Designing Near-Optimal Key Sharing Hierarchies

June 25, 2008

1 Balanced routing tree with uniform cost

In this section, we are going to find good logical key trees for balanced routing trees with uniform cost. Consider the routing tree in Figure 1, where each internal node in the routing tree has d children and the height of the tree is h. So there are d^h leaves in this routing tree; we treat these leaves as group members.

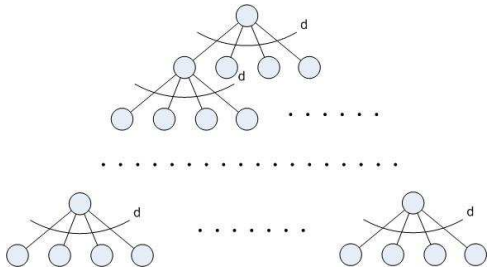

Figure 1: A d-ary routing tree

First try is, we could use this routing tree as a logical tree, which is shown in Figure 2. Then the cost of updating a single member can be calculated as follows:

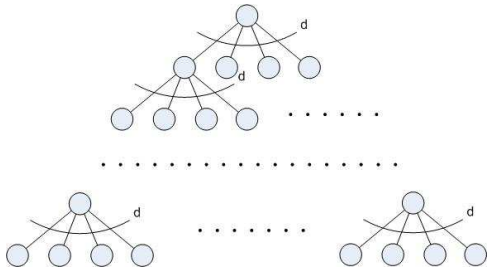

Figure 2: Use d-ary routing tree as logic tree

$$
\begin{aligned}
& dh + d(h-1+d) + d(h-2+d+d^2) + \cdots + d(1+d+d^2+\cdots+d^{h-1}) \\
= \;& d\left[\sum_{i=1}^{k} i + d + (d+d^2) + \cdots + (d+d^2+\cdots+d^{h-1})\right] \\
= \;& d\left[\frac{h(h+1)}{2} + \sum_{i=2}^{h} \frac{d^i - d}{d-1}\right] \\
= \;& \frac{dh(h+1)}{2} + \frac{d}{d-1}\sum_{i=2}^{h} d^i - d\sum_{i=2}^{h}\frac{d}{d-1} \\
= \;& \frac{dh(h+1)}{2} + \frac{d}{d-1}\frac{d^{h+1}-1}{d-1} - \frac{d^2(h-1)}{d-1} \\
= \;& O\left(d^h\right)
\end{aligned}
$$

Notice that the cost is already near optimal (constant factor away). Now we try 2-level logical trees, to see their performance. The first 2-level logical tree we consider is shown in Figure 3, where the root has d children and each of the other internal nodes has d^{h-1} children.

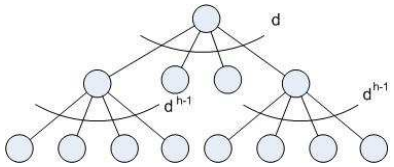

Figure 3: 2-level logical tree

The cost of updating a single member can be calculated as follows:

$$
\begin{aligned}
& hd^{h-1} + d(1+d+d^2+\cdots+d^{h-1}) \\
= \;& hd^{h-1} + d\frac{d^h-1}{d-1} \\
= \;& O\left((h+d)d^{h-1}\right)
\end{aligned}
$$

The second 2-level logical tree we consider is shown in Figure 4, where the root has $d^{h/2}$ children and each of the other internal nodes also has $d^{h/2}$ children.

The cost of updating a single member can be calculated as follows:

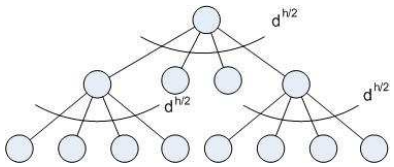

Figure 4: 2-level logical tree

$$\begin{aligned} & hd^{h/2} + d^{h/2}(\frac{h}{2} + d + d^2 + \cdots + d^{h/2}) \\ = \; & hd^{h/2} + \frac{h}{2}d^{h/2} + d^{h/2}\frac{d^{h/2+1} - d}{d-1} \\ \approx \; & \frac{3h}{2}d^{h/2} + d^h \\ = \; & O\left(d^h\right) \end{aligned}$$

The last 2-level logical tree we consider is shown in Figure 5, where the root has d^{h-1} children and each of the other internal nodes has d children.

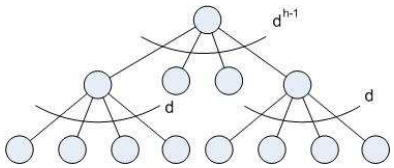

Figure 5: 2-level logical tree

The cost of updating a single member can be calculated as follows:

$$\begin{aligned} & dh + d^{h-1}(h - 1 + d) \\ = \; & dh + (h-1)d^{h-1} + d^h \\ \approx \; & hd^{h-1} + d^h \\ = \; & O\left((h+d)d^{h-1}\right) \end{aligned}$$

From the above calculation, we can see the second 2-level logical tree performs the best. In the following, we generalized this kind of 2-level logical tree, as shown in Figure 6, where the root has d^k children and each of the other internal nodes has d^{h-k} children.

The cost of updating a single member can be calculated as follows:

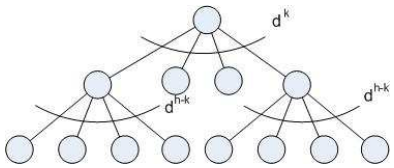

Figure 6: 2-level logical tree

$$\begin{aligned} & hd^{h-k} + d^k(k + d + d^2 + \cdots + d^k) \\ = \; & hd^{h-k} + d^k k + d^k \frac{d^{h-k+1} - d}{d-1} \\ = \; & hd^{h-k} + d^k \left(k - \frac{d}{d-1}\right) + \frac{d^{h+1}}{d-1} \end{aligned}$$

It is not hard to see from this formula, that when $k = h/2$ it achieves the minimum value. Now we start looking at 3-level logical trees. The first one we considered is shown in Figure 7, where the root has $d^{h/2}$ children and each of the other internal nodes has $d^{h/4}$ children.

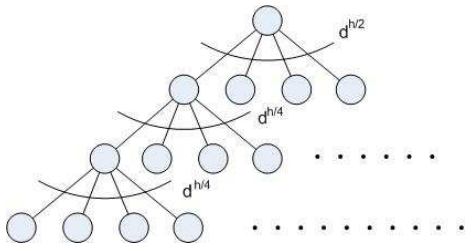

Figure 7: 3-level logical tree

The cost of updating a single member can be calculated as follows:

$$\begin{aligned} & hd^{h/4} + d^{h/4}\left(\frac{3h}{4} + d + d^2 + \cdots + d^{h/4}\right) + d^{h/2}\left(\frac{h}{2} + d + d^2 + \cdots + d^{h/2}\right) \\ = \; & hd^{h/4} + \frac{3}{4}hd^{h/4} + d^{h/4}\frac{d^{h/4+1} - d}{d-1} + \frac{1}{2}hd^{h/2} + d^{h/2}\frac{d^{h/2+1} - d}{d-1} \\ \approx \; & d^h + \frac{h}{2}d^{h/2} + \frac{7}{4}hd^{h/4} \\ = \; & O\left(d^h\right) \end{aligned}$$

The next 3-level logical tree we considered is shown in Figure 8, where every internal node has $d^{h/3}$ children.

The cost of updating a single member can be calculated as follows:

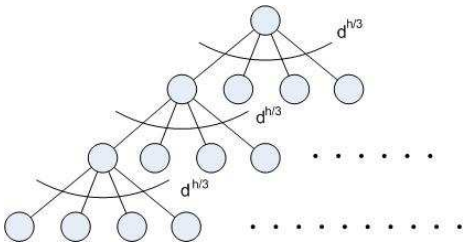

Figure 8: 3-level logical tree

$$
\begin{aligned}
& hd^{h/3} + d^{h/3}\left(\frac{2h}{3} + d + d^2 + \cdots + d^{h/3}\right) + d^{h/3}\left(\frac{h}{3} + d + d^2 + \cdots + d^{2h/3}\right) \\
= \;& hd^{h/3} + \frac{2h}{3}d^{h/3} + d^{h/3}\frac{d^{h/3+1} - d}{d-1} + \frac{h}{3}d^{h/3} + d^{h/3}\frac{d^{2h/3+1} - d}{d-1} \\
\approx \;& d^h + d^{2h/3} + 2hd^{h/3} \\
= \;& O\left(d^h\right)
\end{aligned}
$$

As we can see, both 3-level logical tree are near optimal (constant factor away). But the second leading term in the first 3-level logical tree is smaller the the one in the second 3-level logical tree. Figure 9 shows the generalized 3-level logical tree of this kind.

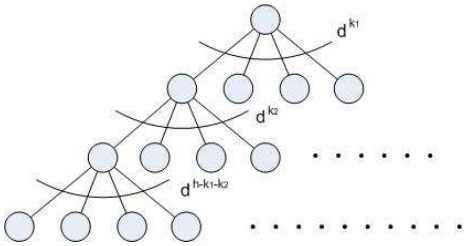

Figure 9: 3-level logical tree

The cost of updating a single member can be calculated as follows:

$$
\begin{aligned}
& hd^{h-k_1-k_2} + d^{k_2}\left(k_1 + k_2 + d + d^2 + \cdots + d^{h-k_1-k_2}\right) + d^{k_1}\left(k_1 + d + d^2 + \cdots + d^{h-k_1}\right) \\
= \;& hd^{h-k_1-k_2} + (k_1+k_2)d^{k_2} + d^{k_2}\frac{d^{h-k_1-k_2+1} - d}{d-1} + k_1 d^{k_1} + d^{k_1}\frac{d^{h-k_1+1} - d}{d-1} \\
\approx \;& d^h + (k_1 - 1)d^{k_1} + d^{h-k_1} + (k_1 + k_2 - 1)d^{k_2} + hd^{h-k_1-k_2}
\end{aligned}
$$

From the formula we can see, when $k_1 = h/2$ and $k_2 = h/4$ it achieves minimum.

Both 2-level logical tree and 3-level logical tree are near optimal. So one thought might be balanced logic trees of this kind are always near optimal. Figure 10 shows a generalized m-level logic tree.

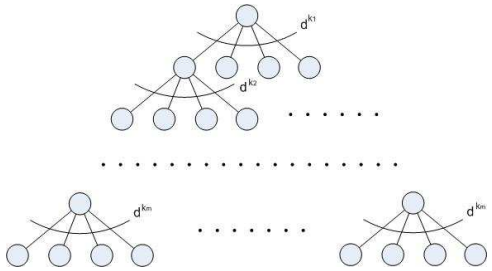

Figure 10: 3-level logical tree

The cost of updating a single member can be calculated as follows:

$$
\begin{aligned}
& hd^{k_m} + d^{k_{m-1}}\left(k_1 + k_2 + \cdots + k_{m-1} + d + d^2 + \cdots + d^{k_m}\right) + \ldots \\
& + d^{k_1}\left(k_1 + d + d^2 + \cdots + d^{h-k_1}\right) \\
= \;& hd^{k_m} + d^{k_{m-1}} \sum_{i=1}^{m-1} k_i + d^{k_{m-1}} \frac{d^{k_m+1} - d}{d-1} + d^{k_{m-2}} \sum_{i=1}^{m-2} k_i + d^{k_{m-2}} \frac{d^{k_m+k_{m-1}+1} - d}{d-1} \\
& + \cdots + d^{k_1} \sum_{i=1}^{1} k_i + d^{k_1} \frac{d^{k_2+k_3+\cdots+k_m+1} - d}{d-1} \\
\approx \;& hd^{k_m} + d^{k_{m-1}} \sum_{i=1}^{m-1} k_i + d^{k_{m-2}} \sum_{i=1}^{m-2} k_i + \cdots + d^{k_1} \sum_{i=1}^{1} k_i + d^{k_{m-1}+k_m} \\
& + d^{k_{m-2}+k_{m-1}+k_m} + \cdots + d^{k_2+k_3+\cdots+k_m} + d^h
\end{aligned}
$$

As we can see from the formula that when $k_1 = h/2$, $k_2 = h/4$, ... , it achieves minimum value. This means $O(\log h)$-level logic tree is the best among this kind of logical trees. However, these logical tree are all near optimal. So the balanced routing tree with uniform cost is not an interesting case.

