Fast group management in IGMP

Luigi Rizzo *

Dip. di Ingegneria dell'Informazione, Università di Pisa
via Diotisalvi 2 – 56126 Pisa (Italy)
email: l.rizzo@iet.unipi.it
http://www.iet.unipi.it/~luigi/

Abstract

The ability to control quickly the expansion/reduction of the multicast distribution tree is central in some recent proposals for multicast congestion control. At the lowest level, these operations are controlled by the IGMP protocol. With the current specification of the protocol, IGMP takes a few seconds to stop distribution of a group after a request. This reduces the effectiveness of congestion control mechanisms based on this feature.

In this paper we propose a mechanism to make the `LEAVE` operation act instantaneously, basing on prediction techniques similar to those used in processors for optimizing jump performance. The proposed mechanisms acts on the router side only, and it is fully compatible with the existing IGMP protocol. An implementation of the mechanisms described in this paper has been included in the latest `mrouted3.9` sources and is available from the author.

1 Introduction

The IP multicast infrastructure (*MBone*) has been successfully used in recent years to support a number of applications, ranging from audio and video conferences to reliable file distributions. A significant obstacle to the large scale deployment of the MBone is the lack of proper congestion control mechanisms such as those present in TCP. Without such mechanisms, applications cannot adapt to variable network conditions, and tend to use an uneven share of the network's capacity. As a consequence, providers and network managers might choose not to support multicast services in order to eliminate a potential source of congestion.

* Source code for the modified multicast router discussed in this paper is available at http://www.iet.unipi.it/~luigi/mrouted/

Article published in Proceedings of Hipparc'98 (1998)

Research on multicast congestion control mechanisms has recently produced some proposals based on a layered data organization [4,7], and relying on the dynamic reconfiguration of the multicast distribution tree. In these proposals, transmission occurs over multiple multicast groups using a layered data distribution approach, and the ability of multicast routers to start/stop data forwarding on demand is used to simulate sender's rate adaptation on different branches of the distribution tree.

For efficiency and scalability reasons, low level multicast routing protocols [1] do not keep exact group membership information. As a consequence, even when the last receiver exits a group, a router can only stop traffic forwarding to an interface after a polling phase to determine group membership has timed out without replies. The effect of this polling phase is a general slowdown of the responsiveness of the rate adaptation mechanism.

In this paper we show how to modify the IGMP protocol (in particular, we will refer to IGMP version 2 and later, which support the pruning of multicast groups) to speed up the response to group leave requests, and improve, among other things, the behaviour of layered congestion control mechanisms. Our proposal is based on the prediction of the poll outcome basing on previous results, and resembles the branch prediction techniques used in RISC processors to improve the speed of execution of jump instructions. The modifications only affect the router side, require very little additional state (a few bits) per group, and are fully backward compatible with existing IGMP implementations.

The paper is structured as follows. In Section 2 we briefly discuss the operation of the IGMP protocol and the leave operation in detail. Section 3 describes our proposal for fast leave operation, and the impact on the robustness of IGMP to malicious attacks. Section 4 briefly reports our initial experience with the implementation and illustrates future work.

2 The IGMP protocol

One of the tasks of IP multicast routers is to determine the presence, of receivers for a given multicast group on each interface. This permits to forward multicast traffic only where necessary, and to avoid flooding network segments where there are no receivers interested to a given group. Group membership information between hosts and routers on local networks is exchanged by means of the IGMP protocol [1]. Hosts willing to receive traffic directed to a multicast group send a membership `REPORT` message, that the router uses to enable forwarding of the requested group to the local network segment. Periodically, routers refresh group membership information by sending membership

REQUESTs, and listening for membership REPORTs coming from receivers still interested in the group. Data forwarding for a group is stopped if a request times out without any report received.

To avoid feedback storms when the set of (local) receivers for a group is large, a feedback cancellation technique is implemented in IGMP: hosts interested in a group do not reply immediately to a request, but schedule the transmission of a report at a random time after the request, and cancel the scheduled transmission if a report is received (sent by some other host) before the scheduled time.

