Shared Tree for Application-layer Multi-source Multicast

Juncheng Jia and Qian Zhang
Department of Computer Science and Engineering
Hong Kong University of Science and Technology, Hong Kong, China
{jiajc, qianzh}@cse.ust.hk

Abstract—Group communication is becoming increasingly important with the decreasing cost of broadband access and the growing number of Internet users. Many of today’s content distribution applications, such as video conferencing, distance learning, multi-party network games, etc., require efficient support for group communications. In these applications, a number of participants may have source data (multi-source) to send and each of them wants to distribute its content to all the other nodes in the system. This calls for the efficient multi-source multicast support. Then a question emerges naturally: exactly how many sources can be served simultaneously and how to construct the shared tree in order to approach the theoretical optimal. The simulation results manifest that our proposed heuristic protocol significantly increases the number of sources that can be served in multi-source sessions and the result approaches to the optimal solution.

I. INTRODUCTION

Content distribution is becoming increasingly important with the decreasing cost of broadband access and the growing number of Internet users. Many of today’s content distribution applications, such as video conferencing, distance learning, multi-party network games, etc., require efficient support for group communications. In these applications, a number of participants may have source data (multi-source) to send and each of them wants to distribute its content to all the other nodes in the system. This calls for the efficient multi-source multicast support. Then a question emerges naturally: exactly how many sources can be served simultaneously by the network and further how to provide a service that can serve as many sources as possible?

Due to the limited deployment of network layer multicast infrastructure, application-layer multicast becomes an alternative and turns out to be a quite promising solution in many cases. In an end-system multicast approach participating peers organize themselves into an overlay topology and leverage unicast for data delivery. However, when multi-source multicast is running over the application layer, there are some challenges we must deal with. First, because application-layer multicast relies on end-systems to relay the packets, high bandwidth requirement is imposed on the end-systems. Simultaneous data transmission from multiple sources pushes the bandwidth requirement even higher. Second, the capabilities of end-systems cover a rather wide range in the real Internet. Thus, the decision for the relay nodes in the tree structure should take this node heterogeneity into account.

Multi-source multicast for group communication can be realized in several different ways. Two approaches are in common use. One is to use a single distribution tree shared by all senders (sources). An alternative is to employ one statically allocated distribution tree for each sender for the duration of a session. In this paper, we will focus on using one single shared tree to support multi-source multicast.

Recently, a number of researchers have been investigating services and applications based on overlay networks [1] to provide application-layer multicast or content distribution. These include Yoid [9], HMTP [6], Overcast [8], TMesh [7] and the like. Specific for the multi-source scenario, Narada [2] is a mesh-first, multi-source multicast system designed for small to medium sized multicast groups. Source specific trees are built on top of the mesh for every sender. NICE [3] adopts an implicit hierarchical cluster-based approach to create a control topology among the participant peers. The data delivery topology is implicitly determined by a set of data-forwarding rules. Its protocol uses the system-wide cluster size as one parameter to bound the fan-out of the overlay, rather than the actual fan-out capability for each individual node. This makes it suboptimal in terms of network capability utilization. Based on NICE, FatNemo [4] proposes the idea of emulating fat-tree in overlays for multi-source group communication. It organizes participant end-systems in a tree that closely resembles a Leiserson fat-tree by dynamically placing the nodes with higher bandwidth capacity close to the root and increasing the cluster size as one ascends the tree. It is specifically geared toward allowing multi-source overlay multicast over the tree. However, none of the above mentioned schemes have answered the question of how many sources can be served concurrently and how to construct the shared tree so that it can serve as many sources as possible.

To answer the above questions, in this paper we use two metrics to build a shared tree: maximizing the number of served sources and maximizing the minimum residual capability (specifically the bandwidth capability) of all nodes. By maximizing the number of served sources, we want to take the best usage of the network capability while satisfying the concurrent requirement of different applications; by maximizing the residual bandwidth capability, we try to avoid bottlenecks in constructing the shared tree and leave more space for further incoming sources. In our study, we demonstrate that given global knowledge, there exists an optimal solution to build such a shared tree. The interesting observation we have obtained is that this single shared tree has a rather good feature...
in that it can support at least \( m - 1 \) sources, where \( m \) is the maximum number of sources that can be supported with any number of trees. We further propose a heuristic to build the shared tree only based on local knowledge. A two-phase tree construction is used, which consists of a node joining phase and a local tree improvement phase. The basic idea is to avoid bottleneck nodes whose bandwidth capabilities are easily used up. In the first phase, when each node joins the system, it will choose a parent from a list of candidates to attach to, which can avoid bottlenecks. For the second phase, each node carries out the local tree improvement algorithm, which applies the above optimal solution on the local range. The improvement considers several possible local tree rebuilding schemes and chooses the one with the best performance while ensuring the connectivity of the entire tree. The simulation is conducted to show the effectiveness of our proposed heuristic.

