Congestion Control in HPR

Jim Martin
Networking Software
IBM Corporation
Box 12195, RTP NC 27709-12195
jjm@vnet.ibm.com

Arne Nilsson
Center for Advanced Computing and Communication
North Carolina State University
Box 7914, Raleigh NC 27695-7914
nilsson@ncsu.edu

ABSTRACT: High Performance Routing (HPR) is an enhancement to IBM's Advanced Peer to Peer Networking (APPN[1]) protocol that improves its performance and reliability. In an HPR network, the Rapid Transport Protocol (RTP) is used for end-to-end data transfer.[2] RTP provides a reliable, in-order delivery service using a selective repeat protocol as the end-to-end error recovery mechanism. Additionally, RTP provides an end-to-end, Adaptive Rate-Based (ARB) flow/congestion control mechanism that is designed to avoid congestion and maximize network stability. This paper describes the ARB algorithm. Simulation results are presented illustrating the behavior and the stability of the algorithm.

1 Introduction

In a packet switched network, multiple sessions compete for network resources. Network resources include bandwidth (i.e., the transmission capacity of a link in the network), a router's switching capacity, and a router's buffer capacity. If demand for a resource exceeds the capacity, congestion occurs. Symptoms of network congestion include increased packet delays and packet loss (from buffer overflow). Uncontrolled congestion leads to "congestion collapse" where throughput is reduced to a small fraction of normal [1].

Congestion control schemes can be categorized as either open-loop or closed-loop. Closed-loop schemes are predominant in packet switched networks such as SNA and TCP/IP. Closed-loop schemes are either reactive or preventive. A reactive scheme, such as TCP, reacts to congestion based on evidence of congestion such as a retransmit time-out [2]. A preventive scheme attempts to detect early signs of congestion allowing the source to reduce its offered load before significant congestion occurs.

ARB, which is a closed-loop, preventive control scheme is designed to keep the network from approaching the "cliff" as illustrated by region 1 and 2 of Figure 1 [3]. TCP, on the other hand, typically oscillates over ranges 1 through 4.

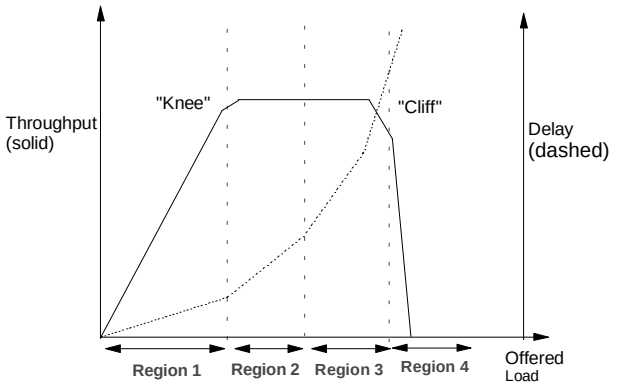

Figure 1: Throughput and delay versus offered load

In this paper we present the ARB algorithm. We provide simulation results illustrating the behavior and stability of the algorithm. To provide some measure of the benefits of ARB, we include a brief comparison between ARB and TCP's congestion control scheme(s).

2 Adaptive Rate Base Congestion Control

ARB is a closed-loop, preventive, rate-based congestion control scheme. ARB employs a distributed algorithm that is implemented at the endpoints of an RTP connection. Each endpoint consists of an ARB sender and an ARB receiver. The ARB sender periodically queries the receiver by sending a rate request to the ARB receiver who responds with a rate reply message. The time between succesive transmissions of rate request packets is defined as a *measurement_interval*. The *measurement_interval* is typically on the order of a round trip time.

The ARB receiver monitors changes in the delay experienced by sequential rate request packets (we will explain the mechanism in detail shortly). ARB assumes that trends in the delay change values (i.e., a running total of delay change

[1] APPN is a registered trademark of IBM.
[2]The HPR specification is available at ftp: //networking.raleigh.ibm.com/aiw/appn/hpr.

samples) are reflective of the level of congestion in the one-way path between the sender and the receiver. Based on this estimate of congestion, the receiver instructs the sender how to react in the rate reply message. The sender adjusts its send rate based on information received in the rate reply message.