2.1 Group leave mechanism

Starting from IGMP version 2, forwarding can be stopped almost "on demand", but not instantaneously, by having a receiver send an explicit LEAVE message to the router (such messages should only come from the last receiver who sent a membership reply for the group). The leave message immediately triggers a membership poll (called *last receiver poll*), whose outcome determines whether or not to keep forwarding data for that group. This mechanism is used by layered congestion control mechanisms such as RLM [4] and RLC [7] to achieve scalable congestion control with rate adaptation to the different parts of the multicast distribution tree.

Because of the feedback suppression mechanism (but also because IGMP messages are not sent reliably), routers do not have exact group membership information. So, upon arrival of a LEAVE message, the router cannot act immediately to stop forwarding the group, but must wait for the last receiver poll to timeout before acting. This poll phase has a duration TR, where T is the interval in which replies are distributed (typically 1 sec) and R is the "robustness factor" defined in the IGMPv2 protocol, i.e. the number of times a membership request is sent before declaring a timeout (typically $R \geq 2$). The few seconds' delay (*leave delay*) of the polling phase partly defeats the purpose of leaving a group for congestion control purposes: because of the delay, after congestion has been detected, unrequested, possibly high bandwidth traffic will keep coming in for the duration of the leave delay, aggravating the congestion situation.

We have shown [7] how the problem can be partly overcome by using some care in deciding when to join/leave groups, and by adopting large time constants in the congestion control mechanism. But such large time constants slow down the responsivity of the congestion control mechanism, and reduce its effectiveness. A fast LEAVE phase would then be highly desirable, both for the effectiveness of the congestion control mechanisms, and for the network

in general since it would permit better filtering of useless traffic, and faster recovery in case of congestion.

2.2 Last Receiver poll

In order to develop a fast leave phase, we need to optimize the behaviour of the router in response to a `LEAVE` request. Ideally, the router could react to the request by immediately stopping the forwarding the group, and running a membership poll must to check for and recover from possible errors in the above action.

Immediate action on a leave request can indeed be very effective in the case of a single receiver for the group in the local network – not an unlikely case with sparse groups – or when the node sending the request is representative of the behaviour of the whole set of local receivers. The latter is also not too far from reality for properly working congestion control mechanisms, such as RLC. Such mechanisms tend to organize receivers behind the same bottleneck so that they will take similar decision on whether to join or leave groups. Thus, the behaviour of any receiver can be used to infer the behaviour of the whole set of receivers, because:

- all receivers will likely decide to join/leave a group approximately at the same time;
- one of them will be allowed to send the LEAVE request to the router;
- no other receivers should respond to a membership query after the LEAVE has been sent.

However, not all applications are likely to achieve such a synchronized behaviour among receivers. As a matter of fact, in most current applications using IP multicast, receivers join/leave a group almost independently of each other, only driven by the user's action. In such a case, a `LEAVE` message could be generated by a random user leaving a group, and immediate action by the router would cause loss of useful data.

Since multicast data is not forwarded reliably by the network, such losses are not too worrisome, provided they are rare: either the application can tolerate them [3], or some higher level mechanism will exist to implement repair actions [2,6]. But if the router immediately and unconditionally blocks a group on a `LEAVE` request, it is possible that the frequency of such events becomes arbitrarily high, especially in presence of malicious behaviour from nodes trying to implement Denial of Service (DoS) attacks. Such nodes could in fact continuously send spurious `LEAVE` messages, thus almost completely disrupting service. In this respect, the last receiver poll acts a verification mechanism that prevents such attacks from being successful.

3 Implementing a Fast Leave Mechanism

To avoid the problems outlined in the previous section, we have designed an adaptive mechanism to speed up the leave phase while minimizing the chance of losing useful data and providing protection against DoS attacks. Our proposal relies on two small modifications to the router side only of the IGMP protocol. These modifications, described in the following, require a minimum amount (a few bits) of per-group additional state, and are fully compatible with existing IGMP implementations.

3.1 Last Receiver Poll Prediction

The first modification consists in the addition of a prediction mechanism that allows the router to anticipate the response to a last-receiver poll and act accordingly.

