Chapter 7

Comments on the Congestion Control Framework

The previous three chapters have described my proposed rate-based congestion control scheme in some detail. There have been several issues raised which require further consideration; these include the available parameters of the scheme and its behaviour. I will address these issues in this chapter.

7.1 The Framework as a Feedback System

It should be apparent that the congestion control framework forms a closed loop feedback system with a delay of one round-trip; this round-trip includes the delay in generation of selective acknowledgments. Note, however, that the framework does not use the value of the round-trip. The feedback loop is shown in Figure 13; compare this figure with Figure 4 (pg. 33), which has more detail.

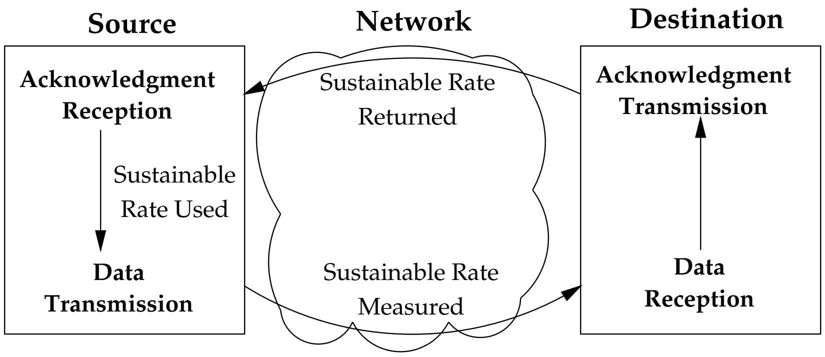

Figure 13: The Framework is a Closed Loop

I have suggested that when the source receives a **Return_Rate** value which lowers its transmission rate, it should not react to further return rate values for at least one round trip time. This allows the effect of the lower source rate to be measured by the network components, and helps to minimise short-term rate fluctuations which would be caused by short-term congestion.

If, on the other hand, the **Return_Rate** value would increase the source's transmission rate, then the network is advertising extra capacity and the source can use this rate value immediately.

In any event, the source's transmission rate cannot go to zero, as this would prevent the source from transmitting due to an infinite inter-packet delay. Similarly, the source's transmission rate cannot go to infinity, as this would cause the source to saturate the network with data. Any Sustainable Rate Measurement function must ensure that the rate returned to a source is neither a zero nor infinity.

When a source begins transmitting, it will not have an initial rate value. There are several alternatives here which are not defined in the TRUMP protocol. If the source has had a similar connection to the same destination in the recent past, it may be able to use the final rate value from that connection. If the source has no historical information, it can start transmitting at a very low data rate. After one round trip, it will have better rate information. If the source performs explicit connection establishment, then it will obtain rate information from the destination before any data has been transmitted over the connection.

The round-trip, and hence the speed of operation of the closed loop, is affected by both the overall link latencies and by the delay in acknowledgment generation. The former is outside of the control of the framework, but a high overall link latency should not penalise a traffic flow. Transport protocols within the framework can control the delay in acknowledgment generation by choosing the number of data packets acknowledged in every selective ack packet: the higher the number, the higher the delay in acknowledgment generation. However, the higher the number acknowledged, the less bandwidth is required for the acknowledgment flow.

It is arguable, in these days of extremely high link bit rates, that the bandwidth required for the acknowledgment flow is negligible. However, if an acknowledgment flow is bottlenecked by a low-speed link, then this is not true. The transport protocol endpoints may examine the sustainable rates and round-trip times for the data and acknowledgment flows and negotiate an appropriate selective acknowledgment size.

In Chapter 11 the effect of different sized selective acknowledgments on the framework's performance is simulated, and the results summarised.