2 Very unbalanced routing tree with uniform cost

As we have seen in last section that for balanced routing tree with uniform cost is very easy to find a near optimal logical tree. Actually, even using the routing tree itself as logical tree is near optimal. In this section, we analyze a very unbalanced routing tree, as shown in Figure 11. In this case, if we use routing tree as logical tree, the cost is far away from optimal. Then we give a way to construct logical tree which improves the cost a lot.

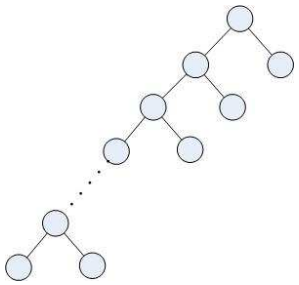

Figure 11: A very unbalanced routing tree

The first logical tree we considered is the same as routing tree, which is shown in Figure 12. Let $c(i)$ represent the update cost when leaf i requests update. Then the cost of this logical tree can be calculated as follows:

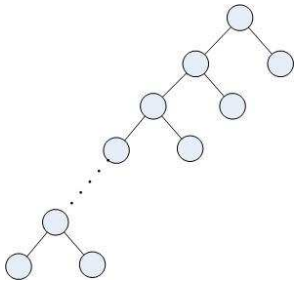

Figure 12: Using routing tree as logical tree

$$
\begin{aligned}
c(1) &= 2n + [(n-1) + (n-1) + 2] + [(n-2) + (n-2) + 2 \times 2] + \cdots + [1 + 1 + 2(n-1)] \\
&= 2n^2 \\
c(2) &= [(n-1) + (n-1) + 2] + [(n-2) + (n-2) + 2 \times 2] + \cdots + [1 + 1 + 2(n-1)] \\
&= 2n(n-1) \\
c(3) &= 2n(n-2) \\
&\cdots \\
c(n) &= 2n \\
cost &= \sum_{i=1}^{n} c(i) = O\left(n^3\right)
\end{aligned}
$$

The next logical tree we considered is 2-level logical tree, which is shown in Figure 13. We group every c members together. There are n/c groups. So the root of logic tree has n/c children and each of the other internal nodes has c children.

Let's assume the first group in the logical tree is a set of "continuous"

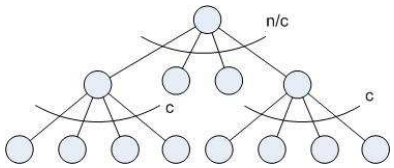

Figure 13: 2-level logical tree

leaves at the bottom of the routing tree. Then the cost of updating a single member in this group can be calculated as follows:

$$
\begin{aligned}
& n + (n-1) + (n-2) + \cdots + (n-c+1) + (n-c+2c) + (n-2c+2c) + \cdots + 2c \\
= \;& cn - \frac{c(c-1)}{2} + n\frac{n}{c} - c\left(1 + 2 + \cdots + \frac{n}{c}\right) + 2c\frac{n}{c} \\
\approx \;& cn - \frac{c^2}{2} + \frac{n^2}{c} - c\frac{n}{c}\frac{n}{c}\frac{1}{2} + 2n \\
= \;& cn - \frac{c^2}{2} + \frac{n^2}{2c} + 2n \\
\approx \;& cn + \frac{n^2}{c}
\end{aligned}
$$

As we see from the formula, when $c = \sqrt{n}$ it achieves minimum, $O\left(n^{1.5}\right)$. Since this group is at the bottom of routing tree, its updating cost is the worst case among all the groups. So the total update cost of this tree is $O\left(n^{2.5}\right)$.

Now we are going to generalize this kind of logic tree. Consider an m-level logic tree which is shown in Figure 14, where each internal node has $n^{1/m}$ children.

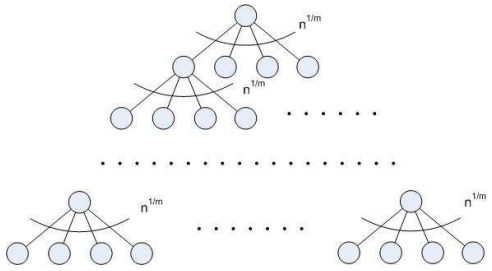

Figure 14: m-level logical tree

Similarly, let's assume the first group in the logical tree is a set of "continuous" leaves at the bottom of the routing tree. Then the cost of updating a single member in this group can be calculated as follows:

$$
\begin{aligned}
& n + (n-1) + \cdots + \left(n - n^{1/m} + 1\right) \\
& + \left(n - n^{1/m} + 2n^{1/m}\right) + \left(n - 2n^{1/m} + 2n^{1/m}\right) + \cdots + \left(n - n^{1/m}n^{1/m} + 2n^{1/m}\right) \\
& + \left(n - n^{2/m} + 2n^{2/m}\right) + \left(n - 2n^{2/m} + 2n^{2/m}\right) + \cdots + \left(n - n^{1/m}n^{2/m} + 2n^{2/m}\right) \\
& + \ldots \\
& + \left(n - n^{\frac{m-1}{m}} + 2n^{\frac{m-1}{m}}\right) + \left(n - 2n^{\frac{m-1}{m}} + 2n^{\frac{m-1}{m}}\right) + \cdots + \left(n - n^{1/m}n^{\frac{m-1}{m}} + 2n^{\frac{m-1}{m}}\right) \\
\approx \quad & \left[nn^{1/m} - \frac{1}{2}n^{2/m}\right] + \left[nn^{1/m} - \frac{1}{2}n^{1/m}n^{2/m} + 2n^{1/m}n^{1/m}\right] + \ldots \\
& + \left[nn^{1/m} - \frac{1}{2}n^{\frac{m-1}{m}}n^{2/m} + 2n^{\frac{m-1}{m}}n^{1/m}\right] \\
\approx \quad & n^{\frac{m+1}{m}} + n^{\frac{m+1}{m}} + \cdots + n^{\frac{m+1}{m}} \\
= \quad & mn^{\frac{m+1}{m}}
\end{aligned}
$$

Since this group is at the bottom of routing tree, its updating cost is the worst case among all the groups. So the total update cost of this tree is $n^{\frac{m-1}{m}} \cdot mn^{\frac{m+1}{m}} \cdot n^{1/m} = O\left(mn^{2+1/m}\right)$. According to this formula, when $m = 3$ the cost is $O\left(n^{7/3}\right)$, when $m = 4$ the cost is $O\left(n^{9/4}\right)$ ...

When the value of m keeps increasing, the logical tree eventually becomes a balance binary tree, which is shown in Figure 15. For each member in this logic tree, it needs to update $\log n$ keys and the cost of updating each key is $O(n)$. So the cost of updating each member is $O\left(n \log n\right)$. Thus, the total update cost is $O\left(n^2 \log n\right)$. And we can show this is the lower bound for all the logical trees. The argument makes use of the theorem that we are going to prove in section 3, "There is always a binary logical tree that is within twice the optimal one". So if we can prove $O\left(n^2 \log n\right)$ is the lower bound for binary logical trees, then we can conclude this is actually the lower bound for all logical trees. In the routing tree shown in Figure 11, there are $n/2$ leaves that are at least $n/2$ away from the root. So in any binary logical tree, there are at least $n/4$ intermedia nodes that have this kind of leaves as children. Notice the multicast cost of any of these intermedia node is $\Omega(n)$. Also at least half of those intermedia nodes (i.e. $n/8$) are $\log n/4$ away from the logical tree root, simply because for binary tree, if all the nodes are at most $\log n - 1$ away from the root, it can at most hold $n/2$ leaves. Thus $O\left(n^2 \log n\right)$ is the lower bound for all logical trees.

Furthermore, for every routing tree with uniform cost, any balanced binary logical tree achieves $O\left(n^2 \log n\right)$ cost.

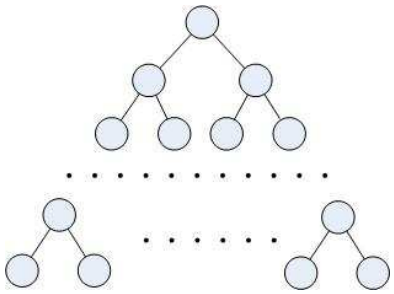

Figure 15: Balanced binary logical tree

3 Balanced binary logic tree is near optimal

In this section, first we are going to show that no matter what the routing tree is, there is a binary logical tree whose cost is within twice the optimal. Then we show there is a balanced binary logical tree that is near optimal. In the end of this section, we give an example which shows the order of members in the balanced binary logical tree does matters (i.e. if we don't choose the order carefully, the cost of logical tree will be arbitrarily large).

3.1 Binary logical tree

Theorem 1. *There is always a binary logical tree whose cost is within three times the optimal one.*

Proof. First we represent the cost of logical tree in a convenient way. Let $M(v)$ be the multicast cost to the leaves in the tree rooted at node v and $PL(v)$ be the number of leaves in the tree rooted at v's parent. Then the cost of logical tree is

$$\sum_{v \in T \ \ v \neq root} PL(v) \cdot M(v)$$

Assume we have the optimal logical tree. Figure 16 shows the corresponding binary logical tree whose cost is within three times the optimal one. For each node in the optimal logic tree, which has more than two children, we group its children into two groups such that it is a $(1/3, 2/3)$ partition of the leaves. If this kind of partition exists, we make changes according to Figure 16 part (a); otherwise, then there is a child with more than 2/3 leaves, and we make changes according to Figure 16 part (b).

