Deterministic Analysis of Flow and Congestion Control Policies in Virtual Circuits

Samar Singh [1]
Department of Computer Science and Engineering,
Indian Institute of Technology,
New Delhi - 110016, India.

Ashok K. Agrawala
Department of Computer Science,
University of Maryland,
College Park, MD 20742, USA.

Srinivasan Keshav
Computer Science Division, Department of EECS,
University of California, Berkeley,
Berkeley, CA 94720, USA.

ABSTRACT

Network congestion can be triggered by transient phenomena that are hard to study using classical stochastic approaches. We show that a deterministic analysis of virtual circuits (VC) allows us to precisely quantify these transients. We analyze some existing window and rate based flow control schemes, and propose a new protocol, 2P, that exploits the deterministic nature of a VC to provide optimal performance.

Our analysis focuses on two issues: utilization of available bandwidth and congestion avoidance. We study the transient behavior of the system when a sudden change occurs, such as loss of a packet or a reduction in the available bandwidth, and give precise expressions for the performance of these schemes. We show that 2P performs better than existing schemes since it adapts to changes in the state of the VC by keeping track of the effective transmission rate as well as the round-trip delay. This is done using a novel rate probing technique based on short bursts.

[1] Research partly supported by a UNDP travelling fellowship under the E&R Network program at IIT, Delhi.

1. Introduction

Recently there has been much interest in congestion avoidance and control in the Internet [Jaco88, KaPa87, Zhan86], as well as in other proprietary networks [Jain86]. A solution to this problem involves issues such as flow-control, packet retransmission, fair resource allocation and routing. Protocol specifications typically do not specify such issues and they are left open to the implementor. The focus of this paper is on flow and congestion control policies at the transport layer of the protocol hierarchy.

Many congestion control schemes have implicit models based on control theory [Jain88], hydrodynamics [Jaco88], queueing theory models [GeKl80] etc. Typically, the solutions obtained are for the steady-state case, and that too after making strong assumptions such as Poisson arrivals and the independence of sources. However, these models appear to be intractable when applied directly to congestion control. Congestion depends on the transient behavior of the network, and stochastic models do not deal adequately with transients. Therefore, schemes proposed in the literature are finally evaluated using simulation [Jain88, Jain89, DeKe89, Zhan89] or implementation [Jaco88]. The lack of a better theoretical approach to this problem is discussed in [BoPl88], which describes a model that does not give closed form solutions, but can be numerically evaluated.

Recent work has shown that networks of round-robin or fair queueing servers isolate VCs from one another, so that the service provided to one VC is partially decoupled from the service provided to any other VC [Hahn86, DeKe89, Morg89]. This suggests a deterministic modeling of the VC. Waclawsky and Agrawala have developed and analyzed such a deterministic model for studying the behavior of window protocols on a virtual circuit [WaAg89]. Other authors have also implicitly or explicitly used some deterministic analysis when discussing the performance of transport protocols [Nagl84, Clar82] (for example, [Jain89] explicitly models the virtual circuit as a series of D/D/1 queues). Thus, a deterministic approach seems well-suited to studying these issues and we use it here to study the behavior of protocols when sudden changes occur in the virtual circuit.

We precisely analyze several transmission policies proposed in the literature. Since we are concerned with congestion, we naturally restrict our attention to situations of high throughput; i.e. the sender wants to send data as fast as it possibly can. Our analysis indicates several deficiencies of present flow control policies. As an alternative, we propose 2P, a transmission control scheme that adapts its behavior by measuring

both the round-trip delay and the inter-arrival time of packets. We show that this scheme performs better than others we have analyzed. Although we still need to verify just how well this scheme performs under appropriate stochastic assumptions and further generalizations, we believe that the directions indicated by these analyses are significant. It also appears that our model can capture much of the reasoning implicit in the schemes in the literature.

The rest of this paper is divided into five sections. In the first, we set up the model and derive some basic results which help us simplify further analysis. In the second, we consider packet loss and retransmission. Next, we describe flow and congestion control schemes for some well known transport protocols, and present our own scheme, 2P. In the fourth section we use our model to analyze and compare the behavior and performance of the chosen schemes. The final section summarizes our results and shows directions for future work.

2. The basic model

We model a virtual circuit (VC) as a series of servers (routers or switches) connected by links. A packet is a point object that starts out from the source and traverses the links and servers until it reaches the destination. The time taken to traverse a link is zero, while the time taken to get service (at each server) is finite but deterministic. If the ith server is idle when a packet arrives, the time taken for service is s_i. If there are other packets from that VC at the server, the packet waits for its turn to get service (we assume a FCFS queueing discipline per VC). We assume a work-conserving discipline, which implies that a server will never be idle whenever it has a packet ready to be served.

To complete the picture, we assume another set of links and servers that constitute a return path from the destination back to the server. This is the path taken by acknowledgment packets (acks). We assume that every packet is acknowledged, so the destination is just another server, and the returning ack is modeled as the same packet looping back to the source. In the sequel, we will use this model of a VC and ignore acks unless specifically required.

This model is similar to the one described in [WaAg89], but it is more general and mathematically simpler.

Remark 1

Differences in the size of the packet and the ack can be ignored, since these are accounted for in the service times of the servers, which includes transmission time and processing time. The important point is that service time at any server is the same for all packets in the VC.

Remark 2

In this paper, we ignore protocols where not every packet is acknowledged (such as SNA). We will consider extensions of our models and results to cover this case in a subsequent paper.

We now describe the formal model, and then restate or derive some elementary results. For a more detailed analysis, the reader is referred to [WaAg89].

Notation

We use subscripts for the server number and superscripts for the packet number. Servers and packets are numbered from 1 onwards, while the sender is server 0.

a_i^j = arrival time of the j th packet at the i th server

s_i = service time of a packet at the i th server

q_i^j = waiting time of the j th packet at the i th server before getting service

d_i^j = departure time of the j th packet from the i th server

In our model, we assume zero delay on the links so $d_{i-i}^j = a_i^j$. We will work mostly with the departure times.

The basic relation governing the system is:

$$q_i^j = 0 \text{ if } a_i^j \geq d_i^{j-1} \text{ and,}$$

$$= d_i^{j-1} - a_i^j \text{ otherwise,}$$

which is better expressed as

$$q_i^j = \max(0, d_i^{j-1} - a_i^j)$$

$$= \max(0, d_i^{j-1} - d_{i-1}^j) \qquad (2.1)$$

This combines with the relation below to describe the system completely.

$$d_i^j = d_{i-1}^j + q_i^j + s_i \quad (2.2)$$

We now derive some simple relationships about inter-packet arrival and departure times that are fundamental to the analysis. While stating results and proofs, we will neglect to make proper use of rounding and truncation to integers, unless we that adds some insight.

Proposition 1

Suppose packets $j-1$ and j arrive at a server i with inter-arrival time s, and the $(j-1)$th packet finds the server free. Then, the inter-departure times of these packets is $\max(s, s_i)$.