7.2 What Constitutes a Traffic Flow?

The Sustainable Rate Measurement function must uniquely identify each traffic flow, and distribute bandwidth to each traffic flow as a set of sustainable bit rates. We saw in Section 6.2.1 that each entry in the Allocated Bandwidth Tables has an identifier for its traffic flow. But what constitutes a traffic flow? There are several possibilities:

1. The Transport Layer is the source of the traffic flow, acting on behalf of an application. The flow is uniquely identified by the tuple (Source Host, Destination Host, Source Port, Destination Port);
2. The Transport Layer is the source of the traffic flow, which is uniquely identified by the tuple (Source Host, Source Port);
3. The Host is the source of the traffic flow, which is uniquely identified by the tuple (Source Host, Destination Host); or
4. The Host is the source of the traffic flow, which is identified by the tuple (Source Host).

The fourth possibility can be immediately dismissed, as a host may be transmitting to multiple destinations with packets taking multiple routes. The second possibility can be discounted for transport protocols such as TCP where an application is tied to a particular port, and may be communicating with several destinations from that port. TRUMP also falls into this category.

We are left with options 1 and 3, and each has quite different characteristics. In the first option, every application to application communication is a traffic flow, and there can be several traffic flows between the same source and destination hosts. Assuming the flows all take the same route, each flow is represented as entries in two ABTs[1] in every traversed router. The routers have a set of desired rates with which to divide up the available bandwidth equitably.

In option 3, each source machine to destination machine communication is a traffic flow. Not only does the Transport Layer demultiplex the incoming packet stream to one or more applications, but it must also demultiplex the sustainable rate returned to it amongst the one or more applications communicating with the same destination. It must also sum the applications' desired rates to form a single **Desired_Rate** value; obviously, ∞ plus a finite value is still ∞.

There are several advantages with option 3. There are fewer entries in each routers' ABTs, as the routers perceive fewer traffic flows. The source can choose its own policy on dividing up the available bandwidth to the applications communicating with the same destination, and it is in a better position to determine the applications' real desires than the network. It also requires fewer timers to perform inter-packet delays.

However, there are several disadvantages with option 3. Because application data flows are aggregated, the routers do not perceive the real number of traffic flows, and may distribute bandwidth fairly on a per-source/destination basis but not on a per-application basis. Although the Transport Layer has fewer timers to deal with, it also has to multiplex the applications' data so as to meet the Leaky Bucket's inter-packet delay. This may cause fluctuations in per-application packet spacing, which may be unacceptable for time-sensitive applications.

Finally, different application data flows have different desired qualities of service, and different desired data rate requirements. A file transfer flow desires an infinite and uniform bit rate, but a remote login flow desires a low and spasmodic bit rate. Aggregating different application data flows may penalise flows with certain characteristics and qualities of service.

In the simulations described in the next few chapters, I chose option 1 to identify each traffic flow.

7.3 What Congestion Fields are Required?

If option 1 in the previous section is chosen, each application data flow is a traffic flow in its own right. I have proposed that each packet transmitted in the congestion control framework has the following fields: the **Flow_Id**,**Desired_Rate**, **Rate**, **Bottle_Node**, **Return_Rate** and **Flow_End** fields.

I would suggest that the rate fields can be floating point numbers at least 32 bits in size, and the node field is most likely to be a network node identifier. The **Flow_End** field can

[1]One for the data flow and one for the acknowledgment flow.

be a single bit. The **Flow_Id** field, combined with the network address of the source, must uniquely identify the flow: a good minimum for this field would be 16-bits, to allow for 65,536 traffic flows per source. Therefore, an implementation of the framework in IPv4 and IPv6 would require at least 113 and 209 extra bits of header per IP datagram, respectively.

The **Rate** and **Return_Rate** fields must exist in each packet, as no feedback would be performed without these fields. The **Desired_Rate** field is extremely useful, as this allows the Sustainable Rate Measurement function to distinguish between flows that will accept any extra bandwidth available, and those which would not make use of the extra bandwidth.