The following defines the parameters and terms that will be used to describe the ARB algorithm.

- *burst_size* : The number of bits the sender is allowed to send in one burst. The default *burst_size* is 65536 bits and typically remains fixed for the lifetime of the connection.[3]

- *burst_time* : Effectively specifies the frequency of bursts (where each burst is of size *burst_size*). For example, if the *send_rate* is 1.5Mbps and the *burst_size* is 65536 bits, then every 43 msecs (calculated by *burst_size/send_rate)* the sender can transmit up to 65536 bits.

- *data_queued*: An estimate of the amount of data queued in the path between the sender and the receiver (in bytes) that is based on the *delay_change_sum* and the *max_rate* value.

- *delay_change:* When a receiver processes a rate request, it assesses the delay incurred by the rate request packet by taking the difference (in seconds) between the *receiver's_measurement_interval* and the *sender's_measurement_ interval.*

- *delay_change_sum:* A running total of all the *delay_change* values.

- *link_capacity:* The transmission speed (in bits per second) of a particular hop in an RTP connection.

- *max_rate*: The minimum *link_capacity* of all links in the path of the RTP connection. The *max_rate* is learned during the HPR route selection process and is used to calculate the *data_queued* value and to set the rate increment values.

- *rate_increment:* The size of a *send_rate* increment in bits per second. The ARB sender modifies this value as the congestion level changes in the network.

- *receiver's_measurement_interval:* The interval between subsequent rate requests received by the ARB receiver.

- *send_rate* : The rate at which the sender is allowed to transmit data into the network (in bits per second). Based on the feedback from the receiver, the ARB sender will either increase/decrease or not change the *send_rate* each *measurement_interval.*

- *sender's_measurement_interval:* The interval between subsequent rate requests transmitted by the ARB sender.

Assuming the sender always has data to send and that the *send_rate* is less than the *link_capacity* of the sender's node, the rate control mechanism leads to on/off behavior. The sender is in the on state for *burst_size/link_capacity* seconds. The length of the idle time that follows the burst is given by *burst_time - (burst_size/link_capacity).* The net effect of the rate control algorithm is to smooth bursty traffic while maintaining the desired *send_rate*.

Figure 2 illustrates RTP's rate control by showing a series of one-way transmissions between an RTP sender and receiver. The sender bursts for a duration of *T_burst* time units followed by an idle period of *T_idle* time units. Figure 2 also illustrates a series of rate request/reply exchanges. Assume that the first packet of each burst contains the rate request. The measurement intervals for the sender and the receiver are labeled *TSmi* and *TRmi* respectively. The time *t0' – t0* represents the one-way effective transmission time to send and receive a rate request packet. The time *t0''- t0* represents the total time required to send and acknowledge the rate request packet (i.e., a round trip time). In Figure 2, we show only one burst each *measurement_interval*. Depending on the *send_rate* and the round trip time, there can be multiple bursts within a *measurement_interval*.