Proof:

Refer to Figure 1. From (2.2) $d_i^j = a_i^j + q_i^j + s_i$, which with (2.1) becomes $= a_i^j + \max(0, (d_i^{j-1} - a_i^j)) + s_i$ so that

$$d_i^j - d_i^{j-1} = a_i^j - d_i^{j-1} + max\,(0, d_i^{j-1} - a_i^j) + s_i \quad (2.3)$$

Now, since $d_i^{j-1} = a_i^{j-1} + s_i$, $d_i^{j-1} > a_i^j$ implies that $a_i^{j-1} + s_i > a_i^{j-1} + s$ *or* $s_i > s$ and we get s_i on the rhs of (2.3). On the other hand, $d_i^{j-1} < a_i^{j-1}$ implies $s > s_i$ and we get s on the rhs of (2.3).

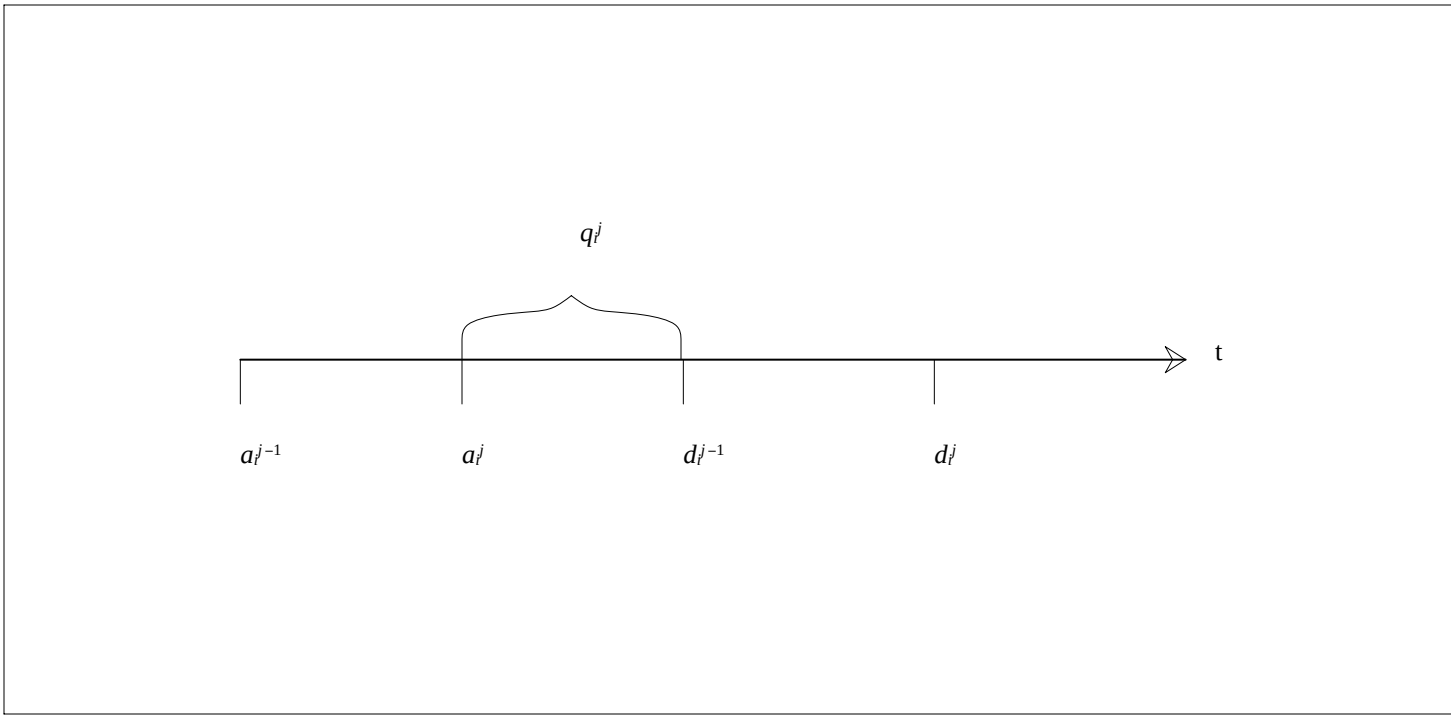

Figure 1

This result can be directly extended by repeated application to give the next two propositions that we state without proof. Detailed proofs of these and similar results can be found in [WaAg89].

Notation

Suppose packets are sent from the source at intervals of s_0 time units. Let

$$s_b = \max(s_i \mid 0 \le i \le n) \qquad (2.4)$$

be the 'bottleneck' service time in the circuit.

Proposition 2

If a source sends packets spaced s_0 time units apart, the acks will return to the source at intervals of s_b time units.

Proposition 3

Under the same conditions, every packet will find server k idle, for all $k > b$ (i.e. there will never be a queue at any of the servers downstream of the bottleneck).

Remark 3

Notice that the conditions (2.4) above imply that packets are being put into the VC at a rate faster than the bottleneck can service them. Thus queues at the bottleneck and, possibly at the servers upstream of the bottleneck, will grow indefinitely. Therefore this result does not make any steady-state assumptions about the system. This is also true of our next result, which assumes the same conditions as in Propositions 2 and 3.

Proposition 4

If at time d_0^j (departure of the jth packet from the source), there are w packets in the path, and packet $j-w-1$ has returned to the source, then the round-trip time for this packet, r_j, is bounded by

$$w.s_b < r_j \le (w+1).s_b \qquad (2.5)$$

Proof:

By Proposition 2, packets return to the source at intervals of sb. Thus,

$$d_n^{j-w-1} + (w+1).s_b = d_j^n \qquad (2.6)$$

and subtracting d_0^j from both sides of (2.6) we get the upper and the lower bound depending on whether $d_n^{j-w-1} - d_0^j$ is less than or equal to zero.

Remark 4

Notice that no packet can take less time than

$$R = \sum_i s_i$$

to return to the source. Thus, if the packet $(j - w - 1)$ has returned to the source, at least that much time has passed since the packet was sent. This implies that $w \geq R/s_0 \geq R/s_b$. Thus, for all values of w, we can write

$$r_j \geq max(R, w.s_b) \qquad (2.7)$$

Remark 5

When these results are applied to a sliding window protocol with window size w, we get the two curves described in [Jain88] for a sequence of D/D/1 queues (Figure 2). The window size corresponds to the offered load. It is easy to include a 'cliff' in these figures, by postulating a maximum buffer-size for the queue at the bottleneck, b_0. Then, a window size larger than $R/s_b + b_0$ will lead to loss of a packet in every window that is transmitted and therefore loss of throughput.

