Chapter 6

RBCC – Measuring a Source's Sustainable Rate

One of the critical components of the proposed congestion control framework is the Sustainable Rate Measurement function. It has the task of determining a set of rates for all traffic flows in the network so as to prevent any router from becoming congested. As each router in the network has an instantiation of this function, each instantiation must communicate with the others to find this set of rates; therefore the function must work correctly in a distributed fashion.

In this chapter, I will outline the requirements for the Sustainable Rate Measurement function, examine an algorithm called RBCC[1] which meets these requirements, and outline how RBCC works within the congestion control framework.

6.1 Requirements of The Sustainable Rate Measurement Function

In order to perform its task correctly, the Sustainable Rate Measurement function should meet the following requirements:

- The function should allocate enough bandwidth to meet each traffic flow's desired rate, if possible. This ensures that each source's throughput is as high as possible, and network utilisation is high.
- The function should allocate bandwidth to flows so that no output interface's available bandwidth is exceeded. This keeps the incoming traffic down to a rate which can be retransmitted.
- A router is a **bottleneck** if the sum of the **Rate** fields for traffic flows crossing an interface exceeds the available bandwidth of that interface. If a router is a bottleneck, the function should communicate its allocated rates to each source affected, and to other routers. Thus, the framework adjusts to take into account the bottlenecks in the network.
- The function should allocate bandwidth to flows, taking into account any bottlenecks upstream (towards the source) and downstream (towards the destination). This is a corollary to the previous point.

[1] Rate Based Congestion Control

- Flows should be free to change their desired rate at any time and to any value. The function should cope with changes in a traffic flow's desired rate.

These requirements are mandatory. Other highly desirable requirements are:

- The function should allocate bandwidth fairly amongst a mix of traffic flows with infinite and finite desired rates.
- The function should treat flows with high round-trip times and a high number of intermediate routers equally to flows with low round-trip times and a low number of intermediate routers.

6.2 RBCC — An Example Function

For many of the proposed framework's functions, existing mechanisms can be used to perform the function. This is not true for the Sustainable Rate Measurement function. To overcome this, I have designed an algorithm called RBCC which satisfies all of the requirements given above for the function.

RBCC is implemented as part of the operations performed in every router. When a packet reaches an RBCC router, it performs the following operations:

1. Receive the packet.
2. Determine the output interface on which the packet must be retransmitted (i.e perform routing).
3. Decide to update any static congestion information based on the values of the congestion fields in the packet.
4. Decide to change the **Rate** and **Bottle_Node** fields in the packet based on the static congestion information.
5. Queue the packet on the output interface for transmission by the Link Layer, or drop the packet if there is no room in the output queue.

Operations 1, 2 and 5 must be done in any router, as was seen in Chapter 1. Operations 3 and 4 are part of the RBCC scheme. To these two we can add two more RBCC operations: the actual update of the static congestion information and the actual change of the congestion fields in the packet. An overview of the RBCC algorithm, and the data required for it to perform its work, is show in Figure 11. Also shown are the operations normally performed by a router.

The data required by RBCC to work are the packet's congestion fields and some static congestion information. The latter is known as the Allocated Bandwidth Table, and is described in the next section.