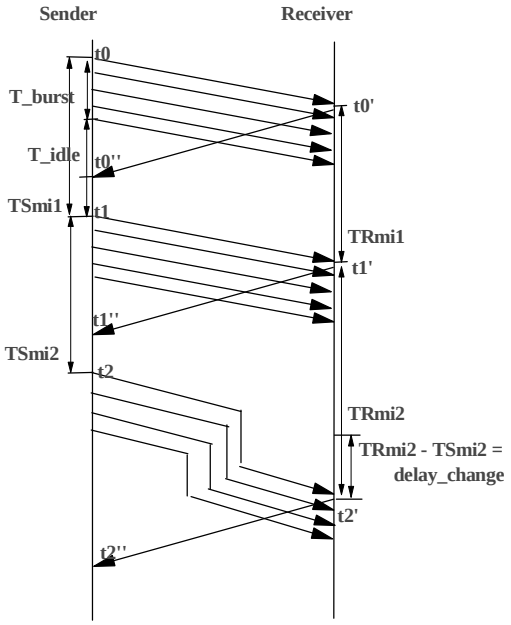

Figure 2: Rate request/reply exchanges

At the beginning of a *measurement_interval*, the sender notes the amount of time since the transmission of the previous rate request. This time increment (which we have defined as the *sender's_measurement_interval*) is inserted in the rate request packet. For example, in Figure 2 at time *t1*, the sender will insert the *TSmi1* value in the rate request packet. When the receiver receives the rate request (at time *t1'*), it calculates the *receiver's_measurement_interval* (i.e., *TRmi1 = t1' – t0').* The receiver subtracts the two measurement intervals to obtain a *delay_change* value. Again using our example, at time *t1'* the receiver calculates the *delay_change* by subtracting *TSmi1* from *TRmi1*.

[3]The *burst_size* will increase if the *burst_time* becomes less than the system timer tick granularity.

ARB assumes that a positive *delay_change* value measured by the receiver is the waiting time the rate request packet experienced in the bottleneck link queue. Continuing with the example illustrated by Figure 2, packets sent after time *t2* are delayed. When the receiver processes the rate request packet at time *t2'*, it calculates a positive *delay_change* value (i.e., *TRmi2 – TSmi2 > 0*).

The ARB receiver sums the *delay_change* values using the *delay_change_sum* variable. Once the *delay_change_sum* (i.e., once the accumulated delay) exceeds a threshold, the receiver will instruct the sender to slow down. Therefore, based on the *delay_change_sum* and also on the availability of receive buffers (i.e., for flow control), the receiver constructs and sends a rate reply packet back to the sender that contains one of the following commands:

- Normal: increment the *send_rate* by the current *rate_increment*.
- Restrain: keep the *send_rate* the same.
- Slowdown1: decrement the *send_rate* by 12.5%.
- Slowdown2: decrement the *send_rate* by 25%.
- Critical: decrement the *send_rate* by 50%.

The ARB algorithm relies on several assumptions. First, ARB assumes that it can accurately track congestion via periodic delay measurements. Clearly, this is true only if the duration of the congestion is longer than a *measurement_interval*. ARB further assumes that a positive *delay_change* implies congestion at the bottleneck link. The implied assumption is that queueing delay observed at the end points of a connection is dominated by the queueing at the bottleneck. This assumption is well-known as observed in [4].

Because ARB assumes that all observed delays occur at the bottleneck link, and because the bottleneck link rate is known (i.e., the *max_rate)*, ARB can estimate the amount of data queued at the bottleneck based on the accumulated delay and the bottlenecklink speed (i.e., *data_queued = delay_change_sum * max_rate)*. Based on this estimate, the ARB receiver provides a rate adjustment command back to the sender.

ARB essentially models a network as a single server queueing system. Using the network shown in Figure 3a as an example, assume that an RTP connection exists between nodes A and D. We are interested in the one-way flow from a sender on node A to a receiver on node D. Assume that at connection setup time, RTP is told that link 2 is the bottleneck link. Figure 3b shows the ARB model: the one-way multihop path simplifies to a single server queueing system located at node B.[4]

[4]A similar model exists for traffic flowing in the reverse direction over the RTP connection.