Now we are going to calculate the increase in cost as a result of making one node binary. According to our new representation, the costs of $u_1, u_2, \ldots, u_m$ are

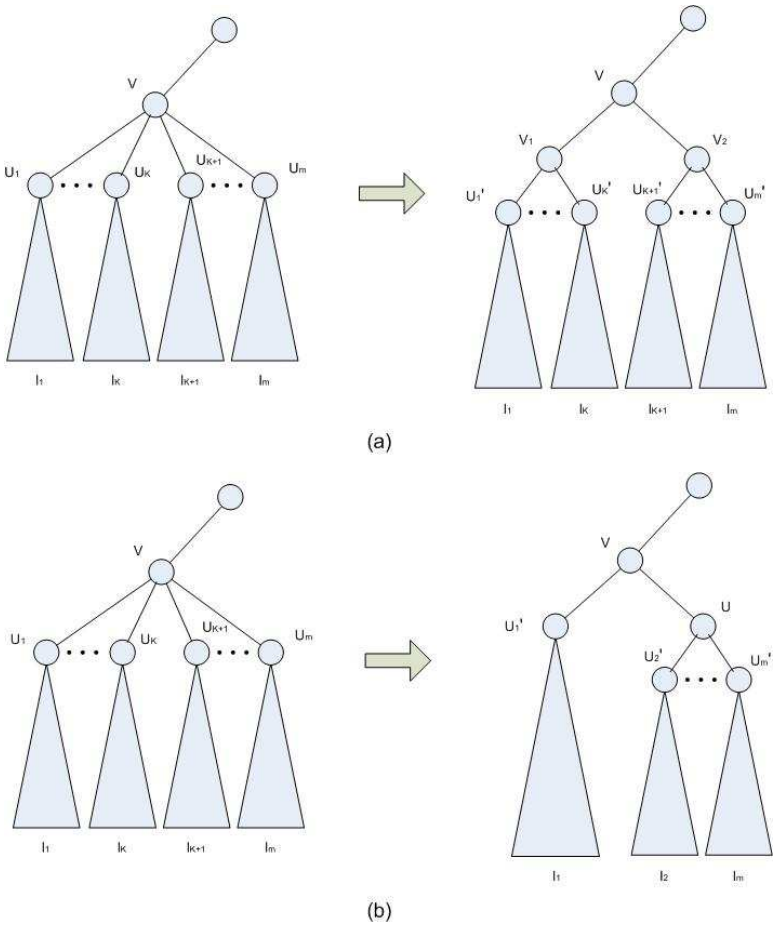

Figure 16: Binary logical tree transformation

$$
\begin{gathered}
(l_1 + l_2 + \cdots + l_m)M(u_1) \\
(l_1 + l_2 + \cdots + l_m)M(u_2) \\
\cdots \\
(l_1 + l_2 + \cdots + l_m)M(u_m)
\end{gathered}
$$

And in Figure 16 part (a), these costs change to the costs of $v_1, v_2, u'_1, u'_2, \ldots, u'_m$ in the binary logic tree, which are

$$(l_1 + l_2 + \cdots + l_m)M(v_1)$$
$$(l_1 + l_2 + \cdots + l_m)M(v_2)$$

$$(l_1 + l_2 + \cdots + l_k)M(u'_1)$$
$$(l_1 + l_2 + \cdots + l_k)M(u'_2)$$
$$\cdots$$
$$(l_1 + l_2 + \cdots + l_k)M(u'_k)$$

$$(l_{k+1} + l_{k+2} + \cdots + l_m)M(u'_{k+1})$$
$$(l_{k+1} + l_{k+2} + \cdots + l_m)M(u'_{k+2})$$
$$\cdots$$
$$(l_{k+1} + l_{k+2} + \cdots + l_m)M(u'_{k+m})$$

Notice that $M(v_1) \leq M(u_1)+M(u_2)+\cdots+M(u_k)$, $M(v_2) \leq M(u_{k+1})+M(u_{k+2})+\cdots+M(u_m)$ and $M(u_i) = M(u'_i)$ for $i = 1, 2, \ldots, m$. So it is easy to see that the cost of $v_1, v_2, u'_1, u'_2, \ldots, u'_m$ is within $1 + 2/3$ times the cost of $u_1, u_2, \ldots, u_m$.

Similarly, for Figure 16 part (b), the cost of $u, u'_1, u'_2, \ldots, u'_m$ is within $(1 + 1/3)$ times the cost of $u_1, u_2, \ldots, u_m$. One observation is, if we want to make v "complete binary", which means any node that has v's original children as children is binary, the cost of sum of v's children's cost at most increases to $\big(1 + (2/3) + (2/3)^2 + \ldots\big) = 3$ times original cost. So after we make every internal node "complete binary" the cost of the new logical tree is within 3 times the original one.

In the above argument, we didn't restrict ourselves to uniform cost condition. The conclusion works for non-uniform cost too. □

One observation about approximation ratio of binary logical tree is, we cannot get any ratio smaller than 2. Consider a "star" shape routing tree with uniform cost. The optimal logical tree should also be a star, in which case the cost is n^2. And the optimal logical tree should be a balanced binary tree, in which case the cost is $2n^2 - 2n$. The ratio between them is 2. This means the ratio between the optimal binary logical tree and the optimal logical is at least 2.

3.2 Balanced binary logical tree

Theorem 2. *There is always a balanced binary logical tree whose cost is within constant times the optimal one.*

Proof. We already know that there exists a binary logical tree that is near optimal. So we are going to prove that there is a balanced binary logical tree that is near optimal binary logical tree.

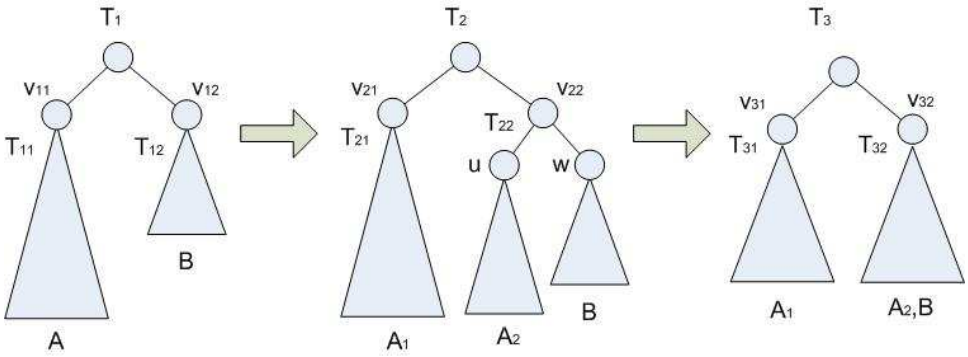

Figure 17: Balanced binary logical tree transformation

As shown in Figure 17, assume we have the optimal binary logical tree, T_1. And $|A| > |B|$. Based on T_1, we create T_2 which balanced the number of members in left subtree and right subtree. $A_1 \subset A$, $|A_1| = n/2$ and $A_2 = A - A_1$. When we separate A_1 and A_2, we reserve the tree structure as it is in A. And we do the same operation to A_2's left subtree and right subtree. Then we will get a balanced binary logical tree, T_3. What we want to prove is, $cost\,(T_3) \leq \alpha \cdot cost\,(T_1)$, where α is a constant.

The costs of T_1 and T_2 can be represented as

$$c\,(T_1) = c\,(T_{11}) + c\,(T_{12}) + c\,(v_{11}) + c\,(v_{12})$$

$$c\,(T_2) = c\,(T_{21}) + c\,(T_{22}) + c\,(v_{21}) + c\,(v_{22})$$

Lemma 1. *$c\,(T_{21}) + c\,(T_u) \leq c\,(T_{11})$, and $c\,(T_w) = c\,(T_{12})$.*

Proof. We prove $c\,(T_{21}) + c\,(T_u) \leq c\,(T_{11})$ by two cases. Take any internal node v in A. If all v's children belong to A_1 or all v's children belong to A_2, then its cost in T_2 is less than or equal to its cost in T_1. Simply because the number of leaves in the tree rooted at its parent node is not increasing and $M\,(T_v)$ keeps the same. If some of v's children belong to A_1 and some belong to A_2, then this node appears in both T_{21} and T_u. Let l be the number of leaves in its parent tree when it is in T_1, and m be the multicast cost. Let l_1 be the number of leaves in its parent tree when it is in T_{21}, and l_2 be the number of leaves in its parent tree when it is in T_u. Then v's cost in T_{21} plus v's cost in T_u is less than or equal to $ml_1 + ml_2 = ml$, which is v's cost in T_1. So, $c\,(T_{21}) + c\,(T_u) \leq c\,(T_{11})$.

The argument of $c\,(T_w) = c\,(T_{12})$ is similar. □

Lemma 2. *$c\,(v_{22}) \leq c\,(v_{11}) + c\,(v_{12})$.*

Proof.

$$\begin{aligned} c(v_{22}) &= n \cdot M(A_2 \cup B) \\ &\leq n \cdot M(A \cup B) \\ &\leq n \cdot [M(A) + M(B)] \\ &= c(v_{11}) + c(v_{12}) \end{aligned}$$

□

By Lemma 1 and Lemma 2, we can have the following list of inequalities and equalities:

$$\begin{aligned} c(T_{21}) + c(T_{22}) &\leq c(T_{11}) + c(T_{12}) + c(u) + c(w) && (1) \\ c(u) &\leq \frac{1}{2} c(v_{11}) && (2) \\ c(w) &= \frac{1}{2} c(v_{12}) && (3) \\ c(v_{21}) &\leq c(v_{11}) && (4) \\ c(v_{22}) &\leq c(v_{11}) + c(v_{12}) && (5) \\ c(v_{31}) &= c(v_{21}) && (6) \\ c(v_{32}) &= c(v_{22}) && (7) \end{aligned}$$

Lemma 3. *$cost(T_3) \leq \alpha \cdot cost(T_1)$, where α is a constant.*

Proof. We are going to prove this lemma by induction on the number of members. And we make use of those 7 inequalities and equalities we got from Lemma 1 and Lemma 2.

$$\begin{aligned} c(T_3) &= c(T_{31}) + c(T_{32}) + c(v_{31}) + c(v_{32}) \\ &= c(T_{31}) + c(T_{32}) + c(v_{21}) + c(v_{22}) \\ &\leq \alpha \cdot c(T_{21}) + \alpha \cdot c(T_{22}) + c(v_{21}) + c(v_{22}) \quad \text{(by I.H.)} \\ &\leq \alpha [c(T_{11}) + c(T_{12}) + c(u) + c(w)] + c(v_{21}) + c(v_{22}) \\ &= \alpha [c(T_{11}) + c(T_{12}) + c(v_{11}) + c(v_{12}) - c(v_{11}) - c(v_{12}) + c(u) + c(w)] \\ &\quad + c(v_{21}) + c(v_{22}) \\ &= \alpha \cdot c(T_1) - \alpha \cdot c(v_{11}) - \alpha \cdot c(v_{12}) + \alpha \cdot c(u) + \alpha \cdot c(w) + c(v_{21}) + c(v_{22}) \\ &\leq \alpha \cdot c(T_1) - \alpha \cdot c(v_{11}) - \alpha \cdot c(v_{12}) + \alpha \cdot \frac{c(v_{11})}{2} + \alpha \cdot \frac{c(v_{12})}{2} + c(v_{11}) \\ &\quad + [c(v_{11}) + c(v_{12})] \\ &= \alpha \cdot c(T_1) + c(v_{11}) \left(-\alpha + \frac{\alpha}{2} + 1 + 1\right) + c(v_{12}) \left(-\alpha + \frac{\alpha}{2} + 1\right) \\ &= \alpha \cdot c(T_1) + c(v_{11}) \left(2 - \frac{\alpha}{2}\right) + c(v_{12}) \left(1 - \frac{\alpha}{2}\right) \\ &\leq \alpha \cdot c(T_1) \quad \text{(as long as } \alpha \geq 4) \end{aligned}$$

By induction, we know this lemma is correct. □

From Lemma 3, we can conclude that there is always a balanced binary logical tree that is near optimal binary logical tree. Hence, there is always a balanced binary logical tree that is near optimal logical tree. Notice, in the above argument, we didn't restrict ourselves to uniform cost condition. The conclusion works for non-uniform cost too. □

3.3 Example of orders of members

Considering the routing tree in Figure 18. This routing tree is totally balanced, and each internal node has $\sqrt{n}$ children. The cost of edges adjcent to root is $\sqrt{n}$, and the cost of edges adjcent to leaves is 1.

