CACHES IN A THEORETICAL MODEL OF MULTICASTING

by

Gregory G. Baker

B.Sc., Queen's University, 1998

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
August 2000

APPROVAL

Name: Gregory G. Baker

Degree: Master of Science

Title of thesis: Caches in a Theoretical Model of Multicasting

Examining Committee: Dr. Ramesh Krishnamurti
Chair

Dr. Arthur Liestman, Senior Supervisor

Dr. Joseph Peters, Supervisor

Dr. Thomas Shermer, Supervisor

Dr. Lou Hafer, External Examiner

Date Approved: _____

Abstract

Multicasting is the process by which a single node, using a sequence of calls, sends a message to a set of nodes in a communication network. The message is passed from the source, through intermediate nodes to the destinations. These intermediate nodes do not remember the message once it is passed on. There is a possibility of transmission failure with each call. A failure forces retransmission from the source.

Some intermediate nodes can be designated as *caches*. Once they have received the message, caches will remember it. Thus, these nodes can also be used to retransmit the message if there is a failure. If the retransmission can be done from a cache node that is closer than the source, the total number of calls necessary will be decreased.

The expected number of calls necessary to complete a multicast is examined. Two methods of counting the number of calls are presented. Upper and lower bounds on the expected number of calls are given.

Placement of a given number of caches is examined in order to determine the locations which minimize the expected traffic. Optimal placements are given for small graphs and a heuristic is given which can be used to place caches in larger graphs.

It is shown that determining if the caches can be placed so that the expected traffic does not exceed a given threshold is NP-complete in directed acyclic graphs.

Acknowledgments

Thanks first to Art. Without him, this thesis would be no more than a random collection of thoughts. Through this whole process, Art seemed to magically give the exact advice I needed at just the right time. I have no idea how, or if, I would have finished this without him.

Also, thanks to my parents for their ever-so gentle prodding. Without their guidance and encouragement, I could never have made it this far.

Finally, thanks to Kat for knowing what I was going through and for pretending to be interested in theoretical networking from time to time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The Internet opens up many problems that could be of interest in theoretical networking. Theoretical networking often deals with store-and-forward models in known, regular network topologies. These are usually said to model a network of processors in a multi-processor system. The Internet, and other networks of computers, however, tend to be much less regular since they have been created over time and usually by several different administrators. Thus, if we are to study these networks in a theoretical way, we must consider a much broader class of topologies.

Here, we will consider the process of *multicasting*, which is sometimes called *1-to-many communication* or *1-to-k communication*. Multicasting is a process where a single source node has a message to be sent simultaneously to several destinations in the network. Some examples are disseminating stock quotes, video broadcasts, audio- and video-conferencing [16]. In such situations, it is obviously more efficient for the source to send to all of the destinations at once than to send to each individually in sequence.

The original specification of the Internet Protocol did not allow for multicasting. An optional multicast extension to IP was defined in 1989 [11]. Internet nodes which are capable of multicasting are part of the *MBONE* or *multicast backbone* [5].

There are two types of error-correcting multicasting: reliable and resilient multicasting. Reliable multicasting, which we will be looking at here, ensures that all information will be passed to all destinations correctly [25]. Resilient multicasting makes a best-effort attempt to inform all destinations within a given time [25]. Resilient multicasting is intended for interactive applications, such as streaming video. In these applications, timing is important and if small amounts of data are occasionally lost, the multicast can continue. Reliable multicasting is used where data corruption is unacceptable, such as software updates or stock quotes.

In this thesis, the process of multicasting consists of the message being sent from a source, through several intermediate nodes, to each of the destinations. These intermediate nodes are called *internal nodes*. A *call* is an attempt to transmit a message from a node to an adjacent node or *neighbour*. The internal nodes have no memory, so if a call fails to correctly transmit the message, the message must be retransmitted from the source.

To lessen the costs incurred by retransmissions, some nodes may be designated as *caches*. These nodes store any messages which are passed through them and can retransmit the

message if required. Since there may be caches closer than the root, they can have the effect of lowering the number of calls necessary to inform all of the destinations, as the message needn't be retransmitted from a distant source.

Since caches are expected to store a great deal of information, they are expensive to implement. Thus, only a few caches will be used in a network. We examine the problem of determining the best locations for a few caches in given networks.

We will look at the total load placed on the communications network. That is, we will attempt to minimize the total amount of network traffic generated by the multicasting process. Caches are added to a network to decrease the total amount of traffic, so we will attempt to maximize their effectiveness.

1.2 Definitions

1.2.1 The Network

Consider a network, represented by an undirected graph, $G = (V, E)$. It is our intention to create a network model that reasonably approximates the Internet.

A *call* is the process of a node sending the message to an adjacent node. We use the term *network traffic* (or simply *traffic*) as the total number of calls used for a given multicast.

Assumption 1.1 *We assume that the path taken between any two nodes in the network does not change.*

This is not strictly true of the Internet where paths can change over time [22]. However, the stability of paths has been studied experimentally and it was found that "in general, Internet Paths are strongly dominated by a single route" [22]. Thus, this assumption is reasonable.

Assumption 1.2 *Consider three nodes a, b and c, with b on the path from a to c. We assume that the combined paths a to b and from b to c are identical to the path from a to c.*

This assumption indicates that nodes make only local choices about routing.

Assumption 1.3 *For each call between two vertices in the network, we assume that there is a certain probability, p, that the message will not be correctly received. We assume that this*

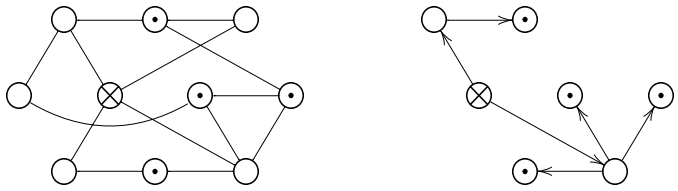


Figure 1.1: A network and a possible directed multicast tree.

probability is constant throughout the network and over time. We also assume that failures occur independently of one another.

This assumption will likely not hold in a real network, where some links will have different probabilities of failure, and failures will be clustered by temporary network problems. We make this simplifying assumption to get a reasonable model of multicasting which can be effectively analyzed.

1.2.2 Multicasting

Suppose that a node $s \in V$ has a message which must be delivered to several other nodes, $D = \{d_1, d_2, \ldots, d_\lambda\} \subseteq V$ in the network. This process is called *multicasting* from s to D. The node s is called the *source node* and D is the set of *destination nodes*.

When considering a particular multicasting instance, we must only consider the edges and vertices of the graph which are used in a multicast from s to D. These vertices and edges form the *multicast subgraph* of G for the multicast from s to D. We can also define a *directed multicast subgraph* which is the multicast subgraph with edges directed in the direction that the message will be transmitted.

1.2.3 Multicast Trees

Assumptions 1.1 and 1.2 imply that the multicast subgraph is always a tree. Thus, we will refer to multicast subgraphs as *multicast trees* or *directed multicast trees*. See Figure 1.1 for an example of a network with its multicast tree. In general, when we draw multicast trees, the source will be indicated with $\otimes$. Destinations will be the leaves of the tree, or will be indicated with $\odot$, if necessary.

The problem of choosing an "optimal" multicast tree has been studied [6, 17, 18]. An optimal multicast tree is defined as follows.

Definition 1.4 (Optimal Multicast Tree) *Given a graph G with a source s and destination set D, an* optimal multicast tree *or* OMT *is a subtree T of G with the following properties: (a) s and all of the nodes in D are vertices in T; (b) the distance from s to $d \in D$ is the same in T as in G; (c) T has the minimum number of nodes of all trees satisfying (a) and (b) [6].*

Given a graph G, along with a source s and destination set D in G, the *optimal multicast tree problem* is the problem of finding the optimal multicast tree for s, D and G.

The following theorem is proved in [6].

Theorem 1.5 *Given a graph G with a source s and destination set D, determining if there is an optimal multicast tree of s, D and G with at most n nodes NP-complete.*

In fact, it has been shown that it is NP-complete to find an optimal multicast tree for several special classes of graphs, including hypercubes [6], graphs with degree at most 3 [6], bipartite graphs [7] and 2D mesh graphs [18].

All of these results assume a multicast model without the possibility of transmission failure. Thus, finding OMTs in a model with transmission failures is also NP-complete since the faultless OMT problem is a restriction of this [13].

Note that in general, destination nodes do not have to be at the leaves of the multicast tree, as they are in Figure 1.1. How these interior destinations behave will depend on the details of the multicast protocol. They may behave as caches (as described in Section 1.2.5) and be able to retransmit the message if necessary, or they may simply receive the message when it goes by, but not be able to retransmit. We will assume the latter and only consider destinations at the leaves.

1.2.4 Multicast Algorithm

We propose a relatively simple reliable multicasting algorithm. We assume that the set D is static, that is, no nodes join or leave the set during the multicast.

Two different types of messages will be sent between nodes. First, *data messages* (or *calls*) are sent down the tree from the source, carrying the information that is to be multicast. Every call carrying a data message will incur one unit of traffic. Data messages fail independently with a given probability, p. Second, *control messages* are sent back up the

tree and are either *ACK* (acknowledge), indicating success, or *NACK* (negative acknowledge) indicating failure. We assume that the control messages never fail and that they create no traffic and take zero time.

These assumptions are reasonable, as control messages will typically be very much smaller than the data messages and thus use a negligible amount of the total traffic and time. Also, we can assume that each node has enough memory to keep track of the control messages it receives; thus, if any control messages are lost, they can be resent locally.

We assume that transmission errors are detectable. This detection can be made either at the sender or the receiver of the call. In the latter case, a NACK must be sent to the sender.

When a node transmits a data message to more than one of its children in the multicast tree, each of these calls will count separately towards the amount of traffic generated. While some networking hardware would make it possible to send to all attached nodes simultaneously, this is not always the case. As a simplifying assumption, we assume that a separate call is needed for each child.

The multicast will be performed using the following algorithm in the multicast tree. The leaves behave as follows:

1. When the message is received, send an ACK to the parent
2. If an incorrect transmission is received, send a NACK

The intermediate nodes behave like this:

1. If an incorrect transmission from the parent is received, return a NACK.
2. If a message is correctly received, forward the message to each child that has not previously returned an ACK. After the forwarding, discard the message.
3. Wait for either an ACK or NACK for every copy forwarded in the last step. If any NACKs were received, send a NACK to the parent. Otherwise, send a ACK.

Finally, the source behaves much like an intermediate node:

1. Send the message to each child.
2. Wait for either an ACK or NACK for every copy forwarded in the previous step. If any NACKs were received, go on to the next step. If not, the multicast is finished.

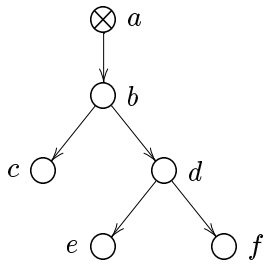


Figure 1.2: A sample directed multicast tree. All of the leaves are destinations.

From	To	Succeed/Fail	Informed	Note
a	b	S	$\{a, b\}$	
b	c	F	$\{a, b\}$	(c sends a NACK to b)
b	d	S	$\{a, d\}$	(b has sent to each child and forgets.)
d	e	S	$\{a, d, e\}$	(e sends an ACK to d)
d	f	F	$\{a, e\}$	(d has sent to each child and forgets. f, then d and then b each send a NACK.)
a	b	S	$\{a, b, e\}$	(a received a NACK so it begins to resend.)
b	c	S	$\{a, b, c, e\}$	(c sends an ACK to b)
b	d	S	$\{a, c, d, e\}$	(b has sent to each child and forgets.)
d	f	S	$\{a, c, e, f\}$	(e already ACKed, so d doesn't send there. f, d and b ACK)

Table 1.1: Sample multicast transmissions for the directed multicast tree in Figure 1.2

3. Send the message to every child that returned a NACK in the last step.

4. Wait for either an ACK or NACK for every copy forwarded in the previous step. If any NACKs were received, go back to the previous step. If not, the multicast is finished.

Consider the directed multicast tree in Figure 1.2. Table 1.1 gives a possible list of data messages used to complete the multicast in this case, that is, a possible execution of the multicast algorithm. In this case, 9 units of network traffic were necessary to multicast.

Note that we haven't specified the order in which a node sends to its children, or whether it can send to all children in the same time step. These both can affect the time used to multicast, but do not affect the traffic.

1.2.5 Caches

In order to decrease the number of retransmissions used in a multicast, one or more *caches* may be added to such a network. A cache is a node which can store messages that are transmitted through it. These caches are set up by network administrators.

A cache node requires a large amount of storage to be available and thus may be quite expensive. Thus, we should expect to only have a few caches at our disposal. We will assume that each cache in the network can hold the entire message until the multicast is complete. This may not be the case if many other transmissions are taking place simultaneously.

Caching has been considered by many others with various network assumptions.

Caches are placed on the Internet to decrease traffic necessary to transmit pages on the World-Wide Web [4]. Web pages are stored at these caches, so that if one is requested by another user, it can be sent from the cache instead of from the original source. Web caches are often set up by network administrators to decrease network traffic on a critical network link, usually an organization's connection to the Internet backbone. [8, 9, 26]

Similar caches are used in multi-processor systems. These caches are maintained to decrease the traffic necessary to service multiple requests for the same information. [3]

In our model, a cache node behaves like both a leaf and a source. Once a cache, a, has been informed, no further retransmissions are needed from s to a. The cache essentially becomes a source for the subtree below it. A cache node behaves as follows in our multicast algorithm:

1. If an incorrect transmission is received, return a NACK.
2. Once the message is correctly received, send an ACK to the parent.
3. Send the message to every child.
4. Wait for either an ACK or NACK from every copy forwarded in the last step. If any NACKs were received, go on to the next step. Otherwise, the cache is finished.
5. Send the message to every child that returned a NACK in the last step.
6. Wait for either an ACK or NACK from each copy forwarded in the last step. If any NACKs were received, go back to the previous step. Otherwise, the cache is finished.