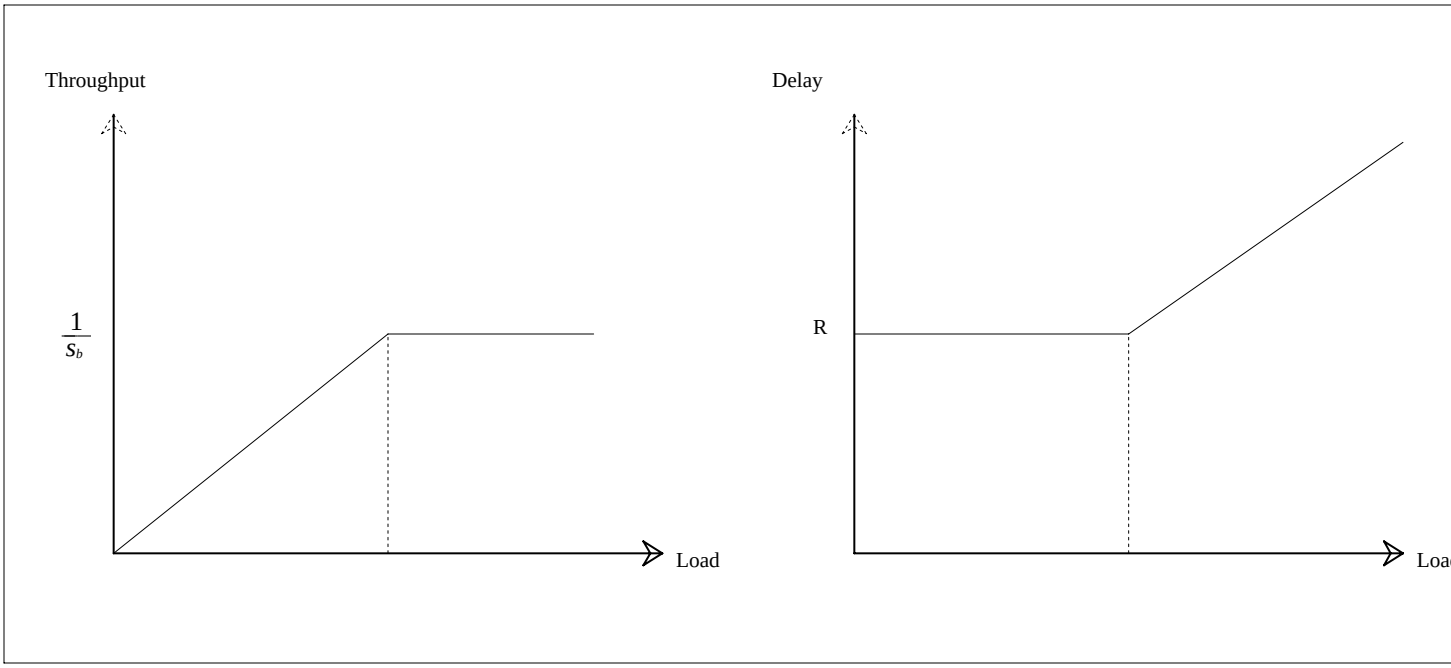

Figure 2

3. Effect of packet loss

We now consider the transient behavior of some protocols when a packet is lost. We wish to measure the time that the bottleneck server remains idle, since this measures the loss of bandwidth.

To concentrate attention on the bottleneck, we make a simplifying assumption. This is a 'worst-case' assumption in the sense that under this assumption a queue will form only at the bottleneck, and therefore this queue will be the largest possible.

Assumption

The fastest possible transmission from the source sends packets at intervals of s_0 where,

$$s_0 > s1, s2, \cdots s_b - 1.$$

We have already seen in Proposition 3 that no queue forms downstream of the bottleneck. This assumption guarantees that no queue forms upstream of the bottleneck, and we can replace these segments of the VC by delay-boxes (Figure 3). In queueing terminology, these boxes are a large number of parallel servers with service time equal to the delay. The first box has a delay,

$$D_1 = \sum_{i=0}^{b-1} s_i \text{ and the second } D_2 = \sum_{i=b+1}^{n} s_i .$$

We now consider a sliding window protocol with $w \geq R/s_b$. Proposition 5 describes the steady state.

Proposition 5

For any $w \geq R/s_b$, $w - R/s_b$ packets require buffering at the bottleneck in the steady state.

Proof:

The source transmits w packets at start-up, and then waits for the first ack to return. Subsequently, it transmits one packet for every ack that returns, and since these return at regular intervals of s_b time units (Proposition 2), packets are transmitted only at those time intervals. The system soon reaches a steady-state queue length at the bottleneck. But there are always w packets in the system. Of these, some are in the delay-boxes at equal intervals of s_b, and the rest are in the queue at the bottleneck or in service there. A simple computation shows that the delay boxes and the bottleneck server can only contain R/s_b packets, so the rest must be queued at the bottleneck. A detailed analysis can be found in [WaAg89].

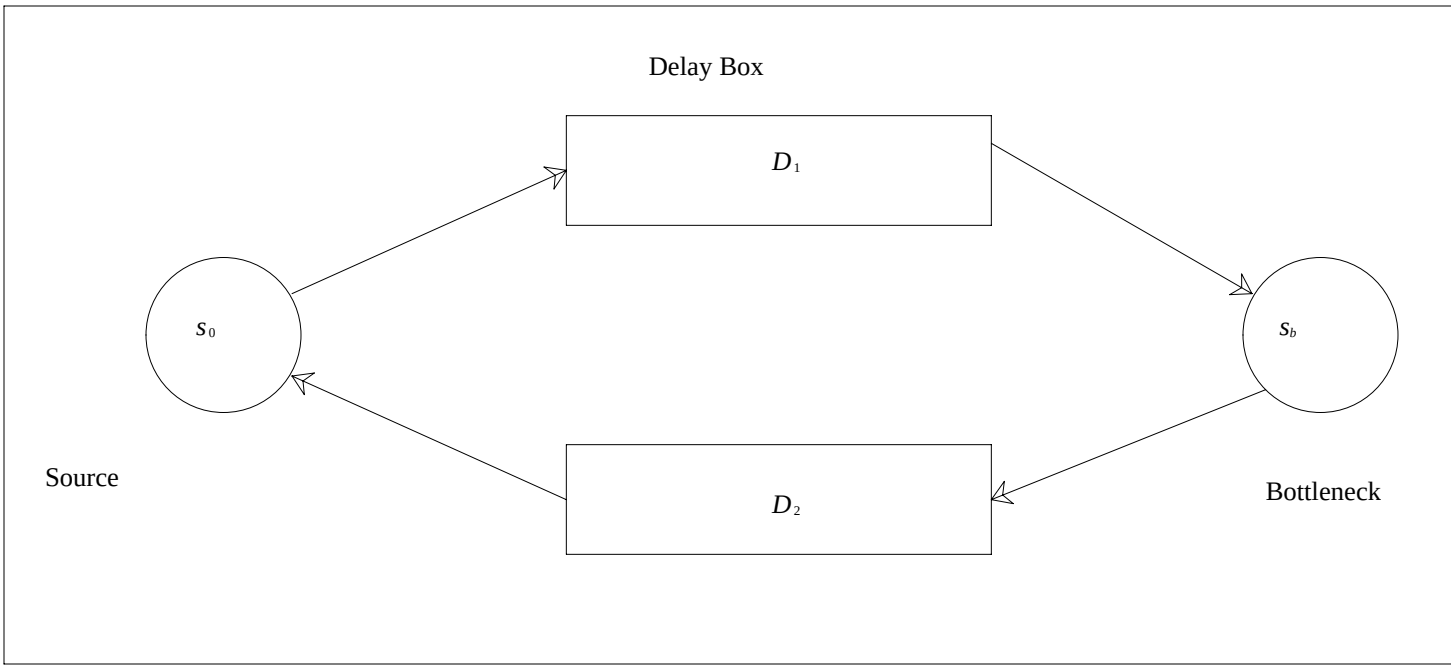

Figure 3

Notation:

Because of the crucial nature of the ratio R/s_b, we will henceforth denote it by V. This is the pipe-line depth when the service rate is $1/s_b$.

