Strawman Specification for TCP Friendly (Reliable) Multicast Congestion Control (TFMCC)

Mark Handley, Sally Floyd
ISI/LBNL
mjh@isi.edu/floyd@ee.lbl.gov

1 Introduction

This document presents a *strawman* specification for TCP-Friendly Reliable Multicast Congestion Control (TFMCC). It is not a standard of any form. We do not suggest basing products on this specification. Eventually, if further research shows this specification to perform acceptably, we envisage moving this document into the IETF for standardization, but the document is not currently at this stage. Many people have contributed to this specification in one form or another - by the time it is ready for standardization we hope that many more people will have contributed ideas and performed simulation work to test the details.

The mechanisms described in this specification are intended to be agnostic as to the choice of reliable multicast protocol used. Clearly we can not completely ignore this issue though. The baseline specification is intended to work in the context of a non-hierarchical RM protocol with NACK-based reliability. This *does not* mean that we reject hierarchical protocols - merely that we envisage this mechanism might function at each level in a hierarchical RM protocol. We also do not reject ACK-based RM protocols - in an environment in which they scale, ACK based protocols have a more complete feedback mechanism than NACK-based protocols, and we envisage that TFMCC will work *better* when modified to provide feedback using the ACK stream. However, we initially specify only NACK-based feedback because this is the most general non-hierarchical case.

We envisage that over time, this document will evolve to contain additional sections indicating how to make use of hierarchy and better feedback mechanisms in environments where these mechanisms are available.

1.1 Assumptions

We make the following assumptions about the application in use:

- The application can tolerate unbounded variation in the time to transmit a fixed amount of data.
- The data flow is one-to-many. Some many-to-many applications may be viewed of consisting of multiple independent one-to-many flows, and so this specification may apply to them too, but we shall not address them explicitly here.
- The data flow is continuous. It does not stop because the application temporarily runs out of data to send.

- The data flow is large - the duration of the flow is expected to be a large number of network round trip times.
- Feedback from receivers to the sender is possible using multicast.
- The desired behaviour is for the sender to transmit at the short-term mean rate that would be achieved by a TCP flow from the sender to the slowest of the receivers.
- It is acceptable for a transport protocol to respond to packet losses on a slower timescale than the one-RTT timescale that TCP uses.

All of these conditions should be true for an application to use TFMCC. It may be possible to modify TFMCC to cope with applications that do not fit these assumptions. For example, a unicast feedback mechanism may be substituted for the multicast mechanism if alternative means are devised to prevent response implosion. We do not deal with such modifications in this document. Companion documents may do so in the future.

The application we have in mind is a bulk-data transfer application, where a large amount of data must be distributed to a large number of sites. However, other applications may also satisfy these assumptions, including unreliable multicast applications. This specification does not address reliability - it is up to the reliable multicast protocol to achieve this within the bandwidth bounds provided by TFMCC.

2 Background

TFMCC is based around modeling the long term behaviour of TCP, and using such a model to achieve similar throughput for a multicast flow. There are several models of TCP long-term behaviour . The range of bandwidths that TFMCC is applicable for depends on the choice of model. Of the existing published models[3][2][1], our simulations show that only the one by Padhye et al[1] is suitable for general purpose use. The other models do not account for TCP retransmission timeouts, and fail to throttle bandwidth appropriately at loss rates above about 5%.

We choose the approximate solution from [1] and ignore the term due to receiver window limiting - this is a self-imposed handicap unrelated to network congestion, so we can safely ignore it. The equation given is:

$$B(l) = \frac{s}{t_{RTT}\sqrt{\frac{2bl}{3}} + t_0 min\left(1, 3\sqrt{\frac{3bl}{8}}\right) l(1+32l^2)} \tag{1}$$

l is the loss fraction, $B(l)$ is the data rate in bytes/sec, and s is the packet size in bytes. In addition, b is the number of packets acknowledged by each TCP ACK, t_0 is the TCP retransmission timeout value, and t_{RTT} is the round trip time from sender to receiver and back.

When applying this equation for multicast congestion control, we face several challenges:

- Choosing a suitable number of transmitted packets to sample the loss rate over.
- Communicating the loss fraction from the receiver that would have the lowest TCP rate in a timely manner without causing a response implosion at the sender.