The ARB *data_queued* variable tracks the queue level at the bottleneck link (i.e., link 2 in Figure 3). The ARB receiver operates in one of five regions where the regions are defined by three threshold values (T1, T2 and T3) that correspond to an amount of data queued at the bottleneck link. The default ARB thresholds are 1000 bytes, 10000 bytes and 100000 bytes of queued data respectively. The following summarizes the ARB operating regions:

- Region 0 (*data_queued* <= 0): This implies there is no congestion. The receiver sends the Normal command to the sender which allows the sender to increase its rate.
- Region 1 (0 < *data_queued* <= T1): Some congestion has occurred, however ARB is not yet at the optimal operating point. Therefore, the sender is issued the Normal command.
- Region 2 (T1 < *data_queued* <= T2): This implies the network has entered the beginning of the desired operating range. The sender is instructed to Restrain.
- Region 3 (T2 < *data_queued* <= T3): This implies that the network has exceeded the desired operating region. The receiver issues a Slowdown1 command to the sender. To ensure fair bandwidth reduction across multiple RTP connections, once the sender is told to decrease its rate, the receiver resets its *delay_change_sum* to zero.
- Region 4 (*data_queued* > T3) ARB assumes that noise (a delay local to the node, perhaps due to operating system scheduling) has occurred. The sender is instructed to Restrain.

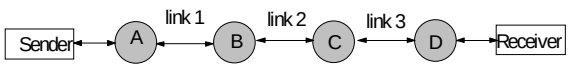

Figure 3a

Figure 3b

Figure 3: ARB network model

Regions 1 through 4 defined above correspond to the regions shown in Figure 1 (Regions 0 and 1 correspond to Region 1 of Figure 1). ARB assumes that the optimal operating point of the network is just past the knee but well before the cliff as illustrated by Region 2 of Figure 1. As the offered load increases beyond the knee, thoughput remains the same but the delay increases. Once the ARB Region 2 is entered, a small amount of data is queued in the network. However, the sender is not instructed to reduce its send rate. The rational is that if the optimal range is too small, ARB's rate increase/decrease algorithm will cause the *send_rate* to

fluctuate too rapidly leading to a less stable network. By issueing a Restrain command, the receiver hopes that the network will stabilize itself without having to reduce the sender's send rate.

When the sender receives the rate reply from the receiver (at time *t0'', t1'', t2''* in Figure 2), it adjusts its rate as instructed by the receiver. If the sender is instructed to increase its rate, it does so by a *rate_increment* amount. When an RTP connection starts, the *send_rate* is initialized to 10% of the capacity of the slowest link in the path. The *rate_increment* is modified by the ARB sender based on observed network congestion trends. The range of increment varies between .2% and .8% of the *max_rate*.

In addition to when a sender processes a rate reply, the *send_rate* is also adjusted at the following other times:

- When a gap detect message is received from the receiver. This implies a packet was lost. The sender retransmits the missing packet(s) indicated in the message and then drops its *send_rate* by 50% (the *rate_increment* is not modified).
- When a rate request time-out occurs. The sender initiates a recovery timer whenever it sends a rate request to the receiver. If this timer pops before the rate reply is received, the sender performs the following:
  - ♦ Cuts the *send_rate* by half.
  - ♦ Increases the rate request time-out value exponentially.
  - ♦ Transmits a rate request at the next send opportunity.

3 Simulation Results

To illustrate the behavior of RTP, we present results from a series of simulations. The simulated network (illustrated by Figure 4) interconnects two LAN's using a 200 Kbytes/second link with a 50ms propagation delay (roughly a cross-country T1 link). All packets contain 1400 bytes of user data along with the appropriate amount of header data. The router buffer size is 50 packets. The source traffic model is bulk data transfer (i.e., a file transfer) with the sender always busy.

