Internet Engineering Task Force RMT WG
INTERNET-DRAFT
draft-ietf-rmt-bb-pgmcc-02.ps
Luigi Rizzo/U. Pisa
Gianluca Iannaccone/SprintLabs
Lorenzo Vicisano/Cisco
Mark Handley/ICSI
27 June 2003
Expires: December 2003

PGMCC single rate multicast congestion control: Protocol Specification

Status of this Document

This document is a product of the IETF RMT WG. Comments should be addressed to the authors, or the WG's mailing list at rmt@lbl.gov.

Abstract

This document describes PGMCC, a single rate multicast congestion control scheme which is TCP-friendly and achieves scalability, stability and fast response to variations in network conditions. PGMCC is suitable for both non-reliable and reliable data transfers. It is mainly designed for NAK-based multicast protocols, and uses a window-based, TCP-like control loop using positive ACKs between one representative of the receiver group (the ACKER) and the sender. The ACKER is selected dynamically and may change over time.

PGMCC is made of two components: a window-based control loop, which closely mimics TCP behavior, and a fast and low-overhead procedure to select (and track changes of) the ACKER. The scheme is robust to measurement errors, and supports fast response to changes in the receiver set and/or network conditions.

Table of Contents

1. Introduction

This document describes PGMCC, a single rate multicast congestion control scheme which is TCP-friendly and achieves scalability, stability and fast response to variations in network conditions.

PGMCC is designed for multicast sessions with one sender and one or more receivers, and is a good match for transport protocols using negative acknowledgements (NAKs) to collect feedback from the receivers. The congestion control scheme implemented by PGMCC closely mimics the congestion-control behavior of TCP, as it uses a window-based control loop which is run between the sender and a selected receiver called the ACKER. The role of the ACKER is to provide timely feedback in the same way as a TCP receiver; additionally, the ACKER is selected dynamically among the receivers as the one which would experience the lowest throughput if separate TCP sessions were run between the sender and each of the receivers.

Scalability in PGMCC comes from the use of negative acknowledgements (NAKs) for collecting feedback from receivers other than the ACKER. As a consequence, the usual techniques for NAK suppression and aggregation can be used to reduce the amount of feedback to the source and improve the scalability of the scheme.

PGMCC is designed to completely decouple congestion control from data integrity. As a consequence, the scheme can work with both reliable data transfer and unreliable communication protocols such as those used for video or audio streaming.

While designed with multicast in mind, PGMCC can be equally used as a replacement for TCP for unicast sessions which require a lower degree of reliability than what TCP offers.

1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 and indicate requirement levels for compliant PGMCC implementations.

2. Protocol Overview

PGMCC is based on two separate but complementary mechanisms:

- o A window-based control loop which closely emulates TCP congestion control.
  The window-based control loop is simply an adaptation of the TCP congestion control scheme to transport protocols where missing (because of network errors or congestion) data packets are not necessarily retransmitted, and so the congestion control scheme cannot rely on cumulative acknowledgements. In PGMCC, the ‘‘congestion window’’ is simulated using a token-based scheme which permits congestion control to be decoupled from retransmission state. One of the receivers in the group operates as the ACKER, i.e. the node in charge of sending positive acknowledgements back to the source and thus controlling the rate of the transfer.

- o A procedure to select the ACKER.
  The purpose of this procedure is to make sure that, in presence of multiple receivers, the

ACKER is dynamically selected to be the receiver which would have the lowest throughput if separate TCP sessions were run between the sender and each receiver.
For the acker selection mechanism, PGMCC uses a throughput equation to determine the expected throughput for a given receiver as a function of the loss rate and round-trip time. Unlike other schemes [2], the TCP throughput equation is not used to determine the actual sending rate, which is completely controlled by the window-based control loop.

In principle, PGMCC's congestion control mechanism works as follows:

- Receivers measure the loss rate and feed this information back to the sender, either in ACK or NAK messages.

- The sender also uses these feedback messages to measure the round-trip time (RTT) to each receiver.

- The loss rate and RTT are then fed into PGMCC's throughput equation, to determine the expected throughput to that receiver.

- The sender then selects as the acker the receiver with the lowest expected throughput, as computed by the equation.

