Optimal Deterministic Timeouts for Reliable Scalable Multicast

Matthias Grossglauser
AT&T Bell Laboratories
Murray Hill, NJ 07974
grossgla@research.att.com *

Abstract

Reliable multicast suffers from the problem of feedback implosion. To achieve scalability, the number of receivers sending feedback in case of loss must remain small. However, losses experienced by different receivers are strongly correlated, since they share resources in the multicast tree.

We present DTRM (Deterministic Timeouts for Reliable Multicast), a distributed algorithm to compute optimal deterministic timeouts for each receiver in a multicast tree as a function of the tree topology and sender-to-receiver delays. DTRM has several desirable properties. First, the computation of the timeouts is entirely distributed; receivers and intermediate nodes only rely on local topology information. Second, NACK implosion is provably avoided for a single loss anywhere in the tree if delay jitter is bounded. Third, feedback information does not need to be processed by intermediate nodes, and receivers do not have to collaborate.

We foresee two possible uses for DTRM. In networks providing hard delay bounds, timeouts can be computed once at session set-up time. In networks with unbounded delays, such as the Internet, timeouts can be adaptively recomputed in response to changes in estimated round-trip times.

1 Introduction

The availability of high speed transmission technology, the ever increasing performance of workstations and personal computers, and the convergence of the data and telecommunications industry will foster the creation of a powerful networking infrastructure. This environment will enable a whole panoply of new applications, including multimedia and distributed computing. It becomes increasingly clear today that many of these applications will rely on an efficient multicasting service. The recent success of Internet multicasting on the MBONE confirms this. Videoconferencing on the Internet is rapidly becoming a standard tool in the research community, despite the rather limited video and audio quality.

Some applications require a *reliable multicast service*, i.e. the error-free transport of information to a group of recipients. Such applications include distribution of "non fault-tolerant" information (such as software, news, or web-pages), distributed computing (database and operating systems), or network management. We believe that the importance of this type of service will increase in the future, when distributed computing becomes commonplace.

It has been observed [1] that a receiver-based multicast protocol achieves better scalability than a sender-based one. But even a receiver-based scheme suffers from the NACK-implosion problem if the losses at different receivers are correlated. This is very likely due to the resource-sharing in a multicast tree. When a packet is lost on some link, then all receivers on the subtree fed by this link experience a loss at the same time and respond.

There are two approaches to solve this problem: *structure-based* and *timer-based*. In structure-based approaches, intermediate nodes in the tree process and combine feedback information. Timer-based approaches, such as the one discussed in this paper, do not rely on processing by network nodes. Rather, they rely on *delayed feedback* to avoid an implosion. They have the advantage of not requiring network support for implosion avoidance, but the potential disadvantage of higher application-to-application delays.

Another dimension in the solution space is receiver collaboration: if receivers are to collaborate, then they have to buffer correctly received data in order to be prepared to respond to other receivers' NACKs. This also implies that feedback has to be multicast to all receivers, i.e. that each receiver is also a multicast source. We assume here that receivers are greedy in the sense that they immediately consume data they have already received. Only the sender buffers data for possible repairs. The receivers only unicast feedback to the source.

Proposals have been made to use random NACK delay timers, which allows to reduce the expected number of NACKs [2, 3]. One reason why only these heuristic approaches have been proposed is that the current Internet does not provide for any Quality of Service guarantees. Next generation networks are expected to offer a more elaborate service model with delay and delay jitter guarantees. Currently, various bodies are working on standardizing such services, e.g. the ATM Forum or the Internet Engineering Task Force (IETF).

This paper presents a rigorous approach to the NACK-implosion problem. We compute *deterministic* timeouts based on the multicast tree topology and sender-to-receiver delays. We establish an optimality

*Current address: INRIA, BP 93, 2004 route des Lucioles, 06902 Sophia Antipolis Cedex, France, email: mgross@sophia.inria.fr.

criterion and present an algorithm, called DTRM (for Deterministic Timeouts for Reliable Multicast), that computes, for every receiver, a timeout value. In a networking environment with delay guarantees, NACK-implosion is provably avoided. The timeout computation is efficient and distributable. Neither sender nor receivers need to have knowledge of group membership or of the complete tree topology. Each receiver only needs to know its round-trip delay to the sender.

The remainder of this paper is structured as follows. Section 2 discusses some related work. Section 3 presents the context of the problem in more detail. Section 4 explains how timeouts have to be set to fulfill the *single-NACK condition*, and establishes an optimality criterion. Section 5 presents an algorithm to compute an optimal timeout allocation. Section 6 discusses the algorithm, and Section 7 concludes the paper.

2 Related Work

In [1], Pingali et al. present an analytical comparison of three generic reliable multicast protocols. They conclude that a receiver-based approach, where the receivers carry the burden of detecting losses and requesting retransmissions, is preferable to a sender-based approach.

The idea of randomly delaying feedback by receivers (slotting) and having receivers suppress redundant feedback (damping) has been introduced in the XTP protocol [2]. These ideas have been used, for example, in the SRM protocol [3]. SRM has been used as the underlying transport system in `wb`, a shared whiteboard application. The authors of SRM present an adaptive algorithm that adjusts the intervals from which the random timers are chosen to the network condition.

