AMCA: an Active-based Multicast Congestion Avoidance Algorithm

Moufida Maimour, Cong-Duc Pham

AMCA: an Active-based Multicast Congestion Avoidance Algorithm

M. Maimour and C. D. Pham
RESO/LIP - ENS, 46 alle d'Italie 69364 Lyon Cedex 07 - France
email:{mmaimour,cpham}@ens-lyon.fr

Abstract

Many works have recently addressed the issue of congestion control for multicast communications and the problem is known to be highly complex. Scalability, responsiveness, stability and fairness with TCP are some of the required properties. In this paper, we present a congestion avoidance scheme for bulk data distribution called AMCA (Active-based Multicast Congestion Avoidance algorithm) that tries to meet these properties. We use the active networking technology to perform on a per-section dialogue to probe for available bandwidth along a multicast tree. The solution uses the RTT variations experienced by every branch to estimate the congestion situation in the multicast tree. The physical multicast tree is also used to appropriately aggregate the RTT variations at intermediate nodes before they reach the source. Simulations show that AMCA converges, makes use of the available bandwidth and reacts rapidly to dynamic changes while being TCP-fair.

1. Introduction

A first congestion control algorithm known as *TCP Tahoe* has been proposed by Jacobson [9] after a series of congestion collapses experienced in 1986. Since then, several variants have been implemented. TCP's congestion control is an efficient mechanism for unicast communications in the Internet. However, when a sender has to send the same information to more than one receiver, unicast becomes inefficient and the use of multicast is a better solution. In this case, congestion control is much more difficult than unicast because of the presence of multiple receivers with different end-to-end paths that could share some branches in the multicast tree.

One early solution requires every receiver to send its own feedback to the source. In order to be scalable, feedback suppression and/or aggregation must be performed. However, feedback suppression could yield to a lack of information at the sender making it very unresponsive to congestion variations. Moreover, inadequate suppression/aggregation of feedbacks is likely to cause the *drop-to-zero* problem [17] where the sender's estimate of the loss rate is much greater than the actual loss rate experienced at every single receiver. A main issue of a congestion control algorithm is how, where and when such feedbacks have to be suppressed and/or aggregated. Motivated by the stochastic nature of loss events especially in a multicast Internet group, RLA [19] uses a probabilistic approach to suppress feedbacks. The sender reacts to a congestion indication with probability $1/n$ where n is the estimated number of the most congested receivers. In this way, the sender responds according to the average instead of the maximum loss rate experienced by the receivers. In LTRC [14], receivers estimate their average loss rates and feed it back to the source piggybacked on NACKs. The sender tries to react only to the most congested paths by using a threshold-based mechanism. The source would apply a high rate change only for persistent losses. Moreover, it reacts only to one loss indication in a time period estimated as the time required by a rate change to take effect in the system. One issue related to this approach is how to make the tradeoff between the "drop-to-zero" problem and the responsiveness of the system. LTRC performs no feedback suppression before reaching the source therefore it still suffer from feedback implosion problem. Other approaches deal with this problem by adopting a feedback suppression combined or not to an aggregation mechanism. One of these approaches consists in choosing one or more receivers to act as representatives of the whole group. These representatives would send feedbacks to the source instead of all the receivers do. In [5], a small set of receivers is dynamically chosen as representatives by the sender based on their feedbacks. Representatives provide immediate feedback to the source allowing it to adjust the current state of the session. For instance, the source in PGMcc [18] selects one receiver to act as a representative designated as the "*acker*" which is the receiver with the lowest capacity. Afterwards, a closed TCP-like control loop is run between the sender and this worst receiver. One concern of a representative approach is to choose the set of the representatives appropriately so they reflect the congestion situations. How to select the right size

and the suitable set of representatives needs further investigations. Other schemes deal with the congestion feedback implosion problem by adopting a hierarchical structure. For instance, TRAM [4] and MTCP [16] are based on a logical tree where feedbacks are propagated and aggregated at the intermediate nodes through the tree structure.

Fairness with other flows and especially with TCP is required since most of the Internet traffic is transported by TCP. To be TCP-friendly, congestion control algorithms emulate the behavior of TCP by using a similar window-based approach (MTCP [16], RLA [19]) or by using the TCP formula [12, 15]. In a formula-based approach (TFMCC [20]), information about the packet loss rate and RTTs are collected, then the TCP formula is used to compute the corresponding rate. The use of the TCP equation is promising to be TCP fair, provided that the receivers RTTs and loss rates are known. However measuring accurately the RTTs and especially the loss rate is a real challenge.