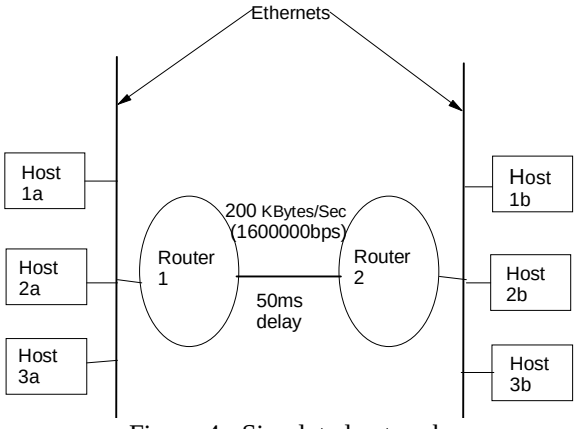

Figure 4: Simulated network

Figure 5 illustrates the behavior of multiple RTP connections that compete for bandwidth. The simulation consists of two one-way RTP connections that start at time 0 and 1 respectively and a late starting connection that starts at time 70 seconds. Traffic from each connections flows in the direction of Router 1 to Router 2. The simulation runs for 200 seconds. The top graph of Figure 5 shows the minimum queue level measured at the bottleneck router using .5 second intervals. We have set the ARB thresholds such that the operating range is between 2 and 11 packets (i.e., the T1/T2 thresholds). Due to the large propagation delay, there are times when the queue levels exceeds the threshold levels (as seen at time 75 seconds). However, once in steady state, ARB keeps the queue levels within the desired operating region.

The dashed line in the lower curve in Figure 5 shows the utilization of the bottleneck link (in the direction of Router 1 to Router 2). The solid lines in the curve shows the "goodput"[5] of the first and third connections (the goodput is sampled in 2 second intervals at the receiver). It is clear that ARB converges slowly to steady state. Once the third connection starts, it takes on the order of 200 seconds for the three connection's throughput to converge. Various parameters of the ARB algorithm can be tuned to increase the rate of convergence (e.g., the rate increment/decrement values, the ARB thresholds and the *burst_size*) however these adjustments typically lead to less stable network behavior (network oscillations and higher packet loss rates).

4 Comparison of RTP with TCP

Rather than an evaluation, the goal of this paper is to describe the ARB algorithm. In this section we expand on the description of ARB provided in the previous section by briefly comparing RTP with TCP. Like RTP, TCP is an end-to-end transport protocol that provides reliable, in-order service. However, unlike RTP, TCP employs a dynamic window, reactive congestion control scheme. End-to-end flow control is integrated with a Go-Back-N error recovery mechanism.

A fundamental aspect of TCP is that it obeys a 'conservation of packets' principle where a new packet is not sent into the network until an old packet has left. TCP implements this strategy via a self-clocking mechanism: acknowledgments received by the sender are used to trigger the transmission of new packets. This self-clocking property is key to TCP's congestion control strategy. If the receiver's acknowledgments arrive at the sender with the same spacing as the transmissions they acknowledge, and if the sender sends at the rate that acknowledgments are received, the sender will not overrun the bottleneck link.

[5] "Goodput" is defined as the rate that data is passed to the application. Therefore, "goodput" accounts for the throughput loss due to retransmissions.

Other elements of TCP's congestion control include the congestion recovery algorithm (i.e., slow start), the congestion avoidance algorithm, and the fast retransmit/recovery algorithms [2,5][6]. Because TCP's sending behavior is regulated by feedback, it is susceptible to unwanted behaviors such as phase effects and a phenomenon known as acknowledgment compression (i.e., ACKs lose their original spacing, becoming compressed at a congested router on their way back to the sender). While each behavior has been observed in real networks, it is unclear how common these behaviors actually are in real networks [6].

Figure 6 illustrates a TCP/Reno simulation that has traffic flowing in both directions (which leads to ACK compression). The maximum window size is set to 36 packets, all packets are 1400 bytes. The simulated network is shown in Figure 4. In one direction, three connections (FTP sources) begin at time 0, 1 and 2 seconds. In the other direction, three connections (two of which are under observation) begin at times 0, 1 and 70 seconds. To eliminate any artificial synchronization induced by the simulation, we add a random delay to the sender and the receiver that is in the range of ten's of microseconds to several milliseconds.