The rest of the paper is structured as follows. In the following, we detail the problem statement. An optimal solution and the theoretical analysis of its property are described in Section III. Section IV presents the distributed heuristic. We evaluate our work with the simulation in Section V. Finally, we conclude the paper and discuss future work in Section VI.

II. PROBLEM STATEMENT

The network model we use is based on the observation that the loss on the backbone links is much smaller compared to the average loss seen by a receiver [5]. We assume the Internet core consisting of routers has infinite capability. End-systems, called nodes, are connected to the Internet core via access links which have limited bandwidth capacities. Under this assumption, the capacity constraints are all imposed on the bandwidth of access links. Capability of a node is used to describe the total bandwidth of all possible access links (including upload and download links) passing through this node. In a multi-source multicast session, a portion of participants want to deliver their content data to all the other nodes and the data transmissions happen simultaneously. The data rate of each source can be different but must be smaller than the capability of the corresponding sender. Under this network model, our target is to build a single shared tree among all the participants such that as many sources as possible can be served and the minimum residual capability of each node is maximized.

To formulate the problem, we use \( V \) to denote the set of nodes with size \( n \). Each node \( v_i \) is associated with its capability \( C(v_i) \). \( S \) is a subset of \( V \) with size \( k \), representing the set of sources who want to simultaneously deliver data to all the nodes in \( V \). Each source \( s_i \) has a data rate \( R(s_i) \) and \( R(s_i) \leq C(s_i) \).

The goal is to build up a single bidirectional spanning tree \( T = (V, E) \), \( E \subseteq V \times V \) to connect the nodes \( V \) for data transmissions, such that the tree can serve a maximum number of sources in \( S \), based on which the minimal residual capability of each node is maximized under each node’s capability constraint.

Denote the degree of node \( v_i \) in the tree as \( d(v_i) \). \( A \) is the set of sources using the tree for delivery. Denote the residual capability for any node with:

\[
RC(v) = C(v) - d(v) \sum_{s \in A} R(s), \forall v \in V.
\]

Then the problem can be formally described to find \( T \) such that:

\[
\begin{align*}
\text{maximize} & \quad |A|, \\
\text{minimize} & \quad \min_{v \in V} RC(v), \\
\text{subject to} & \quad RC(v) \geq 0, \forall v \in V, \\
& \quad d(v) \in \mathbb{Z}^+, \forall v \in V, \\
& \quad \sum_{v \in V} d(v) = 2n - 2.
\end{align*}
\]

One thing that should be stressed is that the two maximization targets have different priorities: maximize \( |A| \) is prior to

maximize \( \min_{v \in V} RC(v) \).

III. THEORETICAL ANALYSIS

In this section, we first give out a greedy solution for the single tree construction problem and prove this solution is actually the optimal one. Then the property of this single tree is depicted.

A. One Greedy Solution

To solve the problem of maximizing the number of served sources and maximizing the minimum residual capability of each node subject to individual capability constraint, we propose a greedy solution. It first chooses small data rate sources, then repeatedly assigns a degree to the node with the maximum residual capability. We elaborate on this basic idea in detail in the following.

The first step is to determine the maximum number of sources the overlay can support with a single tree. We use a greedy method to choose the sources with small data rate first. We use \( SR \) to denote the data rate sum of accepted sources, which is 0 initially. Whenever we choose the source with the \( i \)th smallest data rate, we add this data rate to \( SR \). We use \( SR \) to check the maximum degree for each node to see whether they can build up a spanning tree. If it is possible, then we go on to check the node with the next largest data rate; if not, then we end with the maximum number of sources we can serve. The pseudo-code for this algorithm is shown in the line 2-13 of Fig. 1.

The next step is to construct the single tree which will have the maximum minimum residual capability, as shown in the line 15-24 of Fig. 1. Initially, the degree of each node is set to 1, \( d(v_i) = 1 \), since each node has at least one degree to be connected. Meanwhile, the \( RC \) value of each node is updated. Then we repeatedly increase the degree of the node with the maximal residual capability until the sum of the degree equals to \( 2n - 2 \), which means the degree assignment finishes. In each repetition, the \( RC \) value of each node is also updated.