Recently active networks where routers are able to perform customized services on the packets flowing through them, have been exploited to propose more responsive congestion control algorithms. For unicast, ACC (Active Congestion Control) [6] uses active routers to be more responsive to congestion in the network by installing adequate filters in the routers. For multicast, congestion protocols can benefit from special features such as feedback aggregation [10], hierarchical RTT estimation [8], cache of data packets [1] or the management of multiple groups in case of layered multicast [21]. In this paper, we propose a congestion avoidance scheme for bulk data distribution called AMCA standing for Active Multicast Congestion Avoidance. In this study, AMCA has been integrated into the DyRAM framework [11], but can be applied to any other active reliable multicast protocol. AMCA is a congestion avoidance mechanism rather than a control mechanism because the sender reacts before a loss occurs. AMCA uses the RTT variation as an estimation of the number of queued packets as a congestion measure in a similar way TCP Vegas does. However it appeared that TCP Vegas suffers from path changes which could give erroneous estimation of the RTTs. To avoid the problem of rerouting paths, we adopt a hop-by-hop RTT estimation and bandwidth probing. Using the active networking technology for practical implementation, a per-section dialogue is performed between adjacent nodes including the routers. In this context we consider as a section in a multicast tree, the set of point-to-point links and traditional routers that connect two active routers or an active router to a terminal node (a receiver or the source). In AMCA, a RTT variation is measured for every section. The different RTT variations are aggregated appropriately so the source ends up by receiving the overall RTT variation experienced by the worst end-to-end path of the multicast tree.

Like RMANP [1] and NCA [10], we benefit from the physical multicast tree to perform congestion feedback aggregation and thus overcome the complexity inherent to building and maintaining a logical tree structure [3, 16]. The main difference between AMCA (rate-based) and NCA (window-based) resides in the usage of different congestion measures. Whereas AMCA uses mainly the RTT variation, NCA uses both the loss rate (p) and the RTT to determine the worst receiver. However the loss rate is very difficult to be estimated on a short period while a sufficiently long period would make the congestion control unresponsive. Moreover, when there are no losses, all the receivers, according to their adopted measure, are considered to have the same capacity. Receivers are distinguished among them only and only if at least one receiver experiences a loss. This is also the case of RMANP which detects congestion on packet losses. In our case, the RTT variation measure in AMCA is able to predict congestion situations even in the absence of losses. Unlike RMANP, AMCA do not perform any cache of data packets in the active routers. We believe that an active router has to support multiple multicast sessions and caching means are not always available. The remainder of this paper is organized as follows. An overview of DyRAM is presented in section 2. In section 3, our congestion avoidance algorithm AMCA is presented. Simulation results are given in section 4 and section 5 concludes.

2. DyRAM : an Overview

AMCA is targeted to be used in DyRAM, an active reliable multicast protocol. Nevertheless, the proposed algorithm can be easily used with any other active reliable multicast protocol in addition to unicast. This section gives a brief overview of DyRAM before describing the proposed congestion avoidance mechanism. For more details about DyRAM, the reader can refer to [11]. DyRAM is a reliable multicast protocol with a recovery strategy based on a tree structure constructed on a per-packet basis with the assistance of routers. It uses a receiver-based local recovery where receivers are responsible for both the loss detection (by sending NACKs upon the detection of losses) and the retransmission of repair packets when it is possible. In order to perform flow and congestion control as well as memory management in addition to a more efficient replier election, ACKs are piggybacked on the NACKs. In the absence of NACKs, ACKs are also periodically piggybacked on special messages called “Congestion Reports” (CRs). DyRAM is an active reliable protocol where routers are able to perform loss detection, NACK suppression, subcast of repairs and the election of a receiver (replier) to perform data packet retransmission.

A DyRAM packet contains in its header the multicast address, the source address and the active service identifier

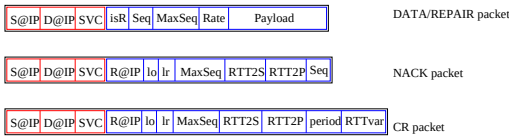


Figure 1. Packets structure.

(SVC) to be performed on this packet (see figure 1). A data packet is uniquely labelled by a sequence number and contains a dedicated field (isR) to distinguish an original transmission from a repair. A data packet contains also the current emission rate which is mainly useful to be known by the receivers in order to update timeouts correctly. A NACK or a CR packet include in addition to their originator address, their last ordered (lo) and last received (lr) data packet. Moreover, a receiver will report its RTT to the source and to its active router using the *RTT2S* and *RTT2P* fields contained in NACK and CR packets. Whereas a NACK contains the sequence number of the requested data packet, a CR contains two other fields which are the CR current period and an aggregated value of the RTT variation experienced by the subtree from which the CR is received. For the purpose of flow control, every receiver includes in the $Maxseq$ field of a CR/NACK (figure 1), the maximum data packet sequence number that corresponds to its available buffering means. The $MaxSeq$ field in a CR or NACK packet is used by the source to control the transmission flow.