Papadopoulos and Parulkar [4] define a reliable multicast taxonomy around an hypothetical optimal algorithm. In their taxonomy, our approach is called *sender-controlled implosion control*. They also outline an algorithm that organizes receivers into *buckets*, based on their round trip delay, but not on topology. Before requesting retransmission upon loss, each receiver has to make sure that all the receivers in buckets with smaller round trip delays have not already requested this retransmission. The maximum number of requests can therefore be limited by limiting the maximum number of receivers in each bucket.

Two approaches for structure-based implosion avoidance are discussed in [5, 6]. The fundamental difference between the two approaches is that in [5], a group member (called *Designated Receiver*) is responsible for processing a region's feedback, while in [6], dedicated servers (called *Logging Servers*) perform this function.

3 Context

As we try to avoid processing of feedback information by intermediate nodes (routers or switches), these simply forward all feedback packets towards the sender.

We assume for the moment that the end-to-end delay for a packet sent from the sender to a receiver α is constant and equal to d^P_α. Also, we assume that the receiver-to-sender delay for NACKs is constant and equal to d^N_α. We call $d_\alpha = d^P_\alpha + d^N_\alpha$ the *round trip delay* of receiver α. Furthermore, we assume that processing at the sender and the receiver is immediate (retransmission due to a NACK packet at the source, detection of loss at a receiver upon reception of an out-of-sequence packet).

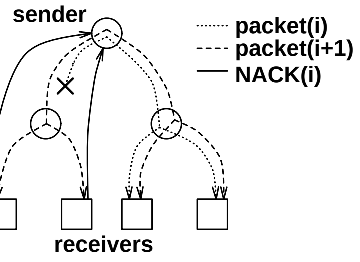

Figure 1: A packet loss seen by multiple receivers results in a NACK being sent back to the sender by each receiver that has seen the loss.

Figure 1 shows how the NACK-retransmission scheme works: if packet i is lost on some link, then each receiver in the subtree attached to that link observes this loss upon reception of packet $i+1$, and starts a timer. When a receiver's timer expires, it sends a NACK(i) packet back to the sender, requesting retransmission of packet i. When a receiver that has started a retransmission timer for packet i receives the retransmitted packet i, it stops the timer without further action.

A NACK-implosion can be avoided if we can make sure that for any loss, one NACK arrives early enough such that the retransmission of the lost packet caused by this NACK prevents further NACKs from other receivers. For this, the retransmitted packet has to arrive at these other receivers before their timers expire. We refer to this requirement as the *single-NACK condition*. The implicit assumption here is that it is reasonable to prevent NACK-implosion at the cost of increased delays in the case of loss.

4 Computing Timeouts

In this section, we first discuss how to determine *some* set of timeouts such that the single-NACK condition is met in the entire multicast tree. Then we establish an optimality criterion. Finally, we describe a distributed algorithm to determine an optimal set of timeouts.

Let us introduce some notation. We assume the existence of a multicast tree. The root of this tree is the *sender*. Non-terminal nodes other than the root are called *switches*, and terminal nodes *receivers*. Where this does not lead to ambiguity, we will sometimes confuse a subtree and its root. Finally, by *downstream* we mean "towards the receivers", and by *upstream* we mean "towards the sender".

4.1 How to Choose Consistent Timeouts

Assume a packet is lost on a link, and call the subtree fed by this link the *loss subtree*. All receivers in the loss subtree, and only they, experience this loss. Consider two receivers α and β in the loss subtree with round trip delays d_α and d_β and timeouts t_α and t_β, respectively.

Assume that an out-of-sequence packet, i.e. the packet following the lost packet, is sent at time 0. The out-of-sequence packet reaches receiver α at time d_α^P and receiver β at time d_β^P. The timer of receiver α therefore expires at time $t_\alpha + d_\alpha^P$, the one of receiver β at $t_\beta + d_\beta^P$. The resulting NACK from receiver α reaches the source at time $t_\alpha + d_\alpha$, and the retransmitted packet reaches receiver β at time $t_\alpha + d_\alpha + d_\beta^P$. Therefore, to ensure that only receiver α sends a NACK back to the sender when both receiver α and β experience a loss, we must have

$$t_\beta \geq t_\alpha + d_\alpha + \epsilon \tag{1}$$

where $\epsilon > 0$ is a small constant. We say that β fulfills the single-NACK condition with respect to α. Note that given the assumption of constant delays, there is no point in choosing t_β larger than $t_\alpha + d_\alpha + \epsilon$, so we assume equality in the sequel.

Our goal is to set timeouts for receivers in a multicast tree such that any single loss on a link only results in a single NACK coming back to the source. This can be achieved by assigning to each subtree A a *representative receiver* α with round trip delay d_α and timeout t_α such that every receiver $\beta \neq \alpha$ in this subtree fulfills condition (1).

Definition 1 *An* allocation ϕ *for a tree is a mapping of the set τ of subtrees (including the tree itself) onto the set ρ of receivers, such that for any subtree $A \in \tau$ with representative receiver $\alpha = \phi(A)$, all non-representative receivers $\beta \neq \alpha$ in A fulfill the single-NACK property with respect to α.*

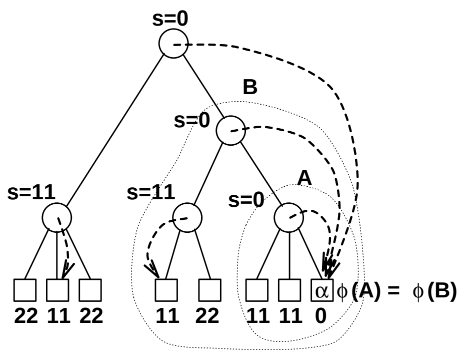

Figure 2: An example allocation.