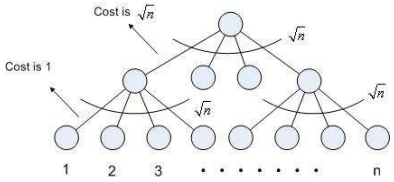

Figure 18: Balanced binary logical tree transformation

The first logical tree we considered is shown in Figure 19, which is the same as the routing tree. We reserve the order of leaves. After calculation, we know the cost of this logical tree is $O\left(n^2\right)$.

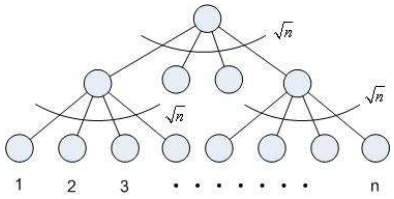

Figure 19: Two level logical tree with good order of leaves

The second logical tree we considered, shown in Figure 20, has the same hierarchy as the one in Figure 19. But we change the order of leaves in order to increase the update cost. After calculation, we know the cost of this logical tree is $O\left(n^{2.5}\right)$. We can see here that the order of leaves in logical trees do play an important roll.

The third logical tree we considered is a balanced binary logical tree. And we reserve the order of leaves, which is shown in Figure 21. After calculation, we know the cost of this logical tree is $O\left(n^2\right)$, which is in the same order as the first logical tree we considered.

The last logical tree we considered is also a balanced binary logical tree. But this time we change the order of leave to increase the update cost. The

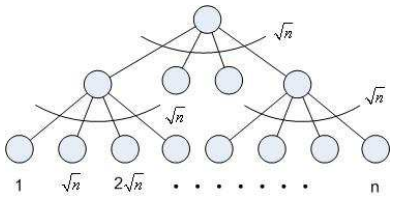

Figure 20: Two level logical tree with bad order of leaves

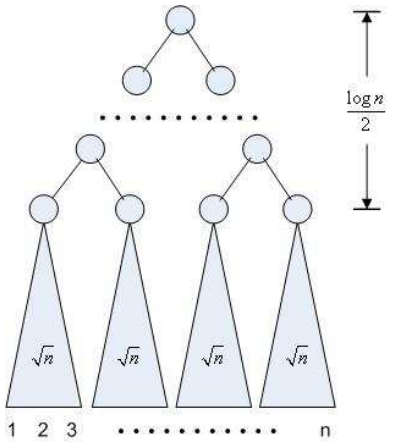

Figure 21: Balanced binary logical tree with good order of leaves

logical tree is shown in Figure 22. After calculation, we know the cost of this logical tree is $O\left(n^2 \log n\right)$. This shows that the order of leaves in logical tree is important for its cost. Also this matches the theorem we proof before that the cost of balanced binary logical tree is at most $O\left(\log n\right)$ away from the optimal.

In sum, the above example tells us, though we have proved that there exists a balanced binary logical tree that is near optimal, we do need to choose the order of leaves in the logical tree carefully. Otherwise the cost of this logical tree could be far away from optimal.

4 An approximation algorithm for routing tree

In this section, we are going to give a simple algorithm, which constructs a near optimal logical tree based on the routing tree.

Here is the description of our algorithm. Given any routing tree, shown in Figure 23, we find the internal node v such that there is a combination of its children whose total number of leaves is between $n/3$ and $2n/3$. We use this node to partition the leaves.

Claim 1. *This kind of partition node can always be found.*

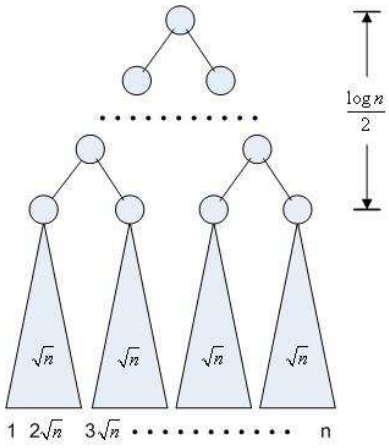

Figure 22: Balanced binary logical tree with bad order of leaves

Proof. Suppose this kind of partition node doesn't exist, which means for all internal node v, its number of leaves is either $< n/3$ or $> 2n/3$. We call nodes with less than $n/3$ leaves small nodes, and nodes with more than $2n/3$ leaves large nodes. Consider the large node with only small nodes as its children, there is a combination of its children whose total number of leaves is between $n/3$ and $2n/3$. This means this kind of partition node exists. □

According to the notations in Figure 23, we can partition the leaves in to two parts, X and Y, such that $n/3 \leq |X| \leq 2n/3$ and $n/3 \leq |Y| \leq 2n/3$. Then we construct the logical tree as shown in Figure 24, where T_1 is the logical tree we construct on X and T_2 is the logical tree we construct on Y by calling our algorithm recursively.

Let $S = X \cup Y$. We are going to show $ALG(S) \leq \alpha \cdot OPT_b(S) + \beta \cdot nM(S)$ using induction. $ALG(s)$ represents the cost of logical tree constructed by our algorithm, and $OPT_b(S)$ means the cost of the optimal binary logical tree. Our induction hypothesis is $ALG(X) \leq \alpha \cdot OPT(X) + \beta \cdot |X|M(X)$ and $ALG(Y) \leq \alpha \cdot OPT(Y) + \beta \cdot |Y|M(Y)$.

Lemma 4. $ALG(S) \leq ALG(X) + ALG(Y) + \dfrac{4c}{3\log 3/2} n \log n + n\left(M(X) + M(Y)\right)$.

Proof. (1) The cost of nodes in $ALG(Y)$ is the same as their cost in $ALG(S)$. (2) Similarly, the cost of nodes in $ALG(X)$ is equal to their cost in $ALG(S) + \frac{4c}{3\log 3/2} n \log n$. The reason we add $\frac{4c}{3\log 3/2} n \log n$ is the multicast cost of each node in $ALG(X)$ increased by c compared to its cost in $ALG(S)$. Since in the worst case $ALG(X)$ has $\log_{\frac{3}{2}} \frac{n}{3}$ levels, the increased cost is at most $2|X| c \log_{\frac{3}{2}} \frac{n}{3} \leq 2\frac{2n}{3} c \log_{\frac{3}{2}} \frac{n}{3} \approx \frac{4c}{3\log 3/2} n \log n$. Combine (1) and (2), then add the cost of the root of $ALG(X)$ and $ALG(Y)$, we know this lemma is correct. □

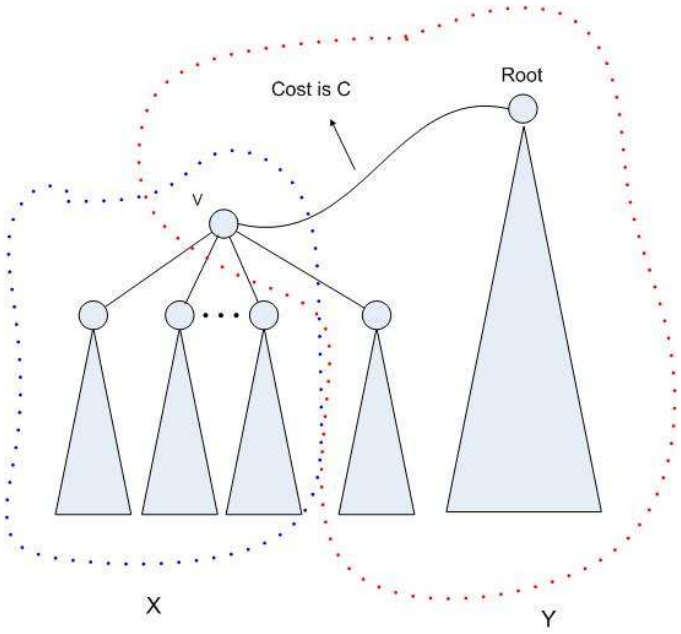

Figure 23: An arbitrary routing tree

Let $OPT_b(S)$ be the optimal binary logical tree. Now we separate this logical tree into two parts, $OPT'_b(X)$ and $OPT'_b(Y)$, as shown in Figure 25. $OPT'_b(X)$ only contains the leaves belong to X, but reserve the tree structure as $OPT_b(S)$. Similarly, $OPT'_b(Y)$ only contains the leaves belong to Y, but reserve the tree structure as $OPT_b(S)$.

Lemma 5. $OPT'_b(Y) \geq OPT_b(Y)$

Proof. Since $OPT'_b(Y)$ is a potential "binary" logical tree for Y, its cost cannot be less than $OPT_b(Y)$. □

Lemma 6. $OPT'_b(X) \geq OPT_b(X) + \frac{c}{4\log 2} n \log n$

Proof. If we calculate $OPT'_b(X)$ only in $ALG(X)$ which is T_1 in figure 24, then similar to Lemma 5 we have $OPT'_b(X) \geq OPT_b(X)$. But the cost of $OPT'_b(X)$ is calculated from the root in the routing tree. Because $OPT'_b(X)$ is binary, there are at least $3|Y|/4$ leaves are $\log_2|Y|-1$ away from the root. So the increased cost is at least $\frac{3}{4}|Y|c\log_2|Y| \geq \frac{c}{4} n \log_2 n$. □

Lemma 7. $OPT_b(S) \geq OPT_b(X) + OPT_b(Y) + \frac{c}{4\log 2} n \log n$

Proof.

$$OPT_b(S) \geq OPT'_b(X) + OPT'_b(Y) \geq OPT_b(X) + OPT_b(Y) + \frac{c}{4\log 2} n \log n$$

□

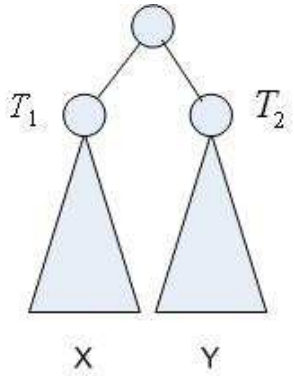

Figure 24: The logical tree constructed by our approximation algorithm

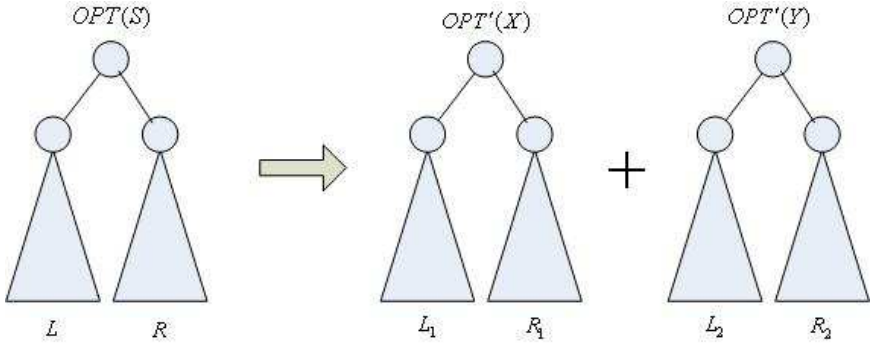

Figure 25: Optimal binary logical tree separation

Lemma 8. $ALG(S) \leq \alpha \cdot OPT_b(S) + \beta \cdot nM(S)$

Proof.