The **Bottle_Node** field is used by the Sustainable Rate Measurement function to determine where a traffic flow is bottlenecked, i.e by the router currently processing the packet, or by another router. Unfortunately, in RBCC's current form, this field is required: this has been confirmed by simulations. If RBCC cannot tell who is the bottleneck of a traffic flow, it cannot determine if any spare capacity can be donated to the flow.

The overhead of 128 bits for the **Bottle_Node** field in IPv6 is large. However, this field only has to indicate which router is the bottleneck. A suggestion would be for each router to randomly choose a smaller unsigned integer as its 'bottlenode' identifier. If a 32-bit identifier was used, a saving of 96 bits would be made, and the probability of an identifier collision across the route of any traffic flow would be very small.

Given these suggestions, a realistic congestion control header for IPv6 requires 192 bits, and is shown in the following diagram. The **Flow_End** flag is indicated by the letter 'E'. The **Version** field allows the congestion control header to be changed without confusing routers using old versions of the congestion control header.

<table>
<tr><td colspan="32">00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td></tr>
<tr><td colspan="8">Next Header</td><td colspan="8">Header Ext Length</td><td colspan="4">Version</td><td colspan="1">E</td><td colspan="11">Unused</td></tr>
<tr><td colspan="32">Flow Id</td></tr>
<tr><td colspan="32">Desired Rate</td></tr>
<tr><td colspan="32">Rate</td></tr>
<tr><td colspan="32">Bottle Node</td></tr>
<tr><td colspan="32">Return Rate</td></tr>
</table>

Figure 14: The Congestion Framework Header in IPv6

7.4 Transport and Network Layer Intercommunication

In the proposed congestion control framework, the Transport and Network Layers in the source of each traffic flow must communicate so that the flow is correctly identified and terminated, and that the transmission for the flow occurs at the correct rate. Here is a description of the intercommunication required between the two layers.

1. When a new flow is created by a Transport Layer protocol, a flow identifier must be chosen by the protocol. The flow identifier must uniquely identify the flow within the source.
2. Every Transport Layer segment given to the Network Layer for transmission must be accompanied by the destination network address, a **Desired_Rate**, a **Flow_Id** value,

and a **Flow_End** value. If the segment is part of an exchange to terminate the flow, the **Flow_End** value must be true; otherwise, it must be false.

3. After prepending a Network Layer header, and before the Network Layer datagram is transmitted, the Network Layer sets the **Rate** field to ∞ and sets the **Bottle_Node** congestion control field to identify the host. The **Desired_Rate**, **Flow_Id** and **Flow_End** fields are set to the values from the Transport Layer.

4. Every Transport Layer segment given to a Transport Layer protocol after reception by the Network Layer must be accompanied by the source network address, the value of the **Return_Rate** field and the value of the **Flow_Id** field.

5. When a **Rate_Quench** datagram is received by the Network Layer, the value of the **Return_Rate** and **Flow_Id** fields must be passed asynchronously to the Transport Layer protocol. As well, the Quench must contain the first N octets of the Transport Layer segment from the datagram which prompted the Quench. These N octets must also be passed to the Transport Layer. This allows the Transport Layer to determine which particular segment prompted the Quench.

7.5 Complexity of RBCC

In the proposed rate-based congestion control framework, most of the control burden has been moved into the network's routers. This flies in the face of traditional congestion research which has concentrated on end-to-end solutions. The experimental results in Chapters 9 to 11 show how significantly better the rate-based framework controls congestion, when compared to the end-to-end scheme used by TCP. Unfortunately, routers face an increasing burden from other areas: higher link bandwidths, packet filtering, CIDR routing extensions, IPv6 extensions and sophisticated dynamic routing protocols. Existing Internet routers must route and retransmit up to hundreds of thousands of packets per second, consulting routing tables with tens of thousands of entries.

Therefore, although the proposed rated-based control framework provides significant congestion control benefits, will the extra workload it places on network routers make the deployment of the framework untenable? This issue is addressed briefly here, and in more detail in Appendix D.

The RBCC algorithm must be implemented on every router within a network using the rate-based framework. This entails an extra CPU overhead and an extra memory allocation overhead on each router.