The analysis of message traffic in a tree with a cache can be simplified. Multicasting in a multicast tree T, where one cache, x, is present can be split into two steps. Let T_x be the

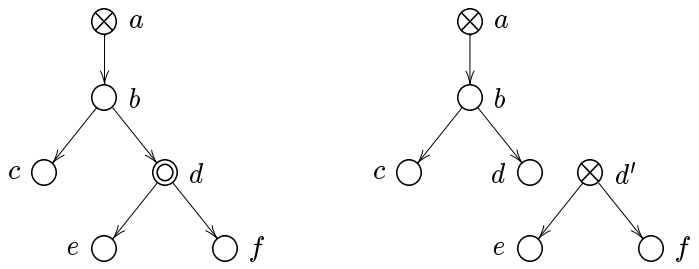


Figure 1.3: Caches effectively split a tree into subtrees. Both the left and right trees can multicast with the same expected traffic. Caches are denoted ◎.

subtree rooted at the cache, x. Let $D_1 \subseteq D$ be the set of leaves in $T \setminus T_x$ and let $D_2 \subseteq D$ be the set of leaves in T_x. In the first step, multicast in $T \setminus T_x$ from the source s to $D_1 \cup \{x\}$. In the second step, multicast in T_x from the cache x to D_2.

Thus, the total traffic is the sum of the traffic to multicast in the subtrees, as in Figure 1.3. When caches are drawn in a figure, they will be indicated by ◎.

For k caches for $k > 1$, a similar decomposition can be done. The original tree T can be decomposed into $k+1$ subtrees rooted at the source and the k caches. The leaves of the subtrees are either destinations or caches. Again, the total traffic is the sum of the traffic in all of the subtrees.

As a result of this, we see that placing a cache at either a leaf or the root will have no effect on the traffic necessary to multicast. In either case, when we consider the subtrees described above, one of the subtrees will be the entire tree and the other will be a single node. Thus, the total traffic will be the same and there is no further need to consider caches at either the source or leaves.

If there is no chance of failure, that is, if $p = 0$, then there will never be a need to retransmit the message. In this case, caches will have no effect on the traffic necessary to multicast.

1.3 Overview

First, in Chapter 2, we will consider the problem of determining the expected traffic to multicast in a given tree. Two methods are presented, one based on simple probability theory, the other on Markov theory. We see that both of these methods give rise to large

computations for all but the simplest cases.

Since we cannot, in general, calculate the expected traffic directly, bounds are developed for expected traffic in Chapter 3. Two operations are given for a tree which are shown to not increase or decrease, respectively, the expected traffic. These are used to create trees for which the expected traffic can be calculated, thus creating upper and lower bounds on the expected traffic.

Given a reasonable method of calculating expected traffic, we can then focus on the problem of cache placement in Chapter 4. For small trees, where the expected traffic can be calculated exactly, it is possible to simply calculate the expected traffic for each possible cache configuration and select the lowest. The optimal cache placements for one and two caches, for all trees with six or fewer vertices are given.

For larger trees, a heuristic algorithm is given which uses the bounds developed previously. The heuristic is used to calculate cache placements for a larger tree to illustrate its use.

Finally, in Chapter 5 the cache placement problem is generalized slightly. This generalization is shown to be NP-complete.

Chapter 2

Counting Traffic in Trees

2.1 Preliminaries

The variable p to denote the probability of failure in any given call.

Given a directed multicast tree T and a node y in T, let T_y be the subtree of T rooted at y. Let x be the parent of y, then let T_y^+ be T_y along with x and the edge from x to y.

For a node x in a directed multicast tree, let $\text{depth}(x)$ be the distance from the source to x. For a multicast tree T, let $\text{depth}(T)$ be the maximum leaf depth in T, that is, $\text{depth}(T) = \max_{x \in T} \text{depth}(x)$.

Let λ be the number of leaves in a tree. Let λ_i be the number of leaves in the tree at depth i from the root.

The following lemma is a very basic result that will allow us to approach the problem of counting network traffic.

Lemma 2.1 *Given a node x with c children, $y_1, y_2, \ldots, y_c$, in a directed multicast tree T with no caches, the traffic necessary to multicast in T_x is the sum of traffic necessary to multicast in each $T_{y_i}^+$.*

Proof: Since T is a tree, calls made in each $T_{y_i}^+$ are independent of one another. Thus, the transmissions made to each of these trees can be considered separately and we can simply add the total number of calls. □

Now, it is possible to begin the problem of counting the expected traffic necessary for a multicast.

2.2 Counting Traffic with Probability Theory

We first count the expected amount of traffic for a multicast using standard probability theory. As we see, this becomes complicated even for small graphs, but it can be used for structural inductions.

2.2.1 Probability Theory

We use standard notation of probability theory. In particular, for a random variable X, $P(X = i)$ will denote the probability that X is i and $E(X)$ will denote the expected value of X.

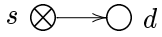


Figure 2.1: The simplest possible directed multicast tree

The expected value of a random variable is the sum of each possible cost times the probability of that cost occurring. That is, $E(X) = \sum i \cdot P(X = i)$.

2.2.2 Using Probability Theory

Consider a network consisting only of a source, s, and a single destination node, d, connected by an edge, as in Figure 2.1.

We first analyze this multicast tree; we wish to find the expected number of calls (expected total traffic) necessary to inform d. Note that the probability of needing i calls to inform v is the probability of failing on the first $i-1$ calls and succeeding on the i-th. Thus, the expected number of calls is

$$\begin{aligned}\sum_{i=1}^{\infty} i(p^{i-1}(1-p)) &= \sum_{i=1}^{\infty} ip^{i-1} - ip^i \\ &= \sum_{i=0}^{\infty}(i+1)p^i - \sum_{i=1}^{\infty} ip^i \\ &= \sum_{i=0}^{\infty} p^i = \frac{1}{1-p}\,.\end{aligned}$$

This value will be used often, so we will use q to denote $1/(1-p)$.

In order to count the traffic, we must first introduce a random variable which will make it possible to recursively express the expected traffic.

Transmission Rounds

Let $\mathsf{relay}(x)$ be a random variable representing the number of times that the node x must receive the message from its parent in order for all of the leaves below it to correctly receive the message. Equivalently, this is the number of times that x must go through the process of forwarding the message to each of its children with uninformed subtrees.

Let $\mathsf{relay}^+(x)$ be a random variable representing the number of times that the parent of x must receive the message in order for all of the leaves below x to correctly receive the

message. Equivalently, this is the number of times that the parent of x must forward the message to x.

Consider a node x with a single child y and suppose that $\mathsf{relay}(y) = i$. Then, the probability that $\mathsf{relay}(x) = n$ for $n \geq i$ is the probability that exactly i of n transmissions from x to y succeed. Thus, summing over i, we have that

$$P(\mathsf{relay}(x) = n) = \sum_{i=0}^{n} \binom{n}{i} p^{n-i}(1-p)^i P(\mathsf{relay}(y) = i)\,.$$

Now, consider the value for $E(\mathsf{relay}(x))$. We expect that y needs to receive the message $E(\mathsf{relay}(y))$ times, and, from above, the expected calls to send a message once from x to y is q. Thus,

$$E(\mathsf{relay}(x)) = q \cdot E(\mathsf{relay}(y))\,. \tag{2.1}$$

If x is a node with children $y_1, y_2, \ldots, y_c$ then we can consider each child separately as in Equation 2.1 and combine them for the total.

$$\mathsf{relay}(x) = \max_{1 \leq i \leq c} (\mathsf{relay}^+(y_i)) \tag{2.2}$$

We have no closed form for this expression. For random variables $X_1, \ldots, X_n$, $E(\max(X_i)) \neq \max(E(X_i))$. The only way to evaluate this expression seems to be to use the definition of expected value and get (for $c = 2$),

$$\begin{aligned} &E(\max(\mathsf{relay}^+(y_1), \mathsf{relay}^+(y_2))) \\ &= \sum_{i_1=1}^{\infty} \sum_{i_2=1}^{\infty} \max(i_1, i_2) P(\mathsf{relay}^+(y_1) = i_1) P(\mathsf{relay}^+(y_2) = i_2)\,. \end{aligned}$$

Note that c sums are necessary for c children. We have not been able to directly evaluate this expression.

Expected Traffic

Let $\mathsf{traffic}(x)$ be a random variable representing the number of calls made in T_x so that all of the leaves below x correctly receive the message. We will also use $\mathsf{traffic}(T)$ to indicate the total traffic necessary to multicast from the root to the leaves of T.

If x is a node with a single child y, then we must send from x to y so that y receives the message $\mathsf{relay}(y)$ times. We can expect this to take $q \cdot E(\mathsf{relay}(y))$ calls. We will also incur

the traffic necessary to multicast to T_y. Thus,

$$E(\mathsf{traffic}(x)) = q \cdot E(\mathsf{relay}(y)) + E(\mathsf{traffic}(y))\,. \tag{2.3}$$

If x has several children $y_1, y_2, \ldots, y_c$, then, Lemma 2.1 immediately gives

$$E(\mathsf{traffic}(x)) = \sum_{i=1}^{c} q \cdot E(\mathsf{relay}(y_i)) + E(\mathsf{traffic}(y_i))\,. \tag{2.4}$$

Unfortunately, this depends on $\mathsf{relay}(y)$ for which we have no closed form. Equations 2.3 and 2.4 can, however, be used to reason formally about the expected traffic and get bounds on it.

2.3 Counting Traffic with Markov Chains

The number of calls necessary to complete a multicast can also be counted using *Markov theory*. This provides an algorithmic method to determine the number of expected calls for a particular multicast tree.

2.3.1 Markov Theory

A *Markov process* is any process where the next state is determined probabilistically from the current state. A *Markov chain* is a Markov process with a finite or countable number of states. For an introduction to Markov Theory, see Norris [21].

Suppose we have a Markov chain with states $\{s_1, s_2, \ldots, s_\sigma\}$. We can define an matrix $P = (p_{i,j})_{\sigma\times\sigma}$ called the *transition matrix*. If the process is in state i, $p_{i,j}$ is the probability that it will be in state j at the next step. Note that we must have that $\sum_{j=1}^{\sigma} p_{i,j} = 1$ and $0 \le p_{i,j} \le 1$. We also must give an initial distribution L, which is a vector giving the probability of the process starting in each of the σ states.

More formally, let P be a transition matrix, $P = (p_{i,j})_{\sigma\times\sigma}$ and let L be an initial distribution, $L = (L_i)_\sigma$. If we let X_i be a random variable representing the state after i transitions, a Markov process has (a) $P(X_0 = i) = L_i$ and (b) $P(X_{n+1} \mid X_0 = i_0, \ldots, X_n = i_n) = p_{i_n, i_{n+1}}$. [21]

Usually, Markov theory uses the concept of "time" to indicate the state transitions. That is, we refer to a process being in some state at *time* κ. There is, however, no reason that we must limit ourselves to interpreting this as actual time. For instance, we wish to

count the number of calls made, instead of the time taken to multicast. Thus, we will say that the multicast is in some state *after κ calls* and each state transition will indicate one call made, instead of one time unit passed.

A state s is called *absorbing* if, once the process enters state s, it cannot leave. It is called *transient* if the probability of returning to that state after visiting it is less than 1. [15]

We are primarily interested in the expected time for the Markov process to go from an initial distribution to a designated state where the process is "completed". We can use the following theorem from [15], Chapter 3, Section 4 to find the expected number of steps.

Theorem 2.2 (Fundamental Markov Method) *Given a Markov chain with σ states, $s_0, s_1, \ldots, s_{\sigma-1}$ where state s_0 is absorbing and all other states are transient. Let P^* be this Markov chain's transition matrix. Let P be the sub-matrix of P^* given by*

$$P^* = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ r_1 & & & \\ \vdots & & P & \\ r_{m-1} & & & \end{bmatrix}$$

Let $\mathbf{1}$ be the column vector of $\sigma - 1$ ones. Then, if $Q = (I - P)^{-1}$, the expected time to get from state s_i to s_0 is the i-th entry of $Q \cdot \mathbf{1}$.

2.3.2 Using the Markov Method

As previously stated, it is possible to view the multicasting problem as a Markov chain. A state includes a list of which vertices are informed. Also, since an internal node is expected to send to each of its children and then forget, the state also includes the child that each node is about to send to, to ensure that each child is called once and that the internal node forgets after sending to each child.

We will adopt the notation $a_{(b)}$ to indicate that node a is informed and it will next send to its child b and $c_{(*)}$ to indicate that c is informed but is not waiting to send. Let M_κ be the set of informed vertices after κ calls.

Note that, in order to apply Theorem 2.2 to get the expected traffic, the Markov process must be set up so that each "step" corresponds to one call. That is, since we wish to count

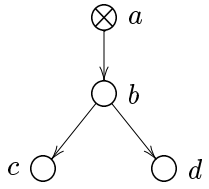


Figure 2.2: A simple multicast tree to demonstrate the Markov method.