Fig. 2 gives an example of an allocation. The round trip delay d_i has been set to 10 for all receivers i, and $\epsilon = 1$. A dashed arrow from a switch to a receiver means that this receiver is the representative receiver of the subtree rooted at this switch. The numbers below the receivers are the timeout values.

Note the following property that is formalized in the lemma below. Suppose a tree A is a subtree of a tree B. Furthermore, suppose that B's representative $\alpha = \phi(B)$ lies in A. All receivers in B, and in particular all receivers in A, must fulfill the single-NACK property with respect to α. Therefore, α is also A's representative.

Lemma 1 *If $\phi(B) = \alpha$, then any subtree A of B such that α is in A has $\phi(A) = \alpha$.*

Proof: This follows directly from the previous paragraph. □

We now give the algorithm that determines the timeout for each receiver in a tree A, given an allocation ϕ. We need the following definition.

Definition 2 *The* set of cotrees *associated with a receiver α and a tree A is the set of subtrees of A not containing α, that are also direct children of some node on the path from the sender to α.*

Consider Fig. 3 for an illustration. Basically, the idea behind cotrees is the following. Assume that we have already chosen timeouts within each cotree of α such that the single-NACK property is fulfilled. Then each cotree is a possible source of *one* unwanted NACK when A is the loss subtree and α is A's representative. By adding a constant offset s_B to each receiver's timeout within a cotree B, we can fulfill the single-NACK property with respect to α while maintaining the single-NACK property within B. This is the idea behind the algorithm below. Given an allocation ϕ, it recursively computes this offset s to be added to every timeout in a subtree (cf. Fig. 3.)

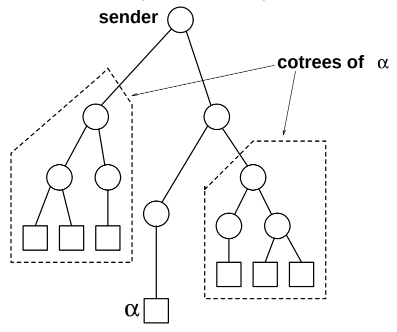

Figure 3: The cotrees of receiver α.

Algorithm 1: Computing timeouts for a given allocation ϕ

```
1   Initialize the timeout of each receiver to zero.
2   Let $A$ be the entire tree.
3   Set the *offset* $s_A = d_{\phi(A)} + \epsilon$.
4   Add $s_A$ to the timeout of all receivers $\in$ cotrees
    of $\phi(A)$.
5   For each nontrivial cotree[1] $A'$ of $\phi(A)$, set $A =$
    $A'$ and repeat step 1.
```

Step 1 ensures that if A is the loss subtree, then only A's representative $\phi(A)$ produces a NACK. Actually, it ensures the single-NACK condition for all subtrees of A containing α (cf. Lemma 1). The only loss subtrees left after this step for which the condition has not been enforced are the subtrees of the cotrees of α. Therefore, the process is repeated recursively for the cotrees.

4.2 Optimal Timeouts

Now that we are able to compute the timeouts given the allocation ϕ, i.e. given a representative receiver for each subtree, we need to establish a sensible cost function to compare allocations. For this, let us consider for a moment how a reliable end-to-end transport protocol based on acknowledgments (negative or positive) works. The goal of the protocol is to deliver data to the application error-free and in order. This means that the transport protocol layer buffers all data following a loss, until the lost data has been retransmitted. Only then can the data be handed over to the application. Now, in our framework, observe what happens to a receiver α in the worst case (the case where nobody else's NACK arrives at the sender earlier than α's) between the moment when the receiver realizes a packet has been lost, and the moment when this packet arrives at this receiver. The receiver runs its timer for time t_α, and then sends a NACK, which takes time d_α^N to arrive at the sender. The retransmitted packet then takes d_α^P to arrive at α. The total time is $t_\alpha + d_\alpha$.

It makes sense to minimize the largest $t_\alpha + d_\alpha$, both to minimize the buffering necessary in the transport protocol layer, and to minimize the delay seen by the application. Furthermore, the transport layer of the sender also has to perform buffering, to allow for the retransmission of lost packets. The sender can be sure that all receivers have received a packet if it has not received a NACK after $\max\{t_\alpha + d_\alpha\}$, where we are maximizing over all the receivers in the multicast tree[2].

We therefore want to minimize

$$T = \max_{\alpha \in \rho}\{t_\alpha + d_\alpha\} \tag{2}$$

where ρ is the set of all receivers in the multicast tree.