Finally, given each node’s degree in the tree, we can rather easily build up such a tree. For the running time, it needs...
1: /*determine the maximum number of served sources*/
2: sort S in an order that \( i < j \), \( R(s_i) \leq R(s_j) \)
3: \( A = \emptyset \)
4: \( SR \leftarrow 0 \)
5: for \( i = 1 \) to \( k \) do
6: /*handle the source with the current smallest data rate*/
7: \( SR \leftarrow SR + R(s_i) \)
8: if \( SR \leq \min_{v \in V} C(v) \) and \( \sum_{v \in V} \left\lfloor \frac{C(v)}{SR} \right\rfloor \geq 2n - 2 \) then
9: \( A \leftarrow A \cup \{ s_i \} \)
10: else
11: break
12: end if
13: end for
14: /*build a tree with max residual capability*/
15: \( SR \leftarrow \sum_{s \in A} R(s) \)
16: for \( i = 1 \) to \( n \) do
17: \( d(v_i) \leftarrow 1 \)
18: \( RC(v_i) \leftarrow RC(v_i) - SR \)
19: end for
20: while \( \sum_{v \in V} d(v) < 2n - 2 \) do
21: \( v_m = \arg \max_{v \in V} RC(v) \)
22: \( d(v_m) \leftarrow d(v_m) + 1 \)
23: \( RC(v_m) \leftarrow RC(v_m) - SR \)
24: end while

Fig. 1: Algorithm for optimal solution

\( O(kn) \) time to determine the maximum number of served sources and \( O(n^2) \) time to assign the degrees. Since \( k < n \), the total running time is \( O(n^2) \).

B. Proof of Optimal Solution

Theorem 1: The greedy selection of sources always gives out the optimal number of sources that can be concurrently supported.

Proof: Suppose a higher data rate node is chosen instead of a lower rate one in an optimal solution, we can just select the lower one and let the one with a higher rate out. This results in more residual capability with each node while ensuring the number of served sources remains unchanged. We repeat the switch until all the lower rate ones are selected. At this time, it is possible to fill in another source; or if it is impossible, at least the residual capability becomes larger. In both cases, the greedy method is better than the optimal, which is a contradiction. \( \square \)

Theorem 2: The greedy degree assignment is optimal and the minimum residual capability of the nodes is maximized.

Proof: Since the degree of each node is at least 1, the actual degrees that need to be assigned are \( n - 2 \), which is illustrated in the line 20-24 of Fig. 1. Here an induction is used to prove this greedy assignment scheme is optimal.

For the basis case, \( \sum_{v \in V} d(v) = n \) at the beginning, which is an optimal case.

For the induction step, we show that if the degree assignment is optimal for \( n \leq \sum_{v \in V} d(v) < K \), by using the greedy method, the degree assignment is also optimal for \( \sum_{v \in V} d(v) = K \). Let \( \text{Min} \text{RC}_{K-1} \) be the minimal \( RC(v) \) for all \( v \in V \), when \( \sum_{v \in V} d(v) = K - 1 \). After the optimal assignment of \( (K - 1) \)th degree, \( \text{Min} \text{RC}_{K-1} \) is maximized. To assign the \( K \)th degree: (1) If \( \max_{v \in V} RC(v) - SR \geq \text{Min} \text{RC}_{K-1} \), the greedy method assigns 1 degree to the node with maximal \( RC \). Then \( \text{Min} \text{RC}_K = \text{Min} \text{RC}_{K-1} \) and \( \text{Min} \text{RC}_K \) is maximized. Otherwise, if an optimal solution results in \( \text{Min} \text{RC}_K' > \text{Min} \text{RC}_K \), since \( \text{Min} \text{RC}_{K-1} \geq \text{Min} \text{RC}_{K-1}' \), then \( \text{Min} \text{RC}_{K-1} > \text{Min} \text{RC}_{K-1}' \). Contradiction. (2) If \( \max_{v \in V} RC(v) - SR < \text{Min} \text{RC}_{K-1} \), after the greedy assignment, \( \text{Min} \text{RC}_K = \max_{v \in V} RC(v) - SR \) and it is maximized. Otherwise, if an optimal solution results in \( \text{Min} \text{RC}_K' > \text{Min} \text{RC}_K \), then \( \text{Min} \text{RC}_K' + SR > \max_{v \in V} RC(v) \), which means \( \exists v' \in V, RC(v') > \max_{v \in V} RC(v) \). Contradiction. \( \square \)

C. Property of Optimal Solution

The optimal solution shown above has a nice feature that if the overlay network can support at most \( m \) sources with any number of trees, then the shared tree built by the our optimal solution can support at least \( m - 1 \) sources. The reason is described as follows.