State	Action	Failure	Success
$\{a_{(b)}\}$	$a_{(b)}$	$\{a_{(b)}\}$	$\{a_{(*)}, b_{(c)}\}$
$\{a_{(*)}, b_{(c)}\}$	$b_{(c)}$	$\{a_{(*)}, b_{(d)}\}$	$\{a_{(*)}, b_{(d)}, c_{(*)}\}$
$\{a_{(*)}, b_{(d)}\}$	$b_{(d)}$	$\{a_{(b)}\}$	$\{a_{(b)}, d_{(*)}\}$
$\{a_{(b)}, c_{(*)}\}$	$a_{(b)}$	$\{a_{(b)}, c_{(*)}\}$	$\{a_{(*)}, b_{(d)}, c_{(*)}\}$
$\{a_{(b)}, d_{(*)}\}$	$a_{(b)}$	$\{a_{(b)}, d_{(*)}\}$	$\{a_{(*)}, b_{(c)}, d_{(*)}\}$
$\{a_{(*)}, b_{(d)}, c_{(*)}\}$	$b_{(d)}$	$\{a_{(b)}, c_{(*)}\}$	$\{a_{(*)}, c_{(*)}, d_{(*)}\}$
$\{a_{(*)}, b_{(c)}, d_{(*)}\}$	$b_{(c)}$	$\{a_{(b)}, d_{(*)}\}$	$\{a_{(*)}, c_{(*)}, d_{(*)}\}$
$\{a_{(*)}, c_{(*)}, d_{(*)}\}$	none	$\{a_{(*)}, c_{(*)}, d_{(*)}\}$	$\{a_{(*)}, c_{(*)}, d_{(*)}\}$

Table 2.1: List of state transitions for the tree in Figure 2.2

traffic, we are not concerned with what happens in the next time unit, but with the next call. Thus, given a state, we need to decide which node should make the next call.

The next call will be made by the lowest informed vertex which has uninformed leaves below it. As long as there are uninformed leaves below, some vertex will have to send information to them. Since there are no lower informed vertices, this vertex must be the one to send the information. This ensures that we are not making an unnecessary call. As long as we continue in this way, counting calls that we know are necessary, we will get the correct number of calls for the multicast.

For example, in Figure 2.2, suppose that we have $M_k = \{a_{(*)}, b_{(c)}\}$. Here, b will send to c since $b_{(c)}$ is the lowest node that is waiting to send. Then, $M_{\kappa+1}$ could be either $\{a_{(*)}, b_{(d)}\}$ (if the transmission fails) or $\{a_{(*)}, b_{(d)}, c_{(*)}\}$ (if the transmission succeeds).

In Figure 2.2, the initial state is $\{a_{(b)}\}$. Table 2.1 gives a list of state transitions for every possible reachable state in this graph; the vertex that will send to make the state transition is listed in the "Action" column. This table corresponds to the following transition matrix. The order of states in the table is the same as the order in the rows and columns in the

matrix.

$$P^* = \begin{bmatrix} p & 1-p & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & p & 0 & 0 & 1-p & 0 & 0 \\ p & 0 & 0 & 0 & 1-p & 0 & 0 & 0 \\ 0 & 0 & 0 & p & 0 & 1-p & 0 & 0 \\ 0 & 0 & 0 & 0 & p & 0 & 1-p & 0 \\ 0 & 0 & 0 & p & 0 & 0 & 0 & 1-p \\ 0 & 0 & 0 & 0 & p & 0 & 0 & 1-p \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

If we rearrange the rows and columns of P^* so that the final state is first, we can apply Theorem 2.2. First, we get

$$P = \begin{bmatrix} p & 1-p & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & p & 0 & 0 & 1-p & 0 \\ p & 0 & 0 & 0 & 1-p & 0 & 0 \\ 0 & 0 & 0 & p & 0 & 1-p & 0 \\ 0 & 0 & 0 & 0 & p & 0 & 1-p \\ 0 & 0 & 0 & p & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p & 0 & 0 \end{bmatrix}.$$

We are now interested in the matrix $Q = (I - P)^{-1}$. In particular, though, we need only be concerned with the first row of Q, since we are only interested in the expected number of steps from the initial state. The first row of Q is

$$\left[\frac{1}{p^3-p^2-p+1} \quad \frac{-1}{p^2-1} \quad \frac{-p}{p^2-1} \quad \frac{p}{p^3-p^2-p+1} \quad \frac{p}{p^3-p^2-p+1} \quad \frac{-1}{p^2-1} \quad \frac{-p}{p^2-1} \right]$$

Finally, the sum of this row is

$$-\frac{2p^2-2p-3}{(p-1)^2(p+1)} = \frac{3q^3+2q^2-2q}{2q-1},$$

which, by Theorem 2.2, is the expected number of calls necessary to multicast to this tree.

Note that for $p = 0$, the expected traffic is the number of edges and as $p \to 1$, the expected traffic goes to infinity. Both of these are true for any multicast tree.

The above method can be used for any multicast tree to determine the expected number of calls. If there are σ states, finding the expected number of calls by this method can be done in time $O(\sigma^3)$, since a matrix of size $(\sigma - 1) \times (\sigma - 1)$ must be inverted [10].

Consider a multicast tree consisting of only a source and λ leaves adjacent to the source. In this case, there are 2^λ states since it is possible that any subset of the leaves may be informed. Thus, we see that finding the expected number of calls by the Markov method yields a $O(2^{3n})$ algorithm, where n is the number of nodes in the multicast tree.

2.4 Basic Traffic Results

We can use the methods from Section 2.2 to get a basic traffic result for paths. In this case, since no node has more than one child, Equation 2.1 is sufficient to calculate the expected traffic on the path.

We will use path(i) to denote a path with i edges and $i+1$ nodes.

Theorem 2.3 *Consider a path* path(d)*, with nodes* $a_0, a_1, \ldots, a_d$ *where* a_0 *is the source and* a_d *is a leaf. The expected amount of traffic necessary to multicast in this network is* $q + q^2 + \cdots + q^d$*. The expected number of transmissions by* a_0 *is* $E(\mathsf{relay}(a_0)) = q^d$*.*

Proof: We count the traffic using the probability theory method. The proof will be by induction on d.

Base Case: It was shown in Section 2.2.2 that for $d = 1$, $E(\mathsf{traffic}(a_0)) = q$. It follows that $E(\mathsf{relay}(a_0))$ is also q.

Inductive Case: Consider a path(d), with nodes $a_0, a_1, \ldots, a_d$. We can assume the result for path($d-1$). Thus, $E(\mathsf{traffic}(a_1)) = q + q^2 + \cdots + q^{d-1}$ and $E(\mathsf{relay}(a_1)) = q^{d-1}$.

Now, Equation 2.1 gives

$$E(\mathsf{relay}(a_0)) = q \cdot E(\mathsf{relay}(a_1)) = q^d\,,$$

and Equation 2.3 gives

$$E(\mathsf{traffic}(a_0)) = q \cdot E(\mathsf{relay}(a_1)) + E(\mathsf{traffic}(a_1)) = q^d + q^{d-1} + \cdots + q^2 + q\,.$$

So, we have both results for $d \geq 1$. □

Unfortunately, the analogous approach for more general trees requires the use of Equation 2.2 for which we have no closed form. By using the probability theory method, we will give some bounds on traffic in more general trees in Chapter 3. Other calculations of expected traffic, particularly those used in Chapter 4, will be made using the Markov method.

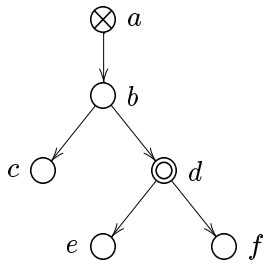


Figure 2.3: Sample directed multicast tree with a cache

2.5 Adding Caches

As noted in Section 1.2.5, adding a cache to a multicast tree has the effect of splitting the tree in two, with the cache in both halves.

We can now use the Markov method from Section 2.3 to calculate the expected traffic for a tree with a cache. Consider Figure 1.3, repeated here as Figure 2.3. Figure 2.4 shows the expected traffic for the tree in Figure 2.3, both with the cache shown and without. In this example, the expected traffic required multicasting with no caches is

$$\frac{3\,q^9 + 10\,q^8 + q^7 + q^6 - 24\,q^5 + 5\,q^4 + 21\,q^3 - 15\,q^2 + 3\,q}{(q^3 + 2\,q^2 - 3\,q + 1)\,(q^2 + q - 1)\,(2\,q - 1)}\,.$$

If we add a cache at d, as shown, the expected cost becomes

$$\frac{3\,q^3 + 6\,q^2 - 4\,q}{2\,q - 1}\,.$$

The cost decreases by a factor of q when a single cache is added.

By placing a single cache at other nodes in such a tree or by using multiple caches, the expected traffic may vary. This will be discussed in more detail in Chapter 4.

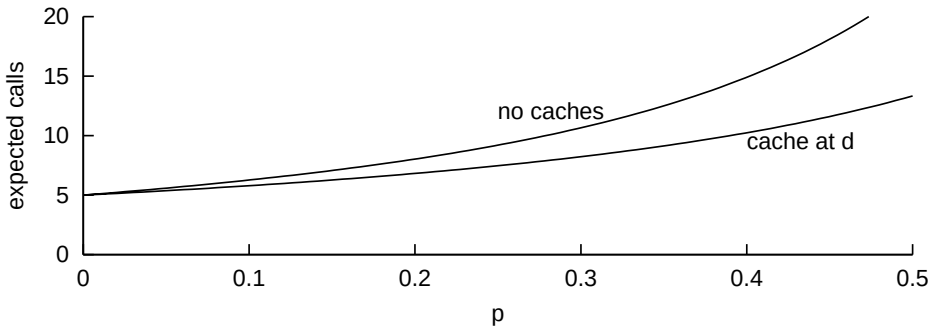


Figure 2.4: Expected calls for the multicast tree in Figure 2.3 with and without its cache, for various probabilities of failure.

Chapter 3

Bounds on Expected Traffic

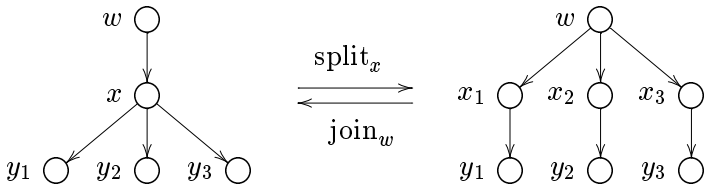


Figure 3.1: A sample of the split and join operation.

In this chapter, upper and lower bounds on the expected traffic necessary to multicast in a given tree are found. This allows us to develop a heuristic method to place caches in the network.

3.1 Terminology

Let T be a directed tree with node w. Let x be a child of w and $y_1, y_2, \ldots, y_c$ be the children of x. Let $\text{split}_x(T)$ be the tree formed by replacing the node x with c nodes $x_1, x_2, \ldots, x_c$ so that each y_i is the only child of x_i and each x_i is a child of w. To do this, replace the edge from w to x in T with c edges from w to $x_1, x_2, \ldots, x_c$, respectively and each edge from x to some y_i by an edge from x_i to y_i.

Similarly, let w be a node in T with children $x_1, x_2, \ldots, x_c$. Let $\text{join}_w(T)$ be the tree formed replacing the nodes $x_1, x_2, \ldots, x_c$ by x so that each child of a x_i in T is a child of x in $\text{join}_x(T)$. That is, if x_i has children $y_{i,1}, y_{i,2}, \ldots, y_{i,c_i}$ in T, then x in $\text{join}_x(T)$ will have each $y_{i,j}$ as a child.

See Figure 3.1 for an example of split and join.

3.2 Upper and Lower Bounds

The two operations split and join will be used to calculate bounds on the expected traffic for a given multicast tree. These bounds are based on the following four lemmas.

First, we note that a split just below the root does not decrease network traffic.

Lemma 3.1 *Consider a directed multicast tree T. Let w be the source, x be a child of w and $y_1, y_2, \ldots, y_k$ be the children of x. Then, $E(\textsf{traffic}(T)) \leq E(\textsf{traffic}(\text{split}_x(T)))$.*

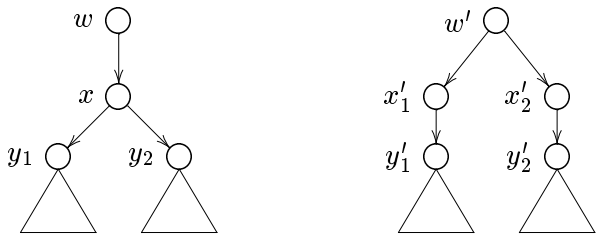


Figure 3.2: Trees T and $\text{split}_x(T)$ for Lemma 3.1.

Proof: Let $T_1, T_2, \ldots, T_k$ be the subtrees below $y_1, y_2, \ldots, y_k$, respectively. We label the nodes of $\text{split}_x(T)$ with a prime to distinguish them from nodes of T. See Figure 3.2 for an illustration of this situation for $k = 2$. We use the counting methods from Section 2.2 to analyze both T and $\text{split}_x(T)$.

