is a* partially spanning subtree *of tree T if T' is a subtree of T, the root of T' is the root of T and the leaves of T' are also leaves in T.*

Corollary 1. *If T is trivalent and T' is a partially spanning tree of T then $S(T') \leq 1$ and $S(T') = 1$ only if $T' = T$.*

Using Corollary 1 it is possible to implement termination detection at the root of a multicast tree. Every message would count the number of times it was duplicated on its way to the leaf and this value would be returned to the root as the depth of the leaf. The root would add up the weights of the leaves and the algorithm would terminate when the total sum reaches 1. While multicast trees are not necessarily trivalent, the tree created by ignoring non-duplicating nodes is trivalent, and this tree has the same number of leaves as the original multicast tree. We call such trees *bifurcation trees* (See Fig. 2).

Definition 4. *The* bifurcation tree *of a multicast tree T is the tree induced by the bifurcation of messages as they traverse T.*

Corollary 2. *Bifurcation trees are trivalent trees.*

The code for the root node and the other nodes of a multicast tree is shown in Alg. 1. Every node in the tree keeps a list of children, C, where $||C|| \leq 2$ and

Algorithm 1 Stateless termination detection in binary trees.

```
    [root node r]
    C       : const     list of children;                    // ||C||=0..2.
    b_max   : integer ;                  // Maximal bifurcation encountered.
    n       : integer   init 0;                              // Numerator.

    Initialization:
1:      if ||C||=0 then exit;                          // Tree is a single node.
2:      b_max=log ||C||;
3:      for every x in C do send ⟨msg,r,b_max⟩ to x;

    Upon arrival of a ⟨ack,b⟩ message from node u:
4:      if b <= b_max then n := n + 2^(b_max − b);
        else
5:          n := 2^(b − b_max) * n+1;
6:          b_max := b;
7:      if n = 2^b_max then exit;                      // Algorithm terminated!

    [other nodes]
    C       : const     list of children;                    // ||C||=0..2.

    Upon arrival of a ⟨msg,r,b⟩ message:
8:      if ||C||=0 then send ⟨ack,b⟩ to root r.                       // Leaf.
9:      for every x in C do send ⟨msg,r,b + log ||C||⟩ to x;  // Bifurcate if necessary.
```

$||C|| = 0$ if the node is a leaf. Since handling fractions can be inefficient and inaccurate, the root node keeps track of the sum of weights, $S(T)$, by holding the numerator, n, and the denominator, $2^{b_{max}}$, separately (the denominator is named b_{max} because the least common multiple for the received denominators is always equal to $2^{(\text{maximal received depth})}$). When the algorithm starts (Lines 1-3) the root checks that it is not the only node in the tree, initializes b_{max} to 0 or 1 according to the log of the number of its children and finally sends the multicast message to all its children. In general, the root waits until $S(T)$ equals 1 (Line 7). As messages traverse the tree they are processed by the internal nodes (Lines 8-9). Whenever a message is duplicated by a node, the bifurcation counter, b, in both duplicates is increased by one (Line 9). When the message reaches a leaf, it is redirected to the root (Line 8) and if the message's b-count, b, is equal to the current b_{max} then the root simply increases n by 1 (Line 4). In general, however, there are two possible cases (Lines 4-7):

1. If an acknowledge message arrives from leaf u with b-count $b \leq b_{max}$, then the weight of leaf u, $w(u) = \frac{1}{2^b}$, must be adjusted to the current least common multiple of the denominators, $2^{b_{max}}$, before adding it to the total sum. Thus, the nominator of $w(u)$ (which is equal to 1) is multiplied by $2^{(b_{max}-b)}$ before being added to n (Line 4).

2. If an acknowledge message arrives from leaf u with b-count $b > b_{max}$ then the least common multiple of the denominators is 2^b and the total sum must be adjusted to it before $w(u)$ added. The nominator of the sum, n, is multiplied by $2^{(b-b_{max})}$ and only then the nominator of $w(u)$ (which is equal to 1) is added to it (Line 5). Finally, b_{max} is updated to represent the new least common multiple (Line 6).

To prove the correctness of Alg. 1, we show that during any point in the execution of the algorithm, $\frac{n}{2^{b_{max}}}$ is equal to $S(T')$, where T' is the partially spanning subtree of the bifurcation tree that was fully traversed by messages and that the acks of its leaves returned to the root (Theorem 2).

