Scalable Multicast Forwarding

Björn Grönvall (bg@sics.se)
Swedish Institute of Computer Science and *Luleå University of Technology*

The goal of this work is to develop efficient algorithms and data structures suitable for software-based forwarding of multicast datagrams. The algorithms should scale to very large numbers of simultaneously active multicast groups. In an example configuration with 32 interfaces a new algorithm can forward 20,000 simultaneously active multicast groups using only 64Kbytes of memory and a leak probability of 2.4×10^{-5}. This algorithm can perform almost 10 million forwarding decisions per second on a 800MHz Pentium III processor

The task of a multicast-forwarding engine is to forward packets along branches of a distribution tree. The branches of the tree consists of network segments and the nodes in the tree are routers. There are a few different classes of distribution trees. For simplicity only bidirectional trees will be considered in this discussion.

Abstractly, a bidirectional-distribution tree is a bidirectional acyclic graph. It has a root called center, core or, rendez-vous point. Forwarding is performed by having traffic flow up the tree towards the center, and down the tree towards the receivers.

A router must for each distribution tree maintain a list of interfaces that connect the tree. To forward a packet, first find the interface list corresponding to the group. If the packet arrived on an interface in the list, then, forward the packet over the remaining interfaces in the list.

It is possible to perform this forwarding function using probabilistic methods. Observe that forwarding will be successfully performed if the interface list includes *at least* all interfaces that the packet must be forwarded over. Since the interface list can potentially be to long, a packet will occasionally *leak* out the wrong interface. However, at a minimum, packets will always be forwarded over the distribution tree. Since the leak probability can be made arbitrarily small, this may not be a problem in practice.

Assume a network configuration where packets are allowed to leak with a maximum probability P. The probability that a leaked packet will leak through the next router is P^2, and through a third P^3. Thus, leak probability decays exponentially as we travel down (or up) the tree. Also, hosts configure their network adaptors to filter out unwanted multicasts. If a packet should sneak through this hardware filter, then software filtering sets in. Thus, hosts on leaf networks are already prepared to handle the problem with traffic leaks.

The suggested probabilistic-forwarding algorithm has an efficient implementation based on Bloom filters. The algorithm uses a fixed small number of memory references (e.g eight) and a small number of multiplications and shift instructions. Memory requirements scale linearly with the maximum allowed number of groups and is also independent of the number of interfaces. Surprisingly, memory requirements are independent of group address length. This is a result of that addresses are not stored in the data structure. Thus, 112 bit IPv6 addresses use no more forwarding memory than 28 bit IPv4 addresses.

An important quality of the algorithm is that it is independent of address-space usage. Performance is unchanged if addresses are picked uniformly random or from some other distribution. This is important since it is impossible to predict how a future multicast address space will be utilized.

There is always a downside to using probabilistic methods. In this case we introduce traffic leakage, i.e traffic will with a small probability leave the distribution tree. The probability that a group will leak through an interface is $[1 - (1 - 1/M)^{KN}]^K$. Here N represents the number of active groups, M is the number of bits in the filter, and K is the number of memory references used per lookup.

Outline of algorithm: Recall that multicast routing is basically about finding the appropriate interface list and then forward the packet over all those interfaces. This mapping from group to interface list can be implemented using a number of parallel Bloom filters. For each interface we maintain one Bloom filter. If a group is *active* with respect to an interface, then the corresponding Bloom filter *includes* this group.

Bloom filters are essentially arrays of bits. By organizing these bit arrays as columns in a bit matrix it is possible to evaluate all filters in parallel using word operations. Thus, a 32 bit micro processor can efficiently calculate a 32 interface list in parallel.

More information

For more information contact *bg@sics.se* or look at the web page *http://www.sics.se/cna/scalable-multicast.html*.