$$\begin{aligned}
ALG(S) &\leq ALG(X) + ALG(Y) + \frac{4c}{3\log 3/2} n\log n + n\left(M(X)+M(Y)\right) \quad \text{(Lemma 4)} \\
&\leq \alpha\cdot OPT_b(X) + \beta\cdot|X|M(X) + \alpha\cdot OPT_b(Y) + \beta\cdot|Y|M(Y) \\
&\quad + \frac{4c}{3\log 3/2} n\log n + n\left(M(X)+M(Y)\right) \quad \text{(I.H.)} \\
&\leq \alpha\left[OPT_b(X)+OPT_b(Y)\right] + \frac{4c}{3\log 3/2} n\log n + \left(\frac{2}{3}\beta+1\right) n\left(M(X)+M(Y)\right) \\
&\leq \alpha\left[OPT_b(S) - \frac{c}{4\log 2} n\log n\right] + \frac{4c}{3\log 3/2} n\log n \quad \text{(Lemma 7)} \\
&\quad + \left(\frac{2}{3}\beta+1\right) n\left(M(X)+M(Y)\right) \\
&\leq \alpha\left[OPT_b(S) - \frac{c}{4\log 2} n\log n\right] + \frac{4c}{3\log 3/2} n\log n + \left(\frac{2}{3}\beta+1\right) n\left(M(S)+c\right) \\
&\qquad (M(S) \geq M(X)+M(Y)-c) \\
&= \alpha\cdot OPT_b(S) + \left(\frac{2}{3}\beta+1\right) nM(S) - \alpha\cdot\frac{c}{4\log 2} n\log n + \frac{4c}{3\log 3/2} n\log n \\
&\leq \alpha\cdot OPT_b(S) + \beta\cdot nM(S) \quad \text{(as long as } \alpha\geq 9 \text{ and } \beta\geq 3)
\end{aligned}$$

□

Theorem 3. *This algorithm is a $(3\alpha+\beta)$-approximation.*

Proof. Let $OPT(S)$ be the optimal logical tree for S. By theorem 1, we know $OPT_b(S) \leq 3\cdot OPT(S)$. So

$$ALG(S) \leq \alpha\cdot OPT_b(S) + \beta\cdot nM(S) \leq 3\alpha\cdot OPT(S) + \beta\cdot nM(S) \leq (3\alpha+\beta)OPT(S)$$

□

In the proof of lemma 8, we know $\alpha \geq 9$ and $\beta \geq 3$, which means this algorithm is a 30-approximation. By slightly modify the constant in lemma 6, we can bring down the approximation constant very close to 23.52 ($\alpha = 6.84$).

The following gives another way to prove this algorithm is a constant approximation algorithm. We are still doing induction on the number of leaves and the inductive hypothesis is $ALG(S) \leq \alpha\cdot OPT(S) + \beta\cdot|S|M(S)$.

Lemma 9. $ALG(S) = ALG(X) + ALG(Y) + n\left[M(X)+M(Y)\right]$

Proof. Here when we apply the algorithm on X, it means apply the algorithm on the tree rooted at v plus the original root and an edge between them with weight c (as shown in Figure 23). The rest of the proof is basically the same as lemma 4. □

Lemma 10. $OPT(S) \geq OPT(X) + OPT(Y)$

Proof. The same as lemma 7. □

Theorem 4. *This algorithm has a constant approximation ratio.*

Proof. We are going to prove this theorem by two cases. First, when $c \leq M(S)/4$

$$\begin{aligned}
ALG(S) &= ALG(X) + ALG(Y) + n\left[M(X) + M(Y)\right] \\
&\leq \alpha \cdot OPT(X) + \beta \cdot |X| M(X) + \alpha \cdot OPT(Y) + \beta \cdot |Y| M(Y) \\
&\quad + n\left[M(X) + M(Y)\right] \\
&\leq \alpha\left[OPT(X) + OPT(Y)\right] + \left(\frac{2n}{3}\beta + n\right)\left[M(X) + M(Y)\right] \\
&\leq \alpha \cdot OPT(S) + \left(\frac{2}{3}\beta + 1\right) n\left[M(X) + M(Y)\right] \\
&\leq \alpha \cdot OPT(S) + \left(\frac{2}{3}\beta + 1\right) n\left(M(S) + c\right) \\
&\leq \alpha \cdot OPT(S) + \left(\frac{2}{3}\beta + 1\right)\left(1 + \frac{1}{4}\right) nM(S) \\
&\leq \alpha \cdot OPT(S) + \beta \cdot nM(S)
\end{aligned}$$

as long as $(2\beta/3 + 1)(1 + 1/4) \leq \beta$, which is $\beta \geq 7.5$. Second case is when $c \geq M(S)/4$. The cost of optimal binary logical tree on X is at least

$$OPT_b(X) \geq \frac{|X|}{2} \cdot \log_2 |X| \cdot 2c \geq \frac{c}{3 \log 2} n \log n$$

The cost of optimal logical tree on X is at one third of the optimal binary logical tree cost, which is

$$OPT(X) \geq \frac{1}{3} OPT_b(X) \geq \frac{c}{9 \log 2} n \log n$$

The cost of logical tree constructed by our algorithm on S is at most

$$\begin{aligned}
ALG(S) &\leq n \cdot \log_{3/2} n \cdot 2M(S) \\
&= 18 \cdot \frac{\log 2}{\log 3/2} \cdot \frac{M(S)}{c} \cdot \frac{c}{9 \log 2} n \log n \\
&\leq 18 \cdot \frac{\log 2}{\log 3/2} \cdot 4 \cdot OPT(X) \\
&\leq 72 \cdot \frac{\log 2}{\log 3/2} \cdot OPT(S)
\end{aligned}$$

Combine both cases, we know this algorithm is a constant approximation algorithm. Notice, we need to set $\alpha \geq 72 \cdot \frac{\log 2}{\log 3/2}$. □

5 An approximation algorithm for routing graph

In this section, we are going to give a simple algorithm, which constructs a near optimal logical tree based on routing graph.

In our algorithm, we make use of LAST (Light Approximate Shortest-path Tree)[1] of the routing graph, which is an (α, β)-approximation of Shortest Path Tree and Minimum Spanning Tree. This can guarantee that the distance from root and any other vertex in LAST is at most α times the shortest distance from root to this vertex in the original graph, and the weight of LAST is at most β times the weight of a minimum spanning tree.

Input: S, which is a set of vertices in the routing graph.
Algorithm:

1. Compute the complete graph on $S \cup \{root\}$. The weight of an edge (u, v) is the length of shortest path between u and v in the original routing graph.
2. Compute the MST on this complete graph. Call it MST(S).
3. Based on MST(S), compute the LAST, which is an (α, β)-approximation of SPT and MST(S).
4. Using this LAST to find the partition node and split S into X and Y.
5. Call the algorithm recursively on X, which returns T_1.
6. Call the algorithm recursively on Y, which returns T_2.
7. Return the binary logical tree constructed using T_1 and T_2, shown in Figure 24.

Now we are going to prove this is a constant approximation algorithm. And we are using the LAST which is a $\left(1 + \sqrt{2}\gamma, 1 + \sqrt{2}/\gamma\right)$-approximation in the following proof.

When we calculate the multicast cost to a subset of terminals in the routing graph, the optimum multicast is by a minimum Steiner tree, computing which is NP-hard. But we can easily approximate it, for instance by selecting a minimum spanning tree in the metric space connecting the root to the desired terminals (the metric being the shortest path cost in the routing graph). So in the following proof, define $M(S)$ to be the cost of the minimum spanning tree connecting the root to S in the complete graph $G(S)$ whose vertices are the ones in $S \cup \{root\}$ and the weight of an edge (u, v) is the shortest path distance between u and v in the routing graph. And the cost we calculated by this definition is a 2-approximation of the cost calculated by the optimal multicasts in the graph.

Lemma 11. $ALG(S) = ALG(X) + ALG(Y) + n\,[M(X) + M(Y)]$

Proof. The proof of this lemma in the routing tree case can be used here. □

Lemma 12. $OPT(X) \geq OPT(X) + OPT(Y)$

Proof. Same as lemma 10. □

Lemma 13. $M(X) + M(Y) \leq \left(1 + \sqrt{2}/\gamma\right) \cdot M(S) + c$

Proof. Let $M_L(S)$ be the multicast cost to S in LAST. From the description of LAST we know $M_L(S) \leq \left(1 + \sqrt{2}/\gamma\right) \cdot M(S)$. Also we have

$$M_L(S) \geq M_L(X) + M_L(Y) - c \geq M(X) + M(Y) - c$$

So $\left(1 + \sqrt{2}/\gamma\right) \cdot M(S) \geq M(X) + M(Y) - c$. □

Theorem 5. *This algorithm has a constant approximation ratio.*

Proof. We are going to prove this by induction. And we want to show $ALG(S) \leq \alpha \cdot OPT(S) + \beta \cdot nM(S)$. Similar to the second proof in the routing tree cases, we are going to show this by two cases. Let $c \leq x \cdot M(S)$.

$$\begin{aligned}
ALG(S) &= ALG(X) + ALG(Y) + n\,[M(X) + M(Y)] \\
&\leq \alpha \cdot OPT(X) + \beta \cdot |X| M(X) + \alpha \cdot OPT(Y) + \beta \cdot |Y| M(Y) \\
&\quad + n\,[M(X) + M(Y)] \\
&\leq \alpha\,[OPT(X) + OPT(Y)] + \left(\frac{2n}{3}\beta + n\right)[M(X) + M(Y)] \\
&\leq \alpha \cdot OPT(S) + \left(\frac{2}{3}\beta + 1\right) n\,[M(X) + M(Y)] \\
&\leq \alpha \cdot OPT(S) + \left(\frac{2}{3}\beta + 1\right) n \left(\left(1 + \sqrt{2}/\gamma\right) \cdot M(S) + c\right) \\
&\leq \alpha \cdot OPT(S) + \left(\frac{2}{3}\beta + 1\right) \left(1 + \sqrt{2}/\gamma + x\right) nM(S) \\
&\leq \alpha \cdot OPT(S) + \beta \cdot nM(S)
\end{aligned}$$

The above inequality holds as long as $(2\beta/3 + 1)\left(1 + \sqrt{2}/\gamma + x\right) \leq \beta$. We have $\gamma > 2\sqrt{2}$, $x < \frac{1}{2}\left(1 - 2\sqrt{2}/\gamma\right)$, and $\beta \geq \dfrac{3\left(1 + \sqrt{2}/\gamma + x\right)}{1 - 2\sqrt{2}/\gamma - 2x}$. This is our first case (c is far away from $M(S)$). The second case is $c > x \cdot M(S)$. The distance from root to any vertex in LAST is at least c. So the shortest path from the root to the vertex is at least $c/\left(1 + \sqrt{2}\gamma\right)$. The cost of optimal binary logical tree on X is at least

$$OPT_b(X) \geq \frac{|X|}{2} \cdot \log_2 |X| \cdot 2 \cdot \frac{c}{1 + \sqrt{2}\gamma} \geq \frac{c}{3\left(1 + \sqrt{2}\gamma\right) \log 2} n \log n$$

The cost of optimal logical tree on X is at one third of the optimal binary logical tree cost, which is

$$OPT(X) \geq \frac{1}{3} OPT_b(X) \geq \frac{c}{9\left(1+\sqrt{2}\gamma\right)\log 2} n \log n$$

The cost of logical tree constructed by our algorithm on S is at most

$$\begin{aligned} ALG(S) &\leq n \cdot \log_{3/2} n \cdot 2M(S) \\ &= 18\left(1+\sqrt{2}\gamma\right) \cdot \frac{\log 2}{\log 3/2} \cdot \frac{M(S)}{c} \cdot \frac{c}{9\left(1+\sqrt{2}\gamma\right)\log 2} n \log n \\ &\leq 18\left(1+\sqrt{2}\gamma\right) \cdot \frac{\log 2}{\log 3/2} \cdot \frac{1}{x} \cdot OPT(X) \\ &\leq \frac{18}{x} \cdot \left(1+\sqrt{2}\gamma\right) \cdot \frac{\log 2}{\log 3/2} \cdot OPT(S) \end{aligned}$$

Combine both cases, we know this algorithm is a constant approximation algorithm. In order to know what this constant is, we need to find the γ and x to minimize the approximation ratio.