The dynamics of the acker selection mechanism are sensitive to how the measurements are performed and applied. In the rest of this document we suggest specific mechanisms to perform and apply these measurements. Other mechanisms are possible, but it is important to understand how the interactions between mechanisms affect the dynamics of PGMCC.

2.1. Packet Contents

Before specifying the sender and receiver functionality, we describe the information required by PGMCC to perform its tasks. This information is carried in the data packets sent by the sender, and in the feedback packets sent by the receiver. As PGMCC will be used along with some transport protocol, the actual data and feedback packets will contain further information for use by the protocol itself. For this reason, we do not specify packet formats, as these depend on the details of the transport protocol used.

Note that the requirements of the transport protocol in terms of packet generation may differ from those of PGMCC. As an example, most NAK-based reliable multicast protocols do not use positive acknowledgements, but PGMCC requires ACKs for clocking out data packets; unreliable transport protocols might have no interest in generating NAKs for data integrity purposes, yet PGMCC depends on NAKs reaching the data sender in order to elect the ACKER.

Implementors may decide to insert PGMCC-related information in already existing protocol packets whenever possible, but in cases such as the ones described in the previous paragraph, it might be necessary to define and generate new packets exclusively for congestion control purposes. As an example, in a prototype implementation of PGMCC on top of the PGM protocol [7], some of the information used by PGMCC is already present in the original protocol packets, and PGMCC-specific information is carried as PGM options in ODATA and NAK packets. However, a new packet type has been defined for ACKs, which are generated according to the rules defined in this document.

2.1.1. Data Packets

Each data packet sent by the data sender contains the following information:

- o A SEQUENCE NUMBER. This number is incremented by one for each data packet transmitted. The field must be sufficiently large that it does not wrap causing two different packets with the same sequence number to be in the receiver's recent packet history at the same time.

- o A TIMESTAMP (or equivalent information, see Section 2.5) indicating when the packet with this sequence number has been sent. There is no requirement for synchronized clocks between the sender and the receivers. The timestamp is used to measure network round-trip times, so needs sufficient resolution for this task. A resolution of 1ms would be adequate.

- o The ACKER IDENTITY, i.e. the identity of the receiver in charge of sending an acknowledgement for this data packet. The ACKER is elected as a result of the process described in Section 2.3.
  A special value is used to indicate that no ACKER is designated for this packet -- this can happen at the beginning of a session or when the current ACKER leaves the group. Receivers interpret this value as a request to elect a new acker.

2.1.2. Feedback Packets

There are two types of feedback packets used by PGMCC: ACK packets and NAK packets. ACK packets are generated by the current ACKER, and are used to detect loss or successful delivery of packets, and to regulate the throughput accordingly. ACK packets also contain information used to determine the TCP-equivalent throughput for the ACKER.
NAK packets are sent by any receiver who experiences loss. They contain information used to determine the TCP-equivalent throughput for that receiver. In an actual protocol instantiation (such as PGM [7]), NAK packets might also be used by the protocol to request the retransmission of specific packets, and indicate the identity of the packet being requested.

Both ACK and NAK packets are sent by data receivers, and contain the following information:

- o The TIMESTAMP (or equivalent information) derived from the most recently received data packet according to one of the techniques described in Section 2.5.
  This value is used by the sender to measure the RTT to the receiver who generated this feedback packet.

- o ''p'', the receiver's current estimate of the LOSS RATE. The loss rate is measured by receivers as described in Section 2.6

In addition to the above, ACK packets (sent by the acker designated in the corresponding data packets) must also contain the following information:

- o RX_MAX, the highest sequence number among received data packets (taking care to deal with sequence number wrapping correctly).
- o ACK_BITMAP, a bitmap indicating the receive status of the latest N (typically N=32) data packets with sequence numbers RX_MAX-(N-1) to RX_MAX.

This information is used by the sender to record which packets have been received or lost, and manipulate the transmit window accordingly. Note that each ACK packet contains information about multiple packets, and this increases the robustness of the scheme to loss of ACK packets. This is necessary because ACKs are not sent reliably (unlike TCP's ACKs, which are cumulative).

2.1.3. Field sizes and formats

The following sizes and formats are suggested for the various fields used by PGMCC and transmitted over the network:

- o SEQUENCE NUMBERS
  32 bit, unsigned, network order.

