INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Encodings of Multicast Trees

Vijay Arya — Thierry Turletti — Shivkumar Kalyanaraman

N° ????

Dec 2004

Thème COM

Rapport de recherche

N 0249-6399

Encodings of Multicast Trees

Vijay Arya , Thierry Turletti , Shivkumar Kalyanaraman*

Thème COM — Systèmes communicants
Projet Planète

Rapport de recherche n° ???? — Dec 2004 — 21 pages

Abstract: In this paper, we present efficient ways of encoding multicast trees. Multicast tree encodings provide a convenient way of performing stateless and explicit multicast routing in networks and overlays. We show the correspondence of multicast trees to theoretical tree data structures and give lower bounds on the number of bits needed to represent multicast trees. Our encodings can be used to represent multicast trees using both node identifiers and link indexes and are based on balanced parentheses representation of tree data structures. These encodings are almost space optimal and can be read and processed efficiently. We evaluate the length of these encodings on multicast trees in generated and real topologies.

Key-words: Explicit Multicast, Multicast Tree Encoding, Multicast state Reduction

* Dept. of ECSE, Rensselaer Polytechnic Institute, NY USA

Encodage des Arbres de Transmission Multipoint

Résumé : Dans ce rapport, on présente différents procèdés pour encoder les arbres de transmission multipoints. L'encodage des arbres est une manière efficace de faire du routage multipoint explicite et sans état dans les routeurs. Il peut être utilisè à la fois pour du multipoint natif et du multipoint applicatif. Avec une étude théorique, on donne des bornes inférieures sur le nombre de bits nécessaires pour représenter les arbres multipoints. Les techniques d'encodage que l'on décrit peuvent être utilisées pour représenter de manière efficace les arbres multipoints en utilisant des identificateurs de noeuds ou de liaisons et sont basées sur la structure de représentation d'arbres avec des parenthèses. On donne une évaluation des performances obtenues pour des arbres multipoints générès artificiellement et pour de réelles topologies.

Mots-clés : encodage des arbres multipoint, multipoint explicite, réduction d'état multipoint

1 Introduction

Current IP multicast routing protocols such as DVMRP [22] and PIM [10, 9] install a per group state in the routers. To support these protocols, a router needs to maintain a multicast forwarding table entry corresponding to every multicast group whose distribution tree passes through the router. Since the entries corresponding to different multicast groups cannot be aggregated, the forwarding table size grows with the number of active multicast groups, violating the stateless principle followed by routers. As the number of multicast groups increase, the cost and performance of routers are adversely affected. The state maintenance problem is further aggravated by the fact that even small multicast groups can introduce significant amount of state into the network. The amount of state introduced by a multicast group in the network depends on the size of its tree. The size of the multicast tree in turn is a function of both the relative locations of the source and various receivers, and the number of receivers itself. A small multicast group consisting of receivers which are spread uniformly in the Internet may introduce more state than a large multicast group concentrated in one region of the Internet. Thus, even a large number of small multicast groups may cause a state explosion. In order to reduce the state overhead at routers, one of the approaches followed is to move the state from the network to the packet, i.e., to use a representation of the multicast tree within every data packet. Each router can then function in a stateless manner and perform multicast forwarding by reading and processing packet headers. To use this approach even in small to medium sized multicast groups, a compact encoding of multicast tree is required.

Due to deployment problems in Inter-domain Multicast, several Application Level Multicast protocols have been proposed [19, 24]. However, the problem of per group state maintenance remains even in application level multicast. Scribe [19] and Bayeux [24] are application level multicast protocols for Pastry [18] and Tapestry [23] overlays respectively. Pastry and tapestry are large scale DHT overlays which allow the interconnection and routing between millions of nodes in a scalable manner. To implement multicast in these overlays, both Scribe and Bayeux introduce a per group multicast state within the overlay nodes which are part of the multicast distribution tree. Due to constraints of per group state maintenance, these overlays cannot support a large number of multicast groups (small or big). State maintenance is a serious issue in overlay nodes since they are constrained by memory and processing power (normally PCs/servers).

Reduction of per group state is just one of the several motivations for encoding Multicast trees. In Distributed Interactive Applications such as large scale virtual environments (multiplayer games) and distributed interactive simulations, participants act as senders and receivers in several multicast groups. Minimizing the control overhead due to group dynamics is a major design consideration here [13, 17]. In these applications, nodes are clustered according to communication needs and multicast distribution trees are constructed within the clusters. As nodes join and leave, nodes send control messages to each other to repair the multicast distribution trees. On average, the number of messages is linear to the number of nodes in the tree. Since there are a large number of multicast groups with frequent joins and leaves, significant control messages flow in the network. If the multicast trees are encoded

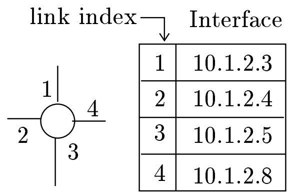


Figure 1: Link Index concept: A router with its link indexes

within data packets, control overhead is reduced and on-the-fly tree repair is made possible at forwarding nodes[15].

Multicast tree encodings also provide a convenient way of choosing explicit multicast trees in networks and overlays. In overlay networks such as RONs [1], Akamai, OMNI [21], by embedding multicast trees within data packets, multicast traffic can be routed on specific links to perform load balancing or traffic engineering. In secure group communication, group keys must be updated when nodes join or leave [12]. When nodes leave, the updated keys cannot be communicated via the old multicast tree since the leaving nodes would receive the new key. Multicast tree encoding can be used to communicate the new key via an explicit multicast tree containing only those nodes which need to receive the new key.

The above motivations notwithstanding, very little is known about optimal encodings of multicast trees. Indeed, there are two proposals which encode multicast trees non optimally [15, 3]. In this paper, we consider the multicast trees which could occur at network or overlay level and show the correspondence of these trees to theoretical tree data structures. Using this correspondence, we give lower bounds on the number of bits needed to encode multicast trees and give efficient ways of encoding them.