First, we count the expected traffic for T. Starting with x, we get

$$E(\mathsf{relay}(x)) = E\left(\max(q \cdot \mathsf{relay}(y_i)\right) = q \cdot E\left(\max(\mathsf{relay}(y_i)\right) ,$$

and,

$$E(\mathsf{traffic}(x)) = q \sum E(\mathsf{relay}(y_i)) + \sum E(\mathsf{traffic}(y_i)) .$$

Then, we have

$$\begin{aligned} E(\mathsf{traffic}(w)) &= qE(\mathsf{relay}(x)) + E(\mathsf{traffic}(x)) \\ &= q^2 \cdot E\left(\max(\mathsf{relay}(y_i)\right) + q \sum E(\mathsf{relay}(y_i)) + \sum E(\mathsf{traffic}(y_i)) . \end{aligned}$$

Now, counting traffic for $\text{split}_x(T)$,

$$E(\mathsf{relay}(x'_i)) = qE(\mathsf{relay}(y'_i)) ,$$

and

$$E(\mathsf{traffic}(x'_i)) = qE(\mathsf{relay}(y'_i)) + E(\mathsf{traffic}(y'_i)) .$$

Then, we have

$$\begin{aligned} E(\mathsf{traffic}(w')) &= q \sum E(\mathsf{relay}(x'_i)) + \sum E(\mathsf{traffic}(x'_i)) \\ &= q^2 \sum E(\mathsf{relay}(y'_i)) + q \sum E(\mathsf{relay}(y'_i)) + \sum E(\mathsf{traffic}(y'_i)) . \end{aligned}$$

Now, comparing $E(\mathsf{traffic}(w))$ and $E(\mathsf{traffic}(w'))$, and noting that $\mathsf{relay}(y_i) = \mathsf{relay}(y_i')$ and $\mathsf{traffic}(y_i) = \mathsf{traffic}(y_i')$ we see that

$$\begin{aligned}
&E(\mathsf{traffic}(w')) - E(\mathsf{traffic}(w)) \\
&= q^2 \sum E(\mathsf{relay}(y_i')) + q \sum E(\mathsf{relay}(y_i')) + \sum E(\mathsf{traffic}(y_i')) \\
&\quad - q^2 \cdot E\left(\max(\mathsf{relay}(y_i)\right) - q \sum E(\mathsf{relay}(y_i)) - \sum E(\mathsf{traffic}(y_i)) \\
&= q^2 \left(\sum E(\mathsf{relay}(y_i')) - E\left(\max(\mathsf{relay}(y_i)\right)\right) ,
\end{aligned}$$

and since the expected value of the max cannot be more than the sum of the expected values, we have that $E(\mathsf{traffic}(w')) \geq E(\mathsf{traffic}(w))$. $\square$

We must also consider the values of $\mathsf{relay}(w)$ and $\mathsf{relay}(w')$. Let s be the source in a multicast tree with a single edge, as in Figure 2.1. Now,

$$\begin{aligned}
E(\mathsf{relay}(w)) &= E(\mathsf{relay}^+(x)) \\
&= E(\mathsf{relay}(s) \cdot \mathsf{relay}(x)) \\
&= E(\mathsf{relay}(s) \cdot \max \mathsf{relay}^+(y_i)) \\
&= E(\mathsf{relay}(s) \cdot \max \mathsf{relay}^+(y_i')) \\
&= E(\mathsf{relay}(s) \cdot \max \mathsf{relay}(x_i')) ,
\end{aligned}$$

and

$$\begin{aligned}
E(\mathsf{relay}(w')) &= E(\max \mathsf{relay}^+(x_i')) \\
&= E(\max(\mathsf{relay}(s) \cdot \mathsf{relay}(x_i')))
\end{aligned}$$

Since the max of several copies of $\mathsf{relay}(s)$ will be larger than a single copy, $E(\mathsf{relay}(w)) \leq E(\mathsf{relay}(w'))$.

Lemma 3.1 can be used to show that a split anywhere in the tree does not decrease expected traffic, as follows.

Lemma 3.2 *Given a directed multicast tree, T and a node x which is neither the source nor a leaf, $E(\mathsf{traffic}(T)) \leq E(\mathsf{traffic}(\mathrm{split}_x(T)))$.*

Proof: Let $P(U)$ be true iff $E(\mathsf{traffic}(U)) \leq E(\mathsf{traffic}(\mathrm{split}_x(U)))$. Note that if U doesn't contain x, we have equality.

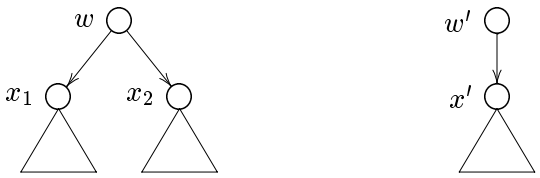


Figure 3.3: Trees T and $\text{join}_w(T)$ for Lemma 3.3.

Base Case: Let w be the parent of x. By Lemma 3.1, we have that $P(T_x^+)$. Since the other children of w are unaffected by the split operation, we have $P(T_w)$. Also, from the above note, we have $E(\mathsf{relay}(w)) \leq E(\mathsf{relay}(w'))$.

Inductive Case: Given b, an ancestor of x with $P(b)$, let a be the parent of b. Let $b_1, \ldots, b_c$ be the other children of a. We have $P(b)$ and $P(b_i)$ for each other child. Also, the values of $\mathsf{relay}(b)$ and $\mathsf{relay}(b_i)$ are the not greater in T than for $\text{split}_x(T)$.

Thus, $\mathsf{relay}(a)$ is not not less in $\text{split}_x(T)$ and

$$\mathsf{traffic}(a) = \mathsf{traffic}(T_b^+) + \sum \mathsf{traffic}(T_{b_i}^+)\,.$$

Now, we immediately have $P(a)$. $\square$

Now, we can get analogous results, showing that a join operation does not increase the expected traffic.

Lemma 3.3 *Consider a directed multicast tree T. Let w be the source, $x_1, x_2, \ldots, x_k$ be the children of w. Then, $E(\mathsf{traffic}(T)) \geq E(\mathsf{traffic}(\text{join}_w(T)))$.*

Proof: Let $T_1, T_2, \ldots, T_c$ be the subtrees below $x_1, x_2, \ldots, x_c$, respectively. As in Lemma 3.1, we label the nodes of $\text{join}_w(T)$ with a prime. See Figure 3.3 for an illustration of this situation for $k = 2$. Let y_{ij} be the j-th child of x_i. We use the counting methods from Section 2.2 to analyze both T and $\text{join}_w(T)$.

First, we count the expected traffic for the x_i,

$$E(\mathsf{relay}(x_i)) = E(\max_j(q \cdot \mathsf{relay}(y_{ij}))) = q \cdot E(\max_j(\mathsf{relay}(y_{ij})))\,,$$

and,

$$E(\mathsf{traffic}(x_i)) = \sum_j q \cdot \mathsf{relay}(y_{ij}) + E(\mathsf{traffic}(y_{ij}))\,.$$

Then, for w, we get

$$E(\mathsf{relay}(w)) = E(\max(q \cdot \mathsf{relay}(x_i))) = q^2 \cdot E(\max_i(\max_j(\mathsf{relay}(y_{ij})))),$$

and,

$$\begin{aligned} E(\mathsf{traffic}(w)) &= \sum_i q \cdot E(\mathsf{relay}(x_i)) + E(\mathsf{traffic}(x_i)) \\ &= \sum_i \left(q^2 \cdot E(\max_j(\mathsf{relay}(y_{ij}))) + \sum_j q \cdot \mathsf{relay}(y_{ij}) + E(\mathsf{traffic}(y_{ij})) \right) \\ &= q^2 \cdot \left(\sum_i E(\max_j(\mathsf{relay}(y_{ij}))) \right) + \sum_i \sum_j q \cdot \mathsf{relay}(y_{ij}) + E(\mathsf{traffic}(y_{ij})). \end{aligned}$$

Now, counting traffic for $\mathrm{join}_w(T)$,

$$E(\mathsf{relay}(x')) = E(\max_{i,j}(q \cdot \mathsf{relay}(y'_{ij}))) = q \cdot E(\max_{i,j}(\mathsf{relay}(y'_{ij}))),$$

and,

$$E(\mathsf{traffic}(x')) = \sum_{i,j} q \cdot \mathsf{relay}(y_{ij}) + E(\mathsf{traffic}(y_{ij})).$$

Finally,

$$E(\mathsf{relay}(w')) = q \cdot E(\mathsf{relay}(x'))$$

and,

$$\begin{aligned} E(\mathsf{traffic}(w')) &= q \cdot E(\mathsf{traffic}(x')) + E(\mathsf{relay}(x')) \\ &= q^2 \cdot E(\max_{i,j}(\mathsf{relay}(y'_{ij}))) + \sum_{i,j} q \cdot \mathsf{relay}(y_{ij}) + E(\mathsf{traffic}(y_{ij})) \end{aligned}$$

Now, comparing $E(\mathsf{traffic}(w))$ and $E(\mathsf{traffic}(w'))$, and noting that $\mathsf{relay}(y_i) = \mathsf{relay}(y'_i)$ and $\mathsf{traffic}(y_i) = \mathsf{traffic}(y'_i)$ we see that

$$\begin{aligned} &E(\mathsf{traffic}(w')) - E(\mathsf{traffic}(w)) \\ &= q^2 \cdot E(\max_{i,j}(\mathsf{relay}(y'_{ij}))) + \sum_{i,j} q \cdot \mathsf{relay}(y_{ij}) + E(\mathsf{traffic}(y_{ij})) \\ &\quad - q^2 \cdot \left(\sum_i E(\max_j(\mathsf{relay}(y_{ij}))) \right) - \sum_{i,j} q \cdot \mathsf{relay}(y_{ij}) + E(\mathsf{traffic}(y_{ij})) \\ &= q^2 \left(E(\max_{i,j}(\mathsf{relay}(y'_{ij}))) - \sum_i E(\max_j(\mathsf{relay}(y_{ij}))) \right), \end{aligned}$$

and again, since the expected value of the max cannot be more than the sum of the expected values, we have that $E(\mathsf{traffic}(w')) \leq E(\mathsf{traffic}(w))$. $\square$

Lemma 3.4 *Given a directed multicast tree, T and a node x which not a leaf,* $E(\mathit{traffic}(\mathrm{join}_x(T))) \leq E(\mathit{traffic}(T))$.

Proof: The proof of this is analogous to that of Lemma 3.2, using Lemma 3.3 instead of Lemma 3.1. $\square$

3.3 General Bounds

First, a very simple lower bound can be presented, based on Theorem 2.3.

Theorem 3.5 *Let T be a directed multicast tree. Let P be a path from source s to destination d, with length* $\mathrm{depth}(T)$. *Then,* $E(\mathit{traffic}(T)) \geq E(\mathit{traffic}(P)) = q + q^2 + \cdots + q^{\mathrm{depth}(T)}$.

Proof: Since P is a subgraph of T, the expected traffic cannot be less for T than for P. Theorem 2.3 gives $E(\mathsf{traffic}(P)) = q + q^2 + \cdots + q^{\mathrm{depth}(T)}$. $\square$

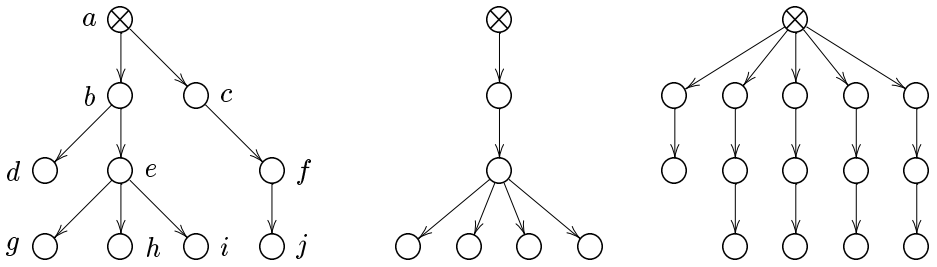
An upper bound and a better lower bound can be found by using the split and join operations.

We define the *broom graph* of a tree T in order to get a lower bound on the expected traffic of T. The broom graph of T consists of path of length $\mathrm{depth}(T) - 1$ from a vertex x to a vertex y, with the source at x and $\lambda_{\mathrm{depth}(T)}$ pendant vertices attached to y. Thus, the broom graph of T is a tree rooted at x with $\lambda_{\mathrm{depth}(T)}$ leaves at depth $\mathrm{depth}(T)$. The broom of T has a total of $\mathrm{depth}(T) + \lambda_{\mathrm{depth}(T)}$ vertices. Given a multicast tree T, the broom graph of T can be formed by repeatedly joining vertices in T.

Figure 3.4(b) gives an example of a broom graph. The broom graph is formed from T by joining b and c and then joining d, e and f.

A broom graph with a path of length a and b leaves is denoted $\mathrm{broom}(a, b)$. The node above the leaves is called the *branch node*. For example, the broom graph in Figure 3.4(b) is $\mathrm{broom}(3, 4)$.

Theorem 3.6 *Let T be a multicast tree. Let T' be the broom graph of T. Then,* $E(\mathit{traffic}(T)) \geq E(\mathit{traffic}(T'))$.



(a) (b) (c)

Figure 3.4: A multicast tree (a) with its broom graph (b) and spider graph (c).

Proof: The broom graph of T can be formed by repeatedly joining vertices. Thus, by Lemma 3.4, we have $E(\mathsf{traffic}(T)) \geq E(\mathsf{traffic}(T'))$. □

An analogous upper bound on the traffic can also be found. Let T be a multicast tree, with λ leaves at depths $a_1, a_2, \ldots, a_\lambda$, respectively. The *spider graph* of T consists of λ disjoint paths of lengths $a_1, a_2, \ldots, a_\lambda$, respectively, with a common endpoint at the root, s. The spider graph of T can be formed by repeatedly splitting vertices of T.