3. AMCA Description

A good characteristic of a congestion control algorithm is its ability to probe extra bandwidth in the network. We show in the following, how by using periodic feedbacks, we can estimate the available bandwidth in a simple connection. A generalization to the case of a multicast tree will be provided.

3.1. Probing Extra Bandwidth of a Connection

In this section, we consider the case of a connection with one queue between a sender and one receiver (figure 2). Let r_b be the bottleneck rate of the connection and r the transmission rate of the sender in bits/s. An intuitive observation is that while the transmission rate is less than the bottleneck rate, the reception rate is equal to the emission rate. The reception rate could not exceed the connection bottleneck rate r_b even if the emission rate is increased. When the transmission rate exceeds the bottleneck rate, a queue of packets will build up through the connection. Let $q(t)$ be this queue size in packets at time t. Let S be the packet size and Δq be the positive or negative variation in the queue length during a given time period T (figure 2):

$$\Delta q = q(t+1) - q(t) = (r - r_b)\,\frac{T}{S} \tag{1}$$

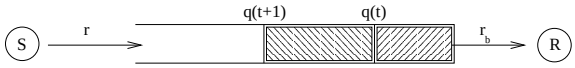


Figure 2. Queue length variation.

To estimate Δq we can employ a mechanism based on round trip time variations since a growing up (or the opposite case) queue produces an increased (or a decreased) RTT. The RTT variation $\Delta\tau$ can be expressed in terms of Δq as follows:

$$\Delta\tau = \Delta q\,\frac{S}{r_b} \tag{2}$$

which gives:

$$r_b = \Delta q\,\frac{S}{\Delta\tau}$$

Hence,

$$r_b = r\,\frac{T}{T+\Delta\tau} \tag{3}$$

This gives an expression of the bottleneck rate and is used by our congestion avoidance algorithm to adjust the rate so the transmission rate will be the closest to the bottleneck rate. From (1) and (2) we derive the next equation which gives an expression of Δq as a function of r, T, S and $\Delta\tau$:

$$\Delta q = \frac{r\,T}{S}\,\frac{\Delta\tau}{\Delta\tau+T} \tag{4}$$

We can easily obtain accurate values for r, T and S. However for $\Delta\tau$ we have to provide an estimation mechanism. To do so in such a simple connection, the source could have an estimation of the RTT variation simply by exploiting feedbacks information. For instance if we suppose that the receiver sends an ACK per packet received then the source can use the difference between the two last experienced RTTs. The T period in this case is S/r_b.

The problem when using such a method to probe available bandwidth appears when we have a session with multiple links. The path between the sender and the receiver can change and this would give erroneous results. [13] pointed out that TCP Vegas suffers from similar situations since rerouting a path may change the propagation delay of the connection which could result in a substantial increase/decrease in throughput. To overcome this problem, we propose to use a per-section bandwidth probing approach. A section is defined as the set of point-to-point links and traditional routers that connect two active routers or an active router to a terminal node (a receiver or the source). The main concern is to be able to estimate the RTT variations in a per-section fashion and then how these RTT variations are aggregated before giving them to the source. To do so, we have to establish communications between every node and its neighbor in a connection.

3.2. RTT Estimation

A straightforward solution to estimate the RTTs between each receiver and the source consists in sending ping messages periodically from the receivers to the source. This

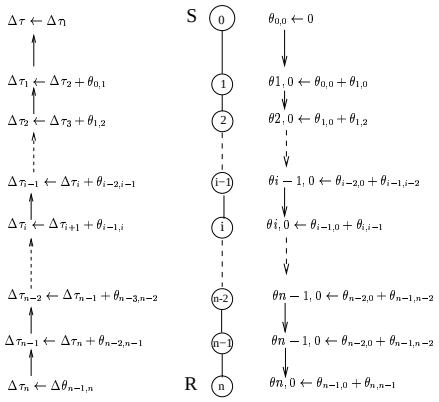


Figure 3. RTT estimation and CRs aggregation

naive solution would however overload the source in the presence of a large number of receivers. Instead of estimating directly the RTTs between the source and each receiver, we begin by estimating the RTTs between every node and its parent in the multicast tree. This mechanism meets also the per-section RTT measurements required by our hop-by-hop bandwidth probing procedure (section 3.1). To illustrate our method we show in figure 3, one linear path of the multicast tree that connects the sender (labelled by 0) to one of the receivers with label n. Intermediate nodes labelled from 1 to $n-1$ are the routers. The generalization of our method to the other receivers and intermediate nodes is straightforward.

Special messages called heartbeat messages (HB and HB_RESP) are periodically exchanged between each node i and its parent $(i-1)$ in the same way the ping/pong messages do. The first HB message is sent by the receiver. The reception of a HB by a router triggers the emission of a HB message to its parent in addition to responding with a HB_RESP message. Let $\theta_{i,j}$ be the computed RTT between the ith and the jth node (we have $\forall i,j : \theta_{i,j} = \theta_{j,i}$). Consequently, $\theta_{i-1,i}$ is the RTT between node i and its parent, and $\theta_{0,i}$ is the RTT between node i and the source.