Let E be an execution of Alg. 1 and let $\alpha = m_1, m_2, \ldots$ be the sequence of messages that return to the root. Let β be some prefix of α. Denote by T_β the partially spanning subtree of the bifurcation tree created by taking all the leaves that sent messages in β along with all their ancestors up to the root.

By the model, messages are not lost and nodes do not fail. Since every internal node forwards the multicast message to each of its sons (Lines 3, 9) every leaf in the tree must eventually return an ack.

Corollary 3. *Tree T_α is equal to the entire bifurcation tree.*

Theorem 2. *For every execution E and for every prefix β in E, $\frac{n}{2^{b_{max}}} = S(T_\beta)$.*

Proof. Assume that the theorem is correct for some prefix $\beta_k = m_1, m_2, \ldots, m_k$. We show that the theorem holds for $\beta_{k+1} = \beta_k, m_{k+1}$.

Assume that message m_{k+1} came from leaf u with bifurcation counter, b (i.e., $w(u) = \frac{1}{2^b}$). By the induction hypothesis, $\frac{n}{2^{b_{max}}} = S(T_{\beta_k})$. Since $T_{\beta_{k+1}} \supset T_{\beta_k}$ and since only leaf nodes have weights it follows that $S(T_{\beta_{k+1}}) = S(T_{\beta_k}) + w(u) = \frac{n}{2^{b_{max}}} + \frac{1}{2^b}$.

If $b \leq b_{max}$ then the least common multiple of the two denominators is $2^{b_{max}}$. Thus b_{max} does not change, n is set to $n + 2^{(b_{max}-b)}$ (Line 4) and the theorem holds. If $b > b_{max}$ then the least common multiple of the two denominators is 2^b. Thus n is set to $n * 2^{(b-b_{max})} + 1$ (Line 5), b_{max} is set to b (Line 6) and the theorem holds. □

3 General Trees and DAGs

It is possible to use Alg. 1 as is to detect termination in multicast trees with nodes of any degree. However, if the number of children of some of the nodes is not a power of 2 then roundoff errors may accumulate when the bifurcation count is calculated.

We suggest constructing the bifurcation tree of a general multicast tree as a binary tree. Each node u in the multicast tree is represented in the bifurcation tree as a binary trivalent subtree, where the leaves of the subtree are the children of u (see Fig. 3). It can be shown that the number of leaves in the total bifurcation tree is equal to the number of leaves in the multicast tree.

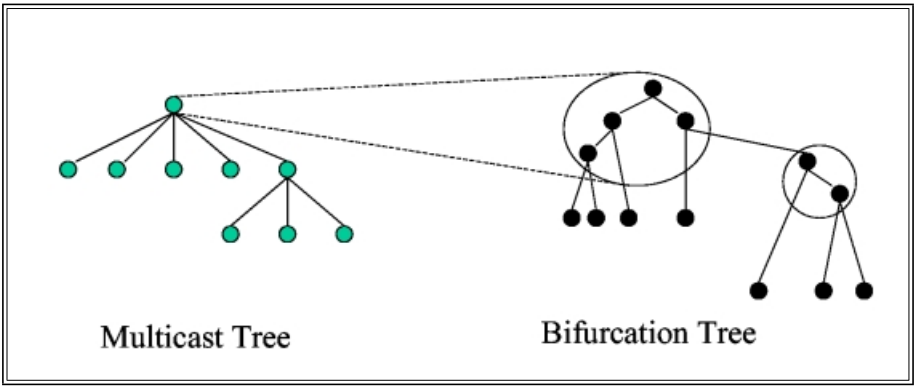

Fig. 3. A general tree and its trivalent bifurcation tree.