Figure 3.4(c) gives an example of a spider graph. The spider graph of T is formed by splitting e and then splitting b in T.

The notation spider$(a_1, a_2, \ldots, a_\lambda)$ denotes a graph with a source attached to paths of lengths $a_1, a_2, \ldots, a_\lambda$. The spider graph of T has the same number of leaves, at the same depths, as T. The spider graph has $1 + a_1 + a_2 + \ldots + a_\lambda$ vertices. The spider graph in Figure 3.4(c) is spider$(2, 3, 3, 3, 3)$.

Theorem 3.7 *Let T be a multicast tree. Let T' be the spider graph of T. Then, $E(\mathsf{traffic}(T)) \leq E(\mathsf{traffic}(T'))$.*

Proof: The spider graph of T can be formed by repeatedly splitting vertices until no further splits can be performed. Thus, by Lemma 3.2, we have $E(\mathsf{traffic}(T)) \leq E(\mathsf{traffic}(T'))$. □

Now that we have the upper and lower bounds given by Theorems 3.6 and 3.7, we need to determine the expected traffic to multicast to the broom and spider trees.

First, note that Theorem 2.3 gives that a path with d nodes has expected cost $q + q^2 + \cdots + q^{d-1}$. Combining this with Lemma 2.1, we immediately have the expected cost for a

spider graph. The tree spider$(a_1, a_2, \ldots, a_\lambda)$ has expected cost

$$E(\mathsf{traffic}(\mathrm{spider}(a_1, a_2, \ldots, a_\lambda))) = \sum_{i=1}^{\lambda} \left(q + q^2 + \cdots + q^{a_i}\right) . \tag{3.1}$$

Let Λ_i be the number of leaves at depth i *or more* from the root. Then, for a spider graph T,

$$E(\mathsf{traffic}(T)) = \sum_{i=1}^{\mathrm{depth}(T)} \Lambda_i q^i .$$

The expected traffic of a broom graph is more difficult to calculate. A lower bound on its traffic will be given. This gives a looser lower bound on traffic, but it is still better than the lower bound given in Theorem 3.5.

Let T be the tree broom(a, b). Let the vertices on the path be $x_1, x_2, \ldots, x_a$ so that x_1 is the source and x_a is the branch point. We will bound the expected traffic on T using the probability methods of Section 2.2.

If we start with x_a, Lemma 2.1 gives that the expected traffic is $E(\mathsf{traffic}(x_a)) = b \cdot q$. We must also count the number of times that x_a receives the message from its parent. First, note that

$$P(\mathsf{relay}(x_a) = i) = \sum_{j=1}^{b} \binom{b}{j} \left[p^{i-1}(1-p)\right]^j \left(1 - p^{i-1}\right)^{b-j} .$$

Thus,

$$\begin{aligned} E(\mathsf{relay}(x_a)) &= \sum_{i=1}^{\infty} i \cdot P(\mathsf{relay}(x_a) = i) \\ &= \sum_{i=1}^{\infty} \sum_{j=1}^{b} i \binom{b}{j} \left[p^{i-1}(1-p)\right]^j \left(1 - p^{i-1}\right)^{b-j} . \end{aligned} \tag{3.2}$$

I have been unable to evaluate this expression to a closed form. Although, since $E(\mathsf{relay}(x_a))$ must be increasing with b, its value can be bounded below by q, which is the expected number of transmission rounds at x_a if $b = 1$. So, $E(\mathsf{relay}(x_a)) \geq q$.

Now, we can use Equations 2.2 and 2.3 to get

$$E(\mathsf{relay}(x_{a-1})) = q \cdot E(\mathsf{relay}(x_a)) \geq q^2 ,$$

and

$$E(\mathsf{traffic}(x_{a-1})) = q \cdot E(\mathsf{relay}(x_a)) + E(\mathsf{traffic}(x_a)) \geq q^2 + bq .$$

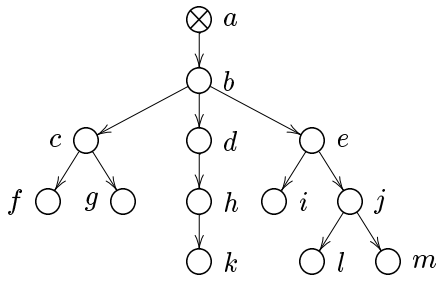


Figure 3.5: A 13-vertex multicast tree to demonstrate Corollary 3.10.

Continuing up to x_1, we get

$$E(\mathsf{traffic}(T)) = E(\mathsf{traffic}(x_1)) \geq q^a + q^{a-1} + \cdots + q^2 + bq\,. \tag{3.3}$$

This bound is not particularly tight and could be improved if a better approximation of Equation 3.2 could be found.

We can restate Theorem 3.6 as

Theorem 3.8 *Let T be a multicast tree with $\lambda_{\mathrm{depth}(T)}$ leaves at distance* $\mathrm{depth}(T)$ *from the root. Then,*

$$E(\mathit{traffic}(T)) \geq E\left(\mathit{traffic}(\mathrm{broom}(\mathrm{depth}(T), \lambda_{\mathrm{depth}(T)}))\right)\,.$$

Note that a similar argument can be used for nodes at other distances.

Theorem 3.9 *Let T be a multicast tree with b nodes at distance a from the root. Then,*

$$E(\mathit{traffic}(T)) \geq E\left(\mathit{traffic}(\mathrm{broom}(a, b))\right)\,.$$

Proof: Let T' be the tree T with all nodes below depth a pruned. Since T' is a subtree of T, $E(\mathsf{traffic}(T')) \leq E(\mathsf{traffic}(T))$. Applying Theorem 3.6 to T' gives the result. □

This immediately gives

Corollary 3.10 *Let b_a be the number of nodes at depth a from the root of a multicast tree T. Then,*

$$E(\mathit{traffic}(T)) \geq \max_{1 \leq a \leq \mathrm{depth}(T)} \left(\mathit{traffic}(\mathrm{broom}(a, b_a))\right)\,.$$

For example, consider the tree in Figure 3.5. This yields the bound

$$E(\mathsf{traffic}(T)) \geq \max\left\{q^4 + q^3 + q^2 + 3\,q, q^3 + q^2 + 5\,q, q^2 + 3\,q, q\right\}\,.$$

3.4 Summary

Let T be a multicast tree and let $T_{\rm sp}$ be the spider graph of T and $T_{\rm br}$ be the broom graph of T.

Let $a_1, a_2, \ldots, a_\lambda$ be the depths of the leaves of T, so that $T_{\rm sp} = \text{spider}(a_1, a_2, \ldots, a_\lambda)$. Let $d = \text{depth}(T)$ and recall that λ_d is the number of leaves of T at depth d, so that $T_{\rm br} = \text{broom}(d, \lambda_d)$. Let b_a be the number of nodes at depth a from the root of T.

The results of this chapter are combined in the following Corollary.

Corollary 3.11 *For T as described above,*

$$\max_{1 \le a \le \text{depth}(T)} (\mathit{traffic}(\text{broom}(a, b_a))) \le E(\mathit{traffic}(T_{br})) \le E(\mathit{traffic}(T)),$$

and

$$E(\mathit{traffic}(T)) \le E(\mathit{traffic}(T_{sp})) = \sum_{i=1}^{\lambda} q^{a_i} + q^{a_i - 1} + \cdots + q\,.$$

Proof: This follows immediately from the theorems and traffic results in Section 3.3. □

Note that for both the upper and lower bound above, the highest order term is q^d since $a_i = d$ for at least one value of i.

Chapter 4

Cache Placement

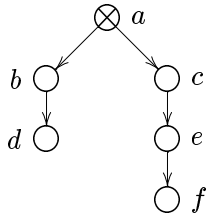


Figure 4.1: A multicast tree to demonstrate cache placement calculations

We now have several tools to calculate the expected traffic to multicast in a given tree. These tools can now be used to explore the optimal cache locations in these trees.

4.1 Calculation

We begin by considering cache placements in small trees. A sample of the calculations necessary to determine the best cache placement will be given, using the tree, $T = \text{spider}(2, 3)$, in Figure 4.1.

Using the Markov method from Section 2.3, implemented in Maple, as described in Appendix A, we find that the expected traffic to multicast to T with no caches is

$$E(\mathsf{traffic}(T)) = q^3 + 2\,q^2 + 2\,q\,.$$

The expected traffic is the same if we place a cache at the source a since the source already acts as a cache. It is also the same if we place a cache at the leaves, d or f. Caches at the leaves are not used since the leaves never send the message.

Thus, we can consider placing a single cache at b, c or e to decrease the expected traffic. We can calculate the expected traffic of each of these placements by using the Markov method and the subtree decomposition described in Section 1.2.5.

First, consider placing a cache at b. The subtrees in this case are spider$(1, 3)$ rooted at a, with b and f as leaves, and a path of length 1 (a path(1)) with b as the source. In this case, the expected traffic is

$$\begin{aligned} &E(\mathsf{traffic}(\text{spider}(1,3))) + E(\mathsf{traffic}(\text{spider}(1))) \\ &= (q^3 + q^2 + 2\,q) + q \\ &= q^3 + q^2 + 3\,q\,. \end{aligned}$$

If we place the cache at c, the subtrees are spider$(2,1)$ and spider(2). Thus, the expected traffic for T with a cache at c is

$$E(\mathsf{traffic}(\text{spider}(2,1))) + E(\mathsf{traffic}(\text{spider}(2))) = 2\,q^2 + 3\,q\,.$$

Finally, placing the cache at e gives subtrees spider$(2,2)$ and spider(1). The expected traffic in this case is

$$E(\mathsf{traffic}(\text{spider}(2,2))) + E(\mathsf{traffic}(\text{spider}(1))) = 2\,q^2 + 3\,q\,.$$

Note that this is the same as the expected traffic when a single cache is placed at c.

The interval $0 \leq p < 1$ where p is valid corresponds to $q \geq 1$. Since the functions $q^3 + q^2 + 3\,q$ and $2\,q^2 + 3\,q$ do not cross in this interval, the optimal placement of a single cache is the same for all q. The placement is not unique; placing a single cache at either c or e gives the lowest expected traffic.

We can also consider placement of two caches in T. There are three possible placements in the interior vertices, $\{b,c\}, \{b,e\}, \{c,e\}$.

First, if caches are placed at b and c, the tree is decomposed into three subtrees, spider$(1,1)$ with a as the root, path(1) with b as the root and path(2) with c as the root. Thus, the expected traffic for T with these caches is

$$\begin{aligned}
&E(\mathsf{traffic}(\text{spider}(1,1))) + E(\mathsf{traffic}(\text{path}(1))) + E(\mathsf{traffic}(\text{path}(2))) \\
&= 2\,q + q + (q^2 + q) \\
&= q^2 + 4\,q\,.
\end{aligned}$$

If two caches are placed at b and e, the subtrees are spider$(1,2)$ rooted at a, path(1) rooted at b and path(1) rooted at e. The expected traffic here is

$$\begin{aligned}
&E(\mathsf{traffic}(\text{spider}(1,2))) + E(\mathsf{traffic}(\text{path}(1))) + E(\mathsf{traffic}(\text{path}(2))) \\
&= (q^2 + 2\,q) + q + q \\
&= q^2 + 4\,q\,.
\end{aligned}$$

Finally, if caches are placed at c and e, the subtrees are spider$(2,1)$, path(1) and path(1). The expected traffic is

$$\begin{aligned}
&E(\mathsf{traffic}(\text{spider}(2,1))) + E(\mathsf{traffic}(\text{path}(1))) + E(\mathsf{traffic}(\text{path}(2))) \\
&= (q^2 + 2\,q) + q + q \\
&= q^2 + 4\,q\,.
\end{aligned}$$

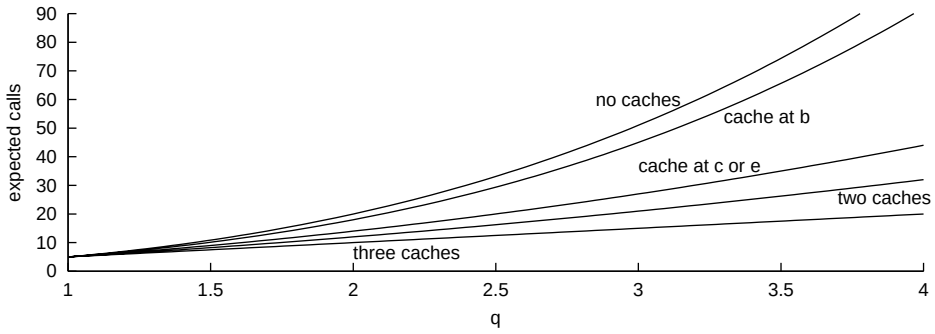


Figure 4.2: Expected calls for various cache placements in Figure 4.1

For two caches, all three possible placements give the same expected traffic. Thus, two caches could be placed at any pair of the interior nodes, producing the same expected traffic.

Lastly, we can place three caches, with one at each of the interior nodes. For these caches, the subtrees are spider$(1, 1)$ rooted at a and three copies of path(1) rooted at each of b, c and e. Thus, the expected traffic is

$$\begin{aligned} & E(\mathsf{traffic}(\text{spider}(1,1))) + 3\,E(\mathsf{traffic}(\text{path}(1))) \\ &= (2\,q) + 3\,q \\ &= 5\,q\,. \end{aligned}$$

Figure 4.2 shows a comparison of the expected traffic for each of the possible cache configurations in T. All of the functions meet at $q = 1$, as they must since $q = 1$ corresponds to $p = 0$, where the caches will have no effect on the multicast. Also, none of the functions cross elsewhere, indicating that the optimal cache location does not depend on the value of q. We will see in Section 4.3 that this is not always the case.