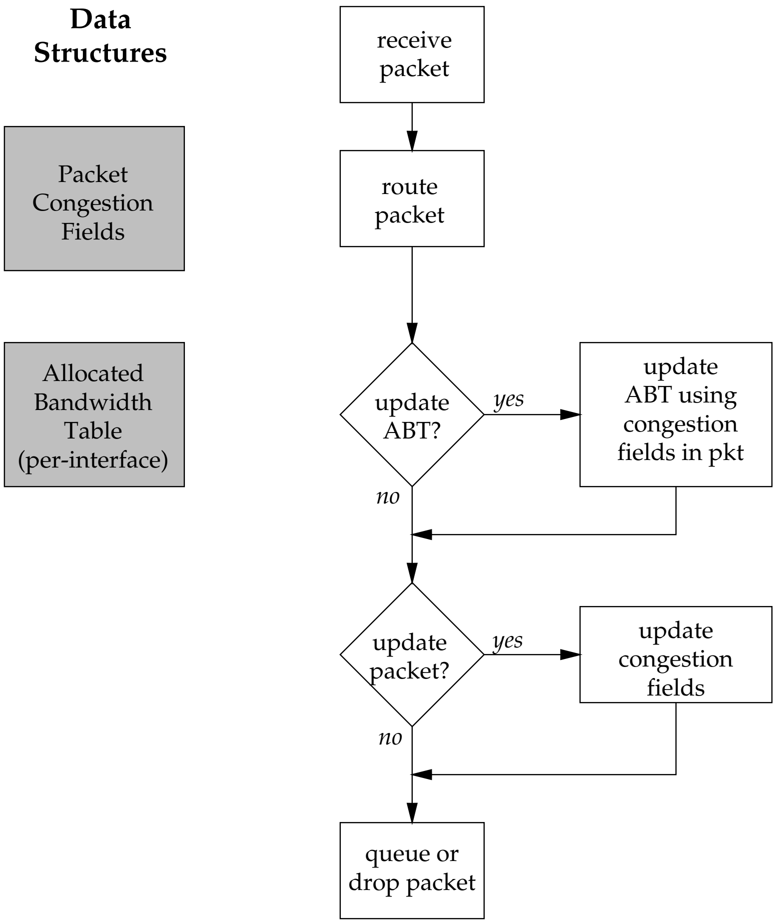

Figure 11: Simplified Execution Flow Diagram for RBCC

6.2.1 The Allocated Bandwidth Table

In order to calculate sustainable rates for traffic flows, RBCC needs to maintain information about these flows. Each router has a number of output interfaces which connect it to other routers, or to sources and destinations of data. For each output interface, RBCC keeps a table of information which describes the bit rate that has been allocated for each flow of data currently passing through the output interface. This table is known as the *Allocated Bandwidth Table*, or the **ABT**. Each entry in the table has the following fields:

Flow_Id, a value which uniquely identifies this flow of data.

Desired_Rate, the bit rate which the source of this traffic desires; it can be ∞.

Limiting_Rate, the limiting rate for the flow, described in Section 6.2.3.

Rate, the bit rate which has been allocated by the router for this flow of traffic; it cannot be zero or a value greater than the overall bit rate of the output interface. Similarly, the sum of the **Rate** fields in the table must not exceed the overall bit rate of the output interface.

Bottle_Node, the router which this router believes is the bottleneck on this flow of traffic. It may or may not be the current router.

A discussion on the **Flow_Id** is delayed until Section 7.2. The next sections describe the four operations performed by RBCC, and how the packet fields and the Allocated Bandwidth Tables are used by RBCC, with reference to Figure 11.

6.2.2 Deciding to Update the ABTs

The first operation in RBCC is to decide if the Allocated Bandwidth Tables need to be updated. This decision is made heuristically, and is performed using the congestion fields in the newly-arrived packet and the flow's ABT entries.

The Allocated Bandwidth Tables do not need to be updated on every packet arrival; in many cases, the packet contains the same congestion information as the last packet for this traffic flow. An optimisation for RBCC would be to cache the last-received congestion fields for every traffic flow; if the newly-arrived packet's fields are the same as the cached fields, the ABTs do not need to be updated. This optimisation is omitted below for clarity, but is performed in the simulations described in Chapters 8 to 11.

For every packet received by a router, there are *two* ABTs which could be updated. The first ABT is for the output interface to which the packet is destined. The second ABT belongs to the interface which retransmits packets coming in the reverse direction (i.e destination to source). Only the **Return_Rate** congestion field is used to update the second ABT. RBCC decides to update these ABTs on the conditions outlined below.

The first/forward ABT can be updated when:

- The packet is for a new traffic flow through the output interface. This new traffic flow must be allocated some bandwidth by the router.

  The ABT **Desired_Rate**, **Rate** and **Bottle_Node** field are set to the packet's values. A **Flow_Id** is created which identifies the new traffic flow.