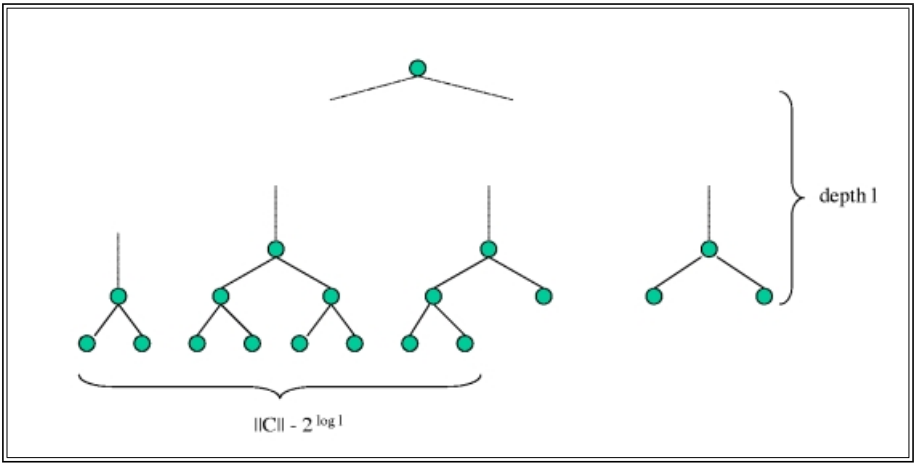

Fig. 4. A trivalent tree with $||C||$ leaves ($l = \lfloor \log ||C|| \rfloor$).

There are many ways to exchange an internal node, u, having $||C|| > 0$ children with a trivalent tree of $||C||$ leaves. One such way is to construct an almost-balanced tree, where the depth of the leaves is either $\lfloor \log ||C|| \rfloor$ or $\lfloor \log ||C|| \rfloor + 1$. The number of leaves at each depth can be calculated, and is presented in Fig. 4. Since the abstraction of the bifurcation tree is created in the internal nodes, changes in the code are not necessary in the root (Alg. 2).

Although the algorithm was presented for trees, it works as is for DAGs. The bifurcation tree would include all the possible paths in the DAG, from the root to the leaves.

Algorithm 2 Updating the bifurcation count in general trees.

```
[other nodes]
C          : const     init array of children of size ||C||;
l          : integer   init 0;

Upon arrival of a ⟨msg,r,b⟩ message:
1:  if ||C|| = 0 then send ⟨msg,b⟩ to root r;                    // Leaf.
2:  l := ⌊log ||C||⌋;
3:  for i:= 1 to ||C|| do
4:      if (i ≤ 2^(l+1) − ||C||) then send ⟨msg,r,b + l⟩ to C[i];
5:      else send ⟨msg,r,b + l + 1⟩ to C[i];
    od;
```

4 Future Work

Graphs that contain loops present a difficult problem for stateless algorithms. Since there is no state in the nodes, there is no way for a message to detect it returned to the same node twice. It is possible to solve this problem if we assume that there is a spanning tree embedded in the graph. Since the same spanning tree can be concurrently used by many sessions of the stateless algorithms, it satisfies our model.

Assuming that termination detection is available on general graphs, it is possible search for practical stateless algorithms for various problems, such as finding the shortest path between any two nodes or finding the minimal spanning tree. But since the result of these algorithms may be the entire graph, the best solution might be to recreate the graph at the root. A more challenging problem may be to find the length of the shortest path between any two nodes for example.

Acknowledgments. I want to thank Yehuda Afek and Dan Touitou for helpful discussions and support. In particular, I want to thank Manor Mendel for bringing to my attention Theorem 1 and how it simplifies the explanations and proofs.

References

1. David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, 1997.
2. E. Dijkstra and B. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, 1980.
3. Nir Shavit and Nissim Francez. A new approach to detection of locally indicative stability. In *Automata, Languages and Programming*, pages 344–358, 1986.

4. Gerard Tel. *Introduction to Distributed Algorithms.* Cambridge Uni Press, Cambridge, 1994.
5. E. Dijkstra, W. Feijen, and A Gasteren. Derivation of a termination detection algorithm for distributed computations. In *Inf. Proc. Lett. 16, 5*, pages 217–219, 1983.
6. E. Dijkstra. Shmuel safra's version of termination detection. Technical Report EWD998, The University of Texas at Austin, 1987.
7. Donald F. Towsley, James F. Kurose, and Sridhar Pingali. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. *IEEE Journal of Selected Areas in Communications*, 15(3):398–406, 1997.
8. Brian Neil Levine and J. J. Garcia-Luna-Aceves. A comparison of reliable multicast protocols. *Multimedia Systems*, 6(5):334–348, 1998.
9. Eric W. Weisstein. Binary tree. http://mathworld.wolfram.com/.