Consider the point when the above tree can not support source \( s_t \) with the next largest data rate, we are faced with one of three situations:

1) \( \sum_{i=1}^{n} RC(v_i) < (2n - 2)R(s_t) \) \hspace{1cm} (3)

This means the total residual capability of the overlay can not support source \( s_t \). We can do nothing even if we use multiple trees.

2) \( \sum_{i=1}^{n} RC(v_i) \geq (2n - 2)R(s_t) \)

\hspace{1cm} \forall v \in V, RC(v) < R(s_t) \hspace{1cm} (4)

Again, no further improvement is possible, because the overlay has enough total bandwidth, but there exist some inherent bottlenecks.

3) \( \sum_{i=1}^{n} RC(v_i) \geq (2n - 2)R(s_t) \)

\hspace{1cm} \forall v \in V, RC(v) \geq R(s_t) \)

\hspace{1cm} \sum_{i=1}^{n} \left| \frac{RC(v_i)}{R(s_t)} - 1 \right| < 2n - 2 \hspace{1cm} (5)

This means there may exist some small fraction of capability among nodes that we can utilize. If that is the case, we can modify the current tree to collect such unutilized capability into a few nodes so that source \( s_t \) can also be served.

From situation 3, because \( \sum_{i=1}^{n} \left| \frac{RC(v_i)}{R(s_t)} - 1 \right| < 2n - 2 \), we get \( \sum_{i=1}^{n} RC(v_i) < (3n -
2) \( R(s_t) \). So finally:

\[
(2n - 2)R(s_t) \leq \sum_{i=1}^{n} RC(v_i) < 2(2n - 2)R(s_t),
\]

which means with the help of additional trees, we can satisfy at most one more source. After we build other trees and fill the data stream on these trees, the total residual capability is:

\[
\sum_{i=1}^{n} RC(v_i) < nR(s_t) < (2n - 2)R(s_{t+1}).
\]

As a result, this approach can be one of the optimal solutions for our first requirement: maximize \( |A| \). This property means theoretically using a single shared tree is not a bad solution. However, to achieve such an optimal single shared tree is not easy in a distributed manner due to the lack of global information.

IV. DISTRIBUTED HEURISTIC

Our distributed approach aims to build a single data delivery tree for the multi-source multicast session only based on local information. The construction includes 2 stages: node joining and tree improvement. In each stage, local information is used to achieve a local optimal. The metric is to avoid bottleneck, which is in turn to maximize the bandwidth on each link in the local range. By this, the best effort is made to serve more sources. The details are given in the following sub-sections.

A. Node Joining

The single tree is built up as each node joins. The first node becomes the root of the tree. The protocol assumes that a well-known Rendezvous Point (RP) exists, acting as the bootstrapping node. It maintains the IP addresses of joined members. When a new node wants to join the session, it first contacts the RP. The RP randomly chooses several existing members as the candidate list (CL) and gives the list to the new coming node. From this CL, the new node makes the decision of which node is its parent in the tree. The new node sends a query packet to each candidate. Then each candidate replies with its capability and current degree in the tree. With this information, the new node selects one candidate to attach to and meanwhile the degree of this candidate should be updated.

Since at the node joining stage early joined nodes do not know about later joined sources, the decision made by a new node does not rely on the information of sources. Instead, the decision criteria is that \( \min_{t \in CL} \frac{\text{capability}_t}{\text{degree}_t} \) should be maximized. By this, we make sure a local area consisting of the candidates can support the most number of future sources.

After joined the network, each node maintains a parent candidate list, CL, in addition to its parent and its children list. For the new node, the list got from the RP becomes the initial parent candidate list.

B. Local Tree Improvement

Since the node joining process only achieves local optimal results, it may cause some nodes to become the "hot-spots" and eventually the bottlenecks globally. So our protocol relies on a local tree improvement process to tune the tree from time to time. After the node joining period, each node will carry out the tree improvement process periodically. Again, the improvement is based on local information available to the improvement-initializing node. Specifically, the local adjusting range is limited to one-hop distance from that node in our work.

In a sample improvement case illustrated in Fig. 2, there are 3 sources \((s_1, s_2, s_3)\) whose data rates are 100, 100, 200 respectively. The original tree is shown as in Fig. 2a, where node X has 3 children \((Ch_1, Ch_2, Ch_3)\) and has the capability of 400. X is the bottleneck in this setup, so only \(s_1\) can be served. When X initiates the improvement process, it sends query packets to its neighbors, including its parent \((P)\), children \((Ch_1, Ch_2, Ch_3)\) and candidates in the CL \((C_1, C_2)\). Each of these neighbors responses with its capability and degree. We assume the information of all sources (e.g. data rate) is available for each node.