- Discovering the RTT to the receiver that would have the lowest TCP rate.
- Approximating t_0 at the receiver that would have the lowest TCP rate.

To send at the correct rate, the sender must continuously know the values of $B(l)$ (or l, t_{RTT} and t_0) at all receivers and not just the value of l. This is because the receiver with the highest loss fraction is not necessarily the receiver that would achieve the lowest rate with TCP because there may be a receiver with a slightly smaller loss rate and a much higher RTT.

In the general case, this cannot be done without causing a response implosion, although it may be possible for some small groups using ACK-based protocols. Thus we specify ways to approximate the behaviour that would be obtained with the full information using only incomplete information. Clearly protocols that can provide more complete information than we assume can be achieve better approximations.

Note that later versions of this specification may substitute different functions for $B(l)$ that remedy deficiencies of the Padhye equation when RED gateways are employed and when modeling TCP-SACK. These revised functions should be able to be plugged into this specification without requiring further changes and result in slight improvements to performance.

3 Calculating loss fraction at the receiver

In every data packet, the sender transmits the current data rate it is transmitting at, the estimate of worst case t_{RTT}, the timestamp when it sent the packet, and two sequence numbers, MSEQ and DSEQ. MSEQ is the message sequence number which is increased by one for every data packet sent regardless of whether it is a retransmission, and DSEQ which uniquely identifies the data packet itself for the purposes of retransmission requests. We do not specify the form that DSEQ takes. Some protocols may be be able to use MSEQ only.

The receiver keeps track of the MSEQ space maintaining information about which packets were received and which were lost. For now, we assume that it keeps an infinite history, but will address how to make this finite later.

The receiver calculates the inverse of the function $B(l)$ using the rate parameter supplied by the sender and it's locally calculated t_{RTT} and t_0 values (see below). This gives the fraction of packets, l_{exp}, that would be expected to be lost to give this data rate. The value k/l_{exp} then gives us the number of packets of history to examine to determine the loss fraction. k is a constant value that is used to trade off responsiveness against sampling accuracy. We recommend a default value of $k = 4$, but this should be the subject of future study.

To calculate the loss fraction we count loss events rather than lost packets. Thus lost packets are ignored if they are within one RTT of a lost packet that was not ignored. This models TCP-SACK behaviour where only one backoff is performed per RTT. If a receiver has an estimate of its own RTT it uses that. If not, it uses the RTT advertised by the sender.

If k/l_{exp} is greater than the length of the history, the length of the history should be used instead.

If k/l_{exp} does not include several loss events, the behaviour may be unstable and cause oscillation. Thus if k/l_{exp} does not include E loss events, we calculate the loss fraction by including all the packets back to the Eth most recent loss event or the beginning of the history if E loss events have not occurred. We recommend a default value of $E = 4$ to be used based on provisional simulation results, but this should be the subject of future study.

4 Calculating the RTT

Ideally we'd like each receiver to be able to calculate or approximate the RTT to the sender. In ACK-based protocols, with small numbers of receivers, this is possible, and should be done.

With larger numbers of receivers, this is not possible to achieve without synchronized clocks at the sender and receivers. We STRONGLY RECOMMEND that reliable multicast senders either have an external clock source such as GPS or use NTP to obtain a good clock reference.

If the sender knows unambiguously that it has a clock synchronized using NTP or an external source such as GPS, it indicates this in the data packets along with the sender timestamp. Any receiver that knows unambiguously that it has a clock similarly synchronized SHOULD use its local clock to calculate the one-way propagation delay and double this to approximate the RTT. If the resulting value is less than 100ms, assume 100ms is the RTT until the RTT can actually be measured, because NTP cannot be relied upon to synchronize to very small absolute differences on machines with poor inbuilt clocks. We RECOMMEND that multicast receivers have a clock synchronised to an accurate external time source or use NTP to obtain a good clock reference.

If the sender does not have or does not know it has a synchronized clock, or if a receiver does not have a synchronized clock, the congestion control scheme can still function, but it will be more conservative in its estimates of RTT. Such a receiver should initially assume that the RTT is 500ms. Although the real RTT may be greater than this, it will not be greater by more than an order of magnitude on links that do not leave the Earth/Moon system. We do NOT RECOMMEND using this mechanism for interplanetary communication.