- The **Desired_Rate** field in the packet differs from the **Desired_Rate** field currently in the ABT. The router may need to lower its allocated rate so as not to exceed the new **Desired_Rate**; alternatively, the router may be able to allocate more bandwidth if the **Desired_Rate** has increased.

  The ABT **Desired_Rate** field is set to the packet's value.

- The packet/ABT **Rate** fields differ. This may be due to a change in bottleneck upstream, or a change in the flow's rate allocation upstream. In any case, this router may need to update its own Allocated Bandwidth Table.

  The ABT **Bottle_Node** field is set to the packet's value, and the ABT **Rate** field is set to the packet's **Rate** value.

The second/reverse ABT can be updated when:

- The packet **Return_Rate** field is lower than the ABT **Rate** field. A downstream router has lowered the traffic flow's bit rate, and this router needs to update its ABT to reflect this.

  The ABT **Rate** field is set to the packet's **Return_Rate** value. If the ABT **Bottle_Node** field identifies this router as the bottleneck, then this field is changed to any other value.

Note that the ABT **Bottle_Node** field is only required to identify if the router is, or is not, the flow's bottleneck.

These heuristics ensure that the ABT contains the correct information required to determine the sustainable rates for all traffic flows passing through the output interface. If any ABT fields have been modified, then the appropriate ABT must be updated.

6.2.3 Updating the ABTs

Updating an Allocated Bandwidth Table given the information already in the table and the congestion information in a packet is not trivial. We have considered the conditions which might prompt recalculation of the information in an ABT. Let us now consider the work involved.

There are several criteria which must be taken into account when updating an ABT:

- Allocation of rates amongst the flows in the ABT must be fair.
- A traffic flow's rate allocation must not exceed the desired rate of its source, nor may it exceed the rate set by a bottleneck router which is not this router.
- A traffic flow's rate allocation cannot be set to zero or to ∞.
- The sum of allocated rate in the ABT must not exceed the output interface's overall bit rate.

Rate allocation must be fair, but the ABT may contain flows with many different desired rates; these may be finite or infinite as well. This makes rate allocation difficult. The rule of thumb that I have chosen for RBCC is:

> A flow has a desired rate D and has an existing rate allocation R from its bottleneck router[2]. The flow's *limiting rate* L is D if we are the bottleneck router, otherwise $min\{D, R\}$.
>
> A flow with limiting rate L should receive the same rate allocation A as all flows with limiting rates $\geq L$, unless $A > L$. In the latter case, the flow is allocated L, and the difference, $A - L$, is distributed to the flows with higher limiting rates, according to this rule.

This distribution rule gives the following results:

- Flows with equal limiting rates will be allocated the same bit rate.
- When bandwidth becomes scarce, flows with low limiting rates are more likely to preserve their existing rate allocations, and flows with higher limiting rates are more likely to have their allocations decreased.

In the extreme case when $A < L_{lo}$ for the flow with the lowest limiting rate, *all* flows will receive the rate allocation A. Only when flows cannot use their allocation A is the excess passed on to 'faster' flows. Therefore, the distribution rule treats flows with varying desired and limiting rates fairly.

The ABT update criteria and the distribution rule combine to form the ABT update algorithm used by RBCC. This algorithm is given below in the C language[3].

```
/* Data Structures */
struct conv_type                          /* Table entry for a single flow */
{
    int             flow_id;              /* Unique identifier for this flow */
    int             bottlenode;           /* Node which is the bottleneck */
```

[2] Note that R may be ∞ when there is no bottleneck router.