Remark 6

In this steady state, the bottleneck will be continuously busy. Thus, any loss of throughput because of packet retransmission can be measured as the idle time at the bottleneck. Second, note that the bottleneck queueing delay leads to an increase in the round-trip delay and hence a possible slack in the setting of the VC retransmission timer. Finally, when the window size is at the knee the bottleneck is continuously busy but the queue is zero. The 'slow-start' window-adjustment strategy [Jaco88] can lead to a final window size that is much larger than this value, being limited only by the largest possible size of the bottleneck queue.

We now consider three retransmission schemes, and analyze the loss of bandwidth on packet loss in each case. These are sliding window flow control with go-back-n and selective retransmission, and a rate based flow control scheme with selective repeat. We assume that the time-out is set for

$$T_0 \geq r_t = w.s_b \geq R \qquad (3.1)$$

and further that the jth packet is dropped sometime after being serviced at the bottleneck. The more realistic case, when the packet arrives at the bottleneck and is then discarded, can be similarly analyzed.

Case I: Sliding window with go-back-n Scheme

Proposition 6

The idle time at the bottleneck, L, is given by,

$$L = r_t + \max(0, T_0 + R - 2r_t)$$

Proof:

Note that when the $(j-w)$th packet returned to the source, the jth packet was transmitted. i.e.

$$d_0^j = d_n^{j-w} \qquad (3.2)$$

Thus, at time $d_0^j + (w-1)s_b$, the packet $j-1$ returns to the source and packet number $(j+w-1)$ is transmitted. After this, since the jth packet does not return to the source, no further transmissions will take place. This last packet, $(j+w-1)$, returns to the source at time

$$d_0^j + (w-1)s_b + w.s_b$$

and it finishes service at

$$d_0^j + (w-1)s_b + w.s_b - D_2 \quad (3.3)$$

(We work backwards to avoid the queuing delay at the bottleneck). On the other hand, retransmission will start at $d_0^j + T_0$. The first retransmitted packet will reach the bottleneck at

$$d_0^j + T_0 + D_1 \qquad (3.4)$$

and after that the bottleneck will be continuously busy again. Thus, the time lost on the bottleneck will consist of any idle time after the instant given in (3.3) plus the time wasted in servicing the packets j, $j+1$, $\cdots$ $j+(w-1)$, since all of them have to be re-transmitted. The second term is easily seen to be exactly $w.s_b = r_t$. To find out the idle time, if any, we only need to check whether the time given in (3.3) less than that in (3.4). i.e.

$$idle\ time = \max(0, [d_0^j + T_0 + D_1] - [d_0^j + (w-1)s_b + w.s_b - D_2])$$

$$= \max(0, T_0 + R - 2r_t) \quad \text{and, total loss of bandwidth becomes}$$

$$L = r_t + \max(0, T_0 + R - 2r_t) \qquad (3.5)$$

Case II: Sliding window with selective retransmission

Here, the packets transmitted after the jth one are not wasted. Thus the loss of bandwidth consists almost entirely of actual idle time of the bottleneck. As in the earlier case, (3.3) and (3.4) are valid. But here, the maximum in (3.5) is added to the minimum idle time of the bottleneck, which occurs while the retransmitted packet is being serviced and then returned to the source, $= s_b + D_2$. Further, even though the transmission of the next window starts at that point in time, it takes the first packet an additional time of D_1 to reach the bottleneck. Thus, we have

Proposition 7

$$L = R + \max(0, T_0 + R - 2r_t) \qquad (3.6).$$

Remark 7

In both the above schemes, we can avoid any extra loss of bandwidth because of the second term, as long as we set

$$T_0 \leq 2r_t - R = r_t + (r_t - R) \qquad (3.7),$$

but the selective retransmission scheme always performs slightly better than Go-Back-N (as one would expect). Indeed, with a large window (compared to the 'knee', $R/s_b = V$) we have $r_t \gg R$ and it should be the preferred method. However, it is important to note that a large timeout value for ruling out spurious timeouts has a penalty.

Case III: Rate-based flow control with selective retransmission

The two earlier cases considered a traditional sliding window flow control scheme. Recently there has been interest in rate based flow control schemes. We present one such scheme that is based on NETBLT [ClLa87]. We call it Netblt to acknowledge our inspiration, while denoting that our analysis is for a slightly different subset of NETBLT. In this scheme, a source transmits packets at regular intervals s_0, where the time s_0 is estimated by measuring the time between arriving acks and therefore estimates s_b [WaAg89]. We shall assume that $s_0 = s_b$.

Estimation of s_b is done only for consecutively numbered packets. Thus a missing packet will not disturb the scheme. We set a retransmission timer to any value $T_0 \geq r_t + R$. As soon as the timer goes off,

we retransmit the timed out packet at the next available time slot. In practice, we may not need to wait for the next time slot, since the effect on performance is likely to be small. In this scheme, the sequence number of an ack acknowledges a single packet (and not all lower-numbered packets as well).

We believe that this rate-based scheme captures the essence of many similar schemes. We will add details regarding rate estimation to Netblt in a later section.

Proposition 8

In this scheme, the idle time when a packet is lost is s_b, which is the time wasted in retransmission.

Remark 8

One objection that may be raised is that selective retransmission requires the receiver to buffer the packets that come after the lost packet. The packets to be buffered are precisely those that lie between the two transmissions of the lost packet. The number of such packets is easily seen to be T_0/s_b. Since $T_0 \geq r_t$, this number can be as low as that required by the sliding window scheme operating at the knee. On the other hand, there is no loss of bandwidth incurred by setting the timer to a much larger value.

To make this scheme robust against possible buffer overflow at the receiver, a window should indeed be negotiated. However, the window need not be adjusted during the operation of the protocol from congestion control and avoidance considerations. This clearly separates the function of flow control between sender and receiver, and flow control between sender and bottleneck. (However, the window should be larger than V, so that it does not interfere with the rate-based scheme.)

Remark 9

On a packet loss, both the window protocols disturb the steady-state flow rate by the rapid transmission of an entire window of packets. This requires increased buffer capacity at the bottleneck. Specifically, the bottleneck queue increases to a maximum size Q before settling down to the steady-state value. This is similar to what happens at start-up, and a detailed analysis can be found in [WaAg89]. The scheme proposed by Jacobson and Karels [Jaco88] (JK) and CUTE [Jain86] avoid this by doing a slow-start after every packet loss, However, these schemes cause considerable loss of bandwidth, as we will show later, and they are only justified on the assumption that the packet was

lost because of congestion, which in turn, will take some time to clear. Proposition 9 computes the maximum size of the queue.

Proposition 9

Let $V \le w \le R/s_0$ and assume that the VC is empty when w packets are transmitted by the sender as fast as possible (i.e at intervals of s_0). Then, the queue at the bottleneck will grow to a maximum size

$$Q = w - \frac{(w-1)s_0}{s_b} \qquad (3.6)$$

Proof:

Call the instant at which the first packet arrives at the server time zero. Subsequently, packets arrive at the server every s_0 time units, while one packet leaves every s_b time units. Since, by definition, $s_b \ge s_0$, the queue reaches its maximum size when the wth packet arrives at the server. This occurs at time $(w-1).s_0$. During this time, the number of packets that have left the server is

$(w-1)s_0/s_b$, and hence the proposition.