- o TIMESTAMPS
  32 bit, unsigned, network order. A resolution of 1ms or better is desirable.

- o ACKER IDENTITY
  Same size and format of a network layer address (e.g. 32 bit for IPv4). Note though that using an IP address for the Acker Identify will cause problems with NAT traversal. Transport protocol designers might examine the SSRC mechanism used by RTP [6] as an alternative form of node identifier that could be used as Acker Identity.

- o LOSS RATE (“p”)
  16-bit unsigned integer, in network format, with 0 indicating no loss and 2^16-1 indicating 100% loss.

- o ACK BITMAP
  32-bit, in network format, with least significant bit indicating receive status of packet RX_MAX.

2.2. Window-based controller

In a window-based congestion control scheme such as TCP, the “congestion window” represents, among other things, the maximum amount of packets in flight at any time, which in turn controls the throughput of the session. The sender keeps track of the actual number of packets in flight, basing on its transmissions and the reception of acknowledgements.

The sender may dynamically change the size of the window, according to the congestion control scheme being used. In TCP, and PGMCC, an “Additive Increase Multiplicative Decrease” (AIMD) scheme is used: in absence of loss, the window is increased by some fixed amount (typically one packet) per round trip time (RTT), whereas upon loss the window is reduced to a fraction of its original value (typically halved) in each RTT in which a loss event is experienced.

In PGMCC the window is managed using a token-based mechanism, controlled by two variables:

- o A “Window Size”, W, which describes the current window size in packets.

- o A “Token Count”, T, which indicates the number of packets that can be transmitted. T is bounded above by W. It is decremented every time a packet is transmitted, and incremented every time an ACK is received, according to the rules below.

Note that these two variables need to hold non-integer data. Typically a fixed point representation with at least 16 bits for both integer and fractional parts would be acceptable for implementation purposes.

The information contained in each ACK is used to determine how many new packets are acknowledged by that ACK, and whether there are unacknowledged packets which were not reported in previous ACKs. The sender also schedules a timeout to react in case no ACKs are received.

The sender behaves as follows:

- o INITIALIZATION
  At startup, or after a timeout, both W and T are set to 1.

- o ACK RECEPTION, NO LOSS DETECTED
  If the incoming ACK reports new acknowledged packets, and no loss (as defined in the next paragraph) is detected, then the window is inflated by one packet per RTT.

  NOTE: during the slow-start phase, TCP opens the window exponentially up to the SSTHRESH value, which is computed by TCP according to the dynamics of the session and updated upon losses.

  We do recommend that PGMCC uses a similar strategy, but using a fixed, small value for SSTHRESH (e.g. 4 packets). In fact, due to the dynamicity of the ACKER, which might change on every single packet, it is hard to compute a reliable estimate of the SSTHRESH without keeping state for multiple receivers, and the benefits are small in any event.

  In summary, the reaction to ACK reception on no loss modifies T and W as follows (here, N is the number of new packets acknowledged by the incoming ACK):

```
if (W < SSTHRESH) then
   D = min(N, SSTHRESH - W) // use the first D acks for exp.opening
   N = N - D                // and the remaining ones for linear opening
   T = T + 2*D
   W = W + D
endif
// do linear window opening with the remaining acks
T = T + N * ( 1 + 1/W )
W = W + N/W
```

- o PACKET TRANSMISSION
  One token is consumed for each packet transmitted:

  T = T - 1

- o ACK RECEPTION, LOSS DETECTED
  If the incoming ACK reports an unacknowledged data packet which is followed by at least 3 acknowledged data packets, then the packet is assumed to be lost and PGMCC reacts by halving the window, in the same way as TCP after 3 duplicate acknowledgements. This is achieved by modifying T and W as follows:

  $$T = T - W/2 , W = W/2$$

  to simulate the multiplicative decrease.
  Additionally, all window manipulation is suspended for the subsequent RTT. This is achieved by recording the current transmit sequence number, and canceling any further manipulation of the window until feedback is received for the next transmitted packet, or until a timeout occurs.

2.3. Acker Selection

The ACKER selection process in PGMCC aims at locating the receiver which would have the lowest throughput if each receiver were using a separate TCP connection to transfer data.