RBCC maintains state about the existing traffic flows across an output interface in the interface's Allocated Bandwidth Table. Many of the operations that deal with the packet congestion control fields must consult this table. Let us look at the complexity of each operation in turn.

During the ABT update decision, the ABT entry corresponding to the incoming packet must be found. A linear search has order of complexity O(N), where N is the number of flows in the ABT. For a new flow, a new ABT entry must be inserted. The entries need to be kept in increasing limiting rate order, and so an insertion sort is required. A linear insertion sort has order of complexity O(N).

During the ABT update proper, available bandwidth must be fairly distributed across all traffic flows in the ABT. With the table ordered by limiting rate, the allocation to each flow is independent of the other flows, and the update operation has order of complexity O(N).

Finally, a packet's congestion fields may need to be updated from the ABT. A reasonable implementation should retain a 'pointer' to the entry from the ABT update decision operation, and this operation should have order of complexity O(1).

The overall complexity of the operations performed by RBCC is O(N), where N is the number of traffic flows represented in the output interfaces Allocated Bandwidth Table.

Without reducing the complexity, the speed of the RBCC operations can be improved by such techniques as caching the most-recently-seen congestion fields for each traffic flow. The ABT does not need to be updated if the cached congestion fields match those in a received packet. The results given in Section 10.11 show that caching appreciably reduces the number of ABT updates.

Similarly, an ABT does not need to be updated if all the flows in the ABT are bottlenecked elsewhere, and the sum of the flows does not exceed the output interface bandwidth. Addition of this pre-update test to the simulated code removed approximately 60% of ABT updates due to cache misses.

Appendix D examines the cost of RBCC on current core Internet routers. The conclusion is that RBCC imposes an extra memory and processing burden on Internet routers which is commensurate with their current memory and packet switching CPU burden.

7.6 RBCC and Fair Share

The RBCC function meets the requirements of the Sustainable Rate Measurement function within the congestion control framework. [Charney 94] describes and analyses a similar algorithm for allocating rates to traffic flows, the 'Fair Share' algorithm.

There are several differences which distinguishes Charney's work from RBCC. In essence, the framework in which Charney's algorithm operates assumes that all traffic flows desire an infinite transmission rate; moreover, each instantiation of the algorithm operates independently of the others. The independence of the algorithm instantiations means that only some of the data which can help to determine optimum data rates is utilised. In RBCC, not only is rate information used in **both** directions, but the identity of the current *bottleneck* for each traffic flow is delivered to each router, which assists in determining the optimum data rates.

Charney notes that finite desired transmission rates can be modeled in the framework by inserting a 'fictitious' link between the traffic source and the first router, where the link has a maximum data rate equal to the source's desired rate. The drawback of this fiction is that the desired transmission rate is thus fixed, and cannot be altered by the traffic source once set.

In Charney's framework, some packets carry a "stamped rate" field, which corresponds to the **Rate** and **Return_Rate** fields in this work, described in Section 4.4. A bit in the packets indicates whether the "stamped rate" field is being propagated to the destination (corresponding to the **Rate** field in this work), or to the source (corresponding to the **Return_Rate** field in this work).

Fair Share does not appear to take into account the bandwidth required by reverse traffic flows (e.g by acknowledgment flows for transport protocols), but the "stamped rate" field for each direction must be carried in separate packets, and it is unclear how the "stamped rate" is returned to the source of acknowledgments. In the framework proposed in this thesis, each packet holds rate fields for both directions: **Return_Rate** information does propagate to the source of acknowledgments.

Sources cannot indicate a desired transmission rate to the Fair Share algorithm. This can cause underutilisation of available bandwidth. This is overcome with an extra bit, the *u-bit*, associated with the "stamped rate" field. "Greedy" flows clear this bit, and flows with fixed desired rates set this bit on. Intermediate routers set the *u-bit* when their advertised rate for the flow is **less** than that indicated in the "stamped rate" field. If a "stamped rate" field returns to the source with the u-bit off, then the advertised rate of all links was higher than the stamped rate of the original packet. In this case, flows desiring more bandwidth can increase their rate, and the value in the "stamped rate" field, to an arbitrary value. The algorithm will then converge to a new set of optimum values.