4. Adaptive schemes for congestion avoidance and control

Earlier authors have described some mechanisms that lead to congestion in a virtual circuit and, on this basis, have suggested modifications to transport layer protocols. In these schemes, a source tries to detect the state of the VC and uses this information to adapt its sending rate or flow control window. The schemes differ in the choice of the state probing mechanism, and how they react to changes in the state. For example, Jain has proposed that the round-trip delay, r_t, be measured regularly and used to probe the state of the virtual circuit [Jain89]. The JK scheme [Jaco88] uses packet loss as an indication of congestion. NETBLT, a rate-based scheme, measures packet inter-arrival times to adapt to state changes [ClLa87].

We will describe and analyze four such adaptive schemes. These are

(1) The delay-based scheme mentioned above [Jain89]

(2) The dynamic window adjustment scheme proposed by Jacobson and Karels [Jaco88]

(3) The variant of NETBLT, Netblt, described earlier, with some additions [ClLa87].

(4) A new rate-based flow control scheme, 2P

Schemes 1 and 2 are both window based, and we will refer to them as W-schemes. We do not consider adaptive schemes that depend on explicit control signals sent to the source by the network, such as the DECBIT scheme [Jain88].

Remark 10

We distinguish between congestion avoidance and congestion control [Jain88]. Schemes that interpret the loss of a packet to signal congestion and then react to this event, are really congestion control schemes. To avoid congestion, a scheme must have a more detailed model of the VC. For example, in our VC model, we can avoid congestion by operating at the knee ($w = V$) rather than near the cliff ($w - V = b_0$). The JK scheme and CUTE [Jain86] operate near the cliff and do not even recognize the existence of the knee. The delay scheme, Netblt and 2P attempt to stay at the knee and thus avoid congestion.

Remark 11

We do not propose to discuss how a transmission control policy can achieve fair-allocation of the available bandwidth amongst different virtual circuits. Further, it simplifies analysis and indeed the operation of transmission control schemes, if we assume for the moment some form of fair-queueing at the servers [Hahn86, Nagl84, WoMa89, DeKe89]. Specifically, we will assume that the server is able to recognize the packets of each VC and place them in a separate queue. Service is given to each queue in a round- robin fashion i.e. one packet is serviced from each queue in turn. A queue is skipped only if it is empty. Thus, if one other VC is sharing the bottleneck with ours, when the service time for our packets is s_b, we would expect the service time to drop to $s_b/2$ when that VC stopped transmitting. The assumption here is that both VCs desired to use as much of the bandwidth as possible and that they were therefore each given half of what was available. This way, the results on packet loss are more meaningful, since the packet losses from different VCs are decoupled.

4.1. Delay based scheme

Here, the round-trip delay, r_t, is measured once every round-trip-time [Jain89]. An increase in r_t is interpreted as a signal of impending congestion ($W \geq V$) and triggers a multiplicative decrease in the window size, while no increase in r_t additively increases the window size. Packet loss is taken as a definite

sign of congestion and the window is set to 1. The decision variable is actually the dimensionless quantity

$$NDG = \frac{(r - r_{old})(W + W_{old})}{(r + r_{old})(W - W_{old})} \quad (4.1)$$

where r, r_{old}, W, W_{old} are the round trip delays and window sizes respectively. The decision rule is: *if* $NDG > 0$ then decrease(W) else increase(W). The window changes once in each cycle of length r_t. The increase and decrease policies are:

$$W \leftarrow W_{old} + \delta W \; ; \; W \leftarrow cW_{old}, \; c < 1 \quad (4.2)$$

Since we are in a deterministic framework, we assume that r_t can be measured accurately by timing a single packet, and further, that this is done once every cycle. Note that such a control policy operating in our model of the VC achieves a steady-state cyclic behavior that agrees with the simulation studies in [Jain88]. The window increases linearly until it is just above the knee. At this point there is a multiplicative decrease in the window size and the cycle repeats.

If c is small, or W is large at the knee, then the scheme would constantly lose available bandwidth except during the one cycle that it operates at the knee. Clearly, the effective rate of transmission is the mean of the two extremes, so we state the following proposition without proof.

Proposition 10

The delay scheme underutilises the available bandwidth. Asymptotically, over a long conversation, only a fraction $(1+c)/2$ of the bandwidth is used.

4.2. JK

Jacobson and Karels [Jaco88] propose that the flow control window size be dynamically adjusted according to the network state. Initially, a source sets its window size to 1. As it receives acks, this is slowly increased ('slow-start') to the maximum allowed window size, until a timeout occurs, indicating a packet loss from congestion. At this point, the window is shut down to one, and the process repeats. The window increases exponentially upto a threshold, and then linearly. In the JK scheme, regular measurements of r_t are made, but only with the objective of properly setting the timer.

4.3. NETBLT

In the NETBLT scheme [ClLa87] the receiver controls the source's sending rate based on its estimation of the VC state. The receiver regularly compares the inter-arrival time of the packets to the source sending rate it had specified. An arrival rate below 95% of the transmission rate indicates 'congestion' (as it is called in RFC-998) and the rate is reduced to the previous rate. The rate is increased if the achieved transmission rate is over 95% of the specified rate. This can be considered to be a rate probe that tests whether more bandwidth is available.

In terms of our model, the knee is detected when an increase in the transmission rate does not lead to an increase in throughput. Packet loss would occur later, when the queue grows beyond b_0.

For our analysis, we adapt these features of NETBLT into Netblt. First, with no loss of generality, we shift the rate control to the source. Second, we add the rate probing scheme to Netblt. The amount of increase is a small constant, which we express as the number of extra packets that will be transmitted in one round-trip time, δW. Netblt reduces the sending rate if there is a significant increase in the inter-arrival times compared to the specified inter-transmission times. In terms of our model, we increase the transmission time-interval whenever a returning packet gives us a larger estimate of s_b. In our analysis of Netblt, we assume a steady state at the knee, since the protocol is designed to stay (around) there. Note that no measurement of round-trip time is made with the objective of flow control. This has several interesting consequences.

Netblt ignores some important aspects of NETBLT. Since we are not concerned with fair allocation of bandwidth, we will not analyze the algorithm that attempts to achieve such an allocation, although it does determine the rate-increases for probing the VC. We have also neglected the burst-transmission scheme and multiple-buffering (a consequence) that appear to have been introduced because of the practical difficulty of maintaining timers on a per-packet basis.

Remark 12

We note that the burst-transmission policy of NETBLT is close, in practice, to the 2P scheme (described below). Thus it could easily serve the twin purposes of measuring R and s_b. However, as this was not the stated purpose of burst-transmission, we have preferred to analyze the straightforward rate-based scheme incorporated into Netblt. This will bring into focus the advantages to be

gained by using the round-trip measurement in addition to the rate-measurement.

4.4. 2P

Theory

In a deterministic model, a source performs best when it is exactly at the knee, and thus has V packets outstanding. At the knee, there is no queueing at any node, and the maximum possible throughput is achieved. Since $V = R/s_b$, and R and s_b could change with time, it is necessary to periodically measure these quantities. We measure s_b as a direct consequence of Proposition 2, that is, s_b is the inter-arrival time of acks for packets that are sent out as fast as possible. R is the time between sending out a packet and receiving an ack when all the queues in the VC are empty. This can be approximated by measuring r_t after deliberately reducing the queue(s) in the VC.

