RBCC: A Rate-based Congestion Control Scheme for Packet Switched Networks

W. Toomey

School of Computer Science
University College
The University of New South Wales
Australian Defence Force Academy
Canberra, ACT

Abstract

Congestion control schemes in packet switched networks have usually involved changes in transport layer window sizes (TCP, TP4), feedback of congestion indicators to transport layers (DecBit, Q-Bit, RED), heuristics for output queue reordering (Fair Queueing) or heuristics for packet dropping (Random Drop). This paper diverges from these paradigms by proposing a network layer-based congestion control scheme that calculates a near-optimal rate for all packet sources, and a rate-based transport protocol which uses this rate information in its flow control. The rate-based congestion scheme is compared against TCP Reno's end-to-end flow control in a simulated network scenario, and the results are examined in detail.

Warren Toomey, email: wkt@cs.adfa.oz.au

Contents

1 Introduction

Congestion in connectionless packet-switched wide-area networks is a major problem, and this problem is growing alarmingly in such networks as the Internet, which is suffering an exponential growth in users and network traffic. This technical report outlines a method of congestion avoidance for networks with this architecture, and compares the effectiveness of this method against TCP/IP in a single network scenario.

This technical report is divided into three parts. Part 1 (Sections 1 to 2) gives a brief look at the problem of network congestion and the current solutions for it. Part 2 (Sections 3 to 4) outlines a new solution to network congestion (RBCC) which is rate-based, and details the basis of its operation and its implementation in the REAL network simulator. Part 3 (Section 5) compares RBCC against the congestion mechanisms in TCP/IP in an example network scenario.

1.1 Network Architecture

The network architecture under consideration has the following characteristics:

1. The network is a *wide-area network* with nodes in the network passing data to other nodes along links. The nodes are *arbitrarily* connected; the network does not have a particular topology (e.g spanning tree, hypercube etc.).
2. The network is *connectionless*. There is no reservation of network bandwidth or resources between a source of data transmission and its intended destination. End-to-end "connections" may be set up by the Transport layers in the source and destination, but the other nodes within the network are oblivious to this.
3. The network is *packet-switched*. Packets are individually routed from source to destination. They may take different routes, and may arrive at the destination out-of-order.
4. Packets are routed from one link to the next via nodes, often known as *routers*, *packet switches* or *gateways*. Nodes usually buffer incoming packets before they are transmitted on an outgoing link. This is described in greater detail in the next section.
5. The network uses *semi-static routing*. Routes change slowly over time. Most route changes are performed manually, and most routes are static.
6. The network sets *no bandwidth constraints* on data sources. A host may attempt to transmit data at a rate which will exceed the bandwidth or resources on the links and nodes between it and the data's destination.
7. The network makes *no transmission guarantee*. The links on the network have finite bandwidth, and the nodes have finite buffer space for packets waiting to be switched. If network components are unable to forward data on to the destination, for any reason, the data may be discarded.
8. The network is an *internetwork*. The nodes in the network are connected by links that vary greatly in type. There may be large variations in:
    - The physical characteristics of the links,
    - The Medium Access Control mechanism of the links,
    - The bit rate on the links,
    - The bit error rate on the links, and
    - The packets' size on the links.

1.2 Routers – Bottlenecks in the Network

All the links on a network are joined together by *routers*. These forward packets arriving on incoming links to the appropriate outgoing links, to ensure that the packet is routed to its intended destination.

Incoming packets are buffered in an Input Buffer. Once in the buffer, they are selected by a Packet Selection Function to be passed to a Routing Function. The Routing Function determines on to which outgoing link a packet must be placed for it to get closer to its destination; this is done with a routing table which, as described earlier, is semi-static. When the correct link for the packet is found, the Routing Function passes the packet to a Packet Queueing Function for queueing on the output buffer for the link. When the packet reaches the other end of the queue, it is transmitted over the link to the next router, or to the final destination.

There are two fundamental bottlenecks in the given network architecture. Firstly, there is a minimum time needed for the router to decode the network header of the incoming packet, determine the route for the packet, and pass the packet to an outgoing link for transmission. Secondly, there is a delay for the packet to be transmitted on the outgoing link; this may just be the packet's transmission time for a full-duplex link, or there may be an extra delay for the link to become clear on a half-duplex link.

The second bottleneck indicates that the router must be prepared to buffer output packets, to prevent them from being lost while the router waits for an outgoing link to become clear. The two bottlenecks together indicate that the router must buffer incoming packets, to prevent them from being lost if they arrive too quickly for the router to process.

By definition, the router's buffers are finite. If a buffer becomes full, then no more packets can be queued in the buffer, and the router has no choice but to discard further packets. This causes data loss in the data flow between a source and destination, and usually causes the source to retransmit the data.

Although a router cannot queue a packet if the corresponding output buffer is full, it can choose either to discard the unqueued packet, or to discard a packet already in the output queue, and then queue the unqueued packet. This choice is performed by the Packet Queueing Function.

1.3 Congestion in Connectionless Packet-switched Networks

Given the above architecture, it is possible to see how network congestion can occur. A source of data flow on the network cannot reserve bandwidth across the network to its data's destination. It therefore is unable to determine directly what rate of data flow can be sustained between it and the destination.

If a source transmits data at too high a rate that can be sustained between it and the destination, one or more routers will begin to queue the packets in their buffers. If the queueing continues, the buffers will become full and packets from the source will be discarded, causing data loss and, if the source is attempting to guarantee transmission reliability, retransmission of data and increased transmission time between the source and the destination. Figure 1 from [Jain & Ramakrishnan 88] demonstrates the problem of congestion.

As the load (rate of data transmitted) through the network increases, the throughput (rate of data reaching the destination) increases linearly. However, as the load reaches the network's capacity, the buffers in the routers begin to fill. This increases the response time (time for data to traverse the network between source and destination) and lowers the throughput.

As the routers' buffers begin to overflow, packet loss tends to 100%, response time approaches infinity and the throughput approaches zero; this is the *point of congestion collapse*. This point is known as the *cliff* due to the extreme drop in throughput. Figure 1 also shows a plot of power, defined as the ratio of throughput to response time. The power peaks at the *knee* of the figure.

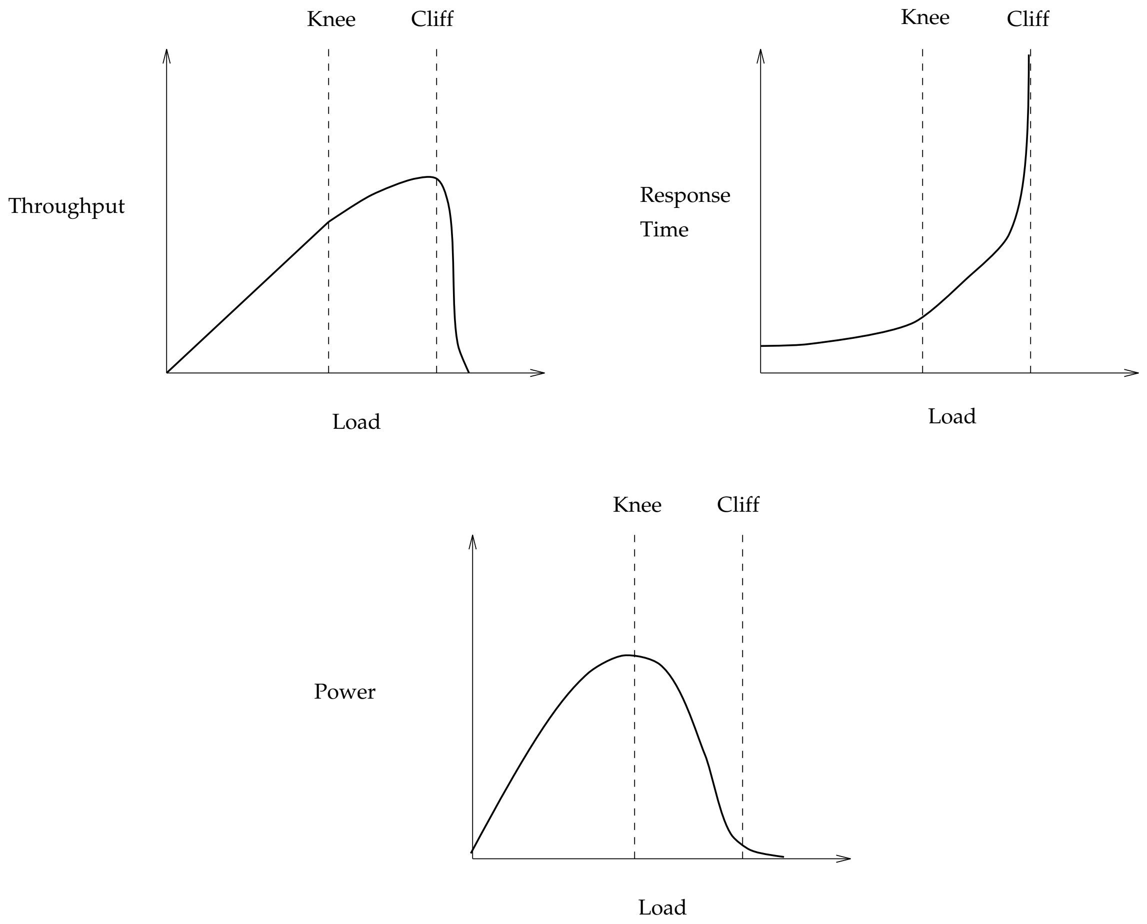

Figure 1: Network Performance as a function of Load

2 Problems with Existing Congestion Control Schemes

There are several congestion control schemes in use and proposed for networks with Internet-like architectures. There is not space in this technical report to give a complete survey of these schemes; however, a few of the main examples have been selected for demonstration.

