Efficient Key Distribution for Secure Multicast

Richard Ladner
Justin Goshi

Multicast Security

- Multicast is broadcast to a group, not to everyone.
- Multicast group – group of members who have a right (have paid) to be in the group.
- Applications
  - Pay-per-view live broadcast
  - Video-on-demand for a fee
  - Software update group
  - Authorized for database access

Key Distribution Problem

- Group key to secure content
  - Change (re-key) on membership change
    - ADD(u) - u does not have access to data prior to operation, but does after operation
    - DELETE(u) - u does not have access to data after operation
  - Distribute group key to current members
  - Additional keys needed to secure group key

Logical Key Trees

(Wong et al., 1998)
(Wallner et al., 1998)

- Shows keys held by each member
  - Nodes represent keys
  - Leaf nodes represent members
  - Members hold all keys on path to root

Encryption

- Each member i has a private key ui which is distributed using public key encryption.
- Each key ki encrypts data multicast to the leaves of the tree rooted at g.
- Member i holds all the keys on its path to the root.

Group keys

- Needed for multicast
- Needed for re-keying
Re-keying Messages

- $E(k_a, k_g)$ is a message to the group rooted at $g$ that is encrypted using $k_g$ and contains a new key $k_a$ for some ancestor of $g$.
- Goal is to minimize the number of these re-keying messages.

Re-keying (ADD)

- 2 level tree
- $ADD(u)$ requires 2 messages per key
- Example: $ADD(u_{10})$
Re-keying (ADD)

- Example: $ADD(u_{10})$

Re-keying (ADD)

- Example: $ADD(u_{10})$

Re-keying (ADD)

- Define $d_u$ to be the depth of $u$.
- $ADD(u)$ cost is $2d_u$ messages

Re-keying (ADD)

- Example: $ADD(u_{10})$

Re-keying (DELETE)

- $DELETE(u)$ requires a linear number of messages, each of the form $E(k'_G, u_j)$

Re-keying (DELETE)

- Example: $DELETE(u_9)$

Re-keying (DELETE)

- Example: $DELETE(u_9)$
Re-keying (DELETE)

• Example: $\text{DELETE}(u_9)$

![Diagram of Re-keying (DELETE) example]

Re-keying (DELETE)

• Example: $\text{DELETE}(u_9)$

![Diagram of Re-keying (DELETE) example]

Re-keying (DELETE)

• Define ancestor weight $w_u$ to be sum of degrees on path from $u$ to root

$\text{DELETE}(u)$ cost is $w_u - 1$ messages

![Diagram of Re-keying (DELETE) example]

Asymmetry

• ADD and DELETE have different costs
  – ADD $2d_u$
  – DELETE $w_u - 1$

• ADD has more freedom than DELETE
  – ADD can go anywhere in the tree
  – DELETE is where it happens

![Diagram of Asymmetry example]

Fundamental Problem

• What is the best key tree for a given mix of ADDs and DELETEs?

• Our approach is to use balanced trees.

• Evaluation of the approach
  – Theoretical worst case analysis
  – Simulation studies

![Diagram of Fundamental Problem example]

Previous Results

(Poovendran and Baras, 2001)

(Snoeyink et al., 2001)

• Amortized worst-case cost lower bound is $\Omega(\log n)$ per operation.

• Constructed static multiway trees that are optimal for ancestor weight.
Cost Components for On-line Algorithms

- Tree structure cost due to $w_u$
- Restructuring cost to maintain structure

Algorithms

- K-ary trees
- B-trees
- 2-k trees (like AVL trees)
- Weight balanced trees

K-ary Trees

(Wong et al. 2000)

- ADD(u) - insert into the tree to keep $w_u$ as small as possible.
- DELETE(u) - simple remove it at cost $w_u$
  - If only one child, then collapse

B-Tree Algorithm

- All leaves have same depth
- All internal nodes (except the root) have degree in the interval \([\lceil t/2 \rceil, \lceil t \rceil]\), where $t$ is the order of the B-tree.
- Use existing algorithms for maintaining B-Tree property
  - John Hopcroft, 2-3 trees, 1970

Height-Balanced 2-k Algorithm

- All nodes are balanced
  - Height of children differ by at most
- All internal nodes have degree in the interval \([2, k]\)
- Extension of the AVL tree algorithm of Rodeh et al. (2001)

Weight-balanced Algorithm

- Node weight of $u = \max w_i$ over all leaf nodes $i$ in sub-tree rooted at $u$
- Node $u$ is weight-balanced if its children differ in node weight by at most 1
**Weight-balanced Algorithm**

- **DELETE(u)** costs proportional to ancestor weight $w_u$
- How about balancing by $w_u$?
  - Can be done for 2-3 and 2-3-4 trees

---

**Worst-case Tree Structure Cost Analysis**

- Optimal worst-case bound is $3 \log n$
- Derived worst-case bounds for our algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$w_u$</th>
<th>Algorithm Bound</th>
<th>Optimal Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height-balanced 2-4</td>
<td>$4 \log n$</td>
<td>$e = 1.618$</td>
<td>$\approx 3.04$</td>
</tr>
<tr>
<td>B-tree of order 4</td>
<td>$4 \log n$</td>
<td>$e = 1.618$</td>
<td>$\approx 2.11$</td>
</tr>
<tr>
<td>Weight-balanced 2-3-4</td>
<td>$\log n$</td>
<td>$b = 1.325$</td>
<td>$\approx 1.30$</td>
</tr>
</tbody>
</table>

---

**Simulation Results**

- Does good tree structure help performance?
  - We do not have a way to analyze restructuring cost
  - Trace data for multicast is problematic
  - Simulation may yield insights

---

**Random Re-keying**

- B-tree
- Height-balanced
- Weight-balanced
- Degree-k

Each point represents 1,000 consecutive operations

---

**Pathological Deletions**

- B-tree
- Height-balanced
- Weight-balanced
- Degree-k

Each point represents 1,000 consecutive operations
Summary: Online Key Tree Algorithms

- Algorithms to maintain balanced trees
- Identified 2 cost components
- Derived worst-case tree weight bounds
- Good performance, especially when tree becomes highly unbalanced

Future Potential

- Crypto technology is probably adequate for deployment
- Future depends on popularity of multicast
- May be other distributed applications that need secure group management
  - Access control in databases
  - Access control in file systems