As in the RTP simulation, the top curve of Figure 6 plots the minimum queue length observed during a .5 second interval. The solid lines in the lower curve show the "goodput" of the first and third connections that flow from Router 1 to Router 2. The dashed line in the lower curve shows the utilization of the link in the direction of Router 1 to Router 2. The RTP results for the same two-way traffic simulation turn out to be virtually identical to the results shown in Figure 5.

5 Conclusions

In this paper, we have presented the congestion control scheme used by IBM's HPR protocol. ARB is a closed-loop, rate-based scheme that is designed to avoid congestion while maximizing network stability. By using periodic measurements, the ARB receiver tracks queue buildup in the one-way path between the sender and the receiver. Based on the latest congestion measurement and based on a history of previous samples, the receiver issues a message back to the sender instructing it how to modify its sending rate (i.e., increase, decrease or no change). Although this paper is clearly not an evaluation, the simulation results illustrate the network stability inherent in the algorithm. We showed that this is in sharp contrast to the sometimes unpredictable behavior of TCP/Reno.

References

1. J. Nagle, "Congestion Control in IP/TCP Internetworks", ACM Computer Communications Review, October, 1984.
2. V. Jacobson, "Congestion Avoidance and Control", Proceedings of the ACM SIGCOMM88, August 1988.
3. R. Jain, K. Ramakrishnam, "Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals and Methodology", Proceedings Computer Networking Symposium, Washington, April 1988.
4. A. Mankin, K. Ramakrishnan, "Gateway Congestion Control Survey", RFC 1254, 1991.
5. W. Stevens, "TCP/IP Illustrated, Volume 1 The Protocols", Addison-Wesley, 1994.
6. J. Mogul, "Observing TCP Dynamics in Real Networks", ACM SIGCOMM92, 1992.

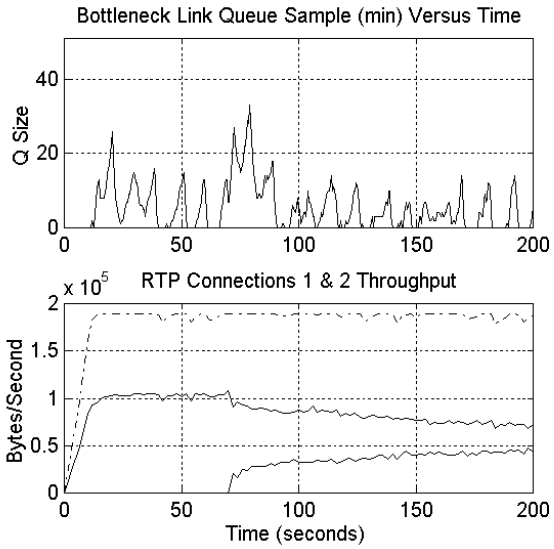

Figure 5: One-way RTP traffic simulation

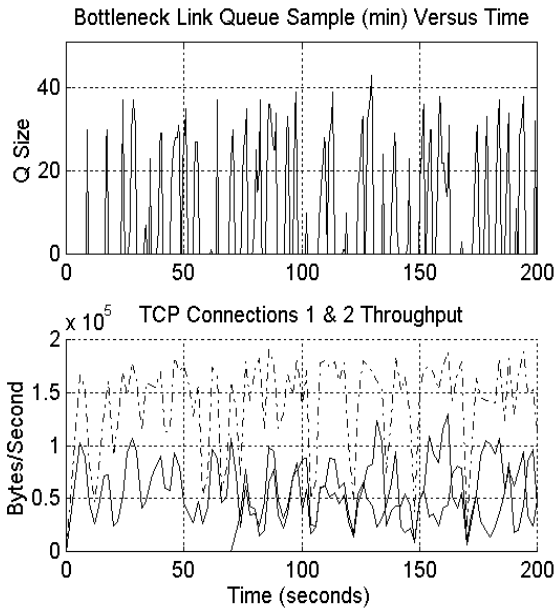

Figure 6: Two-way TCP traffic simulation

[6] The latest version of TCP containing these algorithms is known as TCP/Reno.