For each group, the router keeps the history (in the form of a short bit array) of the outcomes of previous polls. Upon a leave requests, the router starts a poll, but immediately tries to predict its outcome basing on the recorded history, as follows:

- When the history reports all timeouts, a new timeout is predicted. The router then immediately stops forwarding of the group to the local network, and possibly notifies upstream routers. In case a membership reply is received, group forwarding is restarted (possibly contacting the upstream router). Otherwise, no further action is performed at the expiration of the timeout.
- When at least one of the recent polls resulted in a membership reply, the router behaves normally, i.e. waits for the end of the poll to decide how to act.

In both cases, the history is updated by discarding the oldest entry and recording the actual result of the current poll.

The modifications to the state diagram of an IGMPv2 router is shown in Figure 1, while the C pseudo code in Figure 2 almost completely specifies the modified router behaviour (further details related to the interaction with congestion control algorithms will be given in the next Section). In the code, variable `history` is part of the (group,interface) state.

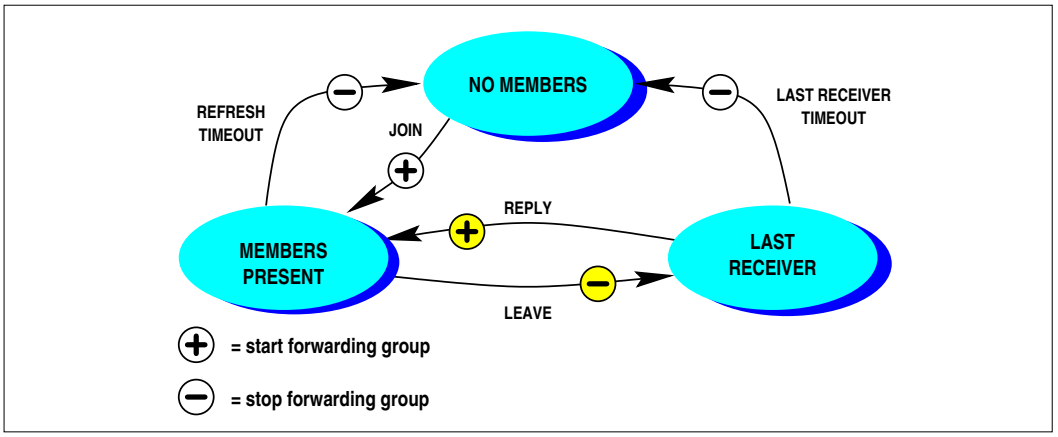

Fig. 1. Simplified state diagram for an IGMPv2 router. The grey symbols represent the additional action performed by our mechanism depending on the prediction.

3.2 Preventing False Membership Reports

There is an inherent race condition in using the pruning mechanism of multicast routers to implement receiver-driven congestion control. When multiple receivers are on the same network segment, they will decide to stop listening to a group (and notify the operating system of their decision) at slightly different times, depending on the host's speed and load for each receiver. The following chain of closely-spaced events could then lead to a false membership report (see Figure 3):

(1) a receiver on host M detects congestion and stops listening to a group;
(2) host M sends a LEAVE request;
(3) the router immediately sends a membership query;
(4) host M' schedules a membership reply and sends it when the timer expires;
(5) a receiver on host M' also detects congestion and stops listening to a group.

This race is unavoidable, and independent of the presence of our prediction mechanism. Depending on the relative speeds of the receivers, multiple false membership reports can be generated. The effect on IGMP of these false membership reports is limited to some additional IGMP traffic on the local network, and possibly some additional load on the multicast router to process such messages.