[3] This code is abstracted from the file `router/rbcc.c` in the modified REAL network simulator, described in Chapter 8.

```
    double            rate;             /* Rate allocated by a node */
    double            desired_rate;     /* Rate desired by the source */
    double            lim_rate;         /* Limiting rate on the flow */
    struct conv_type *next;             /* Pointer to next flow in the list */
};


/* Input Variables */
double bandwidth;                       /* Available bandwidth on interface */
struct conv_type *head_of_flow_list;    /* List of flows requiring bandwidth */
                                        /* on the output interface. Note that */
                                        /* this list is ordered by the */
                                        /* limiting rate field (ascending) */
int flowcount;                          /* Number of flows in the list */
int node;                               /* Router's unique identifier */

/* Local Variables */
double bandwidth_left;                  /* Bandwidth not yet allocated */
int flows_left;                         /* Flows with no allocations yet */
struct conv_type *c;                    /* Pointer to each flow in the list */
double allocation;                      /* Bandwidth allocation chosen for */
                                        /* each flow */


/* Output Variables: the rate and bottlenode fields in the linked list */
/* are updated by the algorithm. */


/* Algorithm */

  /* Fairly divide the bandwidth up amongst the flows. */

  bandwidth_left= bandwidth; flows_left= flowcount;
  for (c= head_of_flow_list; c!=NULL; c= c->next) {

                /* The bandwidth_left variable contains the bandwidth */
                /* yet to be allocated to remaining flows. We assume that */
                /* this flow can use an N'th amount, where there are */
                /* N flows remaining to get allocations */
      allocation = bandwidth_left / (1.0 * flows_left);

                /* If the flow's limiting rate is higher, */
                /* we become its bottleneck and set its rate. */
      if (c->lim_rate > allocation) {
          c->rate= allocation; c->bottlenode=node;

                /* Otherwise, the flow is bottlenecked elsewhere. */
                /* Only give it its limiting rate. The excess from */
                /* the allocation calculated above will be shared amongst */
                /* the flows with higher limiting rates. */
      } else {
          allocation= c->lim_rate; c->rate= allocation;
      }

      /* Subtract the allocation actually given to the flow from the */
      /* remaining bandwidth. */
      bandwidth_left-= allocation; flows_left--;
  }
```

6.2.4 Updating the Packet's Congestion Fields

After the Allocated Bandwidth Tables have been recalculated (if required), there are two instances when a router can change a packet's **Rate** and **Bottle_Node** fields. If the router has allocated a lower bit rate than that indicated by the **Rate** field, then the **Rate** field can be lowered and the **Bottle_Node** field updated to the value in the ABT.

When the router is itself the bottleneck on the traffic flow, the **Rate** field can always be overwritten by the **Rate** field in the ABT. This has the effect of increasing the bandwidth available to the traffic flow when there is excess bandwidth at the router (another traffic flow may have terminated). The **Bottle_Node** field is also updated to be the value in the ABT.

Similarly, in the reverse direction, the **Return_Rate** value can be overwritten if the router is the bottleneck router, as shown in Figure 12.

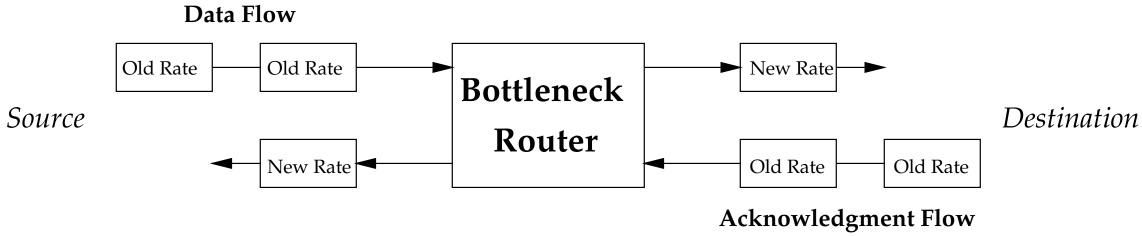

Figure 12: Reverse Congestion Field Update

The bottleneck router has set a new rate for the traffic flow. The **Rate** fields in data packets are updated as they pass through the router. As well, **Return_Rate** fields in acknowledgment packets are also updated. The source learns about the new allocated rate faster than if the **Return_Rate** fields were not being updated.