Let $\beta = \dfrac{3\left(1+\sqrt{2}/\gamma + x\right)}{1 - 2\sqrt{2}/\gamma - 2x}$, $\eta = \dfrac{18}{x} \cdot \left(1+\sqrt{2}\gamma\right) \cdot \dfrac{\log 2}{\log 3/2}$. And we know $\gamma > 2\sqrt{2}$, $x < \dfrac{1}{2}\left(1 - 2\sqrt{2}/\gamma\right)$. When γ increases, β decreases and η increases; when x increases, β increases and η decreases. □

6 Hardness proofs

In this section, we are going to give the hardness result of this problem.

6.1 NP-hard for logical tree with depth constraint

We are going to reduce 3-Partition problem, which is strong NP-hard problem, to our logical key tree problem. First define 3-Partition problem as follows.

Problem 1. Input a set A of $3m$ elements, a bound $B \in Z^+$, and a set of sizes $S(a) \in Z^+$ for each $a \in A$ such that $B/4 < S(a) < B/2$, and $\sum_{a \in A} S(a) = mB$. The question is can A be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that for $1 \leq i \leq m$, $\sum_{a \in A_i} S(a) = B$.

For any instance of 3-Partition problem, which means given A, m and B, we are going to construct a routing tree such that its optimal logical tree has certain cost if and only if A can be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ and for $1 \leq i \leq m$, $\sum_{a \in A_i} S(a) = B$. Assume that the $3m$ elements in A are $a_1, a_2, \ldots, a_{3m}$. Construct the routing tree as shown in

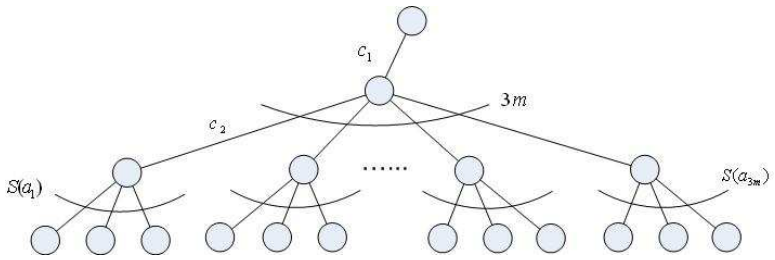

Figure 26: Routing tree constructed by 3-Partition instance

Figure 26. The edge cost of the first level is c_1, the edge cost of the second level is c_2, and the edge cost of the third level is 1.

If A can be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ and for $1 \leq i \leq m$, $\sum_{a \in A_i} S(a) = B$, its corresponding logical tree is shown in Figure 27. And the cost of this logical tree is $(c_1 + c_2 + 1)mB^2 + (c_1 + 3c_2)m^2B + m^2B^2$.

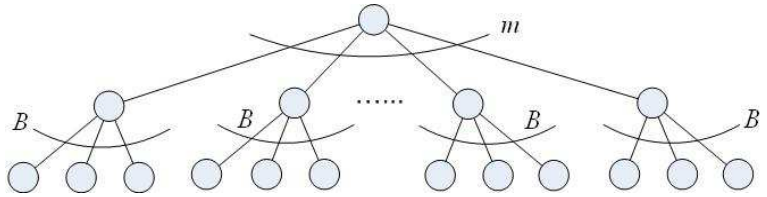

Figure 27: Logical tree corresponding to 3-Partition

Now we need to show that for 3-Partition instance, we can choose c_1 and c_2 such that the cost of the above logical tree is the minimum. First step is to make the minimum among all the 2 level logical trees. The general 2-level logical tree is shown in Figure 28.

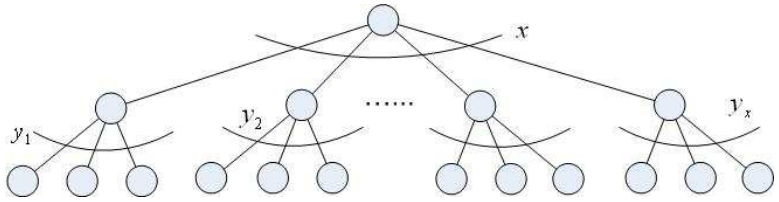

Figure 28: General 2 level logical tree

Its cost can be calculated as follows,

$$
\begin{aligned}
cost &\geq \sum_{i=1}^{x} y_i \left[y_i \left(c_1 + c_2 + 1 \right) + x c_1 + 3 m c_2 + m B \right] \\
&= \left(c_1 + c_2 + 1 \right) \sum_{i=1}^{x} y_i^2 + x c_1 m B + 3 c_2 m^2 B + m^2 B^2 \\
&\geq x \left(c_1 + c_2 + 1 \right) \left(\frac{\sum_{i=1}^{x} y_i}{x} \right)^2 + x c_1 m B + 3 c_2 m^2 B + m^2 B^2 \\
&= \left(c_1 + c_2 + 1 \right) \frac{m^2 B^2}{x} + x c_1 m B + 3 c_2 m^2 B + m^2 B^2
\end{aligned}
$$

when $\left(c_1 + c_2 + 1 \right) \frac{m^2 B^2}{x} = x c_1 m B$, the cost achieves minimum. That is $x^2 = mB \cdot \frac{c_1 + c_2 + 1}{c_1}$. We would like the cost is minimized when $x = m$. So let $mB \cdot \frac{c_1 + c_2 + 1}{c_1} = m^2$, we have $\frac{m - B}{B} = \frac{c_2 + 1}{c_1}$. As long as $m > B$, we could find c_1 and c_2, such that the logical key tree corresponding 3-Partition is the minimum among all 2-level logical trees.

Next step is to show that the logical tree corresponding to 3-Partition is the minimum among all the logical trees whose depth is at most 2, shown in Figure 29.

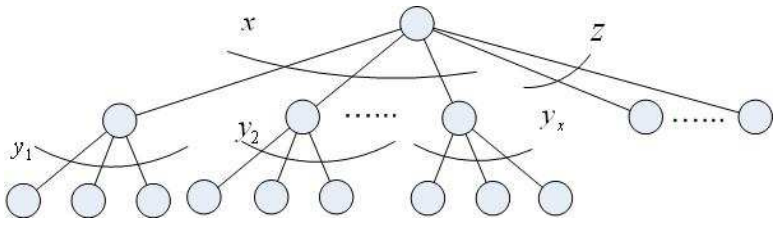

Figure 29: General 2 level logical tree

Let node $i \in z$, which is one step away from the root of logical tree. We analyze this by two cases. The first case is all i's siblings in the routing tree are only one step away from the root in the logical tree. In this case, we group them together by an intermedia node and connect this node to the root in the logical tree, shown in Figure 30, where s_i is the number of i's siblings plus one. And from the definition of 3-Partition problem, we know $B/4 < s_i < B/2$. We want to compare these to logical trees to see which one is less expensive. The cost difference between these two as be calculated as follows ($n = mB$),

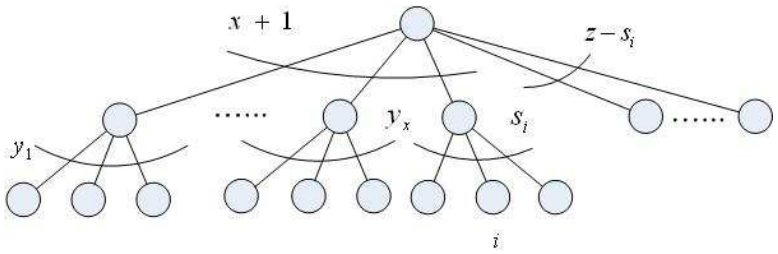

Figure 30: Revised logical tree

$$\begin{aligned}
& s_i \cdot n \cdot (c_1 + c_2 + 1) - [s_i \cdot s_i \cdot (c_1 + c_2 + 1) + n \cdot (c_1 + c_2 + s_i)] \\
= \;& s_i \cdot n \cdot (c_1 + c_2) - [s_i \cdot s_i \cdot (c_1 + c_2 + 1) + n \cdot (c_1 + c_2)] \\
\geq \;& \frac{mB^2}{4}(c_1 + c_2) - \left[\frac{B^2}{4}(c_1 + c_2 + 1) + mB\,(c_1 + c_2)\right] \\
= \;& B\left[\left(\frac{mB}{4} - \frac{B}{4} - m\right) \cdot (c_1 + c_2) - \frac{B}{4}\right]
\end{aligned}$$

as long as $\frac{mB}{4} - \frac{B}{4} - m > 0$ (like $B > 4$), we can make this positive. So by modifying the logical tree we can have a better solution. Similarly, if more than half i's siblings in the routing tree are only one step away from the root in the logical tree, we group them together to make them two steps away, then we have the condition $\frac{mB}{8} - \frac{B}{4} - m > 0$ (like $B > 8$).

The second case is that there exist an i's sibling in the routing tree that is two steps away from the root in the logical tree (notice in this case we can assume that more than half of i's siblings are two steps away from the root of the logical tree). In this case we can group i with its siblings, which is shown in Figure 31. And the cost difference between them can be calculated as follows,

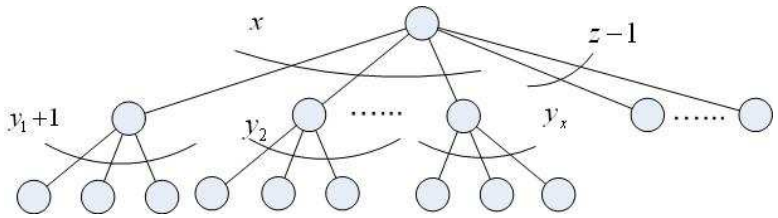

Figure 31: Revised logical tree

$$\begin{aligned}
& n(c_1 + c_2 + 1) - [(y_1 + 1)(c_1 + c_2 + 1) + y_1(c_1 + c_2 + 1) + n] \\
= \;& [mB - (2y_1 + 1)]\,(c_1 + c_2) - (2y_1 + 1)
\end{aligned}$$

as long as $mB > 2y_1 + 1$. If there exists an i's sibling which is in a group with size smaller than $(mB-1)/2$, we can pick large enough c_1 and c_2 to make this positive. If such sibling doesn't exist, then all i's siblings who are two steps away are all in the same group y_1 and $y_1 > (mB-1)/2$. In this case we can separate i's siblings in y_1 from the other nodes in y_1 and have the following logical tree, shown in Figure 32.