Most of the congestion control schemes can be classified as one or a combination of the following methods:

- A transport layer flow control to prevent a source of packets from congesting the network; an example is the sliding window mechanisms in TCP.
- Having congested routers signal other routers or the source of packets about the congestion, so that appropriate action can be taken; an example is the Source Quench and DecBit schemes.
- Heuristics in the Packet Queueing Function that determine which packet is transmitted next, and which packet already in the output queue to drop, if necessary; an example is the Fair Queueing scheme.

Let us look at these examples.

2.1 Transport Layer Design

The Transport layer, as the basic generator of a traffic flow, is the essential "source" of the traffic flow. Unfortunately, while it is responsible for the flow of traffic that may cause network congestion, it is divorced from the Network layer, and is unable to directly see the effect of its traffic flow on the intermediate nodes between it and the destination of the traffic.

The Transport layer is only able to determine that there is network congestion in two ways:

- By being informed of existing network congestion from the Network layer (as is done in the Source Quench mechanism described in Section 2.4).
- By observing the effects of congestion on the traffic flow: delayed arrival of acknowledgments, and acknowledgments that indicate the loss of data.

Delayed acknowledgments and loss of data do **not** specifically imply network congestion. They may be caused by packet corruption and temporary traffic instability due to new flows of data being created. Therefore, transport protocols that rely on the second method to deduce the existence of congestion may not detect congestion when it exists, and may believe that congestion exists when it doesn't.

Transport protocols which perform end-to-end reliability must retransmit data when it is lost in transit. In general, retransmissions are prompted either by the arrival of a negative acknowledgment, or the expiry of a timer set to detect the non-arrival of any acknowledgment. Neither of these events specifically imply the loss of data. Transport protocols which use these events to prompt data retransmission may retransmit data unnecessarily, burdening the network with redundant transmissions and increasing the chance of congestion.

TCP, as a transport layer which has all of the above properties, also has all of the deficiencies described above. Two of its largest problems are the use of a Go-Back-N sliding window and use of a retransmission scheme based on round-trip time.

2.2 TCP's Sliding Window

TCP does not implement any congestion control or congestion avoidance schemes; instead it uses a Go-Back-N sliding window for end-to-end flow control [Postel 81]. TCP's window mechanism has been found to be fraught with several problems and deficiencies. There have been several proposals that remedy these problems, such as the "Silly Window Syndrome" [Clark 82], "Small Packet Avoidance" [Nagle 84] and "Slow Start" [Jacobson 88]. Nearly all implementations have these extensions, and TCP's window operation is now well-behaved. However, TCP's sliding window management is now anything but elegant.

Another problem with a Go-Back-N sliding window mechanism is that if data at point N is lost, the source must retransmit **all** data from N onwards, even is the destination has already received much of this data. This causes unnecessary network traffic.

2.3 Retransmission Timer Deficiencies

TCP uses the expiry of a timer set to detect the non-arrival of an acknowledgment to prompt data retransmission. The dilemma here is that if the timer is set too small, it expires before an acknowledgment can arrive, and data is retransmitted unnecessarily. If it is set too large, the protocol reacts too slowly to the loss of data, reducing the rate of data transmission and throughput.

The acknowledgment for transmitted data should return at the round-trip time. Collective experience has shown that the round-trip time is anything but constant, and it varies due to the load on the network and short- and long-term congestion within the network. Estimating the round-trip time has been shown to be very difficult.

The original TCP round-trip time estimation scheme was simple and worked well when the Arpanet was small. As the network size increased, round-trip time variance increased and the estimation scheme could not cope with this high variance. Jacobson [Jacobson 88] proposed a new round-trip time estimation scheme which could cope with high round-trip time variance, and this is now the recommended scheme for TCP.

Karn [Karn & Partridge 87] added to Jacobson's work by showing an ambiguity in round-trip time estimation due to acknowledgments for packets that have been retransmitted. His algorithm for removing the ambiguity from round-trip time estimates is found in all major implementations of TCP.

Despite these solutions, transport protocols like TCP are unable to accurately deduce the round-trip time in conditions when it has large variance. This leads to network congestion and/or poor throughput and can cause further round-trip time variance.

2.4 ICMP Source Quench

The ICMP Source Quench is the only congestion control mechanism in the Network layer of the TCP/IP protocol suite. Both routers and hosts play a part in the mechanism to control congestion. When a router believes itself to be congested, it sends 'source quench' packets back to the source of the packets causing the congestion. On receipt of these packets, the host will throttle its data rate so as to prevent router congestion. It can then slowly increase its data rate, until source quench packets begin to arrive again.

Source Quench's effectiveness as a congestion control mechanism is flawed due to two problems. Firstly, it is not clearly stated as to how a router or destination decides when it is congested, who to send a source quench message to, how often to send source quench messages, and when to stop. Secondly, it is not clearly stated as to how a host should react to a source quench message, by how much it should reduce its output rate, if it should inform upper layers in the network stack, and how to properly increase its output rate in the absence of source quench messages.

Even worse, a router or destination may be congested and not send any source quench messages. Sources in receipt of a source quench message cannot determine if a router or destination dropped the datagram that caused the source quench message. The source does not know to what rate it should reduce its transmission to prevent further source quench messages.

Since the original specification of the Source Quench mechanism, several further papers have been produced to improve the performance of the mechanism, and to tighten the mechanism's specification ([Nagle 84] and [Braden & Postel 87]). One serious problem, raised by Nagle in [Nagle 84], is that sources who receive source quench messages may not inform the corresponding Transport layer of the source quench. This effectively renders Source Quench useless in some instances.

Some of the deficiencies in Source Quench were recognised in RFC 1009 – Requirements for Internet Gateways [Braden & Postel 87], and while it mandated certain parts of the Source Quench mechanism, it still left others non-mandatory, with suggested behaviour given. The document notes that "... the Source Quench mechanism is known to be an imperfect means for Internet congestion control, and research towards more effective means is in progress ...".

2.5 The Dec Bit Mechanism

The Dec Bit mechanism, also known as the Congestion Indication mechanism, is a congestion avoidance mechanism developed for the Digital Network Architecture at DEC [Jain *et al* 87], and has since been specified as the congestion avoidance mechanism for the ISO TP4 and CLNP transport and network protocols.

In Dec Bit, all network packets have a single bit, the 'Congestion Experienced Bit', in their headers. Sources set this bit to zero. If the packet passes through a router that believes itself to be congested, it sets the bit to a one. Acknowledgment packets from the destination return the received Congestion Experienced Bit to the source. If the bit is set, the source knows there was some congestion along the path to the destination, and takes remedial action. In DECNET, a source reacts to the Congestion Experienced Bit by either increasing its window size by one (if not set), or decreasing the window size by 12.5% (if set). Once the window size changes, new Congestion Experienced Bits are ignored for one round-trip's time, to allow the traffic's effect on the network to be monitored and returned to the source via another Congestion Experienced Bit.

Jain et. al show that Dec Bit is robust and brings the network to a stable operating point with minimal oscillations in sources' window size.

3 TRUMP and RBCC

Because of the problems with window-based flow control schemes, and with the limited amount of congestion information available from single-bit congestion schemes, I was motivated to develop a congestion control scheme that returned appropriate rate information to the source of a traffic flow, and a matching transport protocol which could use this information. During the development of these, I was guided by the following design decisions:

Dropping packets is highly undesirable. A router which is dropping packets is operating in congestion recovery mode, well past the optimum point of operation [Jain 90]. A congestion control scheme should emphasise congestion avoidance, but also provide congestion recovery.

Router queue sizes $>$1 are highly undesirable. Jain [Jain 90] shows that an average router queue length of 1 packet is the optimum value. A congestion scheme should strive to keep queue lengths at length 1.

Use of round-trip timers is undesirable. Round-trip time estimation is complex and difficult to get right. A new transport protocol should avoid the use of round-trip timers if possible.

Use of windows for flow control is undesirable. Window update schemes are also complex and difficult to get right. Poor window updates can also lead to bursty traffic, undesirable both from the point of view of congestion and time-sensitive network traffic. Limited window sizes can also lead to poor utilisation in high-speed networks with large latencies.

Selective retransmission is preferred to Go-back-N error recovery. Go-back-N error recovery will retransmit packets that have been successfully received by the destination. A much better retransmission scheme is selective retransmission.

Separation of error control and flow control is desirable. Packet loss should not be tightly associated with congestion or flow control in a transport protocol. Packets are lost for many other reasons than congestion. A transport protocol should use packet loss to prompt packet retransmission and nothing else.

Transport flow control should use congestion information from routers. Bit-setting techniques help a source identify the presence of congestion, but do not indicate the extent of congestion. Routers should feed back more than one bit of congestion information to transport protocols.

Following these design decisions, I have developed RBCC, a rate-based congestion control scheme, and TRUMP, a rate-based unicast message transport protocol. TRUMP and RBCC have the following characteristics:

1. TRUMP uses a rate-based flow control scheme, using rate information provided by RBCC in routers. TRUMP uses a selective retransmission scheme for error control. The flow control and error control mechanisms are orthogonal.
2. TRUMP admits packets into the network uniformly spaced in time; this minimises short-term congestion due to bursty traffic. Routers attempt to preserve the time spacing of packets in a traffic flow.
3. TRUMP advertises a desired data rate to RBCC. This rate may be infinite (e.g for FTP traffic) or finite (e.g for audio or telnet traffic). RBCC will never allocate more bandwidth than the desired rate.
4. TRUMP does not measure or use round-trip times. A single timer is used to prompt the transmission of the next data packet. The timer is set solely from the suggested data rate provided by RBCC.
5. In the router, the RBCC congestion control scheme calculates the sustainable data rate across the network for each traffic flow. This sustainable rate is passed to the TRUMP instantiation which is the source of the the traffic flow.
6. When performing rate calculations, RBCC partitions the available resources fairly amongst traffic flows, in both uncongested and congested operating regimes.