4.3 Computing an Optimal Allocation

Consider a subtree A with a root, n links and n children $A_1, \dots, A_n$ (cf. Fig. 4). Assume the alloca-

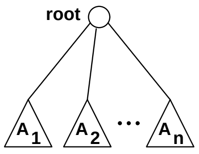

Figure 4: A subtree, consisting of a root, n links and n children.

tions inside the children are fixed, and that timeouts have been determined inside each child such that the single-NACK condition holds. Lemma 1 says that our only degree of freedom in choosing an allocation for A is choosing one of its children's representative as A's representative. If we choose the representative α_i of subtree A_i as the representative of tree A, then we have to add an offset $s_{A_i}[\phi]$ to all the timeouts of receivers in the other children $\{A_1, \dots, A_n\} \setminus \{A_i\}$. because these children are cotrees of α_i. The timeouts of the receivers in A_i remain unchanged.

We now make the following definition:

Definition 3 *For a subtree A and an allocation ϕ, the triple $(s_A[\phi], T_A[\phi], \phi)$ is given by*

$$\begin{aligned} s_A[\phi] &= d_{\phi(A)} + \epsilon \\ T_A[\phi] &= \max_{\alpha \in \rho_A}\{t_\alpha + d_\alpha\} \end{aligned} \tag{3}$$

where ρ_A is the set of receivers of subtree A and where the t_α have been computed using Algorithm 1 in subtree A.

For a subtree A, $T_A[\phi]$ is the cost function we are trying to minimize, and $s_A[\phi]$ is the offset that we have to add to A's siblings if we choose A's representative as A's parent's representative. We call the set of all triples $(s_A[\phi], T_A[\phi], \phi)$ for a tree A the *characteristic set* of A. Note from (3) that $s_A[\phi]$ and $T_A[\phi]$ only depend on the allocation for A and its subtrees. In other words, two different allocations that are equal within A would not give rise to two elements in the characteristic set. For ease of notation, we do not make this explicit. As an example, the following

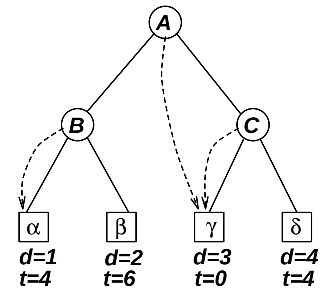

Figure 5: An example tree; an optimal allocation and the resulting timeouts with $\epsilon = 1$ are shown.

tables gives the characteristic sets of nodes A, B and C of the tree in Fig. 5. The optimal allocation shown in Fig. 5 is framed.

$s_B[\phi]$	$T_B[\phi]$	ϕ	t_α	t_β
2	4	$B \mapsto \alpha$	0	2
3	4	$B \mapsto \beta$	3	0

$s_C[\phi]$	$T_C[\phi]$	ϕ	t_γ	t_δ
4	8	$C \mapsto \gamma$	0	4
5	8	$C \mapsto \delta$	5	0

[2]provided the NACK and the retransmitted packet are not lost themselves.

$s_A[\phi]$	$T_A[\phi]$	ϕ	t_α	t_β	t_γ	t_δ
2	10	$A \mapsto \alpha,\ B \mapsto \alpha,\ C \mapsto \gamma$	0	2	2	6
2	10	$A \mapsto \alpha,\ B \mapsto \alpha,\ C \mapsto \delta$	0	2	7	2
3	11	$A \mapsto \beta,\ B \mapsto \beta,\ C \mapsto \gamma$	3	0	3	7
3	11	$A \mapsto \beta,\ B \mapsto \beta,\ C \mapsto \delta$	3	0	8	3
4	8	$A \mapsto \gamma,\ B \mapsto \alpha,\ C \mapsto \gamma$	4	6	0	4
4	8	$A \mapsto \gamma,\ B \mapsto \beta,\ C \mapsto \gamma$	7	4	0	4
5	9	$A \mapsto \delta,\ B \mapsto \alpha,\ C \mapsto \delta$	5	7	5	0
5	9	$A \mapsto \delta,\ B \mapsto \beta,\ C \mapsto \delta$	8	5	5	0

We now show that the characteristic set of each subtree can be computed from the characteristic sets of its children, and that the optimal allocation ϕ_{OPT} can be found this way. If $\phi(A_l) = \phi(A)$, in other words, if A_l's representative is also A's representative, then it follows from Definition 3 and Algorithm 1 that

$$\begin{aligned} s_A[\phi] &= s_{A_l}[\phi] \\ T_A[\phi] &= \max(T_{A_l}[\phi], \max_{i \in \{1,\dots,n\} \setminus \{l\}} \{s_{A_i}[\phi] + T_{A_i}[\phi]\}) \end{aligned} \quad (4)$$

If A is a trivial subtree (i.e. the subtree contains only a receiver α), then α is A's representative. The single corresponding element $(s_A[\phi], T_A[\phi], \phi)$ in A's trivial characteristic set is given as follows.

$$\begin{aligned} s_A[\phi] &= d_\alpha + \epsilon \\ T_A[\phi] &= d_\alpha \\ \phi(A) &= \alpha \end{aligned} \quad (5)$$

This follows readily from Definition 3.

The previous observations can now be used to devise a recursive algorithm that finds an optimal allocation ϕ_{OPT} that minimizes $T_A[\phi]$. Any non-terminal node (switch or sender) in the tree can compute its characteristic set based solely on the characteristic sets of its children using (4). This can be done for each node in the tree in a recursive manner, until the root S of the tree (the sender) knows its characteristic set. The optimal allocation ϕ_{OPT} is then simply the one minimizing $T_S[.]$. To find each receiver's timeout, Algorithm 1 can then be applied.