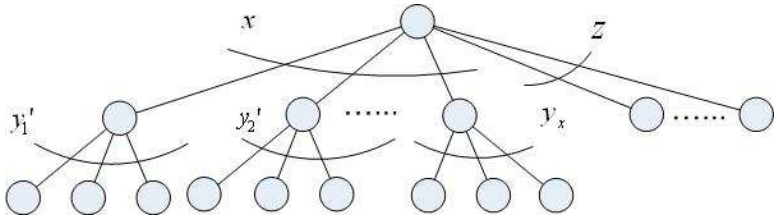

Figure 32: Revised logical tree

y_2' contains all i's sibling who were in y_1, and y_1' contains all the other leaves in y_1. We are going to argue this logical tree has smaller cost then the original one. The cost of subtree y_1 is $y_1^2(c_1+c_2+1)+nM(y_1)$. And this cost changes to $y_1'^2(c_1+c_2+1)+y_2'^2(c_1+c_2+1)+n\left[M(y_1')+M(y_2')\right]$ in the new logical tree. Since $y_1 = y_1' + y_2'$, $B/8 < y_2' < B/2$ and $M(y_1) - [M(y_1') + M(y_2')] = c_1$, we can calculate the cost difference between these two logical trees as follows,

$$\begin{aligned}
& 2y_1'y_2'(c_1+c_2+1) - nc_1 \\
= \;& 2y_1'y_2'\frac{m}{B}c_1 - mBc_1 \\
= \;& mc_1\left[\frac{2y_1'y_2'}{B} - B\right] \\
\geq \;& mc_1\left[2\left(\frac{mB-1}{2} - \frac{B}{8}\right)\frac{B}{8}\frac{1}{B} - B\right] \\
= \;& mc_1\left[\frac{mB-1}{8} - B - \frac{1}{32}B\right]
\end{aligned}$$

as long as $(mB-1)/8 \geq 33B/32$, which means $m \geq 9$, we can make this positive. The above argument shows we can always find a "small" group to place i. Hence the logical tree corresponding to 3-Partition is the minimum among all the logical trees whose depth is at most 2.

Theorem 6. *Logical key tree problem with depth constraint is NP-complete.*

Next we need to show the logical tree that corresponds to 3-Partition is also the minimum among all the logical trees whose depth is greater than 2. Look at the example in Figure 33. We want to compare the cost difference of those two logical trees. The calculation is shown as follows.

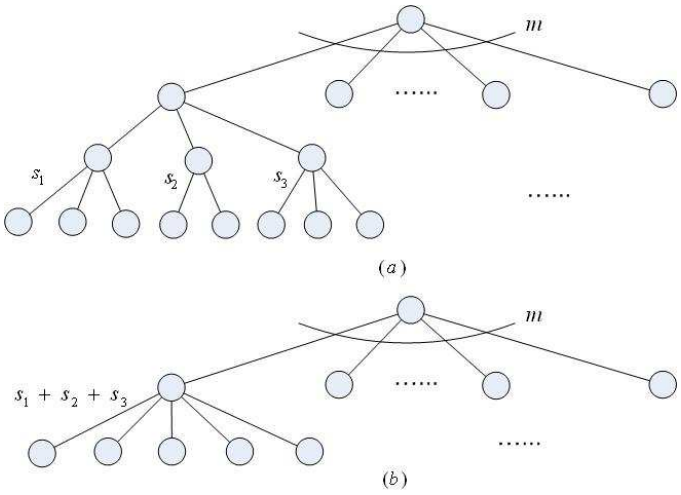

Figure 33: 3-level logical trees

$$
\begin{aligned}
& \left(s_1^2+s_2^2+s_3^2\right)(c_1+c_2+1)+(s_1+s_2+s_3)(3c_1+3c_2+s_1+s_2+s_3) \\
& >(s_1+s_2+s_3)^2(c_1+c_2+1) \\
\Leftrightarrow \quad & \left(s_1^2+s_2^2+s_3^2\right)(c_1+c_2+1)+3(s_1+s_2+s_3)(c_1+c_2)>(s_1+s_2+s_3)^2(c_1+c_2) \\
\Leftrightarrow \quad & \left(s_1^2+s_2^2+s_3^2\right)+3(s_1+s_2+s_3)(c_1+c_2)>2(s_1s_2+s_2s_3+s_3s_1)(c_1+c_2)
\end{aligned}
$$

since $s_1^2+s_2^2+s_3^2 \geq s_1s_2+s_2s_3+s_3s_1$, we know if $c_1+c_2 \leq 1/2$ the above inequality holds. Now consider another example shown in Figure 34. The cost calculation is shown as follows.

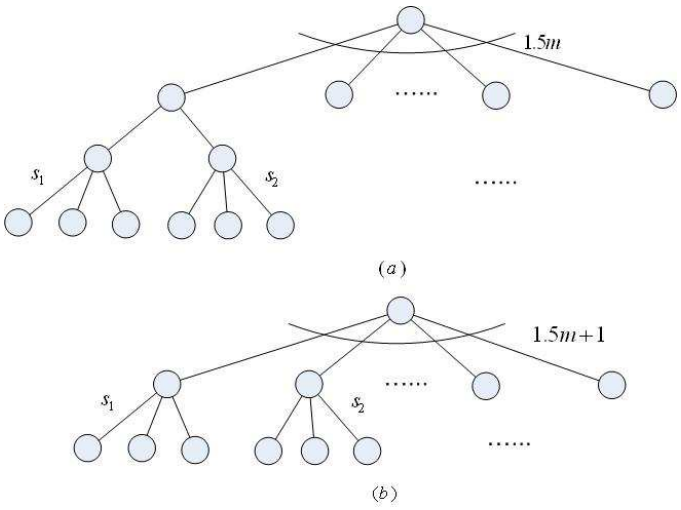

Figure 34: 3-level logical trees

$$
\begin{aligned}
& (s_1+s_2)(2c_1+2c_2+s_1+s_2)+mB(c_1+2c_2+s_1+s_2) > mB(2c_1+2c_2+s_1+s_2) \\
\Leftrightarrow\ & (s_1+s_2)(2c_1+2c_2+s_1+s_2) > mBc_1 \\
\Leftrightarrow\ & (s_1+s_2)^2+2(s_1+s_2)(c_1+c_2) > mBc_1 \\
\Leftrightarrow\ & (s_1+s_2)^2+2(s_1+s_2)\left(\frac{m}{B}-1\right)c_1 > mBc_1
\end{aligned}
$$

6.2 NP-hard for weighted logical tree

In this section, we are going to prove that logical key tree problem, which members can have different probability to request key update, is NP-complete.

Lemma 14. *For the routing tree shown in Figure 35, let S be the set of leaves, $|S| = n$, and each member has the same chance to request key update. $3n\log_3 n$ is the cost lower bound of all logical trees.*

Proof. We are going to prove this by induction on the number of leaves (n) in the routing tree. Base case is when $n = 1$. In this case the optimal logical tree is just a root with a single leaf, which has cost 1. And $1 > 3{\cdot}1{\cdot}\log_3 1 = 0$.

For the inductive case, let d be the degree of the root of logical tree. For the first child, its total number of leaves is m_1, for the second child, its total number of leaves is m_2, ..., for the d^{th} child, its total number of leaves is m_d. Then by inductive hypothesis, we can calculate the cost of logical tree as follows.

$$
\begin{aligned}
cost &\geq d\cdot n + 3m_1\log_3 m_1 + 3m_2\log_3 m_2 + \cdots + 3m_d\log_3 m_d \\
&\geq d\cdot n + d\cdot 3\cdot\frac{n}{d}\cdot\log_3\frac{n}{d} \quad (x\log x \text{ is convex}) \\
&= 3n\log_3 n + d\cdot n - 3n\log_3 d
\end{aligned}
$$

because when $d < 2.5$ or $d \geq 3$, $d \geq 3\log_3 d$. We can have $cost \geq 3n\log_3 n$. And the inequality tights when $d = 3$ and $m_1 = m_2 = m_3 = n/3$. □

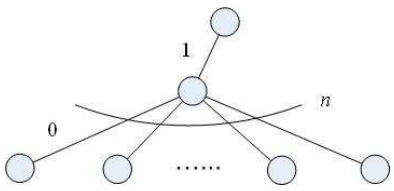

Figure 35: Routing tree

Lemma 15. *If $n = k^3$ ($k \in \mathbb{N}$) and all the leaves have the same chance to request key update, then the optimal logical tree is a degree-3 tree. And this optimal logical tree is unique.*

Proof. First, it is easy to verify that degree-3 logical tree reaches the lower bound, $3n\log_3 n$. So it is an optimal logical tree. Then from the proof of lemma 14, we can see that the inequality only tight when $m_1 = m_2 = \cdots = m_d$ and $d = 3$, which means degree-3 logical tree is the only optimal logical tree for this routing tree. □

Lemma 16. *Given a 3-partition instance, where* $3m$ *(*$3m$ *is the number of numbers) is not the power of* 3*, we can add groups of numbers,* B*,* 0*,* 0*, to make* m *the power of* 3*. And the original 3-partition instance has a solution if and only if the new 3-partition instance has a solution.*

Proof. □

Theorem 7. *Logical key tree problem, which different leaf can have different weight, is NP-complete.*

Proof. We are going to reduce 3-partition problem to this weighted logical key tree problem. First by lemma 16, we can assume that $3m$ is the power of 3. Then, for any given 3-partition instance ($3m$ numbers $w'_1, w'_2, \ldots, w'_{3m}$), we create a logical tree, which has the same structure as the one shown in Figure 35. We add a value w to each of these numbers, $w_i = w + w'_i$, such that $\frac{w_{max}}{w_{min}} < \frac{3n\log_3 n+1}{3n\log_3 n}$, where $w_{max} = \max_i\{w + w_i\}$ and $w_{min} = \min_i\{w + w_i\}$. The top level edge has cost c, the second level edges have cost from w_1 to w_{3m}, and the leaves have weights from w_1 to w_{3m}. Now we are going to show the 3-partition problem has solution if and only if the optimal logical tree has certain cost.

If we make c really huge, then the cost of optimal logical tree is smaller than $c \cdot 3n\log_3 n \cdot w_{max}$. And by lemma 14 and 15, we know any other logical tree, which is not a degree-3 tree, has cost greater or equal to $c \cdot (3n\log_3 n + 1)\cdot w_{min}$. Since we make $\frac{w_{max}}{w_{min}} < \frac{3n\log_3 n+1}{3n\log_3 n}$, we know the optimal logical tree has to be a degree-3 tree. So in the optimal logical tree, the cost caused by the first level edge in the routing tree is exactly $c \cdot 3n\log_3 n \cdot W$ where $W = \sum_i w_i$, which is the same for all degree-3 logical trees.

Now we are going to look at the cost caused by the second level edges in the routing tree. The cost of the first level nodes, root's direct children, is W^2. The cost of the second level nodes is $W_{11}^2 + W_{12}^2 + W_{13}^2$, where W_{1i} is the total weight of the leaves in root's i^{th} child's subtree, and $W_{11} + W_{12} + W_{13} = W$. The cost of the third level nodes is $W_{21}^2 + W_{22}^2 + \cdots + W_{29}^2$, and $W_{21} + W_{22} + \cdots + W_{29} = W$. Similarly, for the j^{th} level, the cost is $W_{j1}^2 + W_{j2}^2 + \cdots + W_{j3^j}^2$ and $W_{j1} + W_{j2} + \cdots + W_{j3^{j-1}} = W$, $j = 1, 2, \ldots, \log_3 3m$. So the cost caused by the second level edges in the routing tree is minimized when $W_{11} = W_{12} = W_{13} = W/3$, $W_{21} = W_{22} = \cdots = W_{29} = W/9$, $\ldots$, $W_{\log_3 3m,1} = W_{\log_3 3m,2} = \cdots = W_{\log_3 3m,m} = W/m = B$. And this cost is $W^2 \cdot (1 + 1/3 + 1/9 + \cdots + 1/m)$, which only the logical tree that cooresponds to the solution of the 3-partition problem has. So

3-partition problem has solution if and only if the optimal logical tree has cost $c \cdot 3n \log_3 n \cdot W + W^2 \cdot (1 + 1/3 + 1/9 + \cdots + 1/m)$. □

7 Approximation of weighted version

In this section we are going to give approximation algorithms for logical key tree problem where different members have different probabilities (weights) to join and leave the group. The underlining routing structure will be routing tree and routing graph respectively.