7. Routers may implement other congestion control mechanisms such as queue reordering and packet dropping schemes to complement RBCC. Routers should drop packets fairly when necessary.

3.1 Congestion Information in Packets

Each packet which traverses the network contains a number of fields which are used by RBCC to set the bandwidth available to the source. In the direction of flow from source to destination (e.g in data packets), three fields are used:

Desired_rate	The bandwidth which the source of this traffic desires; it can be ∞.
Rate	The bandwidth which has been allocated to this flow of traffic by a router.
Bottle_node	The router which set the value of the **Rate** field in the packet.

When a source transmits a packet, it initialises the **Desired_rate** field to the bandwidth which it desires for the flow of traffic; this may be ∞. The **Rate** field is also initialised to this value, and the **Bottle_node** field is set to the identify the source.

As the packet passes through a router, RBCC may lower the value of the **Rate** field if it has allocated less bandwidth to the traffic flow. If it alters the **Rate** field, the **Bottle_node** field is set to identify the router. When the packet reaches its destination, the **Rate** field holds the lowest bandwidth allocated by the routers along the packet's path, and the **Bottle_node** field identifies the node that set this lowest rate.

In the direction of flow from destination to source (e.g in acknowledgment packets), two fields are used:

Return_Rate	The value of the last **Rate** field which reached the destination.
Return_Bottle_node	The value of the last **Bottle_node** field which reached the destination.

These fields are not altered as the packet travels back to the source. Thus, when the source receives the packet, it obtains a measure of the lowest bandwidth available to it for the flow of traffic. It can then adjust its rate to not exceed this bandwidth, so as to minimise the congestion that the traffic flow causes to the network.

Note that acknowledgment packets carry not only the two 'return' fields but also the three fields **Desired_rate**, **Rate** and **Bottle_node**. This is because a bi-directional traffic flow (data and acknowledgment packets) require different amounts of bandwidth and in different directions. Data packets carry the two 'return' fields for the same reason.

3.2 Information Kept in Routers – 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 bandwidth 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 **ABT**. Each entry in the table has the following fields:

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

Desired_rate, the bandwidth which the source of this traffic desires; it can be ∞.

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

The next section describes how the packet fields and the Allocated Bandwidth Tables are used by TRUMP and RBCC.

4 TRUMP/RBCC Operation

As discussed above, the proposed congestion control mechanism is divided into the parts in the source and destination (TRUMP flow control) and in each router (RBCC). Together, these use the congestion information in each packet and in each Allocated Bandwidth Table to control congestion.

I will examine each part of TRUMP and RBCC in turn, describing its operation in detail. I will introduce some pseudo-code, which is based heavily on the RBCC code implemented in the REAL simulator. The pseudo-code will have a C syntax.

4.1 Source Transmission

TRUMP, as a transport protocol, provides not only flow-control, but error correction via packet retransmission and selection of the type of connection required between the source and destination. This report will concentrate on the flow-control aspects of TRUMP. For more details on TRUMP, the reader should refer to [Toomey 93] (which is slightly dated), or contact me for a more up-to-date description of the protocol.

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 current implementation of TRUMP always starts with a two-way handshake:

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 current implementation, S is 1 second and T is 30 seconds.

During data transmission, TRUMP transmits data packets spaced by an inter-packet delay, whose value is calculated as follows:

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

where $rate\ in\ bits\ per\ second$ is obtained from the **Return_Rate** field from either the connection setup acknowledgment packet or the last data acknowledgment packet. A single timer set to this delay prompts the source to transmit the next data packet.

With an initial inter-packet delay set, TRUMP performs the following code:

```
while (there are still packets left to transmit) {

    make_pkt(pkt);
                                   /* Initialise many packet */
                                   /* fields including */
    pkt->seq_no = s++;             /* the sequence number, */
    pkt->desired_rate= pkt->rate=  /* the desired rate and */
                      infinity();  /* initial rate values, */
    pkt->bottlenode = node;        /* the bottlenode */

                                   /* Initialise fields to */
                                   /* hold information about */
                                   /* the ack flow, such as */
    pkt->rtn_rate = ack_rate;      /* the measured ack rate */
    pkt->rtn_bottlenode=           /* and the ack bottleneck */
```

```
                        ack_bottlenode;
      send_pkt(pkt);                  /* Send the packet off */
      set_timer(delay);               /* Perform inter-pkt delay */
  }
```

4.2 Packet Routing

The packet has now entered the network. To reach its destination, it must be forwarded by one or more routers. When the packet reaches a router, the router performs the following operations:

1. Determine the output interface on which the packet must be retransmitted.
2. Decide to update any Allocated Bandwidth Tables based on the values of the five congestion fields in the packet.
3. Decide to change the **Rate** and **Bottlenode** fields in the packet based on the information in the output interface's Allocated Bandwidth Table.
4. Queue the packet on the output interface, or drop the packet if there is no room in the output queue[1].

Operations 1 and 4 must be done in any router. Operations 2 and 3 are part of the RBCC scheme. To these two we can add a third operation, the actual update of the Allocated Bandwidth Tables. Let us look at each of these three operations in turn.

4.3 Deciding to Update the ABTs

The router's Allocated Bandwidth Tables do not need to be updated on every packet arrival. For every packet, 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 opposite direction (i.e destination to source). RBCC updates 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 **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 packet/ABT **Bottle_node** fields are the same, and the packet/ABT **Rate** fields differ. In this case, an upstream bottleneck has changed its allocated bandwidth to the traffic flow, and this router may need to update its own Allocated Bandwidth Table.
- The packet **Rate** field is lower than the ABT **Rate** field. An upstream router has lowered the traffic flow's bandwidth, and this router needs to update its ABT to reflect this.

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 bandwidth, and this router needs to update its ABT to reflect this.
- The packet/ABT **Bottle_node** fields are the same, and the packet **Return_Rate** field differs from the ABT **Rate** field. The flow's bottleneck has altered the allocated bandwidth since a forward packet for the traffic flow was routed. The packet's **Return_Rate** field can thus be changed to be equal to the ABT **Rate** field.

[1] This is first-come first-served queueing. Other alternatives drop a packet already in the queue and queue the latest packet. I have not tested any of these alternatives as yet.

Here is the pseudo-code for the above conditions:

```
c is a pointer to the flow's entry in the ABT;
if (this is a new traffic flow) {
  add an entry to the forward ABT, make c point to the new entry;
  c->desired_rate= pkt->desired_rate;

  /* Prevent infinite rate values in the ABT */
  if (pkt->rate== infinity()) {
        c->rate = 0.0; c->bottlenode = node;
  } else { c->rate = pkt->rate; c->bottlenode = pkt->bottlenode; }
  recalc_rate_table();
}

if (pkt->desired_rate != c->desired_rate) {
  c->desired_rate = pkt->desired_rate;
  recalc_rate_table();
}

if ((pkt->bottlenode == c->bottlenode) && (pkt->rate != c->rate)) {
  c->rate = pkt->rate;
  recalc_rate_table();
}

if (pkt->rate < c->rate) {
  c->rate = pkt->rate;
  c->bottlenode = pkt->bottlenode;
  recalc_rate_table(node, qnum, line_speeds[node][dest]);
}

/* Now test the reverse ABT conditions */
if (pkt->rtn_rate < rev_c->rate or
    (pkt->rtn_bottlenode==rev_c->bottlenode and
                        pkt->rtn_rate != rev_c->rate)) {
  rev_c->rate = pkt->rtn_rate;
  rev_c->bottlenode = pkt->rtn_bottlenode;
  recalc_rate_table();
}
```

4.4 Changing the Packet's Rate Field

There are two instances when a router can change a packet's **Rate** and **Bottle_node** fields. If the router has allocated less bandwidth 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 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). Here is the pseudo-code for these two operations:

```
/* Now update the rate field if necessary) */
if ((pkt->rate > c->rate) || (pkt->bottlenode== node)) {
  pkt->rate = c->rate;
  pkt->bottlenode = c->bottlenode;
}
```

4.5 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 bandwidth amongst the flows in the ABT must be fair.
- A traffic flow's bandwidth 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 bandwidth allocation cannot be set to zero or to ∞.
- The sum of allocated bandwidths in the ABT must not exceed the output interface's bandwidth.

The ABT update algorithm used by RBCC is:

The ABT update algorithm used by RBCC is:

1. If there are no entries in the ABT, there is nothing to do. Stop.
2. If all the traffic flows in the ABT are bottlenecked by this router, then
3. Update the **Rate** entries by dividing the output interface's bandwidth evenly across all flows, without exceeding any flow's desired rate. Stop.
4. Otherwise, calculate H, the highest rate in the ABT for a traffic flow not bottlenecked by this router.
5. Find S, the sum of all the rates in the ABT. For those flows bottlenecked by this router, substitute H for their rate. This ensures that the flows bottlenecked by this router are treated as fairly as other flows in steps 6 and 7.
6. Find the ratio $R = output\ bandwidth/S$.
7. If $R < 1$, then there is less bandwidth on the output interface than required by all flows. Mark all flows as being bottlenecked by this router, and perform step 3.
8. Otherwise, the output interface has excess capacity. Update some **Rate** entries by dividing the excess capacity evenly across all flows bottlenecked by this router, without exceeding any flow's desired rate. Stop.