Finally, although Charney gives a theoretical analysis of Fair Share, this is done to show that the allocated rates converge to the optimum values, when loads on a network are static, and when they are slowly changing (due both to route changes and the set of traffic flows). The problem of network congestion is not considered, and Charney notes:

> In addition, we allow the switches to drop packets as they please, as long as at least some packets of each session continue to get through. While dropping packets can cause a lot of wasteful retransmissions, it is essential to note that our algorithm will still calculate correct optimal rates even in the presence of heavy packet loss.

The thrust of this thesis is to show that RBCC, an algorithm similar to Fair Share, not only allocates optimum transmission rates, but that network congestion is also *effectively controlled*, when RBCC is used as part of a congestion control framework. An area of future research would be to evaluate the performance of Fair Share within my proposed congestion control framework, and to compare it with RBCC.

7.7 Determination of Desired Rate

There are essentially three types of network applications. The first desire an infinite bit rate: file transfers and the like. The second desire a constant bit rate: voice and video transfers. The third have a distribution of bit rates: remote logins by humans, for example.

The source Transport Layer can easily deal with the first two. The third type of application does not fit so well into the rate-based congestion control framework. One solution is to apply an arbitrary rate limit to an application of this type, at the source Transport Layer.

Assume that the Transport Layer advertises the arbitrary rate limit as the flow's desired rate, and is allocated this desired rate. When the application is transmitting below this rate, extra capacity is reserved but goes unused. When the application is transmitting above this rate, the source transport layer buffers the application data and admits it using the Leaky Bucket scheme into the network.

The difficulty is the choice of the arbitrary rate limit. The source should be able to observe the application's short-term bit rate requirements, and move the arbitrary rate limit to utilise the available capacity as best as possible. Note, however, that the arbitrary rate limit becomes the advertised desired rate, and changes in the desired rate cause extra RBCC work to be performed in the routers along the flow's path. Therefore, a compromise needs to be reached between optimal utilisation and excessive RBCC operations. I have not considered this problem in this thesis.

The rate-based control framework is targeted at long-duration traffic flows. Flows which consist of a small number of packets are not specifically catered for. If such flows form only a small fraction of the total network traffic, then the framework will still work, but router queue lengths will be nominally higher. If short-duration traffic flows are a substantial fraction of the network's load, then some other strategy will be required to deal with them.

In any event, short-duration flows should be identified with a special **Flow_Id** value (such as 0) within the framework, as a flag to prevent the creation and destruction of ABT entries over a short period of time.

7.8 Bandwidth Reservation and Resource Monitoring

The RBCC algorithm divides router interface capacity fairly amongst traffic flows, while respecting rate limitations set by other routers. If more flows cross through the router, existing flows may have their rate allocations reduced.

The sources of all traffic flows learn about their new rate allocations. For applications which require a *minimum* transmission rate, the new allocation may be unacceptable. As the framework stands, nothing can be done in this situation.

However, it is relatively straight-forward to alter the RBCC algorithm to *prioritise* certain traffic flows: these priority flows will obtain rate allocations first. An example priority scheme would be to have flows with fixed and equal minimum/maximum rate requirements. These would be given priority over other traffic flows. Once the flow's rate is allocated by a router, it cannot be changed while the flow lasts. If there is not enough available bandwidth at a router to sustain a 'fixed rate' flow, its source can be informed and terminate the in-progress connection.

This example priority scheme, in essence, allows 'fixed rate' flows to reserve bandwidth across the network for the flow's duration, while still providing some bandwidth for 'arbitrary rate' flows. The scheme is easily accomplished by ordering the ABT lists in priority order first, and then by ascending limiting rate order. The current Internet architecture does not allow such bandwidth reservation.