Multicast tree encoding must satisfy two requirements. Firstly, the encoding must be of minimal length since the it is passed in *every* multicast data packet and processed by each forwarding node (router or overlay node). Secondly, the forwarding operation must consume minimal time. Each forwarding node processes the encoding to determine the list of downstream nodes to whom the multicast packet must be forwarded. To minimize both the end-to-end packet delay and the per packet processing load on a forwarding node, the encoding must support the execution of this task efficiently.

The networks in which we consider multicast trees can be divided into two categories: (i) Networks which can *potentially* support forwarding based on *link Indexes*. The link index of a node with value i refers to its ith link. For example, the link index of a router refers to one of its interfaces. A router can potentially order its link interfaces and refer to each interface by its index in this ordering (Fig 1). The link index of an overlay node refers to one of its neighbors. Using a link index, a node can be instructed to forward a packet to one of its neighbors. This operation can potentially be supported in IP networks, static overlays such as OMNI, RON, Akamai, partially upgraded IP networks such as Bananas[11], etc where neighbors of a node do not change frequently. (In Bananas, nodes do forward unicast packets based on link indexes to support multipath-routing). (ii) In dynamic peer to peer networks (e.g. DHTs, unstructured p2ps) both the degree and the neighbors of nodes change frequently as nodes join and leave and hence forwarding based on link indexes cannot be supported. In these networks, node identifiers (e.g. IP addresses) of nodes need to be used to encode the multicast tree.

We show encoding of multicast trees using both Link indexes and Node identifiers. The tree encodings we propose are independent of underlying tree construction protocols and are based on *balanced parentheses representation of ordinal trees*. The rest of the paper is organized as follows. Section 2 places related work in context, section 3 shows correspondence of multicast trees to tree data structures and section 4 presents various tree encodings. Section 5 evaluates the length these encodings on real and generated multicast trees and section 6 concludes.

2 Related work

The encoding of a multicast tree depends on whether it represents the *default* multicast tree supplied by unicast routing framework of the underlying network or whether it represents a *specific or explicit* tree. To represent a specific tree, the multicast tree must be encoded using either the node identifiers or link indexes of the entire tree. To choose the default tree, identities of receivers are sufficient. Default trees can be chosen in any network which supports unicast routing and their main application is reduction of multicast state. On the other hand, explicitly representing trees provides the option of routing multicast traffic as desired, which may be useful to create a multicast routing framework at application level, perform traffic engineering, or multicast state reduction.

In xcast [8], a multicast tree is encoded by listing the IP addresses of receivers in the multicast tree. In IP networks, a packet can be routed to a destination node given its IP address. Due to this reason, explicitly listing the IP addresses of receivers of a multicast group suffices to represent the default multicast tree that exists between the source and the receivers. Forwarding routers perform a routing table lookup for each IP address in the packet and group them based on the common outgoing interface. The multicast packet is then forwarded on the interfaces, each packet containing the corresponding subset of IP addresses. Alternatively, if the last hop routers are allowed to store multicast state, the tree can be represented by listing the IP addresses of last hop routers instead of receivers(xcast+). xcast has two limitations. Since each IP address consumes 32 bits, xcast encoding is suitable for multicast groups containing a small number of last hop routers (6-10 last hop routers need 24-40 Bytes in IPv4). Secondly, each forwarding router needs to perform k IP lookups, where k is the number of last hop routers in its subtree. Large encodings require more IP lookups which are expensive.

The xcast model is also suited to peer to peer DHTs such as Pastry and Tapestry, where packets are routed in the overlay using node identifiers. In Pastry, a node is represented using a 128 bit identifier. By representing multicast trees using receiver identifiers, the state overhead imposed by small multicast groups can be avoided in Scribe and Bayeux. The xcast model is advantageous in DHTs since they guarantee routing in the presence of joins and leaves by intermediate peers.

In Linkcast [3], a multicast tree is encoded using the link indexes of links in the multicast tree. Each forwarding node reads *its* corresponding link indexes and forwards the multicast packet. The multicast tree which is encoded is a reverse path multicast tree constructed using join messages sent from last hop routers towards the source. Each router receiving the join message is expected to forward its identity along with the link index of the interface which received the join message. Thus the source can encode the multicast tree using link indexes. For forwarding, a pointer in the encoding points to the location of the current router in the encoding. Each router uses the pointer to read its outgoing link indexes and updates the pointer for the next hop routers. Although the number of links in the multicast tree is larger than the number of receivers, link indexes can be represented using fewer bits as compared to 32 bits for the IP address of a node. Also, expensive routing table lookups are avoided in Linkcast. We propose a new encoding for Linkcast which requires less space and two new encodings of multicast trees using link indexes.

In [15], authors propose two tree encodings to represent application level multicast trees occurring within clusters of Distributed Interactive Applications. The multicast trees are encoded using IP addresses of participating nodes. Authors propose two encodings - Per level header encoding and Preorder header encodings. In addition to IP addresses of nodes, these encodings spend significant number of additional bits. In a multicast tree of maximum degree d, the per-level header encoding spends $\log d$ bits per node and preorder header encoding spends approximately $\log n$ bits per node. We show an encoding using only 2 additional bits per node.

3 Multicast Trees and Tree data structures

Figure 2(i) shows a multicast tree occurring in an arbitrary network or overlay. The portion of the multicast tree which needs to be encoded is shown highlighted. The last hop routers, which serve one or more receivers are shown grayed. The link indexes corresponding to one of the nodes are shown. Figure 2(ii), (iii) and (iv) show the *Ordinal tree*, *Cardinal tree* and *Arbitrarily labeled tree* data structures corresponding to the multicast tree.