Because the steady-state throughput of a TCP connection can be characterized in a reasonably accurate way in terms of its loss rate and round trip time [3], the throughput for each receiver can be estimated by using these two parameters.

Whenever an ACK or NAK packet from any of the receivers reaches it, the sender is able to compute the expected throughput T_i for that receiver by using the equation shown in Section 2.4, with the round trip time RTT and loss rate p and measured as described in Sections 2.5 and 2.6, respectively. At any given time, the sender stores the expected throughput for the current ACKER, T_acker. This value is updated every time an ACK or NAK from the current ACKER is received (note that, after a new ACKER is selected, the sender will typically receive ACKs from the old ACKER for one RTT, and the feedback from different ACKERs might be interleaved if the paths leading to them have different round trip times).

Whenever an ACK or NAK is received from another node i (a previous ACKER or some other receiver), the expected throughput T_i for that node is computed, and compared with T_acker. Node i is selected as the new acker if

$$T\_i < C * T\_acker$$

where the constant C between 0 and 1 provides some hysteresis and avoids too frequent oscillations in the choice of the ACKER. A suggested value for C is 0.75.

Note that, from an implementation point of view (see Section 2.4), it is more convenient to compute T_i ^(-2), so the above equation must be modified accordingly.

2.3.1. Initial Acker election

Upon reception of a data packet reporting that no acker is currently selected, receivers generate a dummy NAK report which is used to elect the initial acker. The NAK is sent with the usual feedback suppression mechanism dictated by the transport protocol (possibly with shorter time constants) to avoid feedback implosion, and the sender will select the source of the first incoming NAK as the new ACKER.

2.3.2. Acker dropouts

If the ACKER decides to disconnect from the session, it can cause the session to stop. To avoid this, it is recommended that an ACKER deciding to leave the session informs the sender by sending an ACK packet (or a duplicate) carrying an "ACKER_LEAVING" option. The reception of this packet by the sender will in turn trigger an initial acker election phase.

2.4. TCP Throughput Equation

Any realistic equation of TCP throughput as a function of loss event rate and RTT should be suitable for use in PGMCC. Unlike other schemes [2] where the throughput equation directly controls the transmit rate, in PGMCC the equation is used only for acker selection purposes, and the throughput values are only compared among themselves. As a consequence, we can use the following equation, derived from the one presented in [3] by setting RTO = 4 * RTT (as it is common practice):

```
M = 1/T = RTT_i * sqrt(p) * (1 + 9*p * (1 + 32*(p)^2))
```

where

M = 1/T is proportional to the inverse of the throughput for the receiver under consideration;

RTT is the round trip time for the receiver under consideration;

p is the loss rate for the receiver under consideration, between 0 and 1.0;

and multiplying constants are omitted.

The above equation is accurate on a wide range of loss rates, and also covers situations where retransmission timeouts have a significant impact on the throughput of the protocol.

Note that when p=0, the equation yields 1/T = M = 0. This does not constitute a problem as we can still compare the M values computed for different receivers to determine the acker. Also note that it is easier to compute M^2 instead of M, because the former does not require the use of sqrt().

In future, different throughput equations may be substituted for this equation. The requirement is that the throughput equation be a reasonable approximation of the sending rate of TCP for conformant TCP congestion control.

The parameters p and RTT need to be measured or calculated by a PGMCC implementation. The measurement of RTT is specified in Section 2.5; the measurement of p is specified in Section 2.6.

2.5. RTT measurement

In PGMCC, the RTT is measured by the sender making use of the timestamp (or equivalent information) echoed back by each receiver in feedback messages. Three procedures are possible to measure the RTT, as follows. In no case is it required to have clock synchronization between sender and receivers.

2.5.1. Explicit Timestamp

This first technique relies on the transmission of a timestamp TS_j with each data packet j.
The receiver will record the most recently received timestamp, and will echo it back to the source when generating an ACK or a NAK. If the feedback is delayed, the time elapsed between the reception of the timestamp and the generation of the feedback should be added to the echoed timestamp.
The sender computes the RTT by subtracting the received timestamp from the current value of the clock.

The resolution of the timestamp value should be good enough for reasonable precision measurement of typical network round trip times. If receivers need to apply correction for delayed feedback, it is necessary that receivers know the resolution of the timestamp clock. A suggested value is 1ms.