Assuming that we have a baseline approximation for the RTT at a receiver, either the default value above, or one obtained from synchronized clocks, then the receiver uses this value to calculate the $B(l)$. As we indicate below, the receiver with the lowest value of $B(l)$ will report this value along with the arrival time of a recent data packet, the sender timestamp from that packet, and the time the response was sent.

The sender calculates the real RTT from this information, and reports this RTT and the receiver's address in a later data packet. When the receiver receives this data packet, it updates its estimate of RTT, t_{RTT} from the measured RTT, t_{sample}, using the EWMA filter:

If $i = 0$
$t_{RTT,i+1} = t_{sample}$
Else
$t_{RTT,i+1} = 0.25 t_{sample} + 0.75 t_{RTT,i}$

If the receiver never sends a report because its calculated rate $B(l)$ does not require it to do so, it would otherwise not be able to reduce its RTT value. If the real RTT to all receivers were significantly less than 500ms, it would require all receivers to exchange packets with the sender before they could all reduce their calculation of RTT and hence allow the data rate to increase.

To allow the RTT estimates to decay in such cases, we allow the sender to report an estimate of the worst RTT for the whole group based on its sampling of the RTT. Receivers without synchronized clocks that have not exchanged any report packets with the sender may use this value of RTT if the sender reduces it from the initial 500ms value. Receivers with synchronized clocks that have not exchanged any report packets with the sender SHOULD still use their own estimate of the RTT.

Several methods also exist by which a receiver may estimate its RTT to the sender without exchanging

messages directly with the sender.

4.1 Third-Party RTT Estimation Using Synchronized Receiver Clocks

If the sender does not have an externally synchronized clock, but two receivers R_1 and R_2 do have such clocks, then if R_1 measures its RTT to the receiver using a feedback message and timestamps its multicast feedback message, then R_2 may be able to estimate its own RTT to the sender. If the sender sent the packet at time t_s, R_1 received the packet at time t_r, and the measured RTT from the sender to R_1 and back to the sender is t_RTT1, then R_2 can now estimate R_1's clock offset from the sender's clock to be $t_r - (t_s + t_{RTT1}/2)$. Depending on how accurately R_2 believes its clock is synchronized to R_1, this gives it a good measurement of the one-way propagation delay from the sender to R_2. The RTT is estimated by doubling the one-way delay.

If R_1 and R_2 are synchronised by GPS, they may assume their clocks are synchronized to 1ms and R_2 may use the estimate directly. However, if they are synchronized by NTP, the RTT estimate should be used only if it exceeds 100ms, otherwise 100ms should be used as the RTT measurement.

Estimating the offset from RTT information is clearly assuming that the reverse path and the forward path are similar. If the forward path from the sender to R_1 is much different (either longer or shorter) than the reverse path, then the estimate of clock offset will be incorrect by half the difference of the two paths. Thus if R_2 derives a one-way delay value of less than $t_{RTT1}/2$ it should use t_RTT1 as its RTT measurement instead of twice the one-way delay value.

Estimating the RTT from one-way information is unlikely to be a problem so long as the forward (data) path is the one we are principally concerned with.

We are clearly attempting to be conservative here. If the estimate is too low, a receiver may never measure its own RTT, but suffer significant congestion. However, if the estimate is too high, a real measurement will be made if this receiver calculates the lowest value of $B(l)$.

As the senders clock may drift, the estimate of the sender's clock offset should not be used after a long period of time. However, third-party estimation in this manner can significantly speed up the initial convergence of TFMCC.

4.2 Third-Party RTT Estimation Using Hierarchy

Estimates of the RTT can also be made by exchanging messages with a nearby receiver which has itself exchanged a message with the sender. Many different ways exist to do this. Some RM protocols may already include such hierarchies to produce scalable session messages or loss feedback, and we encourage such protocols to make use of their hierarchy to scalably estimate RTTs for receivers that have not exchanged feedback with the sender.

An example of a scalable hierarchical control message scheme that can be applied to a non-hierarchical RM protocol to estimate RTT is found in [4].