A node i, in order to compute its RTT to the source, first computes its RTT to its parent node by sending a HB message timestamped with the emission time. The parent node, on the reception of the HB message, responds with a HB_RESP that contains its RTT to the source that has already been computed. Node i, on receipt of the HB_RESP message is then able to compute its RTT to the source as the sum of the RTT of its parent to the source and its own RTT to its parent as follows:

$$\forall i > 0 : \theta_{0,i} = \theta_{0,i-1} + \theta_{i-1,i} \quad (5)$$

Initially (right side of figure 3), the source responds with a special message HB_RESP_S which contains its own RTT to itself (of course we have $\theta_{0,0} = 0$) in the data packets it sends. Node number 1 will update $\theta_{0,1}$ using (5) and forwards it downstream with the newly computed RTT. Node number 2 will in turn update $\theta_{0,2}$ using the same equation and forwards it downstream and so on (figure 3). Afterwards the use of the HB and HB_RESP messages will be sufficient to estimate the different RTTs.

3.3. Congestion Feedback

In AMCA, every receiver sends a CR to the source every N packets received so the source can learn about the congestion situation in the multicast tree. We chose to send a CR on the reception of a given number of packets rather than sending a CR every a constant period. The congestion situation is then probed proportionally to the emission rate. The resulting period is $T = NS/r$ which represents the expected time to receive N data packets by a receiver where r is the emission rate of the source. In the case where no data packet is received, a NACK could be sent instead of a CR to indicate that a packet is missed by a receiver. With $T = NS/r$, from (4) the queue size variation can be given by:

$$\Delta q = N \frac{\Delta\tau}{\Delta\tau + T} \quad (6)$$

We introduce Δq_p defined as the queue size variation per packet:

$$\Delta q_p = \frac{\Delta q}{N} = \frac{\Delta\tau}{\Delta\tau + T} \quad (7)$$

The periodicity of sending the CRs varies because it depends on the current reception rate. To make the source aware of the current period, a receiver reports it in the $period$ field of a CR packet (figure 1).

In what follows, we note by $\Delta\theta_{i,j}$ the experienced RTT variation between the ith and the jth node. For the ith link that connects the $(i-1)$th and the ith node, the RTT variation is $\Delta\theta_{i-1,i}$. Additionally we note by $\Delta\tau_i$ the RTT variation of the $(i-1)$th intermediate node and the receiver. The overall experienced RTT variation between the source and the receiver is $\Delta\tau = \Delta\tau_1$. Initially, a receiver (left side of figure 3) computes the RTT variation $\Delta\theta_{n-1,n}$ of its upstream link (number n) using the last two RTT measures to its parent with the mechanism described in section 3.2. Afterwards the receiver includes the computed RTT variation in the CR to be sent to the source. The receiver's parent (i.e. node $n-1$) adds the RTT variation reported by the receiver and its own RTT variation to its upstream node to obtain the RTT variation of its parent ($(n-2)$th node) to the receiver:

$$\Delta\tau_{n-1} = \Delta\tau_n + \Delta\theta_{n-2,n-1}$$

Node $(n-1)$ will then report $\Delta\tau_{n-1}$ in the $RTTvar$ field of the CR. In this way, a parent node i, on receipt of a CR from node $(i+1)$ adds its RTT variation to node $(i-1)$ to the received one before forwarding the CR with the newly computed RTT variation:

$$\Delta\tau_i = \Delta\tau_{i+1} + \Delta\theta_{i-1,i}$$

The sender will end up by receiving the RTT variation $\Delta\tau$ experienced on the whole linear connection. This latter can

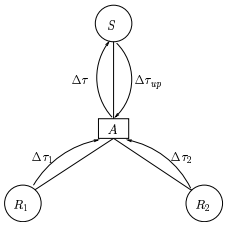


Figure 4. CRs aggregation in a multicast tree

be used in equation (3) or (7) to have respectively a lower bound for the bottleneck rate r_b or the estimated queue length variation per packet Δq_p during the previous period.

3.4. Feedback Suppression/Aggregation

To achieve scalability, a mechanism of aggregation/suppression is performed. We use the physical multicast tree to hierarchically aggregating feedbacks at the intermediate nodes (routers). A suppression mechanism is performed on NACKs thus allowing just one NACK per loss to be sent upstream. Since we do not suppose that all the routers are active, more than one NACK for the same loss could be received by the source. To avoid reacting to duplicate NACKs, the source reacts to the first one and ignores the subsequent ones for a given period of time which depends on the multicast tree structure.