4.2 Examples

For $n = 1, 2, 3, 4, 5, 6$, the number of rooted trees with n vertices is $1, 1, 2, 4, 9, 20$, respectively [23, 24].

Tables 4.1, 4.2, 4.3 and 4.4 list optimal placement of one and two caches, where possible, for multicast trees with three, four, five and six vertices, respectively. Where several cache

#	Tree	1 Cache	2 Caches
1.			
2.			

Table 4.1: Optimal cache placements for all multicast trees with three vertices

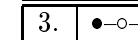
#	Tree	1 Cache	2 Caches
3.			
4.			
5.			
6.			

Table 4.2: Optimal cache placements for all multicast trees with four vertices

placements are presented, they have equivalent expected traffic. For each of these trees, the optimal cache placement does not depend on the value of p.

4.3 Moving Caches

In all of the examples so far, the optimal cache placement has not depended on the value of p. Here, we show that this is not always the case.

Consider placing a single cache in the tree broom$(4,3)$, as seen in Figure 4.3. In this tree, we can place caches at any of b, c or d.

If the cache is placed at b, the subtree containing the root is a path of length 2, with expected cost q. The other subtree, containing the leaves, is a broom$(3,3)$, which has expected cost $(11\,q^6 - 8\,q^5 + 11\,q^4 - 18\,q^3 + 12\,q^2 - 3\,q)/(2\,q-1)(3\,q^2 - 3\,q + 1)$.

If a cache is placed at c, one subtree is a path of length 2, with expected cost q^2+q and the

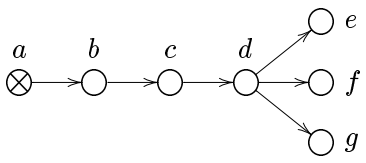


Figure 4.3: A multicast tree where the optimal cache location changes with p.

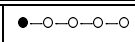
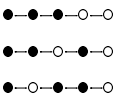
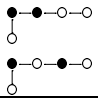
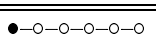
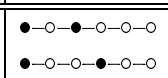
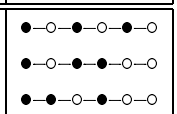
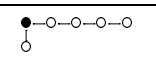
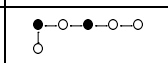
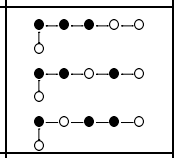
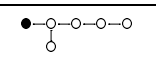
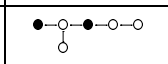
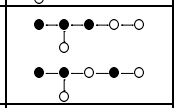
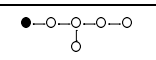
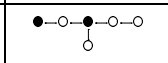
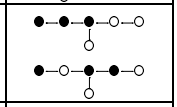
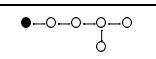
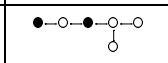
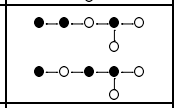
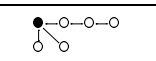
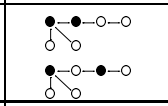
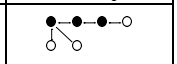
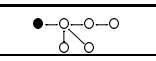
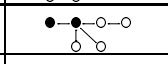
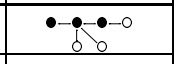
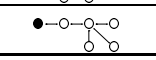
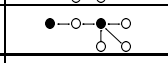
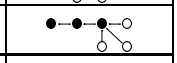
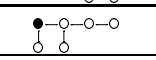
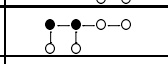
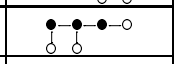
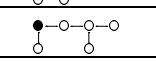
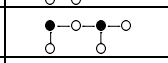
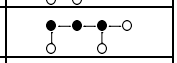
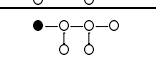
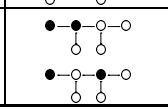
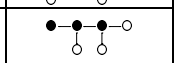
#	Tree	1 Cache	2 Caches
7.			
8.			
9.			
10.			
11.			
12.			
13.			
14.			
15.			

Table 4.3: Optimal cache placements for all multicast trees with five vertices

#	Tree	1 Cache	2 Caches
16.			
17.			
18.			
19.			
20.			
21.			
22.			
23.			
24.			
25.			
26.			

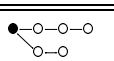
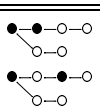
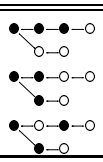
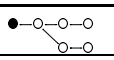
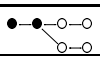
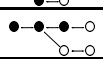
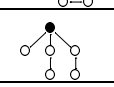
#	Tree	1 Cache	2 Caches
27.			
28.			
29.			
30.			
31.			
32.			
33.			
34.			
35.			

Table 4.4: Optimal cache placements for all multicast trees with six vertices

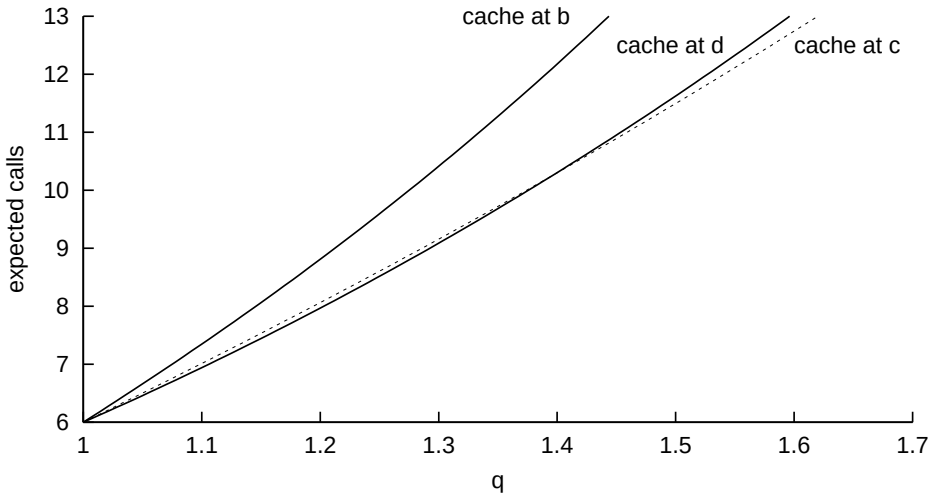


Figure 4.4: Expected calls for various cache placements in Figure 4.3

other is a broom$(2,3)$, with expected cost $(11\,q^5-q^4-15\,q^3+12\,q^2-3\,q)/(2\,q-1)(3\,q^2-3\,q+1)$.

Finally, if the cache is placed at d, the first subtree is a path of length 2, which has cost q^3+q^2+q and the other is a broom$(1,3)$, with expected cost $3q$.

The total expected costs for each cache placement are compared in Figure 4.4. Note that the expected costs for placing caches at c and d cross. The intersection is at $q \approx 1.396$ which corresponds to $p \approx 0.284$. This indicates that for values of p less than 0.284, the optimal cache location is d, as in Figure 4.5(a), but for larger values, the optimal placement is at c, as in Figure 4.5(b).

All of the trees in Tables 4.1, 4.2, 4.3 and 4.4 have been checked and none have this property. Thus, Figure 4.3 is the smallest tree for which the optimal cache placement depends on the value of p.

Further, in the tree broom$(6,5)$ as seen in Figure 4.6, the optimal cache placement moves twice. Similar calculations show that for p below 0.064, the optimal cache placement is at f. For p from 0.064 to 0.462, the optimal cache placement is at e and for larger p, it is at d.

These examples indicate that in general, we cannot simply calculate the optimal cache placements for a tree; we must calculate for a tree *and a specific value of* p.

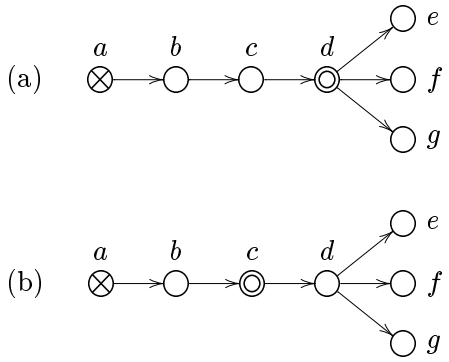


Figure 4.5: Optimal cache locations in Figure 4.3 for $p \leq 0.284$ (a) and $p > 0.284$ (b)

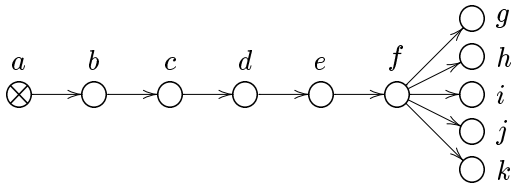


Figure 4.6: A multicast tree where the optimal cache location changes twice with p.

4.4 Heuristic

Since we have no polynomial algorithm for determining the optimal cache placement, we propose a heuristic method. We use the bounds from Corollary 3.11, which can be easily evaluated, to find a good cache placement.

Let T be a multicast tree and let n be the number of nodes in T and n_i be the number of interior nodes in T. Suppose that we wish to place η caches in T.

For each of the $\binom{n_i}{\eta}$ sets of η interior nodes, symbolically calculate the upper bound on the expected traffic for caches at these locations. That is, for each possible cache placement, decompose T into subtrees as described in Section 1.2.5 and sum the upper bounds from Corollary 3.11 for each of these subtrees. This gives an upper bound on the expected traffic for T with these caches.

Keep track of the cache placement with the smallest upper bound for large values of q. This can be done by simply comparing the coefficients of the upper bounds, since we are dealing with large q. This cache placement is chosen by the heuristic algorithm. If there are several alternatives with the same upper bound, choose one arbitrarily.

The upper bound for expected traffic depends only on the distance of the leaves from the nearest cache above. Thus, to calculate the upper bound, it is only necessary to traverse the tree and determine the value of each λ_i for $1 \leq i \leq \text{depth}(T)$. The tree can be traversed and each λ_i calculated in time $O(n)$. Comparison of the upper bounds involves comparing the coefficients calculated. This can take at most $\text{depth}(T)$ steps since there are $\text{depth}(T)$ coefficients. Thus, the heuristic algorithm can be executed in time $O(n \cdot \text{depth}(T) \cdot \binom{n_i}{\eta}) \subseteq O(n^{\eta+2})$.

4.5 Heuristic Examples

In order to illustrate the heuristic method, consider the tree T, in Figure 4.7. The heuristic method of cache placement will be applied to this tree.

4.5.1 A 13-vertex multicast tree, one cache

First, consider the placement of a single cache in this tree. Note that the upper bound is dominated by a term of order $q^{\text{depth}(T)}$. Since the heuristic chooses based on large q, it will choose a cache placement which minimizes the depth of the tallest subtree.

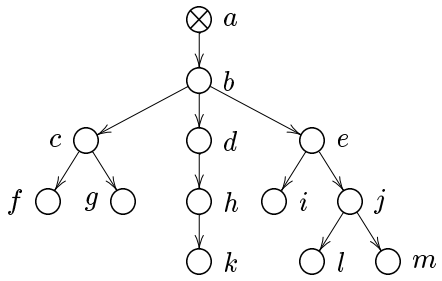


Figure 4.7: A 13-vertex multicast tree to demonstrate the use of the heuristic method.

The subtrees rooted at d and e are the tallest and have the same height. We can consider putting caches at b, d, e, h, i or j since doing so would decrease the height of one of the tallest subtrees.

Since b sits above both of the tallest subtrees, putting a cache there would mean that, in the subtree decomposition, there will be no trees of height greater than 3, and thus the upper bound will be in $O(q^3)$. If a cache was placed at any other node, there would be a subtree of height 4 and the upper bound will be in $O(q^4)$. Thus, the heuristic will place a single cache at b.

Since for large q, the bounds are dominated by a q^d term where d is the height of the tree, b is the optimal cache location for large q. So, in this case, the heuristic produces the optimal solution for large q. Note that this might not be the case for smaller q.

4.5.2 A 13-vertex multicast tree, two caches

Now, consider placing two caches in this tree. The $\binom{6}{2} = 15$ possible placements, along with their bounds are shown in Table 4.5.

Of these, we see that the lowest upper bound for large q occurs for caches at either b and e or b and j. Since the upper bounds are the same, the heuristic indicates that either of these alternatives can be chosen arbitrarily.

Let's now examine how well the heuristic performed.