There are two types of configurations for the improvement, depending on whether candidates in the CL of X are involved (see Fig. 2b and Fig. 2c). In Fig. 2b, the candidates of X are not involved in the improvement. With the knowledge of local nodes (including the parent, children and itself) and all sources, X uses the algorithm similar to the optimal one proposed in Fig. 1. That is to build a small tree on the local nodes such that the number of served sources is maximized and further the minimum residual capability of the local nodes are maximized. The only difference from Fig. 1 is that the original degree of the local nodes must be taken into consideration. As shown in Fig. 2b, both \(s_1\) and \(s_2\) can be served now and the minimum local residual capability is 200.

The other configuration is to involve the candidates in CL of X into the improvement (as shown in Fig. 2c). Now we must ensure the topology after improvement is still a tree, i.e., connected and loop-free. Our scheme is to let X attach to one candidate in CL and become a leaf node. Meanwhile, reconnect X’s original parent P and X’s original children \(Ch_1, Ch_2, Ch_3\) to a local small tree. Again a modified algorithm similar to Fig. 1 is used to achieve to the maximum number of served sources by involved nodes and minimum residual capability of the involved nodes. As shown in Fig. 2c, all \(s_1, s_2\) and \(s_3\) can be served now.

The final potential improvement is to choose the best one from the above two configurations. X determines whether to take it by comparing the potential improved result with the current situation. If the number of served sources can be increased, then X takes improvement; if served number is unchanged but maximum minimum residual capability can be enlarged, then X also take the improvement. In Fig. 2, configuration c will be chosen. X controls the whole improvement. When it finishes the computation, it sends a
specific improvement message to related neighbors. According to the message, the neighbors perform the adjustment.

V. EXPERIMENTAL RESULTS

We use a custom written simulator to evaluate our design. We compare the system performance with 4 different tree construction schemes: random scheme means the tree is built up randomly; optimal scheme uses the optimal solution proposed in Section III with global knowledge; distributed scheme constructs the tree with all nodes join and find their parents with the local information; improved scheme improves the tree set up with distributed scheme applying local tree improvement. Two metrics are used to evaluate the performance of the constructed tree topology: the number of accepted sources and minimum residual capability. For the capability of the nodes, we use a uniform distribution over the interval [1Mbps, 5Mbps], which is reasonable in reality. One round of improvement is simulated, in which each node has one chance to do improvement. Each test case runs for 100 times and the mean results are plotted in figures.

In Fig. 3 and Fig. 4, each node has 5 candidates when joining and each node can be the source. The data rate of sources are 128Kbps and 256Kbps respectively. The result of the random scheme is bad which is not surprising. The distributed scheme on average has a difference of 2 from the optimal solution in terms of the number of accepted sources. The improved one can normally improve the distributed scheme by 0.5. From those two figures, we can see that the heuristic approach gives a fairly good result in terms of the accepted sources.

From the minimum residual capability aspect, Fig. 6 shows the comparison results. Here we limit the total source number to be 1 for Fig. 6(a) and 2 for Fig. 6(b), while the data rate of each source is 128Kbps. The total number of nodes is 500 and each node has 3 candidates. For those trees which can accept the single source in Fig. 6(a), the minimum residual capability obtained using our scheme is almost the same as the one from the optimal solution. In Fig. 6(b), the comparison results for trees which can accept 2 sources are shown. Again our scheme is approaching the optimal.

We also investigate the impact of the candidate number used in the heuristic. The node number is set to be 500 and data rate is 128Kbps. As shown in Fig. 5, when only one candidate is given, the improvement is about 0.7 in terms of the number of accepted sources. As we increase the number of candidates, the performance improvement gain becomes larger, because nodes have more choices when taking local improvement. However, when candidate number increases further, the improvement gain decreases, since nodes can make better decisions with more knowledge when joining.

In summary, through the above simulation results, we can see that our proposed distributed heuristic can achieve rather good performance in terms of both accepted source number and residual capability, which are important for providing good support for multi-source multi-cast applications.

VI. CONCLUSION

In this paper, we target at providing efficient support of the multi-source multicast in the application layer. We propose a single shared tree construction scheme to serve as many sources as possible. An optimal solution for the single tree is given and proved to have a rather good feature. A distributed heuristic protocol is devised to approach the optimal with local information. Using simulation, we show that the results are quite good in terms of the number of accepted sources and minimum residual capability. In the future, we will conduct more comprehensive simulations and investigate other tree-first schemes for the application-layer multi-source multicast.

REFERENCES