We claim that measurement of both these fundamental quantities gives us much finer control over the transmission than could be achieved by measuring either one alone. Our scheme can be adapted to any transport layer transmission protocol by suitable specialization.

Algorithm

2P is a rate based flow control scheme that occasionally skips transmissions to adjust for changes in the network state. It has two parts: start-up and normal transmission.

At start-up, we do not know the value of s_b. Since we do not want to overload the bottleneck with packets, some sort of 'slow-start' is desirable. We combine this with an initial measurement of the VC parameters. At start-up we send a *packet-pair* consisting of two packets sent as fast as possible (back-to-back). The round-trip time of the first packet measures R, and the inter-arrival time of the two packets measures s_b. The current estimate of s_b is denoted by s_e.

During normal transmission we transmit packet-pairs every $2s_e$ time units and s_e is updated to the inter-arrival time between paired acks. We would like to decrease the queue length (to zero, if possible) before measuring R. To do so, we skip one transmission slot periodically, and then send out two packet-pairs back-to-back. The first of these is called a special pair. The round trip time for the special pair gives us R_e, our estimate of R. We skip a slot and generate another special pair whenever the previous special-

pair returns. The first pair transmitted at start-up is considered to be a special pair.

We would like our scheme to react immediately to changes in V. To do so, we recompute R_e/s_e on the arrival of every normal or special pair. If there is a decrease in this value, we skip transmission for some number of slots so that the number of outstanding packets decreases to the latest value of R_e/s_e. This simultaneously ensures that the measurement of R is more accurate (since we are really measuring r_t). A decrease in V automatically triggers a new special pair. These two adjustment mechanisms are more completely specified below.

If we have estimates R_e and s_e for the values of R and s_b respectively, denote the ratio R_e/s_e by V_e. Let V_{new}, V_{old} be the old and new values of V_e using the old and new estimates respectively. Then, we calculate the quantity

$$n_{skip} = \max(\lceil (V_{old} - V_{new})/2 \rceil, x),$$

where $\lceil z \rceil$ computes the smallest integer greater than or equal to z. x is zero for a normal pair and one for a special pair. If the value of n_{skip} is zero, (which can happen only on receipt of an ordinary pair) we simply carry on. Otherwise, we skip n_{skip} transmission slots at regular intervals of $2s_e$ and send a special-pair, followed immediately by another ordinary pair. After that, we continue to send pairs of packets at regular intervals of $2s_e$,until the next special measurement is to be made and so on. Thus, we always skip one time period before sending a special-pair. We compensate for that drop in rate by sending two pairs together at the next slot.

Notice that if any of the ordinary pairs gives us a new value for V, we get a non-zero n_{skip}, so the immediately following pair is a special pair. Any outstanding special pair is then ignored when it comes in. This is a fast-response feature that allows a quick reaction to one of the main causes of congestion: a sudden increase in s_b.

Remark 13

In practice, intermittent bursts of more than two packets may be useful to account for statistical variations in s_b. In our deterministic framework, a pair of packets suffices to illuminate the basic principle.

Remark 14

It is possible to move the rate control to the receiver, as in NETBLT. The receiver looks at the inter-

ack spacing and informs the source whenever it should change its sending rate. This will avoid the need for every packet to be acked. To allow special pairs to measure r_t correctly, we require that the receiver send acks for at least the first packet of a special pair. No other change in the protocol is necessary.

We will see that this simple strategy is enough for adapting to many of the changes that can take place in the VC. Also, by sending all the packets in pairs, we can monitor the VC as often as desired. If this is too frequent and leads to timer overheads, the measurement can be carried out periodically, with some resultant degradation in the performance. In the analysis below we will assume that the measurement is carried out by every pair i.e. we always use the most up-to-date information and get the fastest response to changes possible. Further, with no loss of generality, we will assume that at any time there is only one special-pair in existence.

5. Analysis of Policies

Having described some adaptive flow control schemes, we now turn to a deterministic analysis of their performance. In Section 5.1, we compare the bandwidth utilization on start-up for 2P and JK. Section 5.2 analyzes the reaction of Netblt, W-schemes and 2P to transient changes in the VC state.

5.1. Loss of bandwidth on start-up

5.1.1. JK

JK slow start has two parts: an exponential increase phase, where the window size doubles every round trip time, and then a linear increase where the window size increases by one every round trip time. We study the bandwidth lost until the time the bottleneck reaches full utilization, measured as the idle-time, L, of the bottleneck. We bound the loss due to this scheme from below and above, by schemes that do a strict exponential and a strict linear increase.

Proposition 10

(a) A linear window increase results in an idle time of

$$L = \frac{D_1}{s_b} + \frac{(V-1)(V)}{2}$$

(b) An exponential window increase results in an idle time of

$$\frac{D_1}{s_b} + kV + (2^k - 1)$$

where $k = \lceil \log_2(V) \rceil$.

(c) The loss of bandwidth from the slow start scheme is bounded from above and below by the expressions in (a) and (b) respectively.

Proof:

First, note that events at the bottleneck are time shifted by a time of D_1 or D_1/s_b packet service times. Thus, if the window attains a size of V at the source, this is reflected at the bottleneck only D_1/s_b packet times later. For convenience, we will work with the window size at the source, and adding a correction of D_1/s_b gives us the exact value for L. L is then the time at which the source increases its window to V, corrected by D_1/s_b.

(a) The window starts at one, and increases by one each round trip time, so that in the ith period, i packets are serviced. The window reaches V at the end of the $V-1$th period. In the ith period, the idle time is $V - i$, so that the net idle time is

$$\frac{D_1}{s_b} + \sum_{i=1}^{V-1} i = \frac{D_1}{s_b} + \frac{(V-1)(V)}{2}.$$

(b) The window starts at 1 and doubles each round trip time, so that in period i, the idle time is $(V - 2^i)$. The window reaches V after the $k-1$th time period, where $k = \lceil \log V \rceil$. The idle time is

$$\frac{D_1}{s_b} + \sum_{i=0}^{k-1} (V - 2^i)$$

$$= \frac{D_1}{s_b} + kV + (2^k - 1)$$

(c) Since the slow start algorithm does a exponential increase followed by linear increase, the bounds are clearly those in (a) and (b).

Remark 15

On a high bandwidth long delay channel (e.g. satellite), this loss can be substantial. For example, with R = 300 msec, D_1 = 150 msec and s_b = 1 msec, we find that L, the bandwidth loss in packets is bounded by

$$2339 \le L \le 45000.$$

The DECbit scheme [Jain88] has only a linear increase, and our analysis shows that this could cause a considerable loss of bandwidth. Obviously, such a scheme performs much worse than JK slow start.

On the other hand, the loss for 2P is always the same:

$$L = \frac{D_1}{s_b} + R - 2s_b.$$

We believe that this is unavoidable, since R is the minimum time needed to find out anything about the virtual circuit. In an operating environment, this loss can also be reduced by starting with a flow-rate that is known to be 'safe'. Alternately, this initial measurement can be piggy-backed onto the connection establishment phase of the transport protocol, which necessarily sends just a few packets back and forth. Once we have measured the correct parameters, there can be no loss of bandwidth at all (on a deterministic VC).