In this case, we can perform an exact calculation and see that the two alternatives, caches at b and e or b and j, have the same expected traffic. Since we have a cache at b, we are concerned with placing the other cache in the subtree T_e^+. This tree can be found in Table 4.4 as tree number 26. Thus, placing the second cache at either e or j has the

Caches	Subtrees	Lower Bound	Upper Bound
b, c	$\{a, b\}, \{b, c, T_d, T_e\}, T_c$	$q^3 + q^2 + 5\,q$	$3\,q^3 + 4\,q^2 + 8\,q$
b, d	$\{a, b\}, \{b, d, T_c, T_e\}, T_d$	$q^3 + 2\,q^2 + 4\,q$	$2\,q^3 + 6\,q^2 + 8\,q$
b, h	$\{a, b\}, \{b, d, h, T_c, T_e\}, T_h$	$q^3 + q^2 + 4\,q$	$2\,q^3 + 6\,q^2 + 8\,q$
b, e	$\{a, b\}, \{b, e, T_c, T_d\}, T_e$	$q^3 + 2\,q^2 + 4\,q$	$q^3 + 5\,q^2 + 8\,q$
b, j	$\{a, b\}, \{b, e, i, j, T_c, T_d\}, T_j$	$q^3 + q^2 + 4\,q$	$q^3 + 5\,q^2 + 8\,q$
d, e	$\{a, b, d, e, T_c\}, T_d, T_e$	$q^3 + 3\,q^2 + 5\,q$	$2\,q^3 + 7\,q^2 + 8\,q$
d, j	$\{a, b, d, e, i, j, T_c\}, T_d, T_j$	$q^3 + 2\,q^2 + 7\,q$	$4\,q^3 + 6\,q^2 + 8\,q$
h, e	$\{a, b, d, e, h, T_c\}, T_h, T_e$	$q^3 + 2\,q^2 + 6\,q$	$3\,q^3 + 6\,q^2 + 8\,q$
h, j	$\{a, b, d, e, h, i, j, T_c\}, T_h, T_j$	$q^3 + q^2 + 8\,q$	$5\,q^3 + 5\,q^2 + 8\,q$
c, d	$\{a, b, c, d, T_e\}, T_c, T_d$	$q^4 + q^3 + 2\,q^2 + 5\,q$	$2\,q^4 + 3\,q^3 + 6\,q^2 + 8\,q$
c, e	$\{a, b, c, e, T_d\}, T_c, T_e$	$q^4 + q^3 + 2\,q^2 + 5\,q$	$q^4 + q^3 + 5\,q^2 + 8\,q$
c, h	$\{a, b, c, d, h, T_e\}, T_c, T_h$	$q^4 + q^3 + q^2 + 5\,q$	$2\,q^4 + 4\,q^3 + 5\,q^2 + 8\,q$
c, j	$\{a, b, c, e, i, j, T_d\}, T_c, T_j$	$q^4 + q^3 + q^2 + 5\,q$	$q^4 + 3\,q^3 + 4\,q^2 + 8\,q$
d, h	$\{a, b, d, T_c, T_e\}, \{d, h\}, T_h$	$q^4 + q^3 + q^2 + 4\,q$	$2\,q^4 + 5\,q^3 + 6\,q^2 + 8\,q$
e, j	$\{a, b, e, T_c, T_d\}, \{e, i, j\}, T_j$	$q^4 + q^3 + q^2 + 5\,q$	$q^4 + 3\,q^3 + 4\,q^2 + 8\,q$

Table 4.5: Bounds for expected traffic in Figure 4.7 with two caches

same expected traffic. We see now that the heuristic could not decide between these two alternatives since they are indeed equivalent.

Note that the lower bounds of the other choices are not higher than the upper bound on the cache placement chosen, except those with q^4 terms. Thus, we cannot be sure that we have picked the optimal placement.

For small q, less than about 2, the lower bounds for the possibilities with q^4 terms are lower than the upper bound for the cache placement we chose. Thus, it is also possible that one of these cache placements is better for small q.

We know that the heuristic found a cache placement with minimum order. The placement chosen seems to be good, for this tree.

4.5.3 The broom graph broom$(4, 3)$

In Section 4.3, we saw two multicast trees for which the heuristic algorithm cannot always pick the best cache location for a single cache. The optimal location for a single cache in the trees in Figures 4.3 and 4.6 changes with q. Since the heuristic does not take the value of q into account, it cannot always choose the optimal location.

First, consider applying the heuristic to the placement of a single cache in the broom

graph broom$(4, 3)$, as in Figure 4.3. The heuristic will place the cache at d since this minimizes the height of the tallest subtree. As noted in Section 4.3, this is optimal for large q.

The largest difference between the heuristic placement and the optimal placement is at $q = 1.197$. Here, the expected number of calls in the optimal case is 7.939 and with the heuristic placement the expected number of calls is 8.034. Here, the difference in the expectations is 0.095 calls or just over 1% of the total.

Thus, even though the heuristic does not give the optimal solution for all cases, the solution that it gives is very close to the optimal for all q.

4.5.4 The broom graph broom$(6, 5)$

Now, consider applying this heuristic to the tree broom$(6, 5)$, as in Figure 4.6. The heuristic will place a single cache at d in this tree. Again, this is optimal for large q.

The largest difference is at $q = 1.492$. The difference in the expected number of calls is 0.734 or about 3% of the total.

Again, the heuristic gives a solution which is very close to the optimal solution.

4.5.5 Summary

As these examples illustrate, the heuristic method can be used to suggest cache placements quickly. To evaluate the bounds in Corollary 3.11, it is necessary only to count the number of vertices and leaves at each level in each subtree.

For the particular examples given here, the heuristic performs well, giving optimal or near-optimal placements. To determine how the heuristic performs more generally requires additional investigation.

Chapter 5

A Generalization to Digraphs

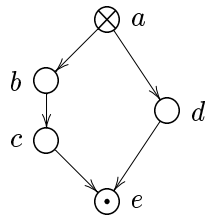


Figure 5.1: A communications graph with source a and destinations $\{e\}$

In previous sections, we have assumed that the multicast tree has been chosen and we wish to determine a cache placement which allows us to minimize the expected traffic for our multicast. We believe this is a hard problem, in general.

From a more systems-oriented viewpoint, both the multicast tree and the cache placement need to be chosen. An obvious problem to consider is to choose the best solution to this larger problem. That is, given G, D and s, find a multicast tree and cache placement that gives the best expected traffic over all pairs of multicast trees and cache placements. In this chapter, we show that this larger problem is NP-complete for directed acyclic networks.

5.1 Example

As an example, consider placing a single cache in the graph in Figure 5.1. In this graph, a cache can be placed at b, c or d. A cache placed at the source or at a destination will have no effect on the traffic, as above.

Suppose a cache is placed at either b or c. The subtree containing a, b, c, e, as in Figure 5.2(a), has expected cost $q^2 + 2\,q$. The other possible subtree, containing a, d, e, as in Figure 5.2(b), has expected cost $q^2 + q$.

Now, suppose a cache is placed at d. The subtree containing a, b, c, e, in Figure 5.2(c), has expected cost $q^3 + q^2 + q$ and the subtree containing a, d, e, in Figure 5.2(d), has expected cost $2\,q$.

Of these, the lowest expected cost is with a cache at d, using the sub-path a, d, e as the multicast subtree. This gives the lowest possible expected traffic for a single cache in this network.

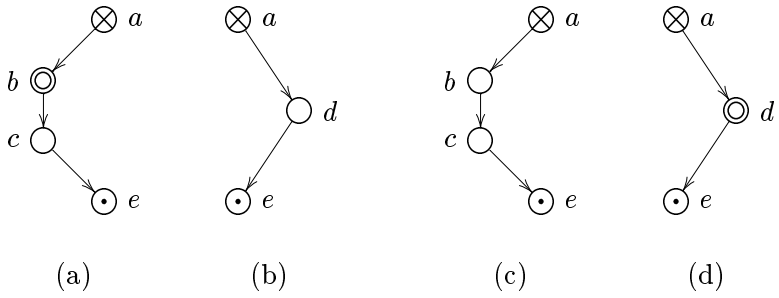


Figure 5.2: The four possible multicast trees and cache placements in Figure 5.1

5.2 NP-Completeness Result

We show that the process of choosing the optimal cache locations in a graph is NP-Complete.

Theorem 5.1 *Given a directed acyclic graph $G = (V, E)$, a source node $s \in V$, a set of destinations $D \subseteq V$, an integer η and a number k. To determine whether there is a placement of η caches in G for which the expected network traffic for some subtree is at most k is NP-complete.*

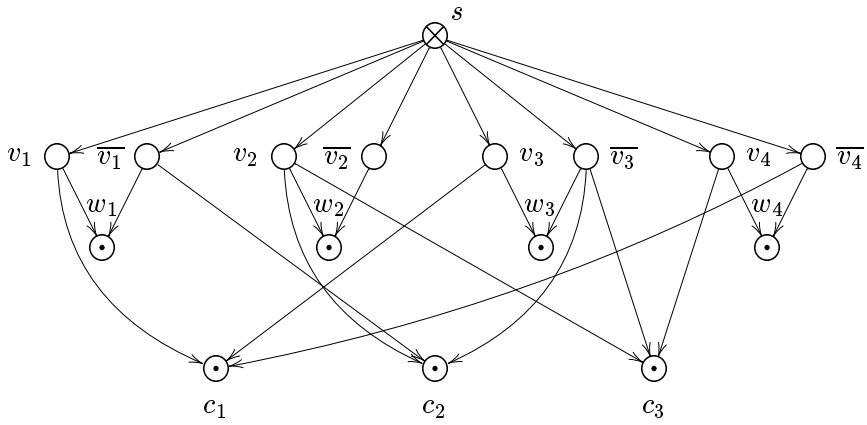
Proof: To prove this, we will show that it can be used to decide 3-SAT [13]. Consider an instance of 3-SAT with variables $v_1, v_2, \ldots, v_\eta$ and clauses $a_{1,1} \vee a_{1,2} \vee a_{1,3}, a_{2,1} \vee a_{2,2} \vee a_{2,3}, \ldots, a_{m,1} \vee a_{m,2} \vee a_{m,3}$ where each $a_{i,j}$ is either v_r or $\overline{v_r}$ for some r.

We will now construct an instance of multicast cache placement on a directed graph which has expected traffic at most a given k iff the 3-SAT instance is satisfiable. Let the following be the vertices.

- a source node, s
- nodes for each literal, $v_1, \overline{v_1}, \ldots, v_\eta, \overline{v_\eta}$
- destination nodes for each clause, $c_1, c_2, \ldots, c_m$
- a destination node for each variable, $w_1, w_2, \ldots, w_\eta$.

Now, create the following directed edges.

- from s to each v_i and $\overline{v_i}$.
- from each v_i and $\overline{v_i}$ to w_i.



$$c_1 = v_1 \vee v_3 \vee \overline{v_4}$$
$$c_2 = \overline{v_1} \vee v_2 \vee \overline{v_3}$$
$$c_3 = v_2 \vee \overline{v_3} \vee v_4$$

Figure 5.3: The multicast graph created in the proof of Theorem 5.1, for the instance of 3-SAT with clauses listed, with $\eta = 4$, $m = 3$.

- for each clause $a_{i,1} \vee a_{i,2} \vee a_{i,3}$, from $a_{i,j}$ to c_i for $j = 1, 2, 3$.

Figure 5.3 gives a sample of this construction for $(v_1 \vee v_3 \vee \overline{v_4}) \wedge (\overline{v_1} \vee v_2 \vee \overline{v_3}) \wedge (v_2 \vee \overline{v_3} \vee v_4$.)

Now, consider the problem of determining the optimal cache placement in this graph. Let the number of caches be η and let $k = q\,(2\eta + m)$.

Caches will not be placed on any of the w_i or c_i vertices. These nodes have no outgoing edges, so a cache placed at any of them could never retransmit the message. Thus, that cache would have no effect on the total traffic. A cache will not be placed at s since the source already acts as a cache. So, again a cache here would have no effect on the total traffic. Thus, all caches must be placed at either v_i or $\overline{v_i}$ nodes.

If caches are placed so that each w_i and c_i has an in-neighbor which is a cache, we can choose a multicast tree like Figure 5.4 where each destination node has a parent which is a cache. In such a tree, the traffic may be calculated using the decomposition described in Section 1.2.5. In this multicast tree, each of the subtrees is a path(1) and has expected cost q. Thus, this multicast tree has the expected number of transmissions $k = q\,(2\eta + m)$, since there are $2\eta + m$ subtrees.

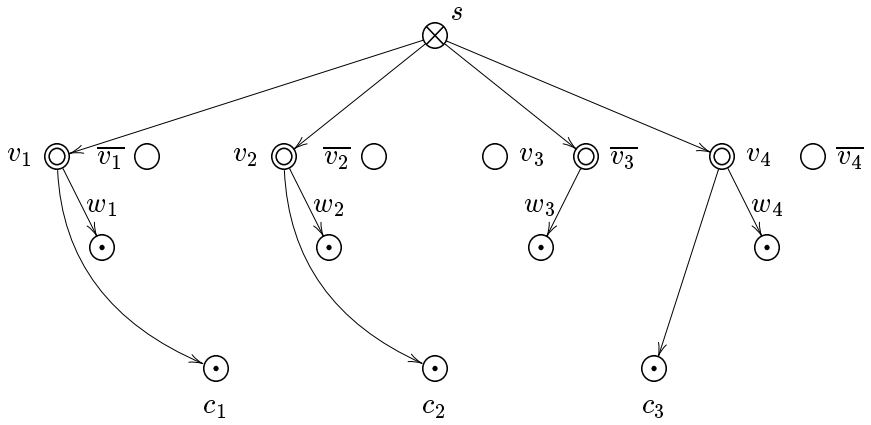


Figure 5.4: An optimal cache placement and subtree for Figure 5.3

Suppose that there is an i such that neither v_i nor $\overline{v_i}$ are cache nodes. Then, w_i has no cache between it and the source. Consider any multicast subtree, T, of this graph; T must contain either v_i or $\overline{v_i}$ in order to contain w_i. Without loss of generality, suppose T contains v_i. Let m_0 be the number of c_j vertices below v_i in T. Then, the subtree $T^+_{v_i}$ is a broom$(1, 1+m_0)$. From Corollary 3.11, the traffic of this graph is bounded below by $q^2 + (1+m_0)q$. Thus, the total expected traffic for T is

$$\begin{aligned} E(\mathsf{traffic}(T)) &= E(\mathsf{traffic}(T^+_{v_i})) + E(\mathsf{traffic}(T \setminus T^+_{v_i})) \\ &= E(\mathsf{traffic}(\mathrm{broom}(1, 1+m_0))) + q\left[2(\eta-1) + (m-m_0)\right] \\ &\geq q^2 + (1+m_0)q + 2q(\eta-1) + q(m-m_0) \\ &= (q^2-q) + q\left(2\eta + m\right). \end{aligned}$$

So, for $q > 1$, the expected traffic is larger than k.