The CRs are aggregated at the intermediate nodes. An active router aggregates the received CRs from the downstream links in one CR to be forwarded upstream. The aggregated CR contains the sequence number of the minimum last ordered and the maximum last received data packets among those reported by the CRs received from downstream. For the aggregation of the RTT variations reported by the CRs from downstream in the case of a multi-path connection, we illustrate our solution with the simple two-level tree depicted in figure 4. We consider two receivers R_1 and R_2 connected to the source S via one active router A. Receivers R_1 and R_2 send CRs to the active router with their respective RTT variations $\Delta\tau_1$ and $\Delta\tau_2$. Once these children CRs are received, the active router A sends its CR to the source with $\Delta\tau$ computed using:

$$\Delta\tau = \Delta\tau_{up} + Max(\Delta\tau_1, \Delta\tau_2)$$

where $\Delta\tau_{up}$ is the RTT variation of the source link (A, S). This latter is extracted from the HB_RESP message sent by the source. Using this method in aggregating the RTT variation, the source ends up by receiving the RTT variation experienced by the worst end-to-end path of the multicast tree.

3.5. Rate Regulation

In AMCA, the regulation parameter is the rate. A minimum and a maximum rates r_{min} and r_{max} are set by the application. Any receiver that could not support the minimum transmission rate has to leave the multicast session. Initially, the source starts to send data packets with a rate equal to the minimum rate r_{min}. Then it tries to increase its rate if no congestion indication is received without exceeding the maximum rate r_{max}. The source uses the information fed back in the CRs and NACKs to update its rate. The RTT variation field of every CR is used by the source to compute the queue size variation per packet Δq_p during the previous period. The goal is to maintain Δq_p as small as possible. During a phase similar to the slow start phase of TCP, the source would increase its rate by S/RTT_{max} (bits per second) every time it receives a CR that gives a queue size variation $\Delta q_p < \epsilon$, where ϵ is a positive number to be chosen in $[0, 1]$. Increasing the rate by S/RTT_{max} where RTT_{max} is the maximum experienced RTT among the receivers, is equivalent to adding 1 to the congestion window for the largest end-to-end connection of the multicast tree. This behavior makes our congestion avoidance algorithm fair with TCP from the beginning of the multicast session.

However, the network could be already congested from the beginning of the session. To avoid aggravating the situation, we need an other congestion indicator to be sure that there is no congestion in the network. This can be done by examining the successive values of the last ordered field of the CRs. This gives a measure of the difference between the emission and the reception rate. This difference is proportional to the number of data packets which are not acked yet which can be given by:

$$\Delta q_a = \max(0, n_s - n_a) \tag{8}$$

where n_s and n_a are respectively the number of sent and acked packets during the previous period. Since a CR is sent to the source every N packets received during a period T, we have:

$$n_s = \frac{T}{S}\, r = N$$

The number of acked packets is computed using:

$$n_a = lo_{i+1} - lo_i$$

where lo_{i+1} and lo_i are respectively the values of the last ordered field of the newly received and previously received CR. Finally:

$$\Delta q_a = \max(0, N - (lo_{i+1} - lo_i)) \tag{9}$$

The source, on the reception of a CR, extracts the current period (T) and the RTT variation ($\Delta\tau$). Afterwards, in addition to the queue size variation per packet Δq_p, the number of data packets not acked yet Δq_a is also computed using (9). The aim is to maintain Δq_a in $[\alpha, \beta]$, where α and β are similar to the two parameters of TCP-Vegas. In the slow start phase, the source continues increasing its rate every time it receives a CR that indicates $\Delta q_p < \epsilon$ and $\Delta q_a < \beta$. The source enters the congestion avoidance phase when it receives a NACK or a CR with $\Delta q_p > \epsilon$ or $\Delta q_a > \beta$. On the receipt of a NACK, the source reduces the rate by

half. Subsequent NACKs are ignored during a period estimated by the difference between the current RTT to the source of the farthest and the closest receiver in the multicast tree ($RTT_{max} - RTT_{min}$). To avoid decreasing dramatically the rate because of isolated losses, we check the lo and lr fields of the NACKs and reduce the rate only if $lr - lo \geq 3$; otherwise we just retransmit the corresponding repair.

During the congestion avoidance phase, on the reception of a CR with a RTT variation $\Delta\tau$ and a period T, the source updates its rate depending on the current values of both Δq_p and Δq_a. When these two measures do not indicate congestion ($\Delta q_p \in [0, \epsilon]$ and $\Delta q_a < \beta$), the rate is multiplied by γ_{inc} chosen slightly greater than 1. In the other cases, the rate is multiplied by $\gamma = T/(T+\Delta\tau)$ which is derived from (3) and allows the rate to converge to the bottleneck rate. When the emission rate is greater than the bottleneck rate, a queue will build up in the bottleneck link. This will be translated by a positive RTT variation $\Delta\tau$ and thus $\gamma < 1$. Multiplying by γ will decrease the transmission rate and this is the appropriate action from the source. In the reverse situation where the emission rate is less than the available bandwidth, $\Delta\tau$ is negative and multiplying by $\gamma > 1$ will increase the transmission rate. In addition to the proposed mechanisms, a severe congestion is detected if no CR, HB or NACK is received during a relatively long period which is set to twice the current period ($2T$). In this case the rate is dropped to its minimum value r_{min}.