Let us look at the size of A's characteristic set as a function of the sizes of the children's characteristic sets. Call A_i's set size m_i. From (4) we see that we have n choices for placing A's representative in one of A's children. Furthermore, in each child A_i, we can choose among m_i allocations. Thus, the size of A's characteristic set is

$$m = n \cdot \prod_{i=1}^{n} m_i. \quad (6)$$

Thus, this algorithm suffers from the combinatorial explosion of the size of the characteristic set, as expressed by (6). In fact, this algorithm does nothing else than explicitly enumerate all possible allocations for the entire tree and compute the cost for each of these allocations. This, of course, becomes rapidly impractical as the tree size increases. In the remainder of this section, we show that only a small subset of the characteristic set, which we refer to as the *constrained characteristic set* (CCS), is necessary at each node in the computation of ϕ_{OPT}, and we find a tight upper bound on the size of this set.

Lemma 2 *Consider two elements $(s_A[\phi], T_A[\phi], \phi)$ and $(s_A[\psi], T_A[\psi], \psi)$ in the characteristic set of a subtree A, where ϕ and ψ differ in A. Assume that $s_A[\phi] \geq s_A[\psi]$ and $T_A[\phi] \geq T_A[\psi]$. Then an optimal allocation ϕ_{OPT} can be found without considering $(s_A[\phi], T_A[\phi], \phi)$ in the computation of the characteristic set of A's parent*[3].

Proof: Assume that ϕ and ψ are identical outside A, i.e. that for any X not subtree of A, we have either $\phi(X) = \psi(X) \notin \rho_A$ (the set of A's receivers), or both $\phi(X) \in A$ and $\psi(X) \in A$ (but possibly different). This poses no restriction on the proof, as the elements of A's characteristic set only depend on the allocation within A, and such a pair of allocations always exists. Let B be the parent subtree of A.

Each element in a subtree's characteristic set corresponds to an allocation in this subtree. Call $(s_B[\phi], T_B[\phi], \phi)$ and $(s_B[\psi], T_B[\psi], \psi)$ the elements of B's characteristic set resulting from (4).

Consider the two cases where (a) B's representative is in A, and (b) where it is not. In case (a), it follows from Definition 3 and the assumption in Lemma 2 that

$$s_B[\phi] = s_A[\phi] \geq s_A[\psi] = s_B[\psi]. \quad (7)$$

Thus,

$$\begin{aligned} T_B[\phi] &= \max(T_A[\phi], \max_{X \in c(B) \setminus \{A\}} \{s_A[\phi] + T_X[\phi]\}) \\ &\geq \max(T_A[\psi], \max_{X \in c(B) \setminus \{A\}} \{s_A[\psi] + T_X[\psi]\}) \\ &= T_B[\psi]. \end{aligned} \quad (8)$$

because $T_A[\phi] \geq T_A[\psi]$ by assumption and $T_X[\phi] = T_X[\psi]$ because X is outside A (i.e. X is not subtree of A), and where $c(B)$ represents the set of B's children.

In case (b), denote with Y B's child in which B's representative receiver $\phi(B) = \psi(B)$ lies. Then

$$s_B[\phi] = s_Y[\phi] = s_Y[\psi] = s_B[\psi]. \quad (9)$$

The fact that X and Y are outside A implies that

$$\begin{aligned} T_B[\phi] &= \max(T_Y[\phi], \max_{X \in c(B) \setminus \{Y, A\}} \{s_Y[\phi] + T_X[\phi]\}, \\ &\quad s_Y[\phi] + T_A[\phi]) \\ &\geq \max(T_Y[\psi], \max_{X \in c(B) \setminus \{Y, A\}} \{s_Y[\psi] + T_X[\psi]\}, \\ &\quad s_Y[\psi] + T_A[\psi]) \\ &= T_B[\psi]. \end{aligned} \quad (10)$$

It follows that the same property that holds for $(s_A[\phi], T_A[\phi], \phi)$ and $(s_A[\psi], T_A[\psi], \psi)$ holds for the two elements of B's characteristic set as well, namely

$$\begin{aligned} s_B[\phi] &\geq s_B[\psi] \\ T_B[\phi] &\geq T_B[\psi] \end{aligned}$$

[3] In the special case of two equalities, we need to consider one of the two elements, but not both.

By induction, it holds for two elements of the root S's characteristic set. As we need to find the element $(s_S[\phi], T_S[\phi], \phi)$ that minimizes $T_S[\phi]$, it follows that there cannot be a single optimal allocation ϕ as there always exists another allocation ψ that is at least as good, i.e. has an equal or lower T_S. Therefore, to find *some* ϕ_{OPT} through (4), it is not necessary to consider $(s_A[\phi], T_A[\phi], \phi)$. This completes the proof. □

The next lemma states that there exists a global optimal such that if B is any subtree and A is a child of B, then the allocation in A is locally optimal (i.e. it minimizes $T_A[\phi]$) if B's representative $\phi(B)$ is *not* in A.