First we need to define how to calculate the costs of logical trees in this model. Similarly to the case with uniform weight, the cost of logical tree can be represented as

$$\sum_{v \in T \ \ v \neq root} w\left(T_{p_v}\right) \cdot M(v)$$

where T_{p_v} is the subtree induced by node v's parent, $w\left(T_{p_v}\right)$ is the weight sum of all the leaves in the tree T_{p_v}, and $M(v)$ is the multicast cost to the members in the subtree rooted at v.

Theorem 8. *There is always a binary logical tree whose cost is within three times the optimal one.*

Proof. Similar to the one for uniform case. □

7.1 Routing tree with weighted members

We are going to give an approximation algorithm for the routing tree with weighted members. The algorithm is very similar to the one for uniform weight case. The difference is when we find the partition node in each iteration, instead of partitioning the number of members, we should partition the weight. If we couldn't find a balanced partition, then there is a single heavy node with weight more than 2/3 of the total weight. In this case let X be a singleton set that contains this heavy node. If $c \leq M(S)/4$, then call the algorithm recursively on both X and Y; if $c > M(S)$, then call the algorithm recursively only on Y and use Huffman Tree on X. The following shows that this is a constant approximation algorithm.

Lemma 17. *If the multicast to every subset of leaves is the same, then Huffman Tree yields the optimal binary logical tree.*

Proof. Huffman Tree yields the minimum average depth of leaves, which in turn shows this is the optimal binary logical tree. □

Lemma 18. *When $c > M(S)/4$, using Huffman Tree on X is a 12-approximation of the optimal logical tree on X.*

Proof. Since $c > M(S)/4$, the multicast cost to any subset of X is between $M(S)/4$ and $M(S)$. By lemma 17, we can see Huffman Tree is at most 4 times the optimal binary logical tree. And by theorem 8, we can conclude that this Huffman Tree is 12-approximation of the optimal logical tree. □

Lemma 19. $ALG(S) = ALG(X) + ALG(Y) + w(S)\,[M(X) + M(Y)]$

Lemma 20. $OPT(S) \geq OPT(X) + OPT(Y)$

Theorem 9. *This algorithm has a constant approximation ratio.*

Proof. We are going to prove this by induction on the number of members. And the induction hypothesis is $ALG(S) \leq \alpha \cdot OPT(S) + \beta \cdot w(S)M(S)$. There are three cases. The first case is $c \leq M(S)/4$ and the partition is balanced.

$$\begin{aligned}
ALG(S) &= ALG(X) + ALG(Y) + w(S)\,[M(X) + M(Y)] \\
&\leq \alpha \cdot OPT(X) + \beta \cdot w(X)M(X) + \alpha \cdot OPT(Y) + \beta \cdot w(Y)M(Y) \\
&\quad + w(S)\,[M(X) + M(Y)] \\
&\leq \alpha\,[OPT(X) + OPT(Y)] + \left(\frac{2}{3}\beta + 1\right) w(S)\,[M(X) + M(Y)] \\
&\leq \alpha \cdot OPT(S) + \left(\frac{2}{3}\beta + 1\right) w(S)\,[M(S) + c] \\
&\leq \alpha \cdot OPT(S) + \frac{5}{4}\left(\frac{2}{3}\beta + 1\right) w(S)M(S) \\
&\leq \alpha \cdot OPT(S) + \beta \cdot w(S)M(S)
\end{aligned}$$

as long as $\frac{5}{4}\left(\frac{2}{3}\beta + 1\right) \leq \beta$, which is $\beta \geq 7.5$.

The second case is $c > M(S)/4$ and the partition is balanced. In this case, we only call the algorithm recursively on Y and use Huffman Tree on X. So we have the following.

$$\begin{aligned}
ALG(S) &= Huff(X) + ALG(Y) + w(S)\,[M(X) + M(Y)] \\
&\leq 12 \cdot OPT(X) + \alpha \cdot OPT(Y) + \beta \cdot w(Y)M(Y) + w(S)\,[M(X) + M(Y)] \\
&\leq \alpha\,[OPT(X) + OPT(Y)] + \frac{2}{3}\beta w(S)M(S) + 2w(S)M(S) \\
&\leq \alpha \cdot OPT(S) + \left(\frac{2}{3}\beta + 2\right) w(S)M(S) \\
&\leq \alpha \cdot OPT(S) + \beta \cdot w(S)M(S)
\end{aligned}$$

as long as $\alpha \geq 12$ and $\frac{2}{3}\beta + 2 \leq \beta$ which is $\beta \geq 6$.

The third case when the partition is not balanced. In this case, our algorithm connect the heavy node directly to the root of logical tree. So we

have the following.

$$\begin{aligned} ALG(S) &= ALG(Y) + w(S)\left[M(X) + M(Y)\right] \\ &\leq \alpha \cdot OPT(Y) + \beta \cdot w(Y)M(Y) + w(S)\left[M(X) + M(Y)\right] \\ &\leq \alpha \cdot OPT(S) + \frac{1}{3}\beta w(S)M(S) + 2w(S)M(S) \\ &\leq \alpha \cdot OPT(S) + \left(\frac{1}{3}\beta + 2\right) w(S)M(S) \\ &\leq \alpha \cdot OPT(S) + \beta \cdot w(S)M(S) \end{aligned}$$

as long as $\frac{1}{3}\beta + 2 \leq \beta$ which is $\beta \geq 3$. In sum, this is a 19.5-approximation algorithm. □

7.2 Routing graph with weighted members

We are going to give an approximation algorithm for the routing graph with weighted members. We use the same idea as the approximation algorithm for uniform case. The algorithm is shown as follows.
Input: S, which is a set of vertices in the routing graph.
Algorithm:

1. Compute the complete graph on $S \cup \{root\}$. The weight of an edge (u, v) is the length of shortest path between u and v in the original routing graph.

2. Compute the MST on this complete graph. Call it MST(S).

3. Based on MST(S), compute the LAST, which is an (α, β)-approximation of SPT and MST(S).

4. Using this LAST to find the partition node and split S into X and Y. If we couldn't find balanced partition, which means there is a heavy node with more than 2/3 the total weight, let X be the singleton set that contains this heavy node.

5. Call the algorithm recursively on X if $c \leq M(S)/4$. Otherwise use Huffman Tree as the logical tree for X. This step returns T_1.

6. Call the algorithm recursively on Y, which returns T_2.

7. Return the binary logical tree constructed using T_1 and T_2, shown in Figure 24.

Now we are going to prove this is a constant approximation algorithm. And again we are using the LAST which is a $\left(1 + \sqrt{2}\gamma, 1 + \sqrt{2}/\gamma\right)$-approximation in the following proof. Notice, lemma 13, lemma 19 and lemma 20 still hold here.

Lemma 21. *When $c > M(S)/4$, using Huffman Tree on X is a constant approximation of the optimal logical tree on X.*

Proof. Since $c > M(S)/4$, the distance from the root to any element in X is at least $\frac{c}{1+\sqrt{2}\gamma} = \frac{M(S)}{4(1+\sqrt{2}\gamma)}$. So the multicast cost to any subset of X is between $\frac{M(S)}{4(1+\sqrt{2}\gamma)}$ and $M(S)$. By lemma 17, we can see Huffman Tree is at most $4\left(1+\sqrt{2}\gamma\right)$ times the optimal binary logical tree. And by theorem 8, we can conclude that this Huffman Tree is $12\left(1+\sqrt{2}\gamma\right)$-approximation of the optimal logical tree. □

Theorem 10. *This algorithm has a constant approximation ratio.*

Proof. We are going to prove this by induction on the number of members. And the induction hypothesis is $ALG(S) \leq \alpha \cdot OPT(S) + \beta \cdot w(S)M(S)$. There are three cases. The first case is $c \leq M(S)/4$ and the partition is balanced.

$$\begin{aligned}
ALG(S) &= ALG(X) + ALG(Y) + w(S)\left[M(X) + M(Y)\right] \\
&\leq \alpha \cdot OPT(X) + \beta \cdot w(X)M(X) + \alpha \cdot OPT(Y) + \beta \cdot w(Y)M(Y) \\
&\quad + w(S)\left[M(X) + M(Y)\right] \\
&\leq \alpha\left[OPT(X) + OPT(Y)\right] + \left(\frac{2}{3}\beta + 1\right) w(S)\left[M(X) + M(Y)\right] \\
&\leq \alpha \cdot OPT(S) + \left(\frac{2}{3}\beta + 1\right) w(S)\left[\left(1+\sqrt{2}/\gamma\right) M(S) + c\right] \quad \text{(lemma 13)} \\
&\leq \alpha \cdot OPT(S) + \left(\frac{5}{4} + \sqrt{2}/\gamma\right)\left(\frac{2}{3}\beta + 1\right) w(S)M(S) \\
&\leq \alpha \cdot OPT(S) + \beta \cdot w(S)M(S)
\end{aligned}$$

as long as $\left(\frac{5}{4} + \sqrt{2}/\gamma\right)\left(\frac{2}{3}\beta + 1\right) \leq \beta$.

The second case is $c > M(S)/4$ and the partition is balanced. In this case, we only call the algorithm recursively on Y and use Huffman Tree on X. So we have the following.

$$\begin{aligned}
ALG(S) &= Huff(X) + ALG(Y) + w(S)\left[M(X) + M(Y)\right] \\
&\leq 12\left(1+\sqrt{2}\gamma\right) \cdot OPT(X) + \alpha \cdot OPT(Y) + \beta \cdot w(Y)M(Y) + w(S)\left[M(X) + M(Y)\right] \\
&\leq \alpha\left[OPT(X) + OPT(Y)\right] + \frac{2}{3}\beta w(S)M(S) + 2w(S)M(S) \\
&\leq \alpha \cdot OPT(S) + \left(\frac{2}{3}\beta + 2\right) w(S)M(S) \\
&\leq \alpha \cdot OPT(S) + \beta \cdot w(S)M(S)
\end{aligned}$$

as long as $\alpha \geq \left(1+\sqrt{2}\gamma\right)$ and $\beta \geq 6$.

The third case when the partition is not balanced. In this case, our algorithm connect the heavy node directly to the root of logical tree. So we have the following.

$$\begin{aligned} ALG(S) &= ALG(Y) + w(S)\,[M(X) + M(Y)] \\ &\leq \alpha \cdot OPT(Y) + \beta \cdot w(Y)M(Y) + w(S)\,[M(X) + M(Y)] \\ &\leq \alpha \cdot OPT(S) + \frac{1}{3}\beta w(S)M(S) + 2w(S)M(S) \\ &\leq \alpha \cdot OPT(S) + \left(\frac{1}{3}\beta + 2\right) w(S)M(S) \\ &\leq \alpha \cdot OPT(S) + \beta \cdot w(S)M(S) \end{aligned}$$

as long as $\beta \geq 3$. So, this algorithm has a constant approximation. □

8 Further directions

In this section, we keep track of further directions of this logical key tree problem.

1. Logical key tree with depth constraint.
2. The underlining routing trees are dynamically changing. How to maintain the logical trees.
3. How to construct a tree which is a good representation of the routing graph.

References

[1] Samir Khuller, Balaji Raghavachari, and Neal E. Young. Balancing minimum spanning trees and shortest-path trees. *Algorithmica*, 14(4):305–321, 1995.