6.3 TRUMP/RBCC Operation

A rate-based transport protocol such as TRUMP uses the sustainable rate measurements from RBCC in order to minimise network congestion. Let us now look at the operation of TRUMP and RBCC in tandem.

6.3.1 Source Transmission

TRUMP, as a transport protocol, provides not only rate-based flow control, but error correction via packet retransmission and selection of the type of connection required between the source and destination. This section will concentrate on the flow control aspects of TRUMP.

A TRUMP Transport Layer can begin transmission either by performing a two-way handshake with the destination Transport Layer, or by starting immediately with data transmission. The two-way handshake is performed as follows:

1. Transmit connection setup packets to the destination with an inter-packet separation of S seconds.
2. If no setup acknowledgment packet is received after T seconds, abort the connection. Stop.

3. If a setup acknowledgment packet is received, discontinue setup packet transmission and start transmitting data packets.

In the simulation of TRUMP and RBCC, S is 1 second and T is ∞. In reality, T would be finite.

During data transmission, TRUMP transmits data packets spaced by an inter-packet delay, whose value is calculated using Equation 4.1, which is given again below:

$$delay = \frac{packet\ size\ in\ bits}{rate\ in\ bits\ per\ second}$$

where $rate\ in\ bits\ per\ second$ is obtained from the **Return_Rate** field from either the connection setup acknowledgment packet or the last received data acknowledgment packet. Remember that an increase in the delay causes TRUMP to ignore **Return_Rate** fields for one round-trip. A single timer per connection is set to the inter-packet delay, and this prompts the TRUMP source to transmit the next data packet.

As data packets pass through the network, their congestion fields are (possibly) set by the intermediate routers according to the RBCC algorithm.

6.3.2 Packet Reception by the Destination

After being forwarded by all the routers between the source and destination, and having the **Rate** and **Bottle_Node** fields set to reflect the bandwidth available to the corresponding traffic flow, a data packet reaches the destination. This may or may not prompt the transmission of a selective acknowledgment. In any event, a selective acknowledgment is eventually transmitted to the source, and its **Return_Rate** field are set to the values received in the most recent packet.

As the stream of selective acknowledgments forms a traffic flow in its own right, the **Desired_Rate** field in each acknowledgment must be set to a sensible value. The method used to determine the desired rate for the flow of acknowledgments back to the source is:

$$desired\,acknowledgment\,rate = most\ recent\ data\ rate\ value * \frac{ack\ packet\ size\ (in\ bits)}{total\ data\ being\ acked\ (in\ bits)}$$

This has the effect of limiting the desired rate of acknowledgment packets to a fraction of the available bandwidth for data packets, with the fraction based on the ratio between acknowledgment packet sizes and the total size of data packets being acknowledged.

The assumption behind the equation above is that the data traffic flow is the limiting factor on the acknowledgment traffic flow. In some circumstances, it may be the other way around: this is not yet catered for.

In the simulation of TRUMP, the transmission of acknowledgment packets are not explicitly separated with any inter-packet delay; it is assumed that the spacing of data packets from the source will correctly space the acknowledgment packets from the destination. The results of this assumption are described in Section 10.10.

6.3.3 Packet Reception by the Source

As with the data packet, the acknowledgment packet is forwarded by several routers before being received at the source. As with the destination, the values of the **Rate** and **Bottle_Node** fields received by the source are returned to the destination as the **Return_Rate** field field in future data packets.

The **Return_Rate** field from the acknowledgment packet is used by the source to recalculate the inter-packet delay, using Equation 4.1 as before:

$$delay = \frac{packet\ size\ in\ bits}{Return\_Rate\ in\ bits\ per\ second}$$

The new delay takes effect after the source's next data packet transmission.

6.3.4 Flow Termination

Correct calculation of a router's ABTs depends upon the router knowing about all the traffic flows through it. It can deduce the start of a new traffic flow by its packets, but it cannot easily deduce the end of a traffic flow from a lack of packets; the flow may have been slowed down by a bottleneck somewhere.