The pseudo-code for the algorithm is below. Note that some of the fields in the ABT are updated by the update decision pseudo-code before this code is executed.

```
/* Firstly, calculate H, the highest rate allocated to a connection */
/* not bottlenecked by us. Also count the # of conns and the # of */
/* conns bottled by us */
for (H=0.0, c = head_conv_list; c; c = c->next) {
  conncnt++;
  if (c->rate > c->desired_rate) c->rate= c->desired_rate;
  if (c->bottlenode == node) bottlecnt++;
  else if ((c->rate != infinity()) && (c->rate > H)) H = c->rate;
}
if (conncnt==0) return;       /* nothing to do */

/* If H remains zero, divide bandwidth */
/* equally to all connections. */
/* They are all bottled by us */
/* The ABT list is ordered by desired rate */
if (H == 0.0) {
  for (c = head_conv_list; c; c = c->next) {
      excess= bw / (1.0 * conncnt);
      c->rate = min(c->desired_rate, excess);
                            /* Subtract allocation from bandwidth */
      bw -= c->rate; conncnt--;
   }
  return;
}

/* Find the sum of rates allocated by us. For connections which are */
/* bottlenecked by us, use the highest rate H above. */
for (sum = 0.0, c = head_conv_list; c; c = c->next) {
  if (c->bottlenode == node) c->rate = min(H, c->desired_rate);
  sum += c->rate;
}

/* Now calculate the ratio of available bw to desired bw */
ratio = bw / sum;

/* If ratio is below 1.0, there is not enough bandwidth */
/* so lower all the allocated bandwidths by the ratio. */
/* If ratio is greater than zero, divide the excess bandwidth */
/* amongst those connections bottlenecked by us. */
if (ratio <= 1.0)
  for (c = head_conv_list; c; c = c->next) {
     c->rate *= ratio; c->bottlenode = node;
  }
else {
  bw -= sum;                /* Leave the excess in bandwidth */
  for (c = head_conv_list; bottlecnt && c; c = c->next)
      if (c->bottlenode==node) {
        excess= bw / (1.0 * bottlecnt);
        c->rate = min(c->desired_rate, excess);
        bw -= c->rate; bottlecnt--;
     }
}
```

4.6 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, the packet reaches the destination. In the current implementation of TRUMP in REAL, the packet is converted into an acknowledgment packet, the values in the **Rate** and **Bottle_node** fields are copied into the **Return_Rate** and **Return_Bottle_node** fields, respectively; the packet's size is changed and the packet is returned to the source. Here is the pseudo-code:

```
pkt->type= ACK;
pkt->dest= pkt->source;
pkt->source= node;            /* We are now the packet's source */
                              /* Return the measured rate */
pkt->rtn_rate= pkt->rate;
pkt->rtn_bottlenode= pkt->bottlenode;

                              /* Guesstimate desired rate */
pkt->desired_rate= pkt->rate=
      pkt->rate * ack_size / pkt->size ;
pkt->bottlenode= node;
pkt->size= ack_size;

send_pkt(pkt);
```

Note in particular the method used to determine the desired rate for the flow of acknowledgments back to the source:

$$desired\ acknowledgment\ rate = measured\ rate * \frac{ack\ packet\ size}{data\ packet\ size}$$

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 data packet sizes. When I get selective retransmission working correctly in TRUMP, the ratio will have to be divided again by the number of data packets that are acknowledged in each acknowledgment packet.

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. I do not take into account this reverse situation at the moment.

I also do not separate the transmission of acknowledgment packets with any inter-packet delay at the moment; I trust that the spacing of data packets from the source will correctly space the acknowledgment packets from the destination. This is probably a glaring omission.

4.7 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 are reused as the **Return_Rate** and **Return_Bottle_node** fields for future data packet transmission.

The **Return_Rate** field from the acknowledgment packet is used to recalculate the inter-packet delay, using the same equation as before:

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

The new delay takes effect after the next packet transmission.

4.8 Connection 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 alleviate this problem, the TRUMP protocol has been designed to perform a two-way handshake at the end of a traffic flow, with *teardown* packets. RBCC as implemented detects these packets, removes the corresponding entry for the traffic flow from the ABT and then recalculates the table:

```
switch (pkt->type) {
  case TEARDOWN:
    delete connection;
    recalc_rate_table();
 }
```

A TRUMP source sends *teardown* packets every second once there are no packets left to send, and stops sending packets after it has received one *teardown* packet.

```
sendteardown is initially 1;
switch (pkt->type) {
  case TEARDOWN:
    sendteardown=0;
    break;
}

if (sendteardown) {
  /* We have successfully sent all our packets, */
  /* now send a teardown packet */
  make_pkt(pkt);
  pkt->seq_no= s;
  pkt->type= TEARDOWN;
  pkt->gen_time= runtime();
  pkt->gs= 0;
  pkt->desired_rate= pkt->rate= infinity(); /* Set initial rates */
  pkt->bottlenode= node;
  pkt->rtn_rate= ack_rate;
  pkt->rtn_bottlenode= ack_bottlenode;
  send_pkt(pkt);
  delay= 1.0e+6;              /* Send teardowns once a second */
}
```

Upon receipt of a *teardown* packet, a TRUMP destination immediately reflects it back to the source.

```
switch (pkt->type) {
  case TEARDOWN:
    /* send a teardown by changing the packet's params */
    pkt->dest = pkt->source;
    pkt->source = node;
    pkt->size = ack_size;
    send_pkt(pkt);
    break;
}
```

5 Comparing TRUMP/RBCC and TCP/IP's Congestion Control

TRUMP and RBCC have been implemented as transport and router mechanisms in the REAL network simulator [Keshav 88][Keshav 91]. This provides an opportunity to compare TRUMP/RBCC's congestion control with an existing method. This section describes a comparison of TCP Reno[2] and TRUMP/RBCC on a REAL network scenario which has an obvious congestion 'hot spot', as shown in Figure 2.

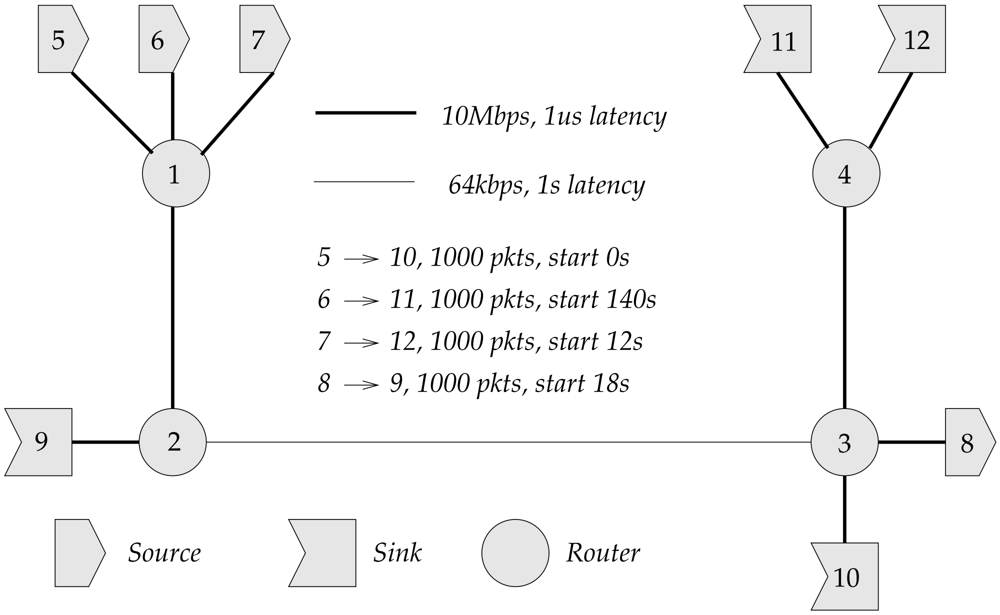

Figure 2: Scenario 7

The scenario shows a number of nodes connected by full-duplex links. All links have a data rate of 10Mbps, except for the link between routers 2 and 3, which is a 64kbps link. The 10Mbps links have a latency of 1 μsec, and the 64kbps link has a latency of 1 second.

There are four data flows, $5 \rightarrow 10$, $6 \rightarrow 11$, $7 \rightarrow 12$ and $8 \rightarrow 9$. All cross the 64kbps link, but all start at different times. Thus the link between routers 2 and 3 is the bottleneck in the scenario, and should cause congestion in those routers.

Note that all data flows have corresponding acknowledgment flows in the reverse direction. Note also that the data flow $8 \rightarrow 9$ is in the reverse direction to the other data flows.

The REAL input file containing Scenario 7 is given in Appendix A.

5.1 Optimum Rates for Scenario 7

Given Scenario 7, it is simple to compute the optimum rates for each of the sources; the calculations are given in Appendix B. The following table summarises the rates; the parameter t is the simulation time in seconds.

[2] TCP with the modifications described in [Clark 82], [Jacobson 88], [Karn & Partridge 87] and [Nagle 84].

Time t (seconds)	Optimum Rate (bps)			
	Source 5	Source 7	Source 8	Source 6
$0 <= t < 12$	64,000			
$12 <= t < 18$	32,000	32,000		
$18 <= t <= 140$	31,168.8	31,168.8	62,337.7	
$140 <= t <= Off_6$	20,779.2	20,779.2	62,337.7	20,779.2
$Off_6 <= t <= Off_8$	31,168.8	31,168.8	62,337.7	
$Off_8 <= t$	32,000	32,000		

In reality, no rate-based congestion scheme could predict these values perfectly. However, they provide a baseline to compare the rates calculated by the RBCC scheme.

5.2 Indicators of Network Congestion

There are several indicators of network congestion:

There are several indicators of network congestion:

- Packet loss as seen by a source,
- Increasing end-to-end delays and round-trip times as seen by destinations and sources respectively,
- High buffer occupancy in routers,
- Packet loss in routers, and
- A higher offered load to a link than its bandwidth capacity.

The first two indicators cannot be used by themselves as congestion indicators. Packet loss is caused by both network congestion and bit errors. Increasing round-trip time is caused by network congestion and lower available bandwidth to the source (competing sources). Window-based transport protocols such as TCP, however, do used these as congestion indicators. The last three indicators are reliable congestion indicators. We will look at all of these indicators and some other measured factors in the following sections.