2.5.2. Implicit timestamp

With this technique, the sender will record a timestamp TS_j for each transmitted data packet j, but the timestamp will not be transmitted with the packet itself.
The receiver will record the most recently received sequence number, and will echo it back to the source when generating an ACK or a NAK.
The sender computes the RTT by looking up the timestamp associated with the sequence number received in the feedback packet, and subtracting it from the current clock value.

If the feedback from the receiver is delayed, as it is commonly the case for NAKs, the receiver can compute, and send back to the source, a correction term corresponding to the time elapsed between the reception of the timestamp and the generation of the feedback. The correction term will then be subtracted by the sender in order to obtain the correct estimate of the RTT.

This RTT measurement technique is equivalent to the previous one, but it saves some space in data packets as the timestamp does not need to be sent explicitly. Feedback packets might become larger if the correction value is transmitted explicitly; but in many cases, the sequence number will already be present for other reasons (e.g. ACK packets), and wherever space is a concern the sequence number and the correction term can be packed in a single 32-bit word without loss of precision.

2.5.3. Sequence numbers

This technique is the least precise, but it does not rely on the presence of a high resolution clock on the nodes.
The sender will not compute any timestamp, and just send data packets with their sequence number j.
The receiver will record the most recently received sequence number, and will echo it back to the

source when generating an ACK or a NAK.
The sender computes the RTT as the difference between the most recently sent sequence number and the sequence number received from the ACK or NAK packet.

Note that in this case the RTT is not measured in seconds, but in "sequence numbers", which are monotonically, but not uniformly, increasing with time. The two measurements are equivalent if the sender transmits at a constant rate. When the data rate changes over time (as it is normally happens, given that PGMCC controls the actual data rate), then the "measured" RTT values grow with the actual transmit rate. This can influence the correctness of the results when comparing two measurement done over different and only partially overlapping time (and sequence number) intervals where the transmit rate incurs a significant change.

2.5.4. Recommendations

Whenever possible, the measurement of the RTT should be carried out using either explicit or implicit timestamps, and by keeping track of the "correction term" (the delay between data reception and feedback generation).

If the receiver does not have a clock with a suitable resolution, the correction term might not be present (or be inaccurate). In this case the timestamps received by the sender on NAK packets might be in error, in the worst case, by as much as the packet interarrival time. This error will normally not be present on ACK packets, which are sent immediately. A suitable correction should be applied by the sender in order to avoid systematic errors.

The measurement based on sequence numbers is less accurate, but also less sensitive to errors due to the lack of the correction term. In fact, the measurement error induced by the lack of the correction term can be at most one unit. This suggests that, when the correction term is not available, measurements based on sequence numbers should be favoured. Simulations have shown that the acker selection mechanism performs moderately better when the RTT measurement is based on timestamps, but performance is reasonably good also with measurements based on sequence numbers.

2.6. Loss rate measurement

The loss measurement in PGMCC is entirely performed by receivers. The measurement results do not directly influence the transmit rate, but are only used for comparison purposes. As a consequence, the scheme is reasonably robust to different measurement techniques, as long as they are not influenced too strongly by single loss events.

The main method suggested for loss measurement is Exponentially Weighted Moving Average (EWMA), which is formally equivalent to a single-pole digital low pass filter applied to a binary signal x_i, where x_i = 1 if packet i is lost, x_i = 0 if packet i is successfully received.

The loss rate p_i upon reception or detection of loss of packet i is computed as

```
p_i = c_p * p_{i-1} + (1 - c_p ) * p_i
```

where the constant c_p between 0 and 1 is related to the bandpass of the filter. Experiments have shown good performance with c = 500/65536, and computations performed with fixed point arithmetic and 16 fractional digits.

As an alternative to EWMA, the technique used in TFRC [2] can be adopted. Simulations have shown a moderate improvement in the acker selection mechanism by measuring loss using the TFRC loss estimator, which is however slightly more expensive to compute than the EWMA loss estimator in presence of packet reordering.

2.7. Timeouts

When a packet is transmitted, the sender schedules a timeout to prevent stalls upon loss of ACKs or disconnection of the ACKER. In TCP, which has a similar problem, the timeout value is computed by accumulating statistics (SRTT and RTTVAR) on RTT samples, starting from a default initial value (3s) when no RTT samples are available.