Since DyRAM implements a local recovery mechanism, we have to be careful about the local recovery problem. Many reliable protocols perform local recovery which leads to the coexistence of multiple data sources. This is why not only the original source has to be congestion controlled, but retransmitting entities (receivers, routers or servers) should also be controlled. Additionally, a feedback suppression is implicitly or explicitly performed which would make the source unresponsive to congestion variations. Suppose that behind a congested link, one or more receivers have sent NACKs that will be received by a replier instead of the source. The replier will retransmit the missed packets and the source is never aware of the congestion and would continue to send with the same rate (even with a higher rate). As a result the congestion situation is aggravated. This is the so called *local recovery problem* [14]. In our scheme, we propose to make the source aware about any NACK that has been forwarded downstream to an elected replier. An active router when aggregating the CRs, includes NACK information mainly in the lo and lr fields of the aggregated CRs. In this way, the source is always aware about losses that occur in the multicast tree even in the presence of local recoveries. The retransmission rate from the repliers is also controlled to do not aggravate a congestion situation until the source adapts its rate. When an elected replier has to retransmit more than one packet, then it sends repairs with a rate equal to the current rate of the source divided by $(lr - lo)S$ which is proportional to the congestion degree. Finally, it is worth mentioning that the DyRAM loss detection service at the routers could be very helpful for the congestion control. A NACK is sent to the source as soon as a loss is detected and the source would reduce its rate rapidly which could enhance the responsiveness of the proposed congestion avoidance algorithm.

4. Simulation Results

A simulation model of AMCA with DyRAM has been implemented using ns-2.1b8 (network simulator [7]). For all the simulations the data packet size is set to 1024 bytes and the NACK/CR packet size is set to 32 bytes. The minimum and maximum rates are set respectively to 1 and 10000 packets per second. While ϵ is set to 0.2, α and β are respectively set to 1 and 3. For the rate regulation, γ_{inc} is set to 1.05. A receiver sends a CR every $N = 32$ data packets received unless said otherwise. For all the simulations, we used the simple topology of figure 5 with four receivers separated by two routers from the source. The bottleneck link L has a bandwidth of $0.9Mbps$. S is the DyRAM source that multicasts data to the receivers $R1$ and $R2$. $S1$ and $S2$ are used as sources for TCP (Vegas) flows to study the fairness of DyRAM with TCP. $R3$ and $R4$ are the TCP sinks of $S1$ and $S2$ respectively.

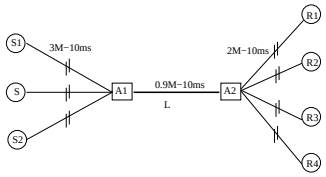


Figure 5. The used topology

4.1. Convergence of the algorithm

A first set of experiments is conducted with just one multicast flow. The source $S1$ multicasts data using DyRAM to the receivers of figure 5. The purpose of this first set of experiments is to show how our congestion algorithm probes and makes use of the available bandwidth. Figure 6 shows the achieved throughput by the receivers for different values of N. We can see that the algorithm converges to the bottleneck available bandwidth of $900Kbps$ and then makes use of the available bandwidth during all the simulation. It is worth showing that there is a tradeoff between the responsiveness (quick reaction) and the stability (in terms of oscillations) depending on the CRs periodicity. We can see that the smaller N is, the more responsive the algorithm is. However the oscillations are smoother when we increase N.

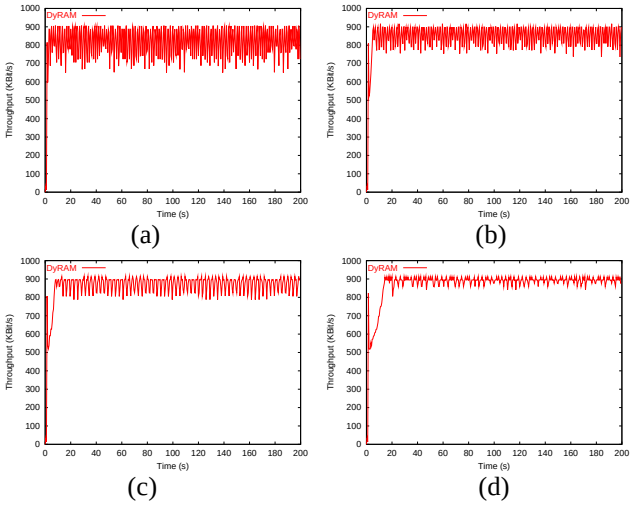


Figure 6. Responsiveness/stability tradeoff, N set to (a) 16, (b) 32, (c) 64, (d) 128

4.2. Fairness with TCP

Since most of the Internet traffic is transported by TCP, the fairness to TCP flows is an important issue. To evaluate the fairness of our scheme, experiments have been conducted on the topology depicted in figure 5. One experiment is performed with one DyRAM flow competing with two TCP flows. The DyRAM source is S and the 2 TCP flows are sent by $S1$ and $S2$ to the receivers $R3$ and $R4$ respectively. The simulation is run for 200 seconds. All of the three flows are started at the same time. Figure 7a shows the achieved throughput by the different flows. We can see that the bottleneck link of $0.9M$ is almost equally shared among the three flows.