An Ordinal tree is a rooted (a unique node is distinguished as the root), unlabeled tree in which each node has an arbitrary degree. A binary tree is an ordinal tree in which the maximum degree of a node is two. The number of ordinal trees on n nodes can be bounded by $O_n = \binom{2n+1}{n}/(2n+1)$. Thus, $\lg(O_n) \approx 2n$ is an information theoretic lower bound on the number of bits needed to represent ordinal trees (lg is $\log_2$). Every ordinal tree of n nodes can be represented using $2n$ balanced parentheses. Fig 2(v) shows the balanced parentheses representation of the ordinal tree in Fig 2(ii). This representation is obtained by a preorder (depth first) traversal of the tree, outputting a "(" while visiting a node for the first time and a matching ")" while visiting the node after visiting its subtree (The correspondence of some parentheses to nodes is shown). By representing a `"("` by `1` and a `")"` by `0`, balanced parentheses can be represented using $2n$ bits.

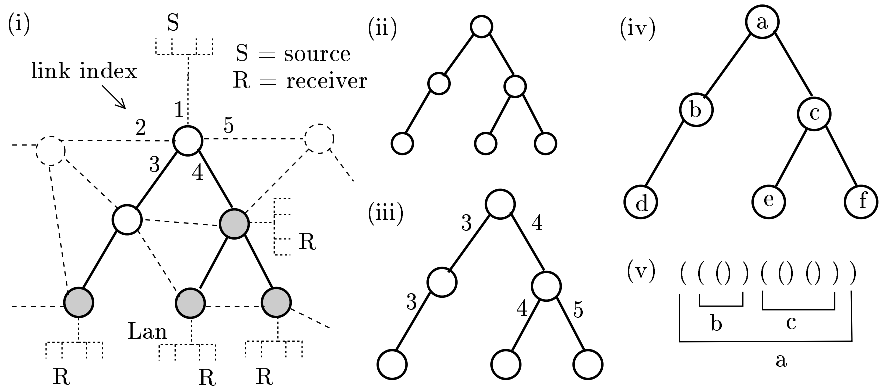


Figure 2: (i) Multicast Tree in a Network (ii) Ordinal tree (iii) Cardinal Tree (iv) Arbitrarily labeled tree (v) Balanced parentheses representation of ordinal tree

Figure 2(iii) shows the cardinal tree which results when a multicast tree is represented using link indexes. A Cardinal tree of degree d is an unlabeled tree in which each node has d positions for an edge to a child (if a child is at the ith position, the corresponding link gets the label i). There are $C_n^d = \binom{dn+1}{n}/(dn+1)$ cardinal trees of degree d on n nodes. $\lg(C_n^d) \approx (\lg d + \lg e)n$ is an information theoretic lower bound on the number of bits needed to encode these trees. A multicast tree in which the maximum link index is d is a cardinal tree of degree d. Figure 2(iii) is a cardinal tree of degree 5. Thus, $(\lg d + \lg e)n$ is a lower bound on the number of bits needed to represent a multicast tree using link indexes.

Figure 2(iv) shows the Arbitrarily labeled tree which results when a multicast tree is represented using node identifiers. Node Identifiers a,b, etc are essentially IP addresses. A *labeled* tree of n nodes is an ordinal tree in which each node is labeled from 1 to n. An *arbitrarily labeled tree* is an ordinal tree in which each node has a *unique* arbitrary label. If each node is represented using a k bit label, then there are $A_n^k = \binom{2^k}{n}\ n!\ O_n$ such trees on n nodes. Thus, $\lg(A_n^k) \approx kn + 2n$ is a lower bound on the number of bits needed to represent a multicast tree using node identifiers.

4 Multicast Tree Representations using Link Indexes

For ease of explanation, we use the following notation. All nodes in the multicast tree have one incoming link (except the root) and zero or more outgoing links. A *terminal or leaf* node has zero outgoing links, a *relay* node has one outgoing link and a *branch* node has more than one outgoing links. In Fig 3(i), a, e, h and d are branch nodes, j, k, m, c, f and g are leaf nodes and b, i, and l are relay nodes. The links of a multicast tree can be divided into branch and relay links. All outgoing links of a branch node are branch links and all outgoing links of a relay node are relay links. For example, ac is a branch link and lm is a relay link. Let l

denote the number of links and n denote the number of nodes, $n = l + 1$. Let $\bar{b}$ denote the number of branch links and $\bar{r}$ the number of relay links, $l = \bar{b} + \bar{r}$. Let b denote the number of branch nodes, r the number of relay nodes, and t the number of leaf nodes, $n = b + r + t$. For tree in Fig 3(i), the preorder node traversal gives $a, b, e, h, j, k, i, l, m, c, d, f, g$ and the preorder link traversal gives $ab, be, eh, hj, hk, ei, il, lm, ac, ad, df, dg$.

4.1 Improvements to Linkcast Encoding

Linkcast consists of two encodings SBM and DBM. These encodings consist of a series of elements. In SBM, each element is a link index or a pointer. A pointer is used to point to the representation of the child node in the encoding. In DBM, each element is either a link index, pointer or a node demarker. SBM requires $\bar{b}$ pointers. DBM reduces this to $\bar{b} - b$, but at the expense of introducing $l + 1$ node demarkers. Each node demarker requires at least two bits to be differentiated from both link indexes and pointers, consuming $2(l + 1)$ bits of space. Utilizing the concept of pointers, we now present a simplified encoding which is shorter than both SBM and DBM and refer to this encoding as Link+. Instead of using node demarkers, our approach is to distinguish between links that terminate at routers which