4.3 Sender's Estimate of Worst RTT

TFMCC needs to have an estimate of receiver population to be able to have the sender estimate the worst RTT. If no estimate of receiver population is available, the sender SHOULD NOT reduce its estimate of worst case RTT because it has no idea how representative a sample it has. Equally, if an insufficient proportion of the receivers have sent responses, the sender SHOULD NOT reduce its estimate. Further research is required to decide what criteria to use to decide what a sufficient proportion is.

The sender should keep a histogram of the RTTs that it has received from receivers. Based on this histogram, it can calculate an estimate of the worst case RTT ignoring outliers. It is necessary to ignore outliers because various buggy router implementations or transient routing loops may cause spurious large RTTs to be reported. The sender chooses the 95th percentile in this histogram (ie the RTT below which 95% of RTT reports fall) as its estimate of the worst case RTT. Clearly this requires at least 20 reports to decide, and we conservatively require 100 reports before the sender switches over from reporting 500ms to reporting the 95th percentile RTT. With typical internet RTTs this switchover will occur after a few tens of seconds.

For long lived sessions, the sender may age the worst-case RTT table so that older RTT values cease to affect the calculation. It may do this in any reasonable manner so long as the histogram represents the several hundred most recent RTT reports at all times. Aging the RTT estimate is a subject for further experimentation.

5 Estimating t_0

t_0 is the TCP retransmission timeout value. In TCP it is normally calculated as:

$$t_0 = t_{srtt} + 2\ t_{rttvar}$$

where t_{srtt} is the smoothed RTT estimate and t_{rttvar} is the variance of the RTT. Both are calculated using EWMA filters from the current sampled RTT, $t_{sampleRTT}$, by:

$$t_{srtt,i+1} = 7/8\ t_{srtt,i} + 1/4\ t_{sampleRTT}$$

$$t_{rttvar,i+1} = 3/4\ t_{rttvar,i} + |t_{sampleRTT} - t_{srtt,i}|$$

However, whereas the RTTs for TFMCC are measured as accurately as possible, in TCP, $t_{sampleRTT}$ is often sampled with course clock granularity. This varies from TCP to TCP, but 500ms is not uncommon. In such systems the minimum value for t_0 is 500ms, not matter what the real RTT is.

Trying to model TCP accurately in this respect is impossible without an RTT measurement for every few data packets from each receiver. This clearly does not scale past very small receiver groups. Also there's a difficult decision of which TCP to model - those with poor clock granularity or those with fine clock granularity. There is no clearly right answer to this question.

In the absence of any better model for TCP that can be scalably deployed in a multicast congestion control mechanism, we recommend:

$$t_0 = M * t_{RTT}$$

The value of M is a subject for future experimentation, but empirical and simulation evidence shows that a value of 4 is not unreasonable.

6 Feedback Mechanism

In a small-scale ACK-based protocol where every few data packets elicit an ACK from every receiver, the calculated value of $B(l)$ and the echo of the senders timestamp should be reported by all receivers in every ACK packet.

In NACK-based protocols with large receiver sets, it is usually infeasible for all receivers to send a report every few packets.

Where a NACK-based protocol uses on-demand packet-level FEC, there will typically be a NACK-suppression mechanism that ensures that the receiver missing the most packets responds indicating how many packets are missing. Such a response would then normally also carry the feedback listed above. However, this is not sufficient to perform congestion control, because the receiver with the highest loss rate may not be the receiver with the lowest value of $B(l)$. Thus a similar suppression mechanism needs to be run in parallel to the loss feedback mechanism so that the receiver with the lowest value of $B(l)$ also sends feedback. We do not recommend a particular loss suppression scheme for this case because a simple variant of the existing scheme for loss reports is probably more appropriate.

For protocols without an existing suppressed feedback mechanism on which to piggy-back congestion feedback, we suggest the mechanism below.

6.1 Sample Feedback Suppression Mechanism

This area requires significant further experimentation to show that it works OK, and to fine tune the mechanism.