5.3 Transmission Sequence Numbers

Packets are transmitted by a source in monotonically increasing sequence number, barring retransmissions, with the derivative of sequence number versus time giving the effective transmission rate of a source. Figure 3 show the sequence number for each packet transmitted by sources 5 and 6 using TCP and TRUMP.

TRUMP's sequence numbers rise smoothly, controlled by the rate values calculated by RBCC. TCP's rate is much more jagged, caused by its window size changes which reflect packets lost by the destination. Note in particular the early rate fluctuation for TCP source 5.

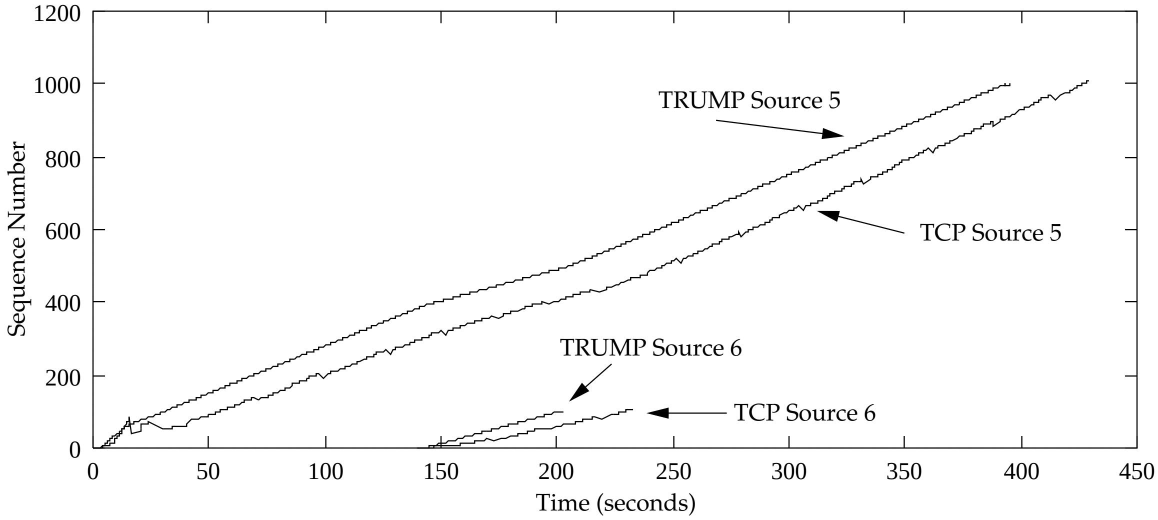

Figure 3: Sequence Numbers for Sources 5 and 6

The plot for sources 7 and 8, Figure 4, shows similar characteristics. TCP transmits packets for source 8 (the reverse data flow) at a much slower rate than TRUMP/RBCC. This is surprising, given the fact that in this scenario TRUMP/RBCC did not lose any packets whatsoever; there must have been adequate bandwidth for source 8 which was not used by TCP. For sources 5, 6 and 7, TCP settles down to a transmission rate which is similar to TRUMP's.

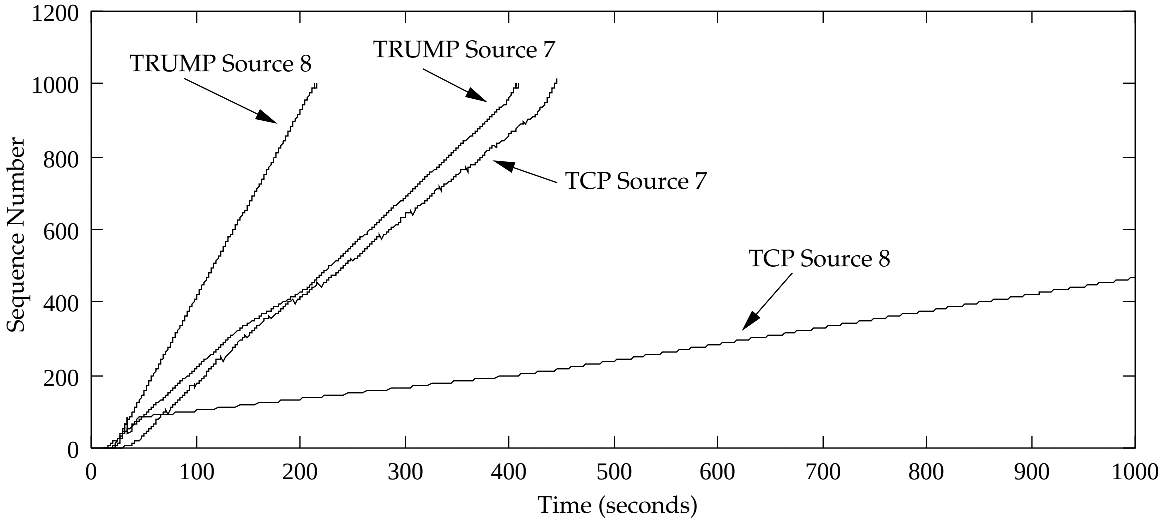

Figure 4: Sequence Numbers for Sources 7 and 8

5.4 Packet Loss

In Scenario 7, TRUMP sources do not lose any packets whatsoever. TCP sources, on the other hand, continually lose packets throughout the life of their transmission. Most packet losses are at the beginning of the data flow (until the optimum window size is reached), and TCP slowly loses packets after that as it tries to open its sliding window to take advantage of any new excess bandwidth. Figure 5 shows the packet losses by TCP.

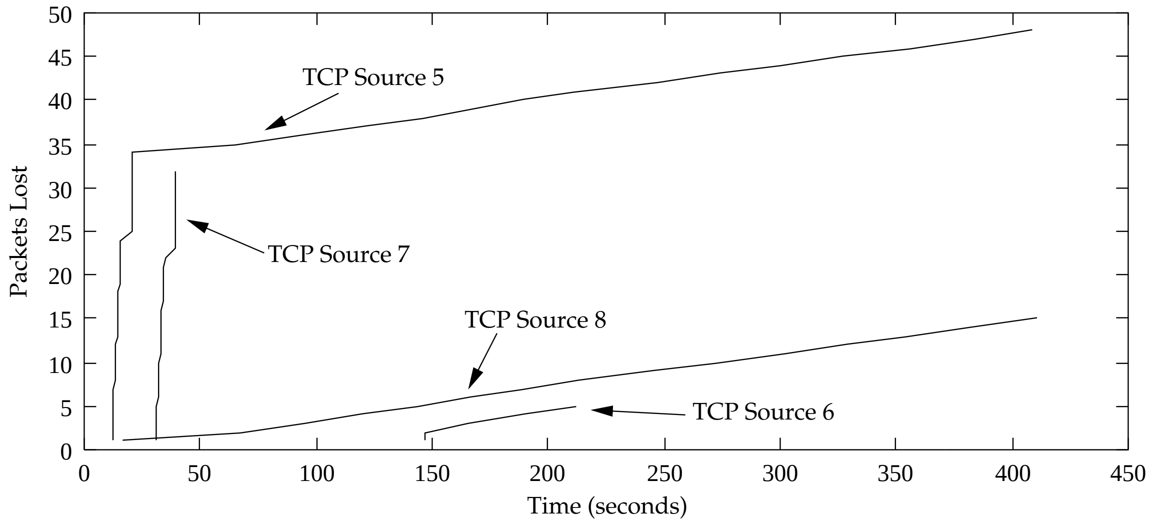

Figure 5: Packet Losses by TCP

5.5 Router Queue Lengths

One of the best indicators of network congestion is a router's average queue lengths for its output interfaces. If a queue has an average length (in packets) less than 1, that interface is underloaded. If a queue has an average length greater than 1, then it is receiving more packets than it can transmit, and is overloaded. A queue length of 1 indicates an optimally loaded interface. As a queue grows in length, packets suffer delays in retransmission at the router, increasing end-to-end delays and round-trip time. All queues are finite in length, and a router must drop packets when there is no space left to queue incoming packets.

In Scenario 7, routers 2 and 3 can only buffer 15,000 bytes (10 data packets). If the router cannot buffer the packet, it is immediately discarded (FCFS queueing and Drop Tail packet dropping). Figure 6 shows the queue lengths (in packets) for router 2.

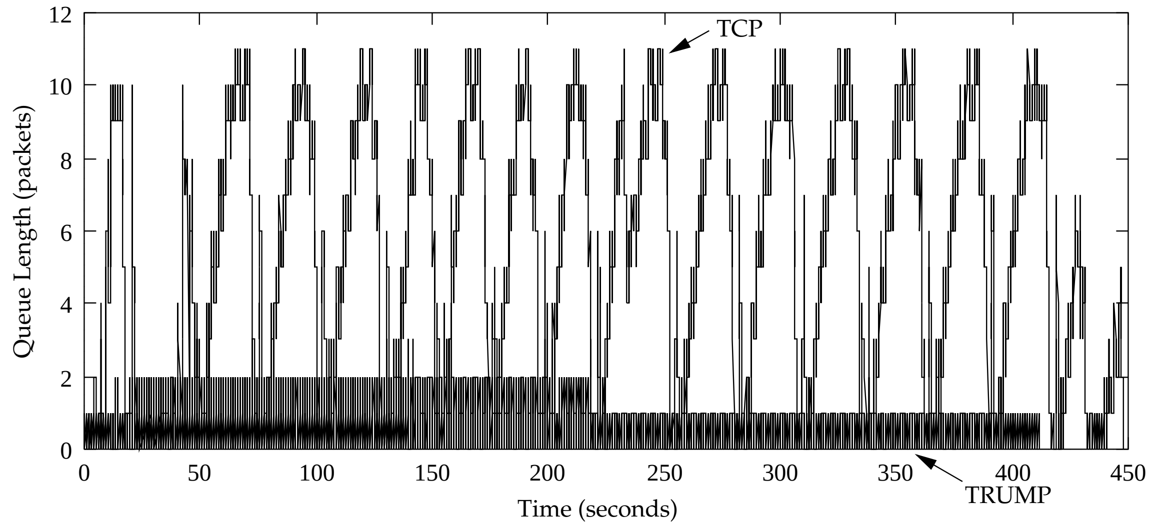

Figure 6: Average Queue Length in Router 2

This plot is cluttered. For router 2, there are several peaks in queue length where TCP's excessive rate has caused lost packets, due to its window resizing mechanism. TRUMP/RBCC nearly always keep the queue length for router 2 between 0 and 2 packets. The only time when the queue length exceeds 2 packets is when source 6 begins transmission at time 140, where the queue length reaches 3 packets.