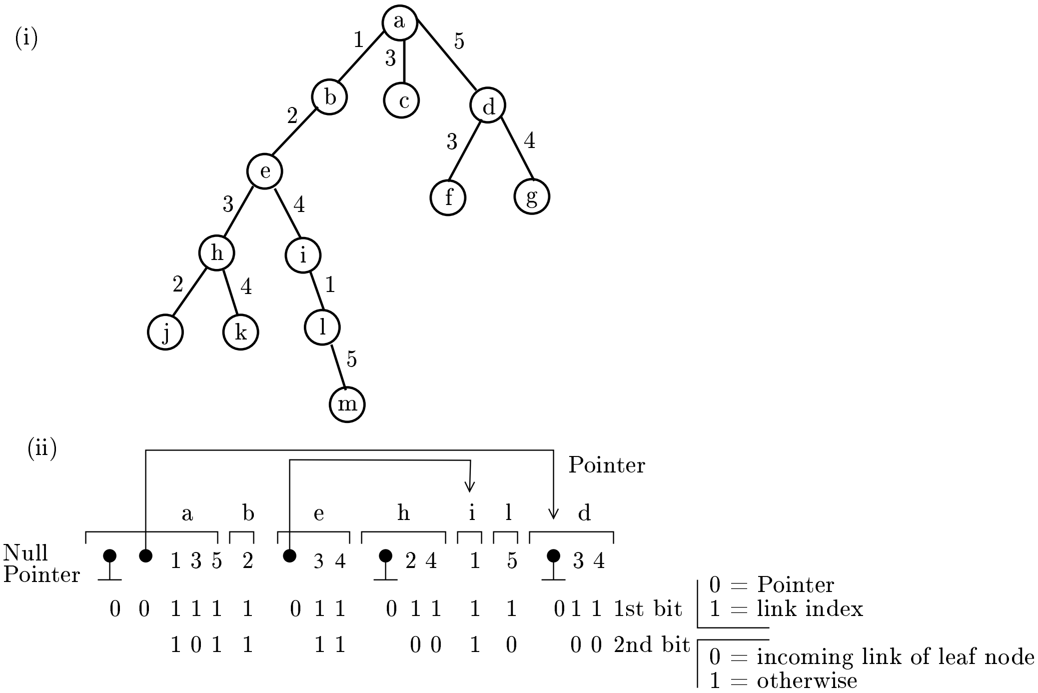


Figure 3: (i) Multicast Tree (ii) Link+ encoding

```
Forward( Enc, ptr )
if ptr == 0 exit;
if ( Enc[ptr].firstbit() == 0 ) /* pointer => branch node */
   i = ptr;
   child = 0;
   while (Enc[i++] is pointer) child++;  /* count pointers */
   for (j=1; j <= child; j++)       /* read outgoing links  */
       OLink[j] = Enc[i + j - 1];
       OPtr[j] = Enc[ptr + j - 1];  /* pointer passed to child */
   child++;
   OLink[child] = Enc[i+child-1];  /* last child */
   OPtr[child] = i + child;
else  // relay or terminal
   if (Enc[ptr] is terminal)
      child = 0;
   else
      child = 1;
      OLink[child] = Enc[i];
      OPtr[child] = i+1;
```

Figure 4: Forwarding Algorithm for Link+

perform forwarding and routers which do not, using one bit per link. One bit is used to distinguish between pointers and link indexes.

To construct Link+ encoding, nodes are visited in preorder. When a node is visited, a string of pointers and outgoing link indexes is outputted. If a node has d outgoing links, then $d-1$ pointers followed by d link indexes are outputted. The $d-1$ pointers point to the output strings of 2nd to dth child nodes. The output string of 1st child node occurs immediately after the output string of current node. Fig 3(ii) shows the Link+ encoding for the tree in Fig 3(i). For example, root node a has three outgoing links, thus 2 pointers followed by three link indexes are outputted. The output string of the first child node b occurs after the output string of a. The 1st pointer points to the output string of 2nd child node c (null). The 2nd pointer points to the output string of 3rd child node d. The first bit in each element is used to distinguish a pointer (`0`) from a link index (`1`). The second bit in each link index is used to distinguish links which terminate on leaf nodes (`0`) from those which do not (`1`).

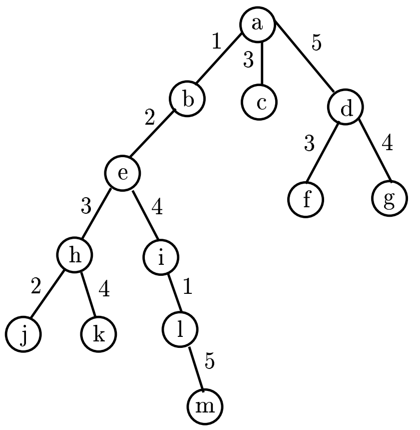
Routing In Link+, each node receives the pointer to its position in the encoding (as linkcast). The root node receives a pointer to the first element in the encoding. Each forwarding node determines the (i) outgoing links to forward the packet (ii) positions of child nodes in the encoding in turn to be forwarded to the children. If the position of the current node starts with a pointer, then it is a branch node, else it is a relay node. If a branch node finds d pointers, it reads the subsequent $d+1$ elements to determine its outgoing links. A relay node receives the position of its outgoing link and its child is the next element. If the second bit of an outgoing link is 0, a null pointer is forwarded and thus a leaf node receives a null pointer. The forwarding operation at a node takes time proportional to its out degree. The detailed algorithm is shown in Fig 4.

In total, the encoding length of Link+ is $(S_l+2)l+(S_p+1)(\bar{b}-b)$ bits. S_l is the number of bits needed to represent the maximum link index in the tree. S_p is the number of bits needed to represent a pointer, $S_p = \lceil \lg(l+\bar{b}-b) \rceil$.