Flow priority and bandwidth reservation schemes, while possible in the congestion control framework outlined, are strictly outside the scope of this thesis, and will not be addressed further.

RBCC routers, with Allocated Bandwidth Tables, are in an ideal position to monitor and enforce the sustainable rates set by the distributed RBCC algorithm. A badly-behaved source which exceeds its rate allocation must still have its packets forwarded by the routers who set the rate allocation. They should be allowed to drop some of the source's packets to keep its transmission within the allocated rate.

Again, the monitoring and enforcement of rate allocations is outside the scope of this thesis: I am only considering well-behaved traffic sources. However, as with flow priorities, this area would be a fertile ground for further research.

7.9 Acknowledgment Flow Problems

One severe problem with the congestion control framework, not yet addressed, is the determination of a good desired rate for a destination's acknowledgment flow. Section 6.3.2 gives an equation to determine an instantaneous desired rate for the acknowledgment flow, but in practice this equation will give fluctuating desired rate values.

Consider: The source transmits packets (evenly spaced by Leaky Bucket) to the destination. Packets may be variable in size. Most transport protocols acknowledge packets rather than data bits. Therefore, a constant bit-rate but variable packet-rate data flow will create a variable packet-rate *and* bit-rate acknowledgment flow. If the sizes of incoming data packets are unpredictable, then the destination cannot accurately determine a good desired rate for its acknowledgment flow.

There is no optimum solution for this problem. Data packet sizes cannot be made constant. A transport protocol which acknowledges data bits rather than data packets could be constructed, but it must deal with acknowledgment of retransmitted and out-of-order data and still keep a constant acknowledgment flow rate: this would be difficult to achieve.

Using low-pass or adaptive filters on the instantaneous desired rates calculated by the equation in Section 6.3.2 would produce an average desired rate, but this would not be optimal. In the simulations described in the next few chapters, I chose to use the highest desired rate ever produced by the equation: this wastes bandwidth in the direction of acknowledgment transmission, but prevents oscillations if the instantaneous desired rate values were used. An adaptive filter would be a much better solution.

7.10 Packet Admission and Readmission

One of the requirements for the rate-based congestion control framework is that:

> Packet admission and readmission should be smooth. Bursty traffic (such as is seen during window updates) not only causes short-term congestion but increases end-to-end and round-trip variance. Transport protocols should admit packets into the network as evenly spaced as possible, and routers should attempt to preserve this spacing.

As presented, an RBCC router makes no attempt to preserve packet spacing, despite the requirement above. Simulation results, given in Section 10.10, show that despite this lack of router functionality, inter-packet spacing is well preserved across a network of RBCC routers. Functionality to perform smooth packet readmission by RBCC routers is not required.

7.11 Summary

We have addressed some issues concerning the architecture, implementation and performance of the rate-based congestion control framework. The complexity of the framework should not add an insurmountable load on the network routers, and it removes much of the work that traditional end-to-end transport protocols must perform in terms of flow and congestion control. The framework can also be configured to allow bandwidth reservation or priorities to certain traffic types, and it can monitor and enforce all allocated rates. The overhead of the extra congestion control fields in all packets is significant, but this can be addressed with appropriate hashing techniques.

The identification of a traffic flow was examined. There are several possibilities here, each with their own advantages. After considering all the alternatives, I resolved that a traffic flow within the framework is the flow of packets which conveys data between two applications. With this option chosen, the intercommunication responsibilities between the Transport Layer and the Network Layer were enumerated.

Several problems have been highlighted in the congestion control framework, notably the problem of determining desired rates for acknowledgment flows. Crude solutions to these problems exist, but the penalty is non-optimum network usage. Further research is needed to address these shortcomings.

In order to demonstrate the effectiveness of the framework in 'real' situations, I now turn to the examination of an implementation of the framework in a packet-based wide-area network simulator.