PGMCC can use a similar scheme to compute the timeouts, remembering that upon ACKER changes (which may be very frequent), the computation of SRTT and RTTVAR must be restarted from the beginning, unless the sender decides to keep state for at least a small number of recent ackers.

Because the ACKER can leave the group without notifying the sender, after a number of successive timeouts the sender MUST force the election of a new ACKER. We recommend this new election to be performed after two successive timeouts.

2.8. Interaction with feedback suppression schemes

Several schemes are used by NAK-based multicast protocols to reduce the amount of feedback directed toward the source and make the protocol scale with large populations of receivers. Such schemes typically rely on randomly delaying NAK generation, and suppressing pending NAKs when an equivalent NAK or a retransmission is heard; or, intermediate nodes such as routers can implement some form of feedback aggregation and filtering.

Such schemes might prevent NAKs from potential ACKER candidates from reaching the source. This filtering might impact the speed at which PGMCC selects the correct ACKER, though initial experience from simulations seem to suggest that PGMCC behavior is not severely affected by NAK suppression schemes.

2.9. Interaction with ECN

PGMCC can use ECN notifications in much the same way as actual losses, and use such notifications to control the throughput of the session.

At the receiver, ECN-marked data packets can be considered as lost packets for the purpose of loss rate computation and ACK/NAK generation. If the ACKER sends an ACK for ECN-marked packets, that ACK MUST report that the packet being acknowledged that was ECN marked.

Similarly the ACKER must indicate in the ACK packet's received packets bitmap that the packet was ECN-marked, or that the packet was lost.

We note that to support use of the ECN nonce, the ACK packet's received packets bitmap would require two bits per packet being reported.

3. Procedures - Sender

The following pseudo-code specifies the complete behavior of the sender in PGMCC.

```
initialization:
  T = 1 ; W = 1 ;   /* initialize window and number of tokens */
  RETRY = 0 ;       /* number of consecutive timeouts so far  */
  < initialize p, RTT for acker to default values >
  ACKER = NO_ACKER; /* no acker is known                      */
  < initialize sequence numbers >
  QUEUED = 0;       /* packets waiting to be transmitted      */

on transmission request:
  send_packet() ;

on timeout expiration :
  T = 1 ; W = 1 ;   /* initialize window and number of tokens */
  if (RETRY < RETRY_MAX)
    RETRY = RETRY + 1
  else
    ACKER = NO_ACKER ; /* old acker is not valid anymore      */
  send_packet() ;

on ACK/NAK reception from receiver I :
  < compute p and RTT for source of this ACK, see Sec. 2.5 and 2.6 >
  RETRY = 0 ;
  if (ACKER == NO_ACKER) { /* select current as acker is no other known */
    ACKER = I ;
    T = T + 1 ;
  }
  if (ACKER != I)
    < select acker according to Sec. 2.3 > ;
  else {
    <update acker statistics (p_ACKER, RTT_ACKER) >
    if (packet_type == ACK) {
      < update_window according to Sec.2.2 >
      send_packet ;
      if (ack_pending)
        update_timeout ;
  }
}
```

```
send_packet() {
  if (QUEUED > 0 && T >= 1) {
    < transmit one packet >
    T = T - 1 ;
    QUEUED = QUEUED - 1 ;
  }
  if ( <there are unacked packet> )
    <set a timeout, see Sec.2.7 >
}
```

4. Procedures -- Receiver

The following pseudo-code specifies the complete behavior of the receiver in PGMCC.

A receiver only transmits an ACK packet when it receives a data packet for which the receiver is designated as the ACKER by the data packet itself. A receiver can transmit a NAK packet after it has detected that a data packet is missing and a suitable delay has passed, as dictated by the feedback suppression rules of the protocol in use.