4.2 Encoding based on Balanced Parentheses

We now present a link index encoding based on balanced parentheses representation. We refer to this encoding as Link*. Link* encoding consists of two parts (i) balanced parentheses representation of the tree (ii) preorder list of link indexes. The first encoding in Fig 5(ii)

(i)



(ii)

At node a : $\underbrace{(\,(\,(\,()\,()\,)\,(\,(\,()\,)\,)\,)}_{b}\;\underbrace{()}_{c}\;\underbrace{(\,()\,()\,)}_{d}$ 1 2 3 2 4 4 1 5 3 5 3 4

At node b : $\underbrace{(\,(\,()\,()\,)\,(\,(\,()\,)\,)}_{e}$ 2 3 2 4 4 1 5

At node c : NULL

At node d : $\underbrace{()}_{f}\;\underbrace{()}_{g}$ 3 4

Figure 5: (i) Multicast Tree (ii) Link* encoding

shows the encoding for the tree in Fig 5(i). The balanced parentheses representation is similar to the balanced parentheses representation of ordinal trees, except that each parenthesis now corresponds to the incoming link of a node, instead of the node itself. The encoding is constructed by a preorder traversal of the tree. When a link is visited for the first time, its link index and a "(" are outputted. When a link is visited after visiting all links in its subtree, a ")" is outputted. This results in $2l$ parentheses and l link indexes. In total, Link* requires $(S_l + 2)l$ bits. S_l as before denotes the number of bits needed to represent the maximum link index.

Routing In Link*, instead of passing a pointer to the position of a node in the encoding, the encoding itself is changed after each forwarding operation. Each node receives only the encoding of its subtree. Fig 5(ii) shows the encoding received by the root node a and the encodings it passes to child nodes b, c and d. A forwarding node reads the balanced parentheses once to determine the outgoing links and the encodings of subtrees to be passed to the child nodes. The ith open parenthesis "(" corresponds to the link index at the ith position in the list of link indexes within the encoding. For example, in Fig 5(ii), root node a reads its balanced parenthesis and determines that it has 3 children. The 1st "(" corresponds to child node b, the 9th "(" corresponds to c and the 10th "(" corresponds to d. Thus a's output links are at positions 1, 9 and 10 in the list of link indexes, i.e., link indexes 1,3 and 5 respectively. The detailed forwarding algorithm is shown in Fig 6. The forwarding operation of a node takes time proportional to reading the balanced parenthesis representation of its subtree. (This encoding is similar to encoding of Munro, et al [4]. However, they consider efficient bit representations of balanced parentheses for very large trees, finding the parent

```
Forward (BPStr, LIStr, n)
if BPStr == NULL exit;
bal = 0;
for i=1 to 2n do    // scan the balanced parenthesis
  if (BPStr[i] == "(")
     open++;
     if bal == 0
        child++;
        OLink[child] = LIStr.link(open);
     else
        LI[child].append(LIStr.link(open)); //encoding passed to children
        BP[child].append("(");
     bal++;
  else
     bal--;
     if bal != 0
        BP[child].append(")");
```

Figure 6: Forwarding Algorithm for Link*

of a node, size of subtrees, etc which are not needed for small and medium sized multicast trees. We use only the bare minimal representation of cardinal trees).

4.3 Improvements to balanced parentheses representation

Several studies [6, 14, 7] have investigated the type of multicast trees which occur in the Internet. Trees occurring in the Internet are constrained by the underlying structure of the Internet. These trees have a large number of relay nodes compared to branch nodes. In Link*, other than the mandatory space of lS_l bits to represent the link indexes, 2 bits are spent for each link (for balanced parentheses). We now present a new encoding Link** which reduces the number of bits on those links which terminate on relay nodes and increases the number of bits on those links which terminate on branch nodes. In Link**, 3 bits are spent

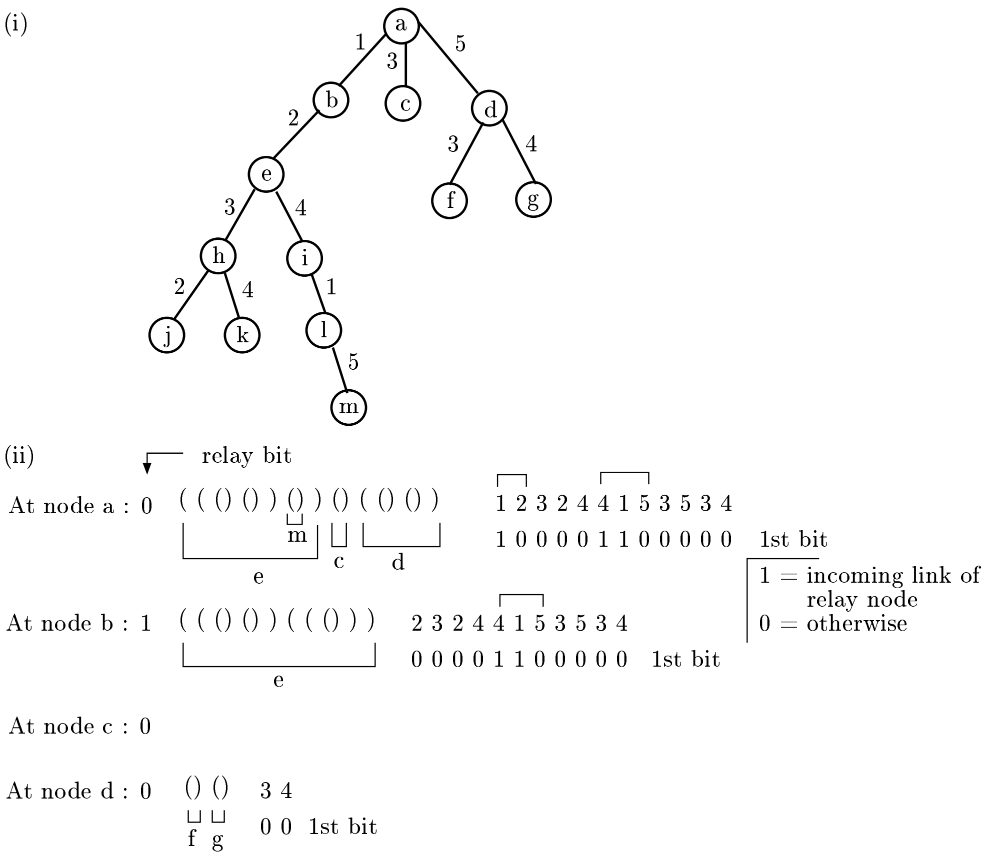


Figure 7: (i) Multicast Tree (ii) Link** encoding

per incoming link of a branch node or leaf node, and only 1 bit is spent per incoming link of a relay node. In total, Link** requires $(S_l + 2)l + b + t - r$ bits. If $r > b + t$, the encoding spends less than 2 bits per link. In section 5, we shall see that for several trees in the Internet, the number of relay nodes is larger than the sum of branch and leaf nodes.