Thus, for each i, either v_i or $\overline{v_i}$ must be a cache to get k expected transmissions. By the pigeonhole principle, since we have η caches, at most one of v_i and $\overline{v_i}$ can be caches.

Now, suppose that this instance of 3-SAT is satisfiable. Then, choose the vertices corresponding to the true literals in the solution as caches, and choose a subtree so that each c_i has a parent which is a cache. From the above, this gives an expected number of transmissions of k.

Conversely, suppose that there is a cache configuration which gives a subtree where each c_i has a cache as a parent. Then, using the cache vertices as true literals, we see that the

3-SAT instance is satisfiable.

Thus, the instance of 3-SAT is satisfiable iff it is possible to distribute caches in this network and choose a multicast tree so that the expected number of calls is at most k. Thus, we see that the cache placement problem on digraphs with no directed cycles is NP-complete. □

The best algorithm we have for the original problem of cache placement in a multicast tree takes exponential time and we have also seen, in the previous theorem, that the generalization of this problem is NP-complete. Thus, we propose the following conjecture.

Conjecture 5.2 *Given a directed multicast tree, T and a maximum number of caches, η, we wish to determine if there is a location of caches that gives an expected number of calls at most k. This problem is NP-complete.*

Chapter 6

Conclusion

6.1 Summary

Two methods were developed in Chapter 2 to count the expected traffic to multicast in a given tree. The probability theory based method of Section 2.2 is difficult to calculate because it is necessary to find the expected value of the maximum of several random variables in Equation 2.2.

The Markov based method in Section 2.3 can be calculated in a much wider variety of cases. However, it is necessary to invert a matrix of size up to $2^n \times 2^n$ where n is the number of nodes in the tree, in order to calculate the expected traffic, using the fundamental Markov method. Thus, this method of calculation is quite slow.

In Chapter 3, the transformations on a multicast tree T, split and join were introduced and shown to not decrease and not increase, respectively, the expected traffic to multicast in T. These were used to create the spider and broom graphs of T. The expected traffic of these graphs was calculated, giving general bounds on the expected traffic to multicast in T. These bounds are summarized in Corollary 3.11.

Having bounds which can be quickly calculated allows us to more closely examine the problem of cache placement in Chapter 4. First, a method to calculate optimal cache placement is given. This relies on calculating the exact expected traffic and thus can be applied only to small graphs. The optimal cache locations were calculated for all trees with 3, 4, 5 and 5 vertices; the results of this were presented in Tables 4.1, 4.2, 4.3 and 4.4. In Section 4.3, it was shown that for some trees, the optimal cache location depends on the value of p.

For larger trees, a heuristic method of determining cache placements was given. This heuristic uses the bounds presented earlier.

Finally, a more general version of the problem was considered in Chapter 5. This more general problem of cache placement shown to be NP-complete in directed acyclic graphs in Theorem 5.1.

6.2 Other approaches

There is an another method of performing Markov calculations based on flow graphs [14, 19, 20]. In this method, the Markov chain is represented by a directed graph (the *flow graph*) where each state corresponds to a vertex in the graph. The edges are weighted to carry the

transition probabilities.

There are several methods of calculating the *generating function* from this graph [20]. The generating function contains a great deal of information about the Markov process. In particular, for a Markov process with the generating function $F(p)$, the expected finishing time is $F'(1)$.

This method of performing the Markov calculations could have advantages over the matrix-based method used here. We must first calculate the generating function in order to determine the expected traffic. Given the generating function, there may be other interesting information which can be easily calculated from it.

It is also possible that the flow graph Markov method is more computationally efficient. Since there are still $O(2^n)$ states where n is the number of vertices in the tree, the algorithm must still take time at least $O(2^n)$. The matrix-based method, as we see in Appendix A, is $O(2^{3n})$.

6.3 Future Work

There are many possible aspects of Internet multicasting that have not been explored.

We found a lower bound on the expected traffic for broom graphs in Corollary 3.10. This is not a particularly tight bound. If a better bound on the traffic of a broom graph could be found, it would lead to a better lower bound for $E(\mathsf{traffic}(T))$ and thus greater certainty in the choices made by the heuristic given in Section 4.4.

On real networks, the probability of transmission failure is quite small. It remains to be seen how effective caches are with values of p and network configurations which correspond to those of the Internet or other real networks.

We have considered the problem of cache placement in a multicast tree. The best algorithm we have to produce an optimal solution for this problem takes exponential time. Thus, we proposed the heuristic algorithm given in Section 4.4. We have also seen in Theorem 5.1 that a generalization of this problem is NP-complete. This lead to Conjecture 5.2.

A different cost function could be considered. Here, we focused on minimizing network traffic. It would also be relevant to consider the time taken to complete the multicast. There are some details of the timing of the multicast algorithm given in Chapter 1 that would have to be specified in order to do this. It would also be possible to consider some aggregate of time and traffic.

In general, the multicast that we are considering is not the only traffic in the network. There could be other communications taking place between nodes in the network which are not part of the multicast process. We could ask how this other traffic interacts with the multicast we are considering.

In particular, there may be other multicasts taking place from other sources to other destinations, which may have started at different times. Is it possible to place some caches in such a network to minimize the traffic necessary to complete both multicasts? The problem of finding a single spanning tree for several originators has been studied [12].

Finally, caches are used on the World-Wide Web to decrease network traffic [9]. This process is similar to the multicast problem considered here, but the destinations request the message at different times. This leads to several complications in the analysis of the problem. First, a reasonable model of the request behaviour must be determined. There has been work in this area [1, 2]. Also, with web traffic, the assumption made here that a cache can hold all of the messages which pass by becomes unreasonable. Thus, we must create and model some policy for removing information from the caches. Work has also been done in this area [8, 26]. Finally, all of these must be worked into a model similar to the one used here and analyzed.

Appendix A

Maple Code

All expected traffic results for specific graphs were calculated using the Maple symbolic manipulation package. The Markov method for calculation was implemented. A description of the algorithms follows.

A.1 Generating Possible States

This algorithm first analyzes the tree to determine a list of valid states. States consist of a set of pairs $[a, b]$, which corresponds to the notation $a_{(b)}$. The pairs consist of two vertices—and informed vertex and the child it will send to next, as in Table 2.1.

This procedure takes the power set of all possible pairs and eliminates sets which don't correspond to possible states. Thus, this algorithm requires $\Omega(2^n)$ time, where n is the number of vertices in the tree.

We let σ denote the number of states found. As stated earlier, $\sigma \in O(2^n)$.

A.2 Generating the Transition Matrix

This algorithm forms a sparse matrix of size $\sigma \times \sigma$ and iterates over the states, filling in the states reached by success and failure in each case, as in the example in Section 2.3.2

Determining the next state is not easy. First, the algorithm determines which vertex should send in this step (if no vertex should send, this is the final state and the process will stay in this state with probability 1). The lowest vertex with uninformed leaves below it, a, is found—it is the vertex which will send. Let b be the receiver of this call. That is, $a_{(b)}$ is in the state.

Recall from Section 2.3.2 that the multicasting process must be decomposed so that a single call occurs in each Markov step. This is done using the method described in Section 2.3.2.

Now that a has sent, $a_{(b)}$ must be removed from the state and, if b is not the last child of a, replaced with $a_{(c)}$ where c is the next child of a. It is, however possible that all of the children of c are already informed. A function is called which detects this and adjusts the state accordingly. This gives the next state if the call fails—p is entered into the appropriate column.

Next, b is added to the state. This gives the next state if the call is successful—$1 - p$ is entered into the appropriate column.

Once the row corresponding to each state has been filled in, a function checks the matrix and determines which states are actually reachable. There may be states which cannot be reached due to slack in the state generation function described in Section A.1. The rows and columns corresponding to unused states are deleted. This speeds the remaining steps.

A.3 Using the Fundamental Markov Method

Finally, the expected number of calls can be found using Theorem 2.2, the Fundamental Markov Method.

This stage involves removing a row and column from P to get Q and calculating $N = (I-Q)^{-1}$. The inversion of this matrix is the most expensive step since it is $O(\sigma^3) \subseteq O(2^{3n})$.

A.4 Cache Placement

The above algorithms can determine the expected cost of multicasting to a given multicast tree. By dissecting a tree, as described in Section 1.2.5, the expected traffic for a tree with caches can be determined.

A procedure takes a tree and a given number, η, of caches. Every η-subset of internal nodes is checked as a possible set of caches. Each set of caches and the expected traffic for that placement is returned.

Note that the optimal cache placement can depend on p, as seen in Section 4.3. Thus, the above algorithm must determine the optimal cache placement for a given p. The various expected costs can also be compared for intersections. If there are any intersections, the optimal cache placement may change at this p.

Bibliography

[1] Ghaleb Abdulla. *Analysis and Modelling of World Wide Web Traffic*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, May 1998.

[2] Martin F. Arlitt and Carey L. Williamson. A synthetic workload model for internet mosaic traffic. (obtained at `ftp://ftp.cs.usask.ca/pub/discus/paper.95-8.ps.Z`), 1995.

[3] Friedhelm Meyer auf der Heide, Berthold Vöcking, and Matthias Westermann. Caching in networks (extended abstract). In *Proceedings of the 11th SODA*, pages 430–439, 2000.

[4] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, August 1994. (obtained at `http://www.acm.org/pubs/articles/journals/cacm/1994-37-8/p76-berners-le%e/p76-berners-lee.pdf`).

[5] Steve Casner. Frequently asked questions (FAQ) on the multicast backbone (MBONE). `ftp://isi.edu/mbone/faq.txt`, December 1994.

[6] Hyeong-Ah Choi and Abdol-Hossein Esfahanian. A message-routing strategy for multicomputer systems. *Networks*, 22:627–646, 1992.

[7] Hyeong-Ah Choi, Abdol-Hossein Esfahanian, and B. Houck. Optimal communication trees with applications to hypercube multicomputers. In *Proceedings of the 6th International Conference on Theory and Applications of Graph Theory*, 1988.

[8] Edith Cohen and Haim Kaplan. Exploiting regularities in web traffic patterns for cache replacement. In *Proceedings of the 31st Annual Symposium on Theory of Computing*, number 31 in Proceedings of the Annual ACM Symposium on the Theory of Computing, 1999. (obtained at `http://www.research.att.com/~edith/Papers/webcache.ps.Z`).

[9] Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford. Improving end-to-end performance of the web using server volumes and proxy filters. In *Proceedings of the ACM SIGCOMM '98*, pages 241–253, September 1998. (obtained at `http://www.acm.org/pubs/citations/proceedings/comm/285237/p241-cohen/`).

[10] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.

[11] S. Deering. *RFC-1112: Host Extensions for IP Multicasting.* Internet Society/Internet Engineering Task Force, August 1989. (obtained at `http://www.faqs.org/rfcs/rfc1112.html`).

[12] Arthur M. Farley, Paraskevi Fragopoulou, David Krumme, Andrzej Proskurowski, and Dana Richards. Multi-source spanning tree problems. In *Proceedings of SIROCCO*, 1999.

[13] Michael R. Garey and David S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[14] Keith O. Geddes. An application of flowgraphs in operating systems. Class project for J.D. Lipson, May 1971.

[15] Dean L. Isaacson and Richard W. Madsen. *Markov Chains—Theory and Applications.* John Wiley & Sons, New York, 1976.

[16] Vicki Johnson and Marjory Johnson. How IP multicast works. `http://www.ipmulticast.com/community/whitepapers/howipmcworks.html`, 1997. An IP Multicast Initialtive White Paper.

[17] Y. Lan, A. H. Esfahanian, and L. M. Ni. Distributed multi-dimension routing in hypercube multiprocessors. In *Proceedings of the 3rd Conference on Hypercube Computers and Concurrent Applications*, pages 631–639, 1988.

[18] Xiaola Lin and Lionel M. Ni. Multicast communication in multicomputer networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(10):1105–1117, October 1993.

[19] John D. Lipson. Flowgraphs and their generating functions. *Information Processing*, 74:489–493, 1974.

[20] John David Lipson. *Flowgraphs and their Generating Functions.* PhD thesis, Harvard University, Cambridge Massachusetts, December 1969.

[21] J. R. Norris. *Markov Chains.* Cambridge University Press, Cambridge, 1997.

[22] Vern Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, October 1997. (obtained at `ftp://ftp.ee.lbl.gov/papers/vp-routing-TON.ps.Z`).

[23] N. J. A. Sloane. On-line encyclopedia of integer sequences. `http://www.research.att.com/~njas/sequences/`.

[24] N. J. A. Sloane. *A Handbook of Integer Sequences.* Academic Press, New York, 1973.

[25] Kunwadee Sripanidkulchai, Andy Myers, and Hui Zhang. A third-party value-added network service approach to reliable multicast. In *SIGMETRICS*, pages 166–177. ACM, May 1999.

[26] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox. Removal policies in network caches for world-wide web documents. In *SIGCOMM '96*, pages 293–305. ACN, August 1996. (revised version available at `http://ei.cs.vt.edu/~succeed/96sigcomm/`).

Index