5.6 End to End Delays

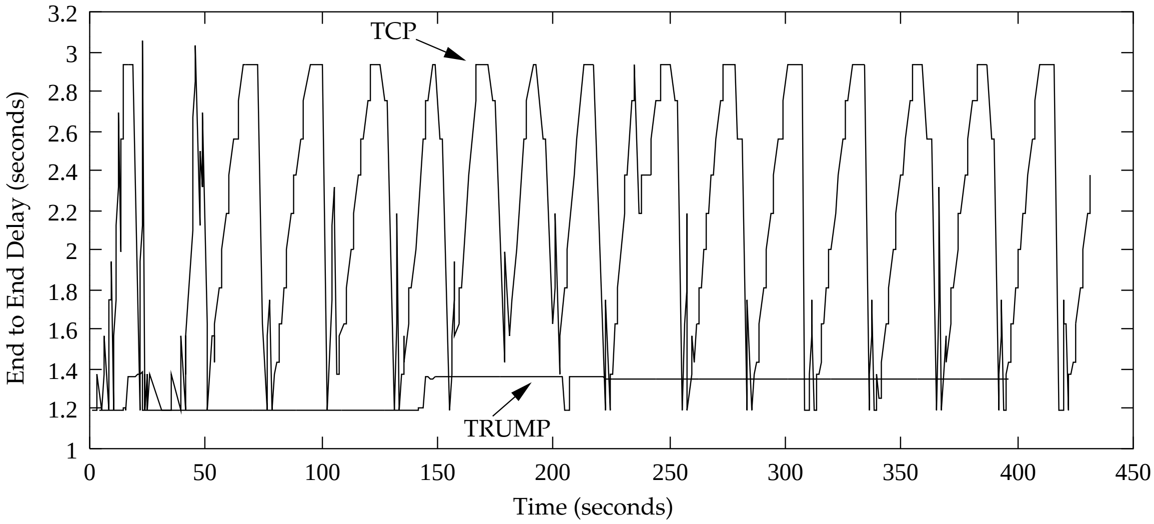

Figure 7: Source/Destination Delay between 5 and 10

In Scenario 7, the end-to-end delay for a packet from source 5 must be at least 1.188 seconds, given the link latencies and packet propagation time. Figure 7 shows the measured end-to-end delay of source 5 with both TCP and TRUMP. TRUMP has an end-to-end delay between 1.19 and 1.38 seconds; TCP's value oscillates between 1.19 and 3.06 seconds, due to the oscillation of the queue size in router 2. The end-to-end delays for sources 6 and 7 show the same behaviour. The low end-to-end delay variance exhibited by TRUMP/RBCC is particularly good for time-sensitive data such as real-time voice or video transmission.

5.7 Link Utilisations

Another indication of network congestion, closely tied to average queue lengths, is the utilisation of a link. Together with the queue lengths, the link utilisation shows whether an output link is underloaded or overloaded. Figures 8 and 9 show the utilisation of the link in the direction of router 2 to router 3 for TCP/IP and TRUMP/RBCC, respectively.

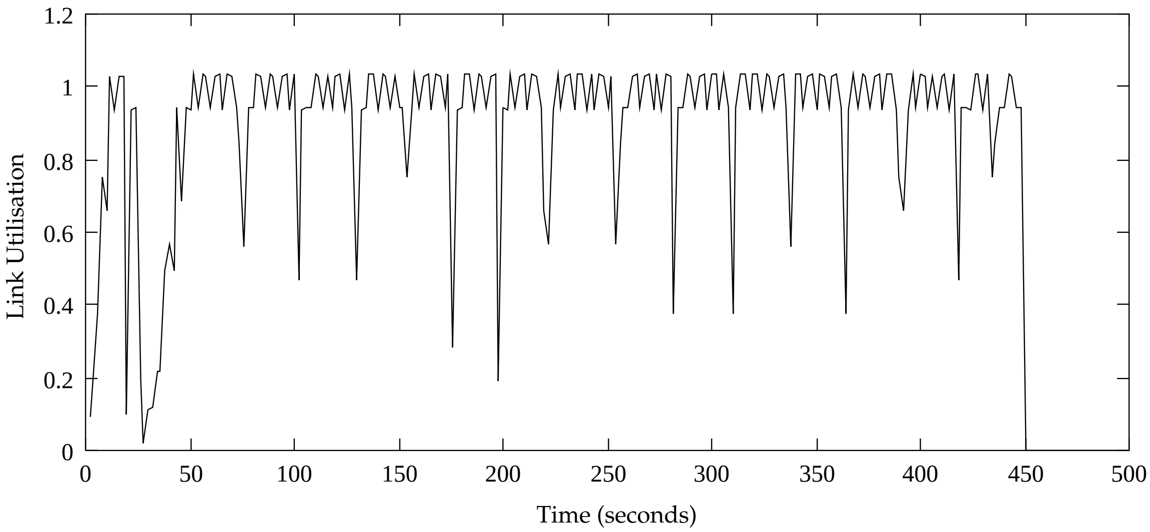

Figure 8: Link 2 $\rightarrow$ 3 Utilisation by TCP/IP

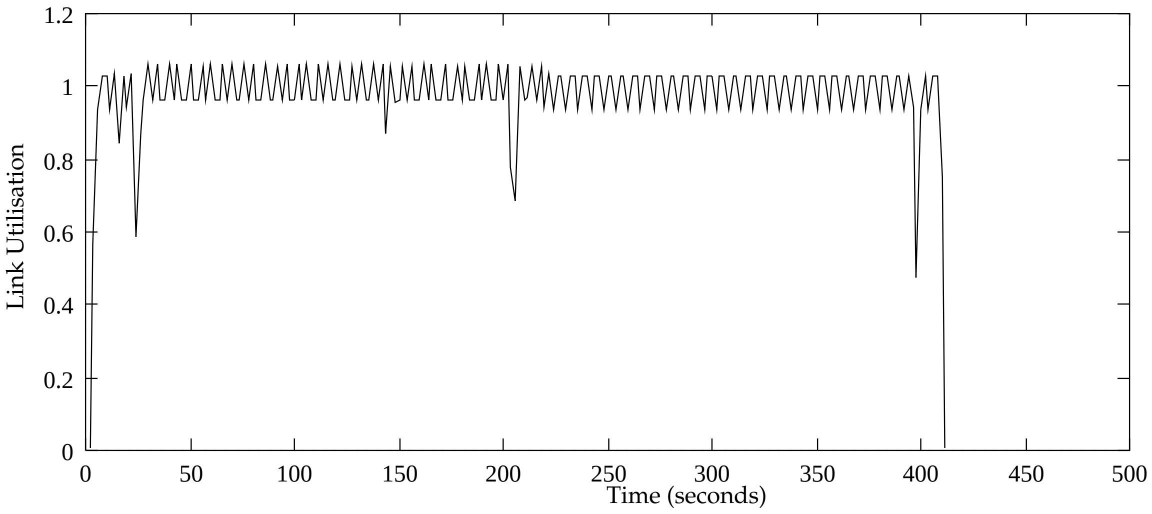

Figure 9: Link 2 → 3 Utilisation by TRUMP/RBCC

Both TRUMP and TCP/IP essentially fully utilise the link. TCP/IP shows occasional dips due to window shutdown, and TRUMP shows dips due to rate recalculations as sources start and stop.

Note that the data obtained by REAL for link utilisation is coarse. REAL periodically calculates the link utilisation by dividing the amount of data transmitted over the link by the time period. If the period is too small, the utilisation information is inaccurate, being sensitive to packet scheduling etc. If the period is too long, the utilisation value is more accurate, but is the average utilisation over a larger time period. I used a time period of 2 seconds for the utilisation measurements.

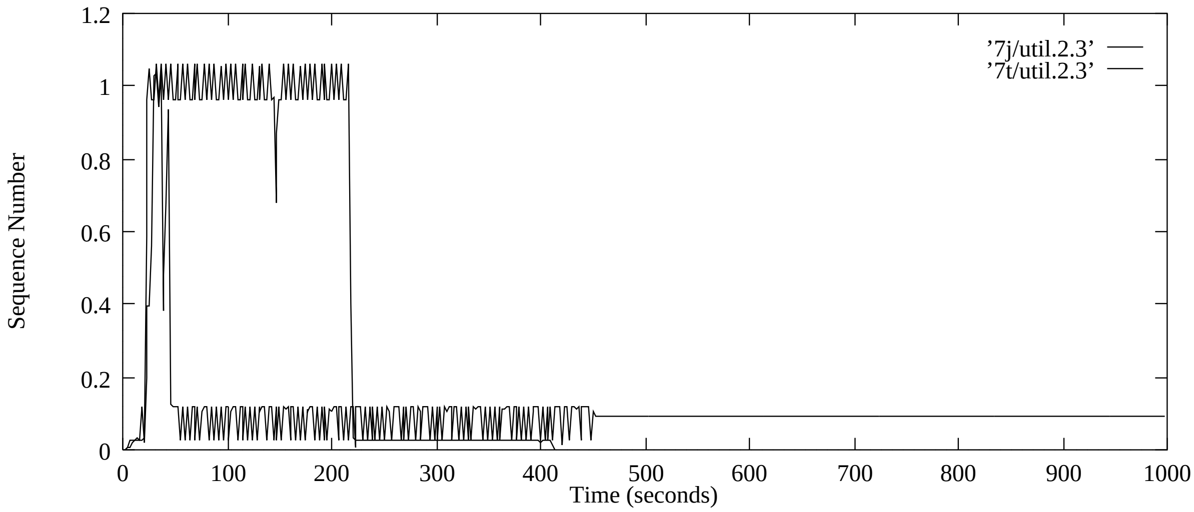

Figure 10: Link 3 → 2 Utilisation

The utilisation of the link in the direction of router 2 to router 3 shows that TRUMP fully utilises the link while source 8 is transmitting, with only acknowledgment traffic passing over the link when source 8 stops. TCP, on the other hand, never fully utilises the link with traffic from source 8.