Link** encoding consists of three parts (i) relay bit (ii) balanced parentheses (iii) pre-order list of link indexes. The encoding is similar to Link*. But it is obtained by encoding a virtual tree in which a path of consecutive relay nodes is treated as a single virtual link whose label is the concatenated list of its individual link indexes. Thus, the balanced parentheses represent only branch and leaf nodes. The first encoding in Fig 7(ii) shows the encoding for tree in Fig 7(i). For example, the path em is treated as a virtual link with label "415", and the path ae is treated as a virtual link with label "12". To determine the start and end of a virtual link, the first bit in each link index is used to distinguish links which terminate on intermediate relay nodes (`1`) from those which terminate on branch or leaf nodes (`0`). The relay bit is used for routing.

Routing The forwarding operation is similar to Link*. Fig 7(ii) shows the encoding received by root node a and the encodings it passes to child nodes b, c and d. The ith open parenthesis "(" corresponds to the ith virtual link in the encoding. For example, in the encoding received by a, the 1st "(" corresponds to the 1st virtual link ae and the 5th "(" corresponds to the 5th virtual link em. During forwarding, a node reads the relay bit to check whether it is a relay node or not. The relay bit is `1` when the encoding is received by a relay node and `0` otherwise. The relay bit received by a node is simply the first bit taken from the link index of its incoming link. For example, when a performs forwarding, it sets the relay bit of child node b equal to first bit of link index of link ab. In Link**, only branch nodes read the balanced parentheses and change it. The relay nodes read only the first link index to perform forwarding. The time complexity of forwarding is constant at relay nodes. The time complexity of forwarding at a branch node is proportional to reading the balanced parenthesis of its virtual subtree. The detailed forwarding algorithm is shown in Fig 8.

4.4 Representing trees using Node Identifiers

Both encodings Link* and Link** can be used *as is* by replacing link indexes of links by node identifiers of nodes which terminate on those links. Figure 9(i) shows the encoding Link* using node identifiers for tree Fig 7(i), as received by root node a. The forwarding algorithm remains the same, except that each forwarding node routes the packet to the corresponding next hop nodes instead of sending it on specific outgoing links. Representing trees with node identifiers using Link* or Link** encodings is suited to application level multicast.

4.5 Representing trees using both Node Identifiers and Link Indexes

In networks which support unicast routing based on node identifiers, when the default multicast tree is represented using link indexes, it may be advantageous to replace certain long paths of consecutive relay links in the tree with the IP address of the last branch or leaf node

```
Forward (BPStr, LIStr, n, relay)
if BPStr == NULL exit;
if (relay == 1)
    child = 1;
    OLink[child] = LIStr.link(1);
    Relay[child] = OLink[child].first_bit();
    BP[child] = BPStr;
    LI[child] = LIStr.without_firstlink();
    return;
bal = 0;
for i=1 to 2n do /* scan the balanced parenthesis*/
  if (BPStr[i] == "(")
     open++;
     if bal == 0
        child++;
        OLink[child] = LIStr.Vlink(open).link(1);
        LI[child].append(LIStr.Vlink(open).without_firstlink());
        Relay[child] = OLink[child].firstbit();
     else
        LI[child].append(LIStr.Vlink(open));
        BP[child].append("(");
     bal++;
  else
     bal--;
     if bal != 0
        BP[child].append(")");
```

Figure 8: Forwarding Algorithm for Link**

in the path. For this reason, we need a representation of tree using both node identifiers and link indexes. For example, if each link index is represented using 6 bits, a path of 10 relay links consumes 60 bits. In Link*, since 2 bits are spent per link, 80 bits are spent in total for this path. Since the default tree is being represented, the path of relay links can be replaced by the 32 bit IP address of the next branch or leaf node.

Both Link* and Link** can be modified to represent certain paths using IP addresses of the terminating node in the path. In Link*, one extra bit is needed per element to distinguish between a node identifier and link Index. In Link**, one extra bit is needed per branch or leaf node to do the same. We call the corresponding new encodings as LN* and LN** respectively. Figure 9(ii) and (iii) show the encodings of the tree Fig 7(i) using LN* and LN** encoding. The path ae is encoded the usual way using link indexes and the path em is replaced by the node identifier m. In LN* and LN**, it is assumed that intermediate

(i)

At node a : (((() ()) ((()))) () (() ()) b e h j k i l m c d f g

b c d

(ii)

At node a : (((() ()) ())) () (() ()) 1 2 3 2 4 m 3 5 3 4

0 0 0 0 0 1 0 0 0 0 1st bit

b c d

1 = node

0 = link

(iii) relay bit

At node a : 0 ((() ()) ()) () (() ()) 1 2 3 2 4 m 3 5 3 4

1 = incoming link of relay node

0 = otherwise

m

1 0 0 0 0 0 0 0 0 0 1st bit

e c d

0 0 0 0 1 0 0 0 0 2nd bit

1 = node

0 = link

Figure 9: (i) Link* encoding using nodes (ii) LN* encoding (iii) LN** encoding

nodes can perform forwarding using both node identifiers and link indexes. If intermediate nodes are IP routers, they need to perform tunneling to route the IP packet to the next branch or leaf node.