Lemma 3 *Assume a subtree A with n children $A_1, \ldots, A_n$, and assume that A_i's representative $\phi(A_i)$ is selected as A's representative. Then in order to find a globally optimal allocation, it is only necessary to consider the elements $(s_{A_j}[\phi], T_{A_j}[\phi], \phi)$ of the characteristic set of A_j, $j \in \{1, \ldots, n\} \setminus \{i\}$ that minimize $T_{A_j}[\phi]$.*

Proof: As A's representative is not in A_j, $s_A[\phi] = s_{A_i}[\phi]$ is independent of the choice of allocation in A_j. Also,

$$T_A[\phi] = \max(T_{A_i}[\phi], \max_{j \in \{1,\ldots,n\} \setminus \{i\}} \{s_{A_i}[\phi] + T_{A_j}[\phi]\}) \tag{11}$$

shows that selecting an allocation ϕ that does not minimize $T_{A_j}[\phi]$ can only increase $T_A[\phi]$. Thus, selecting a ϕ that does not minimize $T_{A_j}[\phi]$ results in elements in A's characteristic set that need not be considered by virtue of Lemma 2. Therefore, if A's representative is not in A_j, we only have to consider elements of A_j's characteristic set that has minimal $T_{A_j}[\phi]$. □

Lemma 2 and Lemma 3 increase the efficiency of the recursive algorithm to find ϕ_{OPT}. Lemma 2 limits the size of the characteristic set at each node, and Lemma 3 makes the computation of a node's characteristic set from its children's characteristic sets more efficient, by limiting the number of combinations that have to be considered. We call the subset of the characteristic set that needs to be retained at each node the *constrained characteristic set (CCS).*

We now prove a theorem that gives a small and tight upper bound on the size of the constrained characteristic set if elements are deleted at each node according to Lemma 2.

Theorem 1 *Let h denote the height of a node, defined as the number of links on the longest path from this node to some leaf in the subtree rooted at this node*[4]*. If elements of the characteristic set of a node are deleted according to Lemma 2, then the size of this set is at most* $\max(h, 1)$.

Proof: The proof makes use of the two following lemmas.

[4]For example, the tree shown in Fig. 2 is of height 3.

Lemma 4 *A trivial tree of height* 0 *(containing just a receiver) has a constrained characteristic set of size one.*

Proof: This follows readily from the fact that there is only one possible allocation in a trivial tree, as noted previously (cf. Eqn. (5)). □

Lemma 5 *A tree of height* 1 *has a constrained characteristic set of size one.*

Proof: A tree A of height 1 consist of a switch and n receivers directly attached to this switch. We can assume w.l.g. that $d_{\alpha_1} < d_{\alpha_2} < \ldots < d_{\alpha_n}$. If two round-trip delays were equal, allocating one or the other is equivalent, and it is therefore enough to consider one of them.

There are n elements in the characteristic set of this tree, corresponding to selecting one of the n receivers as representative. If receiver i is chosen as representative, then the element of the characteristic set, denoted by $(s_A[\phi_i], T_A[\phi_i], \phi_i)$ can be computed from (5) and (4).

$$\begin{aligned} s_A[\phi_i] &= d_{\alpha_i} + \epsilon \\ T_A[\phi_i] &= \max(T_A[\phi_i], \max_{j \in \{1,\ldots,n\} \setminus \{i\}} \{s_A[\phi_i] + T_A[\phi_j]\}) \\ &= d_{\alpha_i} + \epsilon + \max_{j \in \{1,\ldots,n\} \setminus \{i\}} \{d_{\alpha_j}\} \end{aligned} \tag{12}$$

Therefore, $s_A[\phi_1] < s_A[\phi_2] < \ldots < s_A[\phi_n]$. Also, as $\max_{j \in \{1,\ldots,n\} \setminus \{i\}} \{d_{\alpha_j}\} = d_{\alpha_n}$ whenever $i < n$, it follows that $T_A[\phi_1] < T_A[\phi_2] < \ldots < T_A[\phi_{n-1}]$. As for $T_A[\phi_n]$,

$$T_A[\phi_n] = d_{\alpha_n} + \epsilon + d_{\alpha_{n-1}} \geq T_A[\phi_1] = d_{\alpha_1} + \epsilon + d_{\alpha_1} \tag{13}$$

In particular, equality only results for $n = 2$. Therefore, there is only the element corresponding to allocating receiver α_1 with smallest delay as A's representative in CCS$_A$. □

We have proved the assertion for the cases where $h = 0$ and $h = 1$. We are now ready to complete the proof for any h. For this, consider a subtree A and denote the receiver in A with the smallest round-trip time with $\alpha_{\min}$ (assuming it is unique.) Denote the cotrees of $\alpha_{\min}$ in A with $A_1, \ldots, A_n$, as shown in Fig. 6. We prove the theorem in two steps. First, we show that there is exactly one element in CCS$_A$ corresponding to allocating $\alpha_{\min}$ as A's representative. Second, we show that all other possible elements in CCS$_A$ correspond to selecting A's representative in one particular cotree of $\alpha_{\min}$. This results in the desired bound on the size of CCS$_A$.

To show that there is exactly one element in CCS$_A$ corresponding to selecting $\alpha_{\min}$ as A's representative, note that this choice corresponds to an element of CCS$_A$ $(s_A[\phi], T_A[\phi], \phi)$ that minimizes $s_A[\phi] = d_{\alpha_{\min}} + \epsilon$, by definition of $\alpha_{\min}$. Among all allocations that have $\alpha_{\min}$ as A's representative, by virtue of Lemma 3, only one allocation minimizing $T_A[\phi]$ corresponds to an element in CCS$_A$.