5.2. Analysis of adaptation to changes in the VC

We now analyze some transients that arise from abrupt changes in the state of the VC. These changes could be because of a change in the routing tables in the lower layer of the network. A greater concern is when two or more VCs share a server. The start-up or shut-down of transmission on one circuit could cause the others to experience a sudden state change. Thus, such transient analysis lays the foundation for a deeper investigation of the interaction between two or more VCs sharing a server. The behavior of the system depends on the control strategy or transmission policy being used. Therefore we study and compare the behavior for four control schemes

(a) Delay scheme

(b) JK scheme

(c) Netblt

(d) 2P

5.2.1. Increase in D_1 to D_{1new}

When the pipeline delay $D = D_1+D_2$ increases, with no change in s_b, the correct response is to continue sending at s_b, while increasing the window size. In the model of the VC implicit in the delay scheme, this increase in delay is assumed to be indicate an increase in the bottleneck queue length, and thus the onset of congestion. Thus, the scheme will do a shutdown of the window, which is precisely the wrong thing to do. In this analysis, unless otherwise stated, we assume that no timers go off.

5.2.1.1. Delay scheme

Here there are two cases, depending on the previous change to the window.

Case 1:

Assume that the window has just undergone a multiplicative decrease so that $W = cW_{old} = cV$. Clearly, $W < R/s_b < R_{new}/s_b$. Applying the formula for NDG (4.1) we see that, although the delay increases, there are no further multiplicative decreases, and the system reaches equilibrium through a sequence of additive increases. Depending on the value of c used there will be a loss of bandwidth during this period.

Proposition 11

The loss of bandwidth (in excess of normal loss) will be at least

$$\sum_{i=V-V_{old}}^{V} i + \frac{D_{1new} - D_1}{s_b}$$

Proof:

Earlier, the window would have increased additively to V, with some loss of bandwidth. The excess loss occurs while the window additively increases to V from V_{old}. The second term is the change in

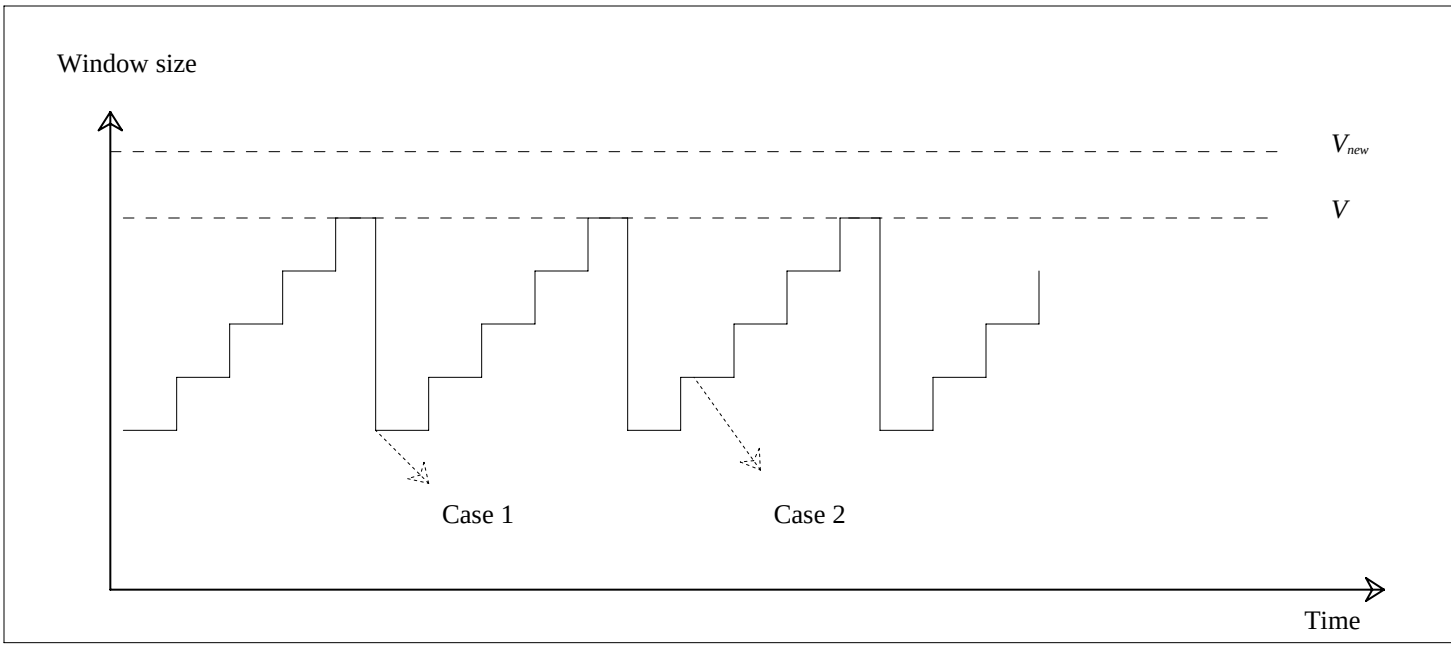

Figure 4

the adjustment factor due to an increase in D_1.

Case2:

Assume that the window has just undergone an additive increase and W and W_{old} are both $\leq V$. At the instant that the new W comes into effect, D_1 changes to D_{1new} so that R changes to $R_{new} > R$. Since the new window is less than $R/s_b < R_{new}/s_b$, each packet will show a new delay of R_{new} (no packet will encounter a queue at the bottleneck). Since R_{new} represents an increase over R, the previous value that had been measured, this is interpreted as an indication of queue build-up at the bottleneck. Thus at the next decision point, W will be multiplicatively reduced to cW, a value considerably below the 'knee' which is now R_{new}/s_b. Then, a full cycle of additive increases will be required before the equilibrium point is reached leading to a loss of bandwidth. In the worst case, this will happen just after the multiplicative decrease, and we have the following proposition:

Proposition 12

The loss in bandwidth can be as large as

$$\sum_{i=V-c\cdot V_{old}}^{V} i + \frac{D_{1new} - D_1}{s_b}$$

Proof:

The window has to increase to V from the value of $c^2 V_{old}$ that it reaches after the second multiplicative decrease, leading to the expression above.

5.2.1.2. JK

The increase in delay causes a delay in the return of acks, and the retransmission timer might go off. If the timer goes off, the source will have to do a slow start from a window size of 1, leading to the loss of bandwidth computed in Section 5.1. If the timer doesn't go off, the exact behavior depends on whether or not slow-start is in progress. If it is, the analysis is similar to that in Section 5.1. If not, and the maximum window size is sufficiently large, the window will just increase till the pipeline is filled, and there is no loss of bandwidth. If the maximum window size is not large enough, the source will be window-limited in the bandwidth that it can use, to a fraction W_{max}/V of the available bandwidth.

5.2.1.3. Netblt

The increase in the delay causes a bubble in the pipeline, and this reduces the computed value of the arrival rate at the receiver. If the reduced arrival rate is less than 95% of the specified sending rate, then Netblt will reduce its sending rate to its earlier rate, leading to a possible small loss of bandwidth.