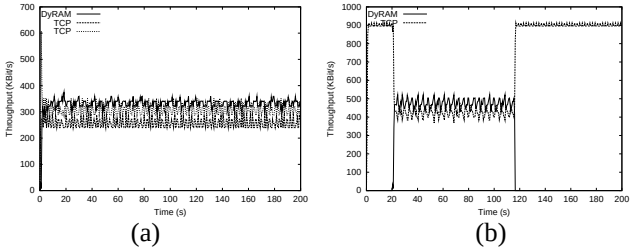


Figure 7. Fairness with TCP, (a) two TCP flows, (b) one TCP flow.

To show the dynamic behavior of our congestion algorithm, we have run a simulation with one DyRAM multicast flow in the background and one TCP flow started at 20s and stopped at 100s. Figure 8a shows the achieved throughput by DyRAM and TCP. We can see that DyRAM makes use of all the available bandwidth while the TCP flow is not started yet. In the presence of the two flows, the available bandwidth is shared equally among them. When the TCP flow is stopped, DyRAM is able again to make use of the available bandwidth. We note that the time required by DyRAM to react to both the existence and the absence of the TCP flow is very small. Figure 8b shows the sequence numbers of data packets received by the DyRAM receivers. We can observe that DyRAM dynamically adjusts its rate and the slope angle of the sequence number line is divided by 2 in the presence of the TCP flow. An other interesting observation is that there is no packet loss during all the simulation.

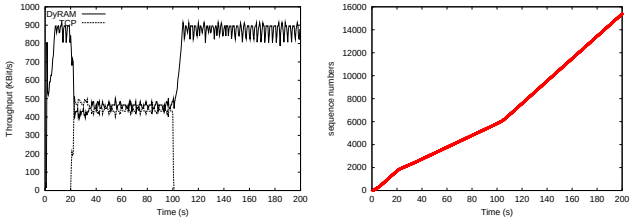


Figure 8. Dynamic behavior of DyRAM with one TCP flow

Figure 9 shows the results of a simulation run with two TCP flows and one DyRAM multicast flow. DyRAM is run from 0 to 200s and the two TCP flows are run respectively from 30 to 100s and 70 to 150s. We can note the same observations as for the previous experiment. DyRAM equally shares the available bandwidth with two TCP flows and always react quickly to the network dynamics. DyRAM is always fair and never aggressive with TCP.

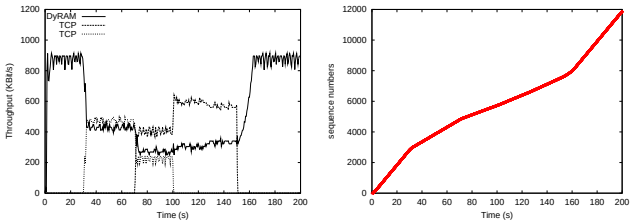


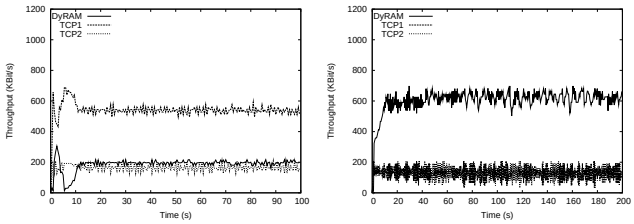
Figure 9. Dynamic behavior of DyRAM with two TCP flows

An other simulation has been performed with TCP in the background and the DyRAM flow run from 20 to approximately 118s (the time required to send 5000 data packets). Figure 7b shows the achieved throughput by the two flows. It is clear that once more DyRAM is proved to be TCP-friendly and shares equally resources with any TCP flow.