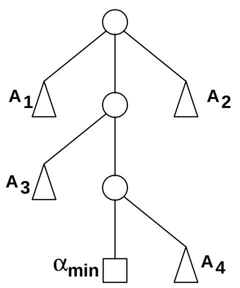

Figure 6: The cotrees of receiver $\alpha_{\min}$ with the lowest round-trip delay in the entire tree.

To bound the number of other elements in CCS_A, call the minimal $T_{A_i}[.]$ in cotree A_i $T_{A_i}^{OPT}$. Let $T_{DC}^{OPT} = \max_{i=1,\ldots,n}\{T_{A_i}^{OPT}\}$, and the corresponding cotree the *dominant cotree DC* (assuming, for the moment, that it is unique). This is the cotree with the highest cost for its local optimum.

Lemma 3 states that if $\alpha_{\min}$ is A's representative, then a necessary condition for an allocation to be in CCS_A is that it corresponds to the locally optimal allocation in each cotree of $\alpha_{\min}$. Therefore, $T_A[\phi] = s_A[\phi] + T_{DC}^{OPT}$.

Let ψ be another allocation such that A's representative $\alpha = \psi(A) \neq \alpha_{\min}$ is not in the dominant cotree DC. It follows that DC is either a cotree of α, or it is a subtree of a cotree of α. For the single-NACK property to hold with respect to α, the offset $s_A[\psi] = d_\alpha + \epsilon$ has to be added to all the timeouts in α's cotree, and in particular, to the timeouts in DC. Then $T_A[\psi]$ must be at least $s_A[\psi] + T_{DC}^{OPT}$, as by definition T_{DC}^{OPT} minimizes $T_{DC}[.]$. As $s_A[\psi] > s_A[\phi]$ and therefore $T_A[\psi] > T_A[\phi]$, ψ is not in A's CCS (Lemma 2).

Thus, A's CCS has always exactly one element corresponding to $\alpha_{\min}$ as A's representative, and possibly other elements corresponding to allocations that place A's representative α in $\alpha_{\min}$'s dominant cotree DC. For each such choice of α in DC, the same argument as before can be used to show that there is only one allocation ψ possible in A's CCS: α uniquely determines $s_A[\psi]$, and thus only the element minimizing $T_A[\psi]$ survives.

It follows that size (A's CCS) $\leq$ size (dominant cotree's CCS) +1. Also, height $(A) \geq$ height (dominant cotree)+1. Therefore, by induction anchored on Lemma 5, the assertion is proved.

If $\alpha_{\min}$'s dominant cotree DC is not unique, i.e. $T_{DC_1}^{OPT} = T_{DC_2}^{OPT} = T_{DC}^{OPT}$, then A's CCS has only one element, corresponding to $\phi(A) = \alpha_{\min}$. To see this, assume there are two dominant cotrees, DC_1 and DC_2. Let ψ be an allocation such that $\psi(A)$ is in DC_1. Due to DC_2, $T_A[\psi]$ must be at least $s_A[\psi] + T_{DC}^{OPT}$. Therefore, there is no element in A's CCS corresponding to ψ. This is easily extended to more than two dominant cotrees. This completes the proof. □

5 Distributed Optimal Timeout Computation

Each element of a node's CCS corresponds to a "candidate allocation", i.e. an allocation that *might* be globally optimal. Suppose that a node that has computed its own CCS from its children's CCSs does not know the complete allocation function ϕ corresponding to each element of its CCS, but only (a) in what child lies its own representative receiver, and (b) which among the possible candidate allocations should be chosen in each child. Then it can be seen that any node's representative receiver for some allocation in this node's CCS can be determined in the following way: start at this node and note which is the candidate allocation in the child containing the representative receiver. Go to this child node, which knows in which of its children is its representative receiver for this candidate application. Do this recursively until you reach a receiver.

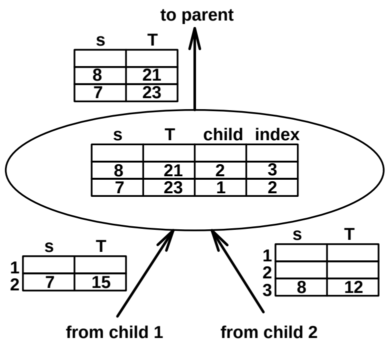

Figure 7: The topology information associated with the allocations in a node's CCS is distributed.

Furthermore, Lemma 3 states that we need to know (b) above actually only for the child containing the representative receiver, because the other children's allocation is locally optimal. Thus, instead of the full allocation function ϕ, we only need to associate with each CCS element (a) a pointer to the child containing the representative receiver, and (b) an index into that child's CCS. Fig. 7 illustrates this: the node receives two CCSs from its two children, and computes its own CCS. Internally, it maintains the pointer to the representative child and this child's CCS index. It sends the CCS to the parent. In the end, the root knows its own CCS. It can then pick the element (s_{OPT}, T_{OPT}) that has the lowest cost function T.

The downstream part of the algorithm computes the actual timeouts. Each node receives an index into its CCS and an offset from its parent. Call the offset S and the selected CCS element $(s, T, \mathit{child}, \mathit{index})$. Each node now sends to its representative child *child* the offset S and *index*. It sends to all of its other children as offset $S + s$, and as index the one corresponding to the child's local optimum (cf. Lemma 3). Finally, the receivers set their timeouts to the offset value they receive from their parent.