5 Simulations

In this section, we evaluate the link based encodings when they are used to represent shortest path multicast trees. We conducted tests using various generated and real topologies of the Internet. Here, we present the results for three representative topologies (Table 1). `ts1000` is the transit-stub topology generated by GT-ITM [5], `scan` topology is a partial map of the Internet collected by SCAN project [16], and `att` is the router level ISP topology of AT&T collected by Rocketfuel [20]. For each of these topologies, we chose random routers as source and last hop routers and constructed shortest path multicast trees from source to last hop routers. To compute shortest paths, for `scan` and `att`, hop count metric was used and for `att`, the generated weights were used. Figure 10 (a),(c) and (e) show the properties of multicast trees for the three topologies (each point is the average of 1000 simulations). As observed, in all topologies the number of relay nodes is significantly larger than the number of branch nodes. However, the total number of branch links is significant and grows linearly with the number of last hop routers. Figure 10(b),(d) and (f) show the encoding lengths of Link+, Link*, Link** and xcast+. The xcast+ encoding length is essentially 32 times

topology	Routers	type
`ts1000`	1040	Generated(GT-ITM)
`scan`	284,805	Real (Internet)
`att`	731	Real (AT&T ISP)

Table 1: Topologies used for simulation

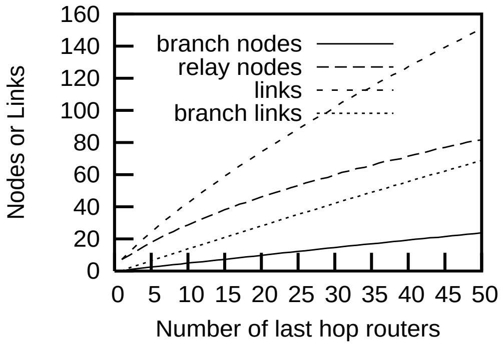
the number of last hop routers. For Link encodings, the link index was represented using 5 bits (routers in the Internet rarely have degree larger than 32). The pointer in Link+ was represented using 8 bits. As seen, the Link+ encoding takes more space than both Link* and Link** to represent pointers corresponding to the branch links (linkcast encodings take at least $l + 2$ more bits than Link+, for l links). For inter-domain topologies `ts1000` and `scan`, Link** encoding takes less space than Link* due to the presence of a large number of relay routers. For intra-domain `att` topology, as the number of last hop routers increase (> 25), Link* takes less space than Link** since the proportion of relay routers reduces compared to branch and last hop routers.

The forwarding performance of link based encodings is better than xcast+ since expensive routing table lookups are avoided. The forwarding performance of encodings depends on two factors (a) Length of encodings read and processed by each node (b) Time taken by the forwarding algorithm to determine the outgoing links to forward the multicast packets. The performance of Link+ forwarding algorithm is better than Link* and Link** since the positions of outgoing links during forwarding operation are obtained directly by reading the pointers. In Link* and Link** the positions of outgoing links are obtained by reading the balanced parentheses. Thus the forwarding algorithm of Link* and Link** takes some extra time compared to Link+. At each forwarding node, this extra time is proportional to the time to read the balanced parentheses representation of the node's subtree. In Fig 11, the average total number of balanced parentheses read on the entire path from the source to leaf nodes is shown in case of Link* and Link** encodings. When the tree is encoded using Link**, less parentheses are read since balanced parentheses are used only for branch and leaf nodes. It must be observed that since each balanced parenthesis is represented using a single bit, the balanced parentheses representation read by each forwarding node is small in length and can be loaded into processor registers and processed at high speeds.

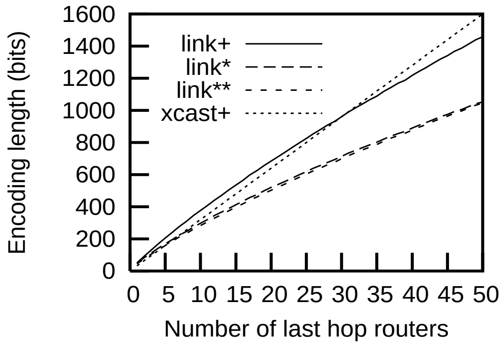
The space and forwarding corresponding to node based encodings is analogous to link based encodings, but of larger scale since more bits are spent for a node as compared to a link.

6 Conclusions

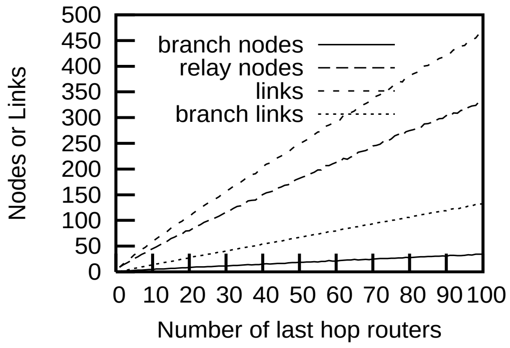
In this paper, we presented efficient ways of encoding specific or explicit multicast trees using link indexes and node identifiers. We presented an improved encoding Link+ and two new encodings Link* and Link** which consume space close to the minimum number of bits