To remove this problem, two more congestion control fields must be added to the four described in Section 4.4. These are:

Flow_Id: A value which uniquely identifies this flow of data. Strictly, the concatenation of the source address and the **Flow_Id** is unique.

Flow_End: A boolean which indicates that the flow has ended.

Transport Layer protocols such as TRUMP must ask the Network Layer to set the **Flow_End** flag for packets which terminate the flow of traffic. When RBCC receives packets with the **Flow_End** flag set, it removes the corresponding traffic flow entry from the ABT and then recalculates the table. Further packets with the the **Flow_End** flag set have no flow allocation, and so should require a negligible overall bit rate.

A TRUMP source sends connection teardown packets every second once there are no packets left to send, and stops sending packets after it has received one teardown packet back from the destination. All connection teardown packets have the **Flow_End** flag set.

Upon receipt of a *teardown* packet, a TRUMP destination immediately reflects it back to the source, again with the **Flow_End** flag set.

6.3.5 Abnormal Loss of a Traffic Flow

If packets from a traffic flow do stop passing through a router, then unless the corresponding ABTs are recalculated, the bandwidth of the output interfaces will be underutilised. If a flow terminates abnormally, or if events such as a route change at another router take place, the router will not see packets with the **Flow_End** flag set.

In this situation, all that the router can do is to assume that the traffic flow has stopped after a certain period of time has expired. The optimal value for this time delay would be the inter-packet delay for the traffic flow scaled up by a small factor K; if no packets are seen after this time, then the router can assume that the flow has stopped.

This introduces a dependency on inter-packet delay estimation, which the Sustainable Rate Measurement function as presented does not perform. In the simulation of RBCC in REAL, a constant $F = 5$ seconds was used. This of course implies that once a flow stops, the router will be under-utilised for up to 5 seconds. Alternatively, if a traffic flow spaces its packets with intervals of $>$5 seconds, the ABTs will be recalculated on receipt of every packet in the flow.

6.3.6 Congestion Avoidance

One of the main factors used to calculate the sustained rate values in RBCC is the actual bandwidth of an output interface, which is constant. As this constant has an overriding effect on the sustained rate values, it can be scaled when a router believes it to be above the knee of the power curve, to provide congestion avoidance.

In my simulations, I chose to scale the interface's bandwidth value as follows:

$$bandwidth\ value\ used = \begin{cases} actual\ bandwidth & \text{if } packets\ queued \leq T \\ actual\ bandwidth * \alpha & \text{if } packets\ queued > T \end{cases} \tag{6.1}$$

In my simulations, values of $T = 5$ and $\alpha = 0.75$ were used, where each router could queue up to 10 data packets, and this worked well as a congestion avoidance scheme.

Other bandwidth scaling possibilities are to have low- and high-watermarks for the number of packets queued, T_{low} and T_{high}, or to scale the actual bandwidth as a function of the number of packets queued. Scaling may be done with or without hysteresis effects.

With any form of sustainable rate scaling, there is the possibility of source rate oscillation with a period of the round-trip time. Any congestion avoidance scheme using sustainable rate scaling must take this into account. With the scaling technique described above, there were minor oscillations observed in the simulations, which had no significant impact on network performance.

A full description of the effect of T and α is given in Chapter 11.

6.4 Summary

The Sustainable Rate Measurement function is the core element of my rate-based congestion control network. In order to perform correctly and fairly, it must meet several requirements which were outlined in Section 6.1. One such function which does meet these requirements is RBCC. It is composed of a number of heuristics and a rate allocation algorithm.

RBCC sets the **Rate** field in each packet based on the information it keeps in a set of Allocated Bandwidth Tables. These tables are updated by the rate allocation algorithm if the heuristics so indicate. The rate allocation algorithm uses congestion information in packets to determine a fair set of sustainable rates for all flows of traffic through a router.

The elements in the rate-based framework, and functional instantiations for two elements, have so far been discussed. The next chapter will examine some of the issues raised during the exposition of the framework. The chapters following will then examine the implementation of the rate-based framework within a network simulator, and the performance of the framework's congestion control.