The following example illustrates the algorithm presented in the previous section. The topology of the

Node	child CCS	own CCS
B	$\mathrm{CCS}_\alpha = \{(2,1)\}$ $\mathrm{CCS}_\beta = \{(3,2)\}$	$\{(2,4,\alpha,1)\}$
C	$\mathrm{CCS}_\gamma = \{(4,3)\}$ $\mathrm{CCS}_\delta = \{(5,4)\}$	$\{(4,8,\gamma,1)\}$
A	$\mathrm{CCS}_B = \{(2,4)\}$ $\mathrm{CCS}_C = \{(4,8)\}$	$\{(2,10,B,1),\underline{(4,8,C,1)}\}$

Table 1: The upstream part of the distributed timeout computation.

Node	child	index	offset
A	B	1	4
	C	1	0
B	α	1	4
	β	1	6
C	γ	1	0
	δ	1	4

Table 2: The downstream part of the computation.

network is depicted in Fig. 5. Table 1 shows the computation each node performs in the upstream part of the protocol. The element chosen in the root's CCS as the best is underlined. Compare this to A's full characteristic set in Section 4.3.

Table 2 shows the downstream part of the protocol. The resulting timeouts are reported in Fig. 5. It is interesting to note that the receiver with the smallest round-trip delay is not the tree's representative in the optimal allocation.

6 Discussion

The algorithm shown in the previous section computes an optimal set of timeouts. Theorem 1 bounds the size of the upstream messages to the height of a node, which is $O(\log n)$ in a balanced tree. For example, assume that a multicast group has 10000 participants, and that each switch only has two children. Then the constrained characteristic set of the root is at most of length $\lceil \log_2(10000) \rceil = 14$. Note that no node in the tree has to know the entire topology, and that communication is local: a switch only communicates with its parent and its children, and receivers only communicate with their parents.

Let us briefly look at what happens when the unique NACK gets lost. If this occurs, then every cotree of the global representative (the entire tree's representative) produces one NACK. Assuming a balanced multicast tree of outdegree m, there are at most $m-1$ cotrees at each level. Thus, the number of additional NACKs produced would be at most $(m-1)\lceil \log_m(n) \rceil$, where n is the number of receivers. In the example above, this number would be 14. In other words, a NACK loss results in a reasonable number of additional NACKs, placing only a small burden on the sender.

We have so far talked about a single sender and multiple receivers. In a multicast group with multiple senders, each receiver would need to know its round trip delay to each of the senders (as in [3]). In shared-tree group multicast (e.g. [7]), the core might take the role of the retransmitter, which would improve scalability by only requiring a single timeout at each receiver despite multiple senders.

7 Conclusion and Future Work

We have presented a rigorous solution to the NACK-implosion problem in reliable multicast. A scalable algorithm to compute optimal timeouts has been derived. This algorithm has several advantages. First, the computation of the timeouts is entirely distributed. No node in the tree (sender, switches, receivers) needs to maintain global information, and communication to compute timeouts is local between neighbors in the tree. The protocol messages remain small even for very large multicast groups. Second, the single-NACK property is deterministic, i.e. we are guaranteed that only one NACK results from a single loss, provided that the NACK and the retransmitted packet are not lost themselves. Third, this algorithm computes a set of timeouts that is optimal with regard to the window size of the transport protocol layer. This maximizes the efficiency of the multicast service by minimizing the buffering requirements in the transport layer, and by minimizing the delays seen by the application. Fourth, switches don't need to process or merge NACK packets – they simply forward them to the sender. This is compliant with the ATM paradigm of end-to-end segmentation and reassembly. Also, switches don't need to do any intermediate buffering of data on behalf of the multicast session.

Acknowledgements

I am indebted to S. Keshav for suggesting this study and many stimulating discussions. My thanks to the anonymous reviewers for many helpful comments. Discussions with J.-C. Bolot, C. R. Kalmanek, C. Lund, S. Phillips and K. K. Ramakrishnan were a valuable contribution to this work.

References

[1] S. Pingali, D. Towsley, and J. F. Kurose, "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols," in *ACM SIGMETRICS '94*, pp. 221–230, May 1995.

[2] W. T. Strayer, B. J. Dempsey, and A. C. Weaver, *XTP: The Express Transfer Protocol.* Addison-Wesley (Reading, Mass.), 1992.

[3] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," in *ACM SIGCOMM '95*, August 1995.

[4] C. Papadopoulos and G. Parulkar, "Implosion Control for Multipoint Applications," in *Proc. 10th IEEE Workshop on Computer Communications*, (Seattle), September 1995.

[5] S. Paul, K. Sabnani, and D. Kristol, "Multicast Transport Protocols for High-Speed Networks," in *Proc. International Conference on Network Protocols*, pp. 4–14, 1994.

[6] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton, "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation," in *Proc. ACM SIGCOMM '95*, pp. 328–341, September 1995.

[7] T. Ballardie, P. Francis, and J. Crowcroft, "Core Based Trees (CBT)," in *Proc. ACM SIGCOMM '93*, (San Francisco, Calif.), September 1993.