5.8 Predicted and Measured TRUMP Transmission Rates

The optimum rates for TRUMP sources in Scenario 7 were given in Section 5.1. Figure 11 shows the actual rates returned from RBCC. As can be seen, the rates used by the TRUMP sources match nearly exactly with the optimal rates. There are slight delays due to round-trips which cause the TRUMP sources to adopt new rates after the optimal time. Initial bitrates are correctly found after TRUMP's two-way connection handshake. The close correspondence between the optimum rates and RBCC's calculated rates indicates that RBCC's rate calculation method is successful, and fair to all traffic sources.

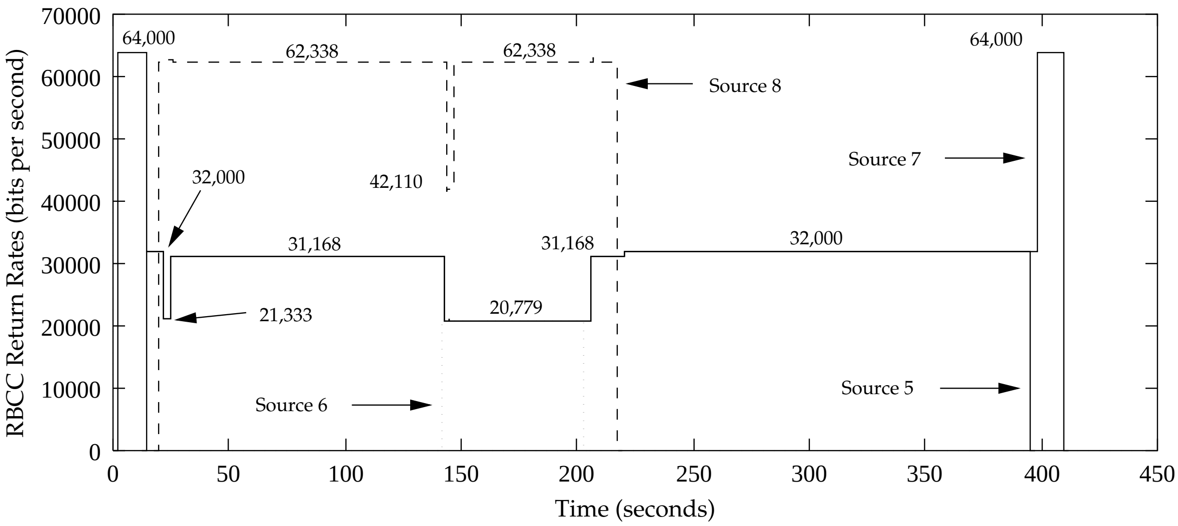

Figure 11: TRUMP's Rates for Scenario 7

6 Evaluation of Comparison Results

The comparison of TCP and TRUMP/RBCC in Scenario 7 shows that TRUMP/RBCC gives quite different traffic characteristics to TCP, with much less network congestion. RBCC determines source rates which will fully utilise output interfaces (if possible), while maintaining very low output queue sizes in routers. This has the effect of preventing long-term congestion, minimising buffering delays and gives end-to-end delays a very low variance, ideal for time-sensitive traffic. The rates calculated by RBCC are quite close to the optimum rates for Scenario 7.

7 Further Work and Conclusions

TRUMP/RBCC as a rate-based congestion control scheme appears to have a lot of promise. Further work with this scheme is required to ensure it is both general and stable. A comparison of TRUMP/RBCC against TCP-Vegas will be performed after RBCC is 'ported' to the *x*-kernel [Hutchinson & Peterson 91], TCP-Vegas is 'ported' to REAL, or both. TRUMP/RBCC needs to be tested on many more network scenarios. Finally, a study of the combination of TRUMP/RBCC and packet reordering schemes such as Fair Queueing [Demers *et al* 89] and/or packet dropping schemes such as Random Drop [Mankin 90] should be fruitful and insightful.

Flow control and congestion control schemes in packet-switched networks need not be window-based, nor need they return just the presence of congestion to traffic sources. This paper has demonstrated that a flow control and congestion control scheme that is rate-based can provide good congestion control, with beneficial side effects such as low packet loss, low router queue lengths, fair bandwidth allocation, and low end-to-end delays & delay variance. Moreover, the scheme allows sources to advertise a desired rate to the network, and this desired rate may be finite or infinite.

8 Acknowledgments

I would like to thank my supervisor and co-supervisor for their wonderful help in preparing this report.

A REAL NetLanguage for Scenario 7

Here are the most relevant parts of the REAL input file for Scenario 7, using TRUMP/RBCC as the congestion control method. I have removed some mandatory lines that control the underlying NEST simulator on which REAL is based.

```
real_params {
        scale_factor = 1.0;                     # Scale time to seconds
        ack_size = 40;                          # ACK size in octets
        buffer_size = 15000;                    # Output iface buf size (bytes)
        ftp_pkt_size = 1500;
        ftp_window = 600;                       # Ftp window size (units?)
        sch_policy = 1;                         # FCFS pkt scheduling
        util_time= 2.0;                         # Measure edge util every 2s
}
nodes{
        default{
                function = dumbtrump;           # Source is TRUMP, no retx
                start_time = 0,100;             # Start at 0 + 1 microsecond
                num_pkts = 1000;                # Send 1000 packets
                average = 1000; }               # Initial rate is 1000 bps
        node 1  { function = rbcc_router; plot=true; }
        node 2  { function = rbcc_router; plot=true; }
        node 3  { function = rbcc_router; plot=true; }
        node 4  { function = rbcc_router; plot=true; }
        node 9  { function = trsink;  plot=true; }
        node 10 { function = trsink; plot=true; }
        node 11 { function = trsink; plot=true; }
        node 12 { function = trsink; plot=true; }
        node 5  { dest = 10;}
        node 6  { start_time = 140,0; dest = 11; num_pkts= 100; }
        node 7  { start_time = 12,0; dest = 12;}
        node 8  { start_time = 18,0; dest = 9;}
}
edges{                            # Bandwidth in bps, latency in microseconds
        default { bandwidth = 10000000; latency = 1; }
        { 1 -> 2; }
        { 2 -> 3; bandwidth = 64000; latency = 1000000;}
        { 3 -> 4; }
        { 5 -> 1; }
        { 6 -> 1; }
        { 7 -> 1; }
        { 8 -> 3; }
        { 9 -> 2; }
        { 10 -> 3; }
        { 11 -> 4; }
        { 12 -> 4; }
}
```

The changes to use TCP/IP instead of TRUMP/RBCC are:

- Change the source from `dumbtrump` to `jk_reno`.
- Change the router from `rbcc_router` to `router`.
- Change the destination from `trsink` to `sink`.

B Theoretical Rate Calculation for Scenario 7

Regarding Scenario 7, the bottleneck in the system is the 64kbps full-duplex link 2 $\leftrightarrow$ 3. We will assume that all acknowledgment rates are limited by their respective data rates by

$$A_{y,x} = R_{ad} * D_{x,y},\ where\ R_{ad} = \frac{ack\ pkt\ size}{data\ pkt\ size} = \frac{40}{1,500} \tag{1}$$

where $D_{x,y}$ denotes a data flow from source x to destination$y$ and $A_{y,x}$ denotes the corresponding acknowledgment flow in the reverse direction.

At time $t = 0$, the flows $D_{5,10}$ and $A_{10,5}$ commence. The optimum rates for both are:

Time	Optimum Rate (bps)	
	$D_{5,10}$	$A_{10,5}$
$0 <= t$	64,000	1706.66

At time $t = 12$, the flows $D_{7,12}$ and $A_{12,7}$ commence. As the link 2 $\leftrightarrow$ 3 is the bottleneck, we can write

$$D_{5,10} + D_{7,12} \leq 64,000 \tag{2}$$
$$A_{10,5} + A_{12,7} \leq 64,000 \tag{3}$$

Obviously (2) is the limiting equation, and by sharing bandwidth equally to the flows, we obtain the rates:

Time	Optimum Rate (bps)			
	$D_{5,10}$	$A_{10,5}$	$D_{7,12}$	$A_{12,7}$
$0 <= t < 12$	64,000	1706.66		
$12 <= t$	32,000	853.33	32,000	853.33

At time $t = 18$, the flows $D_{8,9}$ and $A_{9,8}$ commence. Here, the flows are in the opposite directions to the previous four flows. We can write:

$$D_{5,10} + D_{7,12} + A_{9,8} \leq 64,000 \tag{4}$$
$$A_{10,5} + A_{12,7} + D_{8,9} \leq 64,000 \tag{5}$$

Distributing bandwidth fairly, we have $D_{5,10} = D_{7,12}$ and $A_{10,5} = A_{12,7}$. Assume both links are fully utilised and substituting for the acknowledgment rates using (1), we have:

$$2D_{5,10} + R_{ad} * D_{8,9} = 64,000 = 2R_{ad} * D_{5,10} + D_{8,9} \tag{6}$$
$$2D_{5,10}(1 - R_{ad}) = D_{8,9}(1 - R_{ad}) \tag{7}$$
$$2D_{5,10} = D_{8,9} \tag{8}$$