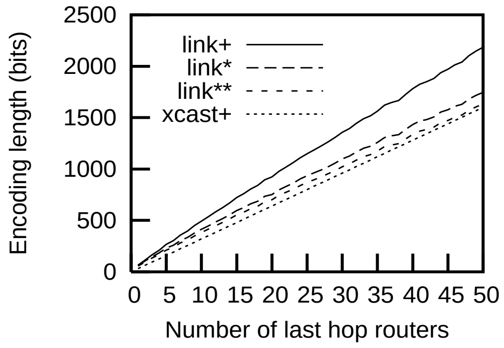
(a) `ts1000`



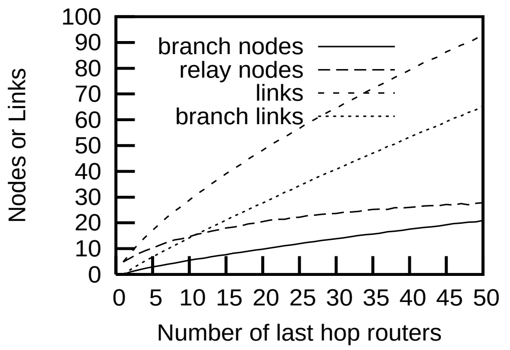
(b) `ts1000`



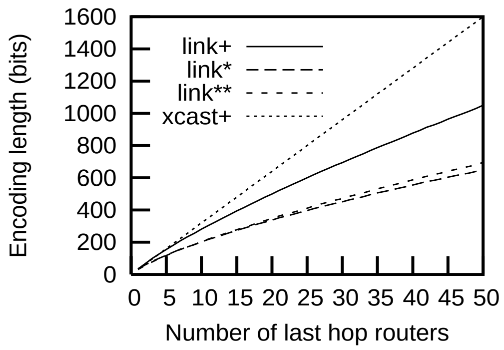
(c) `scan`



(d) `scan`

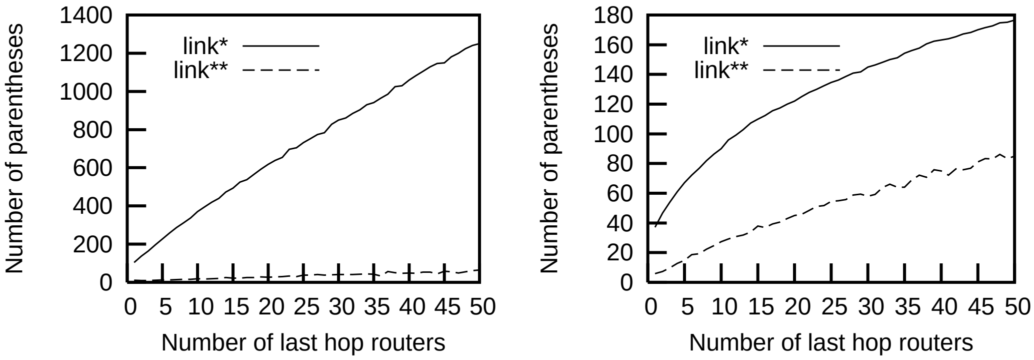


(e) `att`



(f) `att`

Figure 10: Properties of multicast trees and their encoding lengths in various topologies



(a) `scan` (b) `att`

Figure 11: Average total number of parentheses read for the path from source to a last-hop router for Link* and Link** encodings

needed to represent multicast trees. These encodings can be used to represent trees using either link indexes or node identifiers. We evaluated the space consumed by these encodings using shortest path multicast trees in real and generated topologies. These encodings can be used for various applications such as multicast state reduction, traffic engineering and application level multicast, and provide a feasible way of representing trees for small and medium sized multicast groups within data packets.

References

[1] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. The case for resilient overlay networks. In *Proc. of 8th Annual Workshop on Hot Topics in Operating Systems*, 2001.

[2] Vijay Arya, Thierry Turletti, and Shivkumar Kalyanaraman. Encodings of multicast trees. *Inria Research Report*, 2004.

[3] Mozafar Bag-Mohammadi1, Siavash Samadian-Barzoki1, and Nasser Yazdani. Linkcast: Fast and scalable multicast routing protocol. *NETWORKING*, 2004.

[4] David Benoit, Erik D. Demaine, J. Ian Munro, and Venkatesh Raman. Representing trees of higher degree. In *International Workshop on Algorithms and Data Structures*, 1999.

[5] Ken Calvert, Matt Doar, and Ellen W. Zegura. Modelling internet topology. *IEEE Communications Magazine*, 1997.

[6] Robert C. Chalmers and Kevin C. Almeroth. On the topology of multicast trees. *IEEE/ACM Transactions on Networking*, 11(1):153–165, 2003.

[7] Danny Dolev, Osnat Mokryn, and Yuval Shavitt. On multicast trees: Structure and size estimation. In *Proc of IEEE INFOCOM*, 2003.

[8] Rick Boivie et al. Explicit multicast (xcast) basic specification, INTERNET DRAFT draft-ooms-xcast-basic-spec-06.txt. 2004.

[9] S. Deering et al. The pim architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 1996.

[10] S. Deering et al. Protocol independent multicast-sparse mode (pim-sm): Protocol specification. *RFC 2117*, 1997.

[11] Hema Tahilramani Kaur, Shiv Kalyanaraman, Andreas Weiss, Shifalika Kanwar, and Ayesha Gandhi. BANANAS: An evolutionary framework for explicit and multipath routing in the internet. In *Proc. of SIGCOMM FDNA Workshop*, 2003.

[12] C. W Kong, M. Gouda, and S. LamK. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 8:16–30, Feb 2000.

[13] Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Paul T. Barham, and Steven Zeswitz. Npsnet: A network software architecture for large scale virtual environments. *Presence: Teleoperators and virtual Environments*, 1994.

[14] Graham Phillips, Scott Shenker, and Hongsuda Tangmunarunkit. Scaling of multicast trees: comments on the chuang-sirbu scaling law. In *Proc. of Conference on Applications, technologies, architectures, and protocols for computer communication*, pages 41–51, 1999.

[15] George Popescu and Zhen Liu. Stateless application-level multicast for dynamic group communication. *IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'04)*, pages 20–28, 2004.

[16] SCAN project. http://www.isi.edu/scan/mercator/maps.html.

[17] J. Mark Pullen and David Wood. Network technology for DIS. In *Proc. of IEEE*, pages 1156–1167, Aug 1995.

[18] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[19] A. Rowstron, A-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Proc. of NGC*, 2001.

[20] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. *SIGCOMM*, 2002.

[21] S.Y.Shi and J.S.Turner. Routing in overlay multicast networks. *IEEE Infocom*, 2002.

[22] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. *RFC 1075*, Nov 1998.

[23] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *UC Berkeley Technical Report UCB/CSD-01-1141*, 2001.

[24] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proc. of NOSSDAV*, 2001.

Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399