The basic feedback suppression mechanism relies on multicast responses from one receiver suppressing responses from other receivers that have higher values of $B(l)$. Every so often, as indicated by a flag set in a data packet, every receiver sets a feedback timer to a value calculated randomly as below. If the timer expires, that receiver sends its feedback packet. If, however, it hears another receiver with a lower value of $B(l)$ reporting for the same sender timestamp, the receiver cancels the feedback timer. Note that several feedback timers may be running simultaneously triggered by different flagged data packets. Suppression only affects the one timer associated with the same reported sender timestamp.

As we do not know how many receivers received high loss in a reporting interval, we need to use an exponentially weighted random timer. We additionally bias this timer by the bandwidth rate being reported. If the senders current reported sending rate is B_{send}, and the loss rate a receiver wishes to report is $B(l)$, then we set the feedback timer, t_{delay}, to:

$$t_{delay} = D_1 + Rlog_2(2^{D_2/R} * X + 1)$$

Where R is the *senders* current reported value for worst case RTT, X is a random number chosen from the uniform random interval [0:1], and D_1 and D_2 are determined by:

$$D_1 = TBD$$

$$D_2 = TBD$$

We envisage some function of $B(l)/B_{send}$, perhaps weighted by receiver set size will be used to determine D_1 and D_2, but are reluctant to propose precise functions without further simulation or analysis.

7 Messages

We cannot give packet formats in this specification because TFMCC is intended to be integrated into a variety of multicast protocols. However, most protocols are likely to require a similar set of information.

7.1 Sender Messages

Piggy-backed on all (or many) of the senders data packets will be the following information:

Report Flag This indicates that receivers are to set a response timer to make a congestion control feedback report. It is used to synchronize reports so that feedback suppression happens correctly.

NTP Clock Sync Flag This indicates that the sender's clock is synchronized using NTP.

External Clock Sync Flag This indicates that the sender's clock is synchronized using an external clock source accurate to within 1ms.

Send timestamp The absolute time the packet was sent measured in units of milliseconds.

Worst Case RTT The senders current estimate of near-worst case RTT measured in units of milliseconds.

RTT Measurements A list of pairs of IP addresses and the RTTs measured to those hosts using receiver feedback.

Not all of this information is required in every data packet, but if the report flag is set, the send timestamp must be present. Also the worst case RTT should be sent at least once per worst-case RTT.

All data packets must contain a strictly monotonically increasing message sequence number (MSEQ) which must not wrap for significantly longer than the longest message history that will be needed.

7.2 Receiver Feedback Messages

Receiver feedback may be piggy-backed on other control messages so long as this does not adversely affect the timeliness of the reporting or feedback suppression.

Each feedback message contains the following information:

B(1) The value of bandwidth calculated by the receiver from its loss record and RTT measurements.

Sender Timestamp The sender timestamp of the data packet triggering this feedback message.

Receive Timestamp The time, measured at the receiver, that the data packet triggering this message arrived.

Feedback Timestamp The time, measured at the receiver, that this feedback message was sent.

NTP Clock Sync Flag This indicates that the receiver's clock is synchronized using NTP.

External Clock Sync Flag This indicates that the receiver's clock is synchronized using an external clock source accurate to within 1ms.

See section 4.1 for an explanation of the purpose of these flags and why we prefer two absolute timestamps to a single difference of the two.

8 Simulation results

Preliminary simulation results of the use of TCP throughput equations for *unicast* congestion control can be found at: http://north.east.isi.edu/ mjh/results.ps

9 Acknowledgments

This specification is based on the ideas of many people, including Allison Mankin, Brian Whetten, Jim Conlan, Steve Deering, Jon Crowcroft, Jean Bolot, J. Padhye, V. Firoiu, Don Towsley, and Jim Kurose and other members of the Reliable Multicast Research Group. Our apologies to those we've accidentally omitted.

References

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

[2] J. Mahdavi, S. Floyd, "TCP-Friendly Unicast Rate-Based Flow Control", Technical note sent to the end2end-interest mailing list, 1997. http://www.psc.edu/networking/papers

[3] M. Mathis, J. Semke, J. Mahdavi, T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", ACM Computer Communication Review, 27(3):67-82, Jul. 1997.

[4] P. Sharma, D. Estrin, S. Floyd, "Scalable Session Messages in SRM", http://catarina.usc.edu/estrin/papers/infocom98/ssession.ps