4.3. Multiple bottleneck links

To see the behavior of our congestion avoidance algorithm with TCP flows in the presence of multiple bottleneck links, we have performed simulations with the same topology of figure 5 with some lower capacity links. The simulations are performed with 2 TCP flows and one DyRAM

flow. The TCP flows are from S1 and S2 to R3 and R4 respectively. S is the DyRAM source that multicasts to the receivers R1 and R2. We set the link bandwidth of receivers R2 (one of the DyRAM flow receivers) and R4 to $0.2Mbps$ instead of $2Mbps$. Figure 10a shows that DyRAM achieves a throughput that corresponds to the minimum available on the whole multicast tree ($0.2Mbps$). The TCP flow to R4 uses the remaining bandwidth of link L. Figure 10b shows the achieved throughput by the different flows where the bandwidth of the TCP receivers (R3 and R4) links is set to $0.2Mbps$. We can see that the 2 TCP flows use the available $0.2Mbps$ bandwidth for each of them and that DyRAM makes use of the remaining bandwidth on the bottleneck link.



(a) (b)

Figure 10. Multiple bottleneck links

5. Conclusion

In this paper, we have presented AMCA an active-based congestion avoidance algorithm for multicast communications. The proposed scheme is validated with an active reliable multicast protocol (DyRAM). Nevertheless AMCA can be used by any other active reliable multicast protocol and particularly in the case of unicast as an enhancement of TCP Vegas. Our congestion algorithm relies on the active networking technology to perform a per-section dialogue to probe for available bandwidth along a multicast tree. This is done by using the RTT variations experienced by every section in the multicast tree. The RTT variations are then appropriately aggregated by intermediate nodes before they are given to the source.

A set of simulations have been conducted using ns-2.1b8 network simulator to study the behavior of our congestion avoidance algorithm. It has been shown that AMCA converges and makes use of the available bandwidth and reacts rapidly to the dynamic changes in by the multicast tree. In addition, simulations have shown that there are few packet losses because AMCA reacts before a loss occurs. Moreover our algorithm is proved to be fair with TCP.

AMCA has to be experimented on larger topologies that reflect the actual Internet. Moreover the congestion parameters that we used have to be studied in more detail to determine their optimal values. AMCA is a single-rate approach and the emission rate is adapted according to the worst receiver in the multicast group. Currently an adaptation of AMCA to support heterogeneous receivers is under study and results will be available soon.

References

[1] A. Azcorra et al. Multicast congestion control for active network services. *European Transactions in Telecommunications*, 10(3), May/June 1999.

[2] L. Brakmo and L. Peterson. End to end congestion avoidance on a global internet. *IEEE JSAC*, 13(8), October 1995.

[3] D. M. Chiu et al. TRAM: A tree-based reliable multicast protocol. Technical Report TR-98-66, SUN, July 1998.

[4] D. M. Chiu et al. A congestion control algorithm for tree-based reliable multicast protocols. Technical Report TR-2001-97, Sun Microsystems Labs, 2001.

[5] D. DeLucia and K. Obraczka. Multicast feedback suppression using representatives. In *INFOCOM*, 1997.

[6] T. Faber. Experience with active congestion control. In *DARPA ANCE, San Francisco, CA*, May 29-30 2002.

[7] K. Fall et al. Ns notes and documentation ucb/lbnl/vint. http://www.isi.edi/nsnam/ns, July 1999.

[8] S. Golestani and K. Sabnani. Fundamental observations on multicast congestion control in the internet. In *INFOCOM'99*.

[9] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, 1988.

[10] S. Kasera and S. Bhattacharya. Scalabe fair reliable multicast using active services. *IEEE Network Magazine's Special Issue on Multicast*, 2000.

[11] M. Maimour and C. Pham. Dynamic Replier Active Reliable Multicast (DyRAM). In *ISCC'02*, July 1-4 2002, Taormina, Italy. A more recent version is available as technical report RR-4635, INRIA, 2002.

[12] M. Mathis et al. The macroscopic behavior of the tcp congestion avoidance algorithm. *Computer Communication Review*, 27(3):62–82, 199.

[13] J. Mo et al. Analysis and comparison of TCP reno and vegas. In *INFOCOM*, pages 1556–1563, 1999.

[14] T. Montgomery. A loss tolerant rate controller for reliable multicast. Technical Report NASA-IVV-97-011, NASA/WVU, August 1997.

[15] J. Padhye et al. Modeling tcp throughput: a simple model and its empirical validation. In *SIGCOMM*, 1998.

[16] I. Rhee et al. MTCP : Scalable tcp-like congestion control for reliable multicast. Technical Report TR-98-01, 1998.

[17] L. Rizzo. pgmcc: a tcp-friendly single-rate multicast. In *SIGCOMM*, 2000.

[18] T. Speakman et al. PGM reliable transport protocol specifi - cation. internet draft, 1998.

[19] H. Wang and M. Schwartz. Achieving bounded fairness for multicast and TCP traffi c in the internet. In *SIGCOMM'98*.

[20] J. Widmer and M. Handley. Extending equation-based congestion control to multicast applications. In *ACM SIGCOMM (San Diego,CA)*, August 2001.

[21] L. Yamamoto and G. Leduc. An active layered multicast adaptation protocol. In *IWAN*, pages 179–194, 2000.