Substituting (8) into (6) gives $D_{5,10} = 31,168.8$, $D_{8,9} = 62,337.66$ and the table:

Time	Optimum Rate (bps)					
	$D_{5,10}$	$A_{10,5}$	$D_{7,12}$	$A_{12,7}$	$D_{8,9}$	$A_{9,8}$
$0 <= t < 12$	64,000	1706.66				
$12 <= t < 18$	32,000	853.33	32,000	853.33		
$18 <= t$	31,168.83	831.2	31,168.83	831.2	62,337.66	1,662.34

The six flows continue to transmit until time $t = 12$ when the flows $D_{6,11}$ and $A_{11,6}$ commence. Again, distributing bandwidth fairly, we have $D_{5,10} = D_{7,12} = D_{6,11}$ and $A_{10,5} = A_{12,7} = A_{11,6}$. Assume both links are fully utilised and substituting for the acknowledgment rates using (1), we have:

$$3D_{5,10} + R_{ad} * D_{8,9} = 64,000 = 3R_{ad} * D_{5,10} + D_{8,9} \tag{9}$$
$$3D_{5,10}(1 - R_{ad}) = D_{8,9}(1 - R_{ad}) \tag{10}$$
$$3D_{5,10} = D_{8,9} \tag{11}$$

Substituting (11) into (9) gives $D_{5,10} = 20,779.2$, $D_{8,9} = 62,337.66$ and the table:

Time	Optimum Rate (bps)							
	$D_{5,10}$	$A_{10,5}$	$D_{7,12}$	$A_{12,7}$	$D_{8,9}$	$A_{9,8}$	$D_{6,11}$	$A_{11,6}$
$0 <= t < 12$	64,000	1706.66						
$12 <= t < 18$	32,000	853.33	32,000	853.33				
$18 <= t <= On_6$	31,168.83	831.2	31,168.83	831.2	62,337.66	1,662.34		
$On_6 <= t$	20,779.2	554.1	20,779.2	554.1	62,337.66	1,662.34	20,779.2	554.1

As flow $D_{6,11}$ only has 100 packets to send, it finishes quickly, and the rates return to the values before flow $D_{6,11}$ started:

Time	Optimum Rate (bps)							
	$D_{5,10}$	$A_{10,5}$	$D_{7,12}$	$A_{12,7}$	$D_{8,9}$	$A_{9,8}$	$D_{6,11}$	$A_{11,6}$
$0 <= t < 12$	64,000	1706.66						
$12 <= t < 18$	32,000	853.33	32,000	853.33				
$18 <= t <= On_6$	31,168.83	831.2	31,168.83	831.2	62,337.66	1,662.34		
$On_6 <= t <= Off_6$	20,779.2	554.1	20,779.2	554.1	62,337.66	1,662.34	20,779.2	554.1
$Off_6 <= t$	31,168.83	831.2	31,168.83	831.2	62,337.66	1,662.34		

Similarly, flow $D_{8,9}$ has most of the bandwidth across 3 $\rightarrow$ 2, so it is the next to finish, and the rate return to the values before flow $D_{8,9}$ started:

Time	Optimum Rate (bps)							
	$D_{5,10}$	$A_{10,5}$	$D_{7,12}$	$A_{12,7}$	$D_{8,9}$	$A_{9,8}$	$D_{6,11}$	$A_{11,6}$
$0 <= t < 12$	64,000	1706.66						
$12 <= t < 18$	32,000	853.33	32,000	853.33				
$18 <= t <= On_6$	31,168.83	831.2	31,168.83	831.2	62,337.66	1,662.34		
$On_6 <= t <= Off_6$	20,779.2	554.1	20,779.2	554.1	62,337.66	1,662.34	20,779.2	554.1
$Off_6 <= t <= Off_8$	31,168.83	831.2	31,168.83	831.2	62,337.66	1,662.34		
$Off_8 <= t$	32,000	853.33	32,000	853.33				

The flows $D_{5,10}$ and $D_{7,12}$ continue at this rate until one or both finish.

C Deficiencies with RBCC

As with nearly any congestion scheme, RBCC provides several advantages and also several drawbacks. Here are the deficiencies of RBCC that I know of, with some comments on each.

Extra Packet Fields

RBCC as outlined needs five extra fields in every packet to pass the needed congestion information. Three of these hold rates and two hold machine identifiers.

The rates need to be unsigned fixed or floating point numbers with a range of, say, 9 powers of 10 (1bps to 1Tbps), an accuracy of, say, 0.1%, and an ability to represent the value ∞. This would require at least 16 bits per field. It probably would be better to use 32-bit IEEE 754 floating point numbers. The machine identifiers depend on the network layer used. The fields would be 32 bits in length for IPv6 and 128 bits in length for IPng. Overall, this gives an overhead of 20 octets for IPv4 and 44 octets for IPng, a header increase of 50% and 110 respectively.

Rate Table Size and Recalculation Expense

As RBCC keeps a per-source per-interface rate table in every router, this may be a serious use of memory when a router has hundreds or more flows of traffic through it. Even worse, the cost of recalculating a rate table may be extremely high, as the algorithm given is not simple and is probably **O(N)** where **N** is the number of traffic flows through the router.

One way of reducing the cost of rate table recalculation is by performing the recalculation periodically and only when it needs to be recalculated. For example, the period between recalculations may be 100 or 1,000 milliseconds. This lessens the cost of recalculation but will cause more short-term congestion within the network. Also, a good value for the period between recalculations may be difficult to determine, and is likely to depend on the traffic load on the router.

Different Forward/Reverse Routes

One of the assumptions that RBCC makes is that the route taken by packets from the source to the destination is the same as the route in the reverse direction. In real networks, this is not necessarily so. A different route will prevent the reverse packets from affecting the rate table. Further work will examine the affect on RBCC if reverse packets are not seen by an RBCC router. If RBCC performs poorly with the events missing, this will severely limit its use in real networks.

Connections with No Routers

Transport connections where traffic does not pass through a router (i.e where both source and destination are on the same subnet) cannot use RBCC for congestion control. This is a serious deficiency with RBCC.

No Quench Packets

RBCC currently does not have a mechanism for dealing with short-term congestion. It would be useful, for example, to return ‘Quench’ packets to the source so as to quickly lower the source’s rate, rather than waiting a round-trip for acknowledgment packets to return rate information to the source. This idea needs investigating.

Route Changes

RBCC can cope with the arrival of a new traffic flow through a router due to a route change, but not with the loss of a traffic flow through due to a route change. In the latter situation, the router will not detect the loss of the flow and will give less bandwidth to other flows than is available.

One possible solution to this inadequacy would be to use the information from any dynamic route protocol to infer when route changes have affected the list of traffic flows through the router.

D Obtaining TRUMP and RBCC

The REAL simulator additions and changes for TRUMP and RBCC are available via anonymous ftp at *ftp://minnie.cs.adfa.oz.au/pub/RBCC/trumprealpatch.tar.gz*. This technical report is also available via the World Wide Web at *http://minnie.cs.adfa.oz.au/Techreports/RBCC_96a/*.

References

[Braden & Postel 87] R. Braden and J. Postel. *Requirements for Internet Gateways*, RFC 1009, January 1987.

[Clark 82] D. D. Clark. *Window and Acknowledgement Strategy in TCP*, RFC 813, July 1982.

[Demers *et al* 89] A. Demers, S. Keshav, and S. Shenker. *Analysis and Simulation of a Fair Queueing Algorithm*. In *Proceedings, 1989 SIGCOMM Conference*, pages 1–12, September 1989.

[Hutchinson & Peterson 91] Norman C. Hutchinson and Larry L. Peterson. *The x-Kernel: An Architecture for Implementing Network Protocols*. *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1991.

[Jacobson 88] V. Jacobson. *Congestion Avoidance and Control*. In *Proceedings of the SIGCOMM '88 Symposium: Communications Architectures & Protocols*, pages 314–329, August 1988.

[Jain & Ramakrishnan 88] R. Jain and K. K. Ramakrishnan. *Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals and Methodology*. In *Proceedings of the 1988 Computer Networking Symposium*, pages 134–143, April 1988.

[Jain 90] R. Jain. *Congestion Control in Computer Networks: Issues and Trends*. *IEEE Network Magazine*, pages 24–30, May 1990.

[Jain *et al* 87] R. Jain, K. K. Ramkrishnan, and D. M. Chiu. *Congestion Avoidance in Computer Networks with a Connectionless Network Layer*. Technical Report TR-506, Digital Equipment Corporation, 1987.

[Karn & Partridge 87] P. Karn and C. Partridge. *Improving Round-Trip Time Estimates in Reliable Transport Protocols*. In *Proceedings of the ACM SIGCOMM'87*, pages 2–7, August 1987.

[Keshav 88] S. Keshav. *REAL: A Network Simulator*. Technical Report 88/472, Department of EECS, UC Berkeley, 1988.

[Keshav 91] S. Keshav. *Congestion Control in Computer Networks*. Unpublished PhD thesis, Department of EECS, UC Berkeley, August 1991.

[Mankin 90] A. Mankin. *Random Drop Congestion Processing*. In *Proceedings of the SIGCOMM '90 Symposium: Communications Architectures & Protocols*, pages 1–29, September 1990.

[Nagle 84] J. Nagle. *Congestion Control in IP/TCP Internetworks*, RFC 896, January 1984.

[Postel 81] J. B. Postel. *Transmission Control Protocol*, RFC 793, September 1981.

[Toomey 93] W. Toomey. *TRUMP – A Fast Reliable Transport Protocol for Distributed Systems*. Technical Report CS93/1, Department of Computer Science, University College, University of New South Wales, Canberra, 1993.