When/if mechanisms like RLC become more widely deployed, router load might become a concern, because join/leave phases will be much more frequent. False membership reports also have a very disrupting effect on the prediction mechanism, because they pollute the history vector and render the

```
#define HISTORY_LEN 3
#define HISTORY_INIT (1<<HISTORY_LEN)
#define HISTORY_MASK (HISTORY_INIT - 1)
...
int history ; /* last bit represents current poll phase */

STATE: no info present, EVENT: join request
  history = HISTORY_INIT ;
  <behave as specified in IGMPv2>

STATE: Members present, EVENT: receive leave request:
  history =  ( history << 1 ) & HISTORY_MASK ;
  if (history == 0) /* predict timeout */
    <notify routing to stop group distribution> ;
  <behave as specified in IGMPv2>

STATE: Checking membership, EVENT: last timeout:
  if ( history != 0 ) /* timeout was not predicted */
    <notify routing stop group distribution> ;
  <behave as specified in IGMPv2, except routing notification>

STATE: Checking membership, EVENT: receive report:
  if ( history == 0 ) /* timeout was predicted */
    <notify routing to restart group distribution> ;
  /* do not pollute status on first join */
  if (history != HISTORY_INIT)
    history |= 1 ; /* record report received */
  <behave as specified in IGMPv2>
```

Fig. 2. C pseudo code for the last receiver poll prediction.

mechanism ineffective. To reduce the impact of this problem, we decided to introduce a short delay D between the reception of the leave request and the generation of the first membership query. This gives more time to user processes to react to congestion signals, and provide true membership information. This approach has the advantage of being very simple to implement, and also to reduce the amount of control messages exchanged on the local side (membership queries, reports and leaves), and towards the upstream router (prune and graft messages). The drawback is an additional delay in the recovery from timeout mispredictions.

The effectiveness of the prediction mechanism depends almost entirely on the speed of response of local receivers compared to the delay D. We should aim at keeping D as small as possible while making the chance of misprediction negligible. Unfortunately these parameters are highly variable, and this sug-

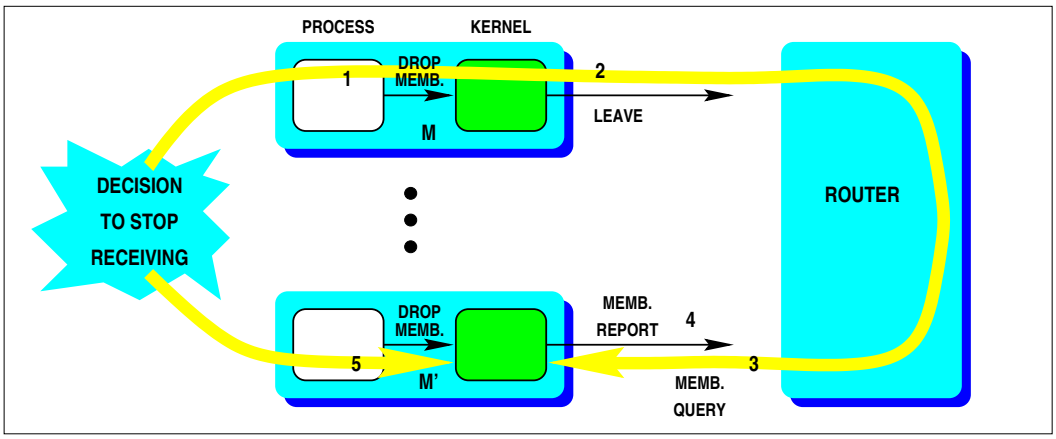

Fig. 3. A possible race condition leading to false membership reports. Depending on the delays in the two chain of events represented by the thick arrows, the kernel on M' can see a different sequence of events.

gests that an adaptive mechanism be used to determine the best value for D.

3.3 Robustness

In order to quantify the impact of errors and the robustness of our proposal to DoS attacks, consider the case where an attacker tries to disrupt service by sending LEAVE requests when the history is zero, the history vector is h bits long, and normal receivers remain in the group for a time T_S. T and R are the parameters used in the LAST RECEIVER poll, and D is the delay in generating the first membership query after a poll ($D = T$ in our implementation). When the attacker succeeds, causing the the algorithm to make a misprediction, a membership reply will be generated after an expected time $E[t] = D + T/(N+1)$, where N is the number of receivers still interested in the group. After the misprediction, the history will have non-zero value for at least T_d seconds, with $T_d = \min(h(TR + D), T_S)$, thus the fraction of time where data can be blocked is

$$f = \frac{E[t]}{T_d} \leq \frac{2}{h(R+1)}$$

This sets an upper bound to the damage that an attacker can do. In practice, an attacker can only succeed once on long lived sessions where other receivers do not leave groups for congestion control purposes, and once every h "true" LEAVE requests in other cases.