The data packet contains acknowledgement status about the most recent 32 sequence numbers known to the receiver.

```
on initialization/session setup:
  < initialize state variables and ACK bitmap >

on DATA packet reception:
  < update p measurement according to Sec.2.6 >
  < record timestamp and packet reception time >
  if (ACKER == this_node) {
    < send an immediate ACK >
  }
  if ( <some data packet is missing and unacknowledged > )
    < schedule a timeout for NAK transmission >

on NAK reception:
  < suppress any pending NAK transmission for the sequence
    number indicated in the NAK >

on timeout:
  if ( < there are missing and unacknowledged packets > ) {
    < send a NAK for one or more of the missing packets >
    < mark such packets as acknowledged >
    if ( <some data packet is missing and unacknowledged > )
      < schedule a timeout for NAK transmission >
  }
```

5. Security Considerations

PGMCC is not a transport protocol in its own right, but a congestion control mechanism that is intended to be used in conjunction with a transport protocol. Therefore security primarily needs to be considered in the context of a specific transport protocol and its authentication mechanisms.

Congestion control mechanisms can potentially be exploited to create denial of service. This may occur through spoofed feedback. Thus any transport protocol that uses PGMCC should take care to ensure that feedback is only accepted from the receiver of the data. The precise mechanism to achieve this will however depend on the transport protocol itself.

In addition, congestion control mechanisms may potentially be manipulated by a greedy receiver that wishes to receive more than its fair share of network bandwidth. A receiver might do this by first reporting inflated loss and RTT samples, in order to get selected as the ACKER, and then generating ACK at the desired rate (including possibly claiming to have received packets that in fact were lost due to congestion). Possible defenses against such a receiver could be based on the sender verifying the correctness of the loss and RTT samples supplied by the receiver. A PGMCC sender SHOULD compare the receiver reports on loss rate and RTT with the information derived directly from the incoming stream of ACKs. In case of discrepancy of the reports, a PGMCC sender SHOULD mark the current acker as ineligible and initiate a new acker election. The decision on how large that discrepancy should be before initiating a new acker election is left to the implementation.

Also, the sender MAY include some form of nonce that the receiver must feed back to the sender to prove receipt. However, the details of such a nonce would depend on the transport protocol, and in particular on whether the transport protocol is reliable or unreliable.

6. Authors' Addresses

Luigi Rizzo
luigi@iet.unipi.it
Dip. Ing. dell'Informazione,
Univ. di Pisa
via Diotisalvi 2, 56122 Pisa, Italy

Gianluca Iannaccone
gianluca@sprintlabs.com
Sprint ATL
1 Adrian Ct., Burlingame, CA, USA

Lorenzo Vicisano
lorenzo@cisco.com
cisco Systems, Inc.
170 West Tasman Dr.,
San Jose, CA, USA, 95134

Mark Handley
mjh@aciri.org
ICSI,
1947 Center St, Berkeley, CA, USA, 94704

7. Acknowledgments

We would like to acknowledge feedback and discussions on equation-based congestion control with a wide range of people, including members of the Reliable Multicast Research Group, the Reliable Multicast Transport Working Group, and the End-to-End Research Group.

[1] Bradner, S., Key words for use in RFCs to Indicate Requirement Levels (IETF RFC 2119) http://www.rfc-editor.org/rfc/rfc2119.txt

[2] Floyd, S., Handley, M., Padhye, J., Widmer, J., "Equation-Based Congestion Control for Unicast Applications", ACM SIGCOMM 2000, Stockholm, Aug. 2000

[3] Padhye, J. and Firoiu, V. and Towsley, D. and Kurose, J., "Modeling TCP Throughput: A Simple Model and its Empirical Validation", Proc ACM SIGCOMM 1998.

[4] Mankin, A., Romanow, A., Brander, S., Paxson, V., "IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols," RFC2357, June 1998.

[5] Rizzo, L., "pgmcc: a TCP-friendly single-rate multicast congestion control scheme", ACM SIGCOMM 2000, Stockholm, Aug.2000

[6] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V., "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, Jan 1996.

[7] Speakman, T., Crowcroft, J., Gemmell, J., Farinacci, D. , Lin, S., Leshchiner, D., Luby, M., Montgomery, T. , Rizzo, L., Tweedly, A., Bhaskar, N., Edmonstone, R., Sumanasekera, R., Vicisano, L., PGM Reliable Transport Protocol Specification, RFC 3208, December 2001. rfc3208.txt also available at ftp://ftp.rfc-editor.org/in-notes/rfc3208.txt

8. Full Copyright Statement