The choice of the length h of the history has an impact on the chance of errors. For the reasons mentioned above, h increases the robustness of the algorithm to errors or attacks, but also increases the time necessary to recover after a misprediction. We should keep in mind that misprediction can also come from transient events in congestion control schemes such as RLC, when a new receiver joins a group where other receivers are already present, and it might take some amount of time to synchronize with other group members. As a consequence, we suggest to use small values for h, e.g. 2 or 3, as a reasonable compromise between robustness and speed.

4 Experience with the implementation

We have implemented the mechanisms described in the previous Section in the mrouted3.9-beta3 code. The implementation required very minor modifications to the mrouted code, basically we just needed to introduce a new variable in the per-group state, and to make those data structures persistent across join/leave phases. The additional code closely reflects the one shown in Figure 2.

Initial tests have been performed using the modified mrouter with standard multicast applications such as `sdr, vat, vic` at the author's site. We have verified in this way the effectiveness of the prediction mechanism in detecting single or multiple listeners for a group and acting consequently.

We have then built a small test setup comprising two networks connected by mrouters through a tunnel. RLC instances have been run on the two local networks, and bandwidth limitations have been enforced on various parts of the network using the `dummynet` software [5] developed by the author. Preliminary experience with this setting has also shown a behaviour conforming to our expectations. These experiments have also given some advice on the setting of time constants in the RLC implementation to improve the interaction with the IGMP protocol.

4.1 Future work

The mechanisms described in this paper were devised mainly to improve the behaviour of RLC in presence of rapidly changing network conditions. Initial experience with the implementation has shown the effectiveness of our proposal, although we have not yet performed any large-scale test in the field. Such tests will be the subject of future experiments, although they are very difficult to set up on a real network because of the presence of non-pruning routers and IGMPv1 clients in many parts of the MBone.

We are now evaluating the behaviour of the modified IGMP in presence of multiple implementations of RLC and heterogeneous receivers. A result that we expect from these experiments is to have an indication of the typical response speeds of receivers, in order to understand the impact of false membership reports and the usefulness of implementing adaptive mechanisms for determining the delay in the generation of membership queries in response to a leave request.

Acknowledgements

This work has been partly supported by the Ministero dell' Università e della Ricerca Scientifica e Tecnologica (MURST) in the framework of the Project "Design Methodologies and Tools of High Performance Systems for Distributed Applications". The author is grateful to Lorenzo Vicisano for his help on this work, and for discussions on the interactions between IGMP and RLC. Many thanks to Bill Fenner who answered numerous questions on the mrouted code.

References

[1] W.Fenner, RFC2236, Internet Group Management Protocol, Version 2, Nov. 1997

[2] S.Floyd, V.Jacobson, C.Liu, S.McCanne, L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing, Scalable Reliable Multicast (SRM)", ACM SIGCOMM 95, available as `ftp://ftp.ee.lbl.gov/papers/srm_sigcomm.ps.Z`

[3] V.Hardman, I.Kouvelas, M.A.Sasse, A.Watson: "A packet loss Robust Audio Tool for use over the Mbone", Research Note RN/96/8, Dept. of Computer Science, University College London, 1996.

[4] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast", SIGCOMM'96, August 1996, Stanford, CA, pp.1-14.

[5] L.Rizzo, *Dummynet: a simple approach to the evaluation of network protocols*, ACM Computer Communication Review, Vol.27, n.1, January 1997, pp.31-41. Source code at `http://www.iet.unipi.it/~luigi/ip_dummynet/`

[6] L.Rizzo, L.Vicisano, *RMDP: an FEC-based Reliable Multicast protocol for wireless environments*, ACM Mobile Computing and Communications Review, Vol.2, n.2, April 1998

[7] L.Vicisano, L.Rizzo, J.Crowcroft: "TCP-like congestion control for layered multicast data transfer", Infocom98, S.Francisco, March 1998