5.2.1.4. 2P

2P will simply keep on transmitting pairs at every $2s_b$ time units with $n_{skip} = 0$. Adjustment to this change is automatic, because the rate of transmission depends only on s_b. There will be a small loss of bandwidth because the bottleneck will be idle for a short while. There is another subtle possibility of loss of bandwidth that needs to be mentioned. That is the case when the first packet of a special-pair reaches the bottleneck after time delay D_1 while the second one reaches there after a delay D_{1new}. The inter-arrival time of these two at the bottleneck is then $\xi = D_{1new} - D_1$. If this is larger than s_b, the protocol will react by skipping some pairs of packets. However, 2P will recover as soon as the next pair of acks arrive.

Proposition 13

The loss of bandwidth will be $2n + \xi/s_b$ packets, where

$$n = \max(1/2\lceil \frac{R_{old}}{s_b} - \frac{R_{new}}{\xi} \rceil, 1)$$

Proof:

The second term is just the bubble size in the pipeline. Since 2P mistakenly estimates that the pipeline depth is R_{new}/ξ, instead of R_{new}/s_b, it skips n slots, and in each slot it loses 2 packets, leading to a net loss of $2n$ packets.

5.2.2. Decrease in delay

From the point of view of congestion, this is the more interesting case. The appropriate response to this event should be to try to reduce the number of packets in the system and therefore the queue length at the bottleneck.

5.2.2.1. Delay scheme

There are two cases. First, assume that this event occurs immediately after there has been an additive increase in the window size and it has reached some value $W \geq R_{new}/s_b$. In that case, the decreased delay causes the window to be further increased, leading to a build-up of the queue at the bottleneck. This is precisely the wrong response, and the window decrease happens only at the next decision point, when a further increase in the window-size gives an increase in the measured r_t. To get an exact expression for the maximum queue that occurs at the bottleneck, we note that an increase in window size to $W+2\delta W$ occurs before it decreases again. The maximum queue-length, from Proposition 5, is

$$W+2\delta W-R_{new}/s_b.$$

If the current window is smaller than V, the new value of V is properly detected, and the system responds correctly to the change.

5.2.2.2. JK

The decrease in the delay causes the queue in the bottleneck to increase. If this causes a packet loss, there is a timeout and a slow-start leading to a loss of bandwidth as computed in Section 5.1. If there is no timeout, the only change is that the window increase cycle that leads to a timeout peaks earlier.

5.2.2.3. Netblt

Since this is a rate-based scheme, with rate being measured by inter-arrival time of the packets, there is no change in the transmission rate. Thus, the V_{old} packets that were in the VC before the change remain in the system causing a queue of size $R/s_b - R_{new}/s_b$ to build up at the bottleneck. i.e. the VC now operates above the knee. This queue will persist, since it is not detected or corrected for.

5.2.2.4. 2P

This behaves just like Netblt and operates above the knee, until a special pair is sent. At that point, an extra delay of $2s_b$ before sending the next pair reduces the queue at the bottleneck and the special pair returns with a reduced value of R, ensuring a sequence of adjustments that finally brings the system back to the knee. If one pair somehow overtakes another, it will have a smaller round-trip time, and will immediately lead to skipping of pairs on its return. This hastens the return of the system to the knee.

5.2.3. Sudden increase in service time

This is the main concern of any flow control policy since it calls for an immediate reduction in the sending rate to avoid congestion. In addition, this will cause an unavoidable extra delay in some packets and thus may affect the the retransmission timers. Increase in s_b can be caused simply by the start-up of transmission by another virtual circuit at the bottleneck, so it is important to correctly respond to this change.

5.2.3.1. Delay scheme

Let us consider the worst case scenario in which the window has just reached its maximum value, $R/s_b + \delta W$. Let the time instant at which the value of s_b changes also be the instant at which the window size has also been increased. Then, δW packets will be immediately injected into the VC by the sender, and after that one packet will be sent every s_b units, as each ack returns. During this entire period, the number of packets in the VC stays at $R/s_b + \delta W$. The first of the extra δW packets will return after an increased round-trip time and signal the sender to decrease its window-size. At the time the first of these acks return, only D_2/s_{new} packets are downstream of the bottleneck. Thus, the upstream portion must contain the rest: $R/s_b + \delta W - D_2/s_{new}$. We assume that at this instant the sender stops transmitting because it has reduced the window size. The last packet just transmitted takes time D_1 to reach the bottleneck and from that point on the queue will stop growing. Now, in time D_1 the number of packets serviced is D_1/s_{new}, so the maximum length achieved by the queue at the bottleneck is:

$$R/s_b + \delta W - (D_1 + D_2)/s_{new},$$

which is essentially the difference between the old window and the ideal new window.

5.2.3.2. JK

A decrease in s_b is treated exactly the same as an increase in delay. The bottleneck queue builds up and the analysis of the situation in identical to that in Section 5.2.2.1.

5.2.3.3. Netblt

Netblt will detect the change at time $s_{new} + D_2$ after it occurs. However, a queue of size $(R/s_b - R_{new}/s_{new})$ builds up at the bottleneck. Unlike the delay scheme above, this queue does not dissipate

and the transmission continues to run considerably above the knee. The consequence of this queue build up is a sudden increase in the round-trip time r_t, of the packet, to a value $R(s_{new}/s_b)$. Given that the service time at the bottleneck could easily double, this suggests a value of at least $2R$ for the timers to avoid a spurious time-out(cf. RFC-793).

5.2.3.4. 2P

In this case there will be a queue buildup until the first pair comes in with the new value s_{new}. Since the round-trip time of this pair would not have changed at all, the value of n_{skip} is positive and some packets will be skipped as required. The queue at the bottleneck should dissipate before a special pair is sent to confirm the new parameters. However, the time taken for the queue to start dissipating will be D_2, the minimum possible. The maximum size reached by the queue is $D_2/s_b - D_2/s_{new}$.

5.2.4. Sudden decrease in service time

This could happen if a VC that was sharing the bottleneck shuts off. The flow control policy should try to use as much of the extra available bandwidth as possible. Notice that only 2P can immediately detect this situation. While a simple rate-based source operating at the knee can probe for such a change by periodically increasing the sending rate it will risk congestion. Similarly, the delay scheme will only achieve the proper rate after a large number of additive increases to the window size.

Our analysis indicates that there is some advantage in operating above the knee. This allows detection and use of the increase in available bandwidth, so that the loss is minimized. This feature can be incorporated into 2P by adjusting n_{skip} so that some fixed number of packets are always present in the bottleneck queue (at the expense of greater delay for each packet but no loss of throughput).

5.2.4.1. Delay scheme

Since such a scheme operates at or near the knee, it can potentially take advantage of the increased bandwidth available only through the additive increase policy. In the worst case, consider the instant where the window has dropped to cW, where $W=R/s_b$, and let s_b change to s_{new} at this instant. The delay scheme will have to increase W by δW until it reaches R/s_{new}. The loss in bandwidth will be $R/s_{new} - cR/s_b$ in the first time period, and will successively reduce by δW each time. Let

$$N = (R/s_{new} - cR/s_b)/\delta W.$$

Then, the total loss can be written as

$$\delta W(\sum_{i=1}^{N} i) = \delta WN(N+1)/2$$

packets, which translates into time by multiplying the expression by s_{new}.

5.2.4.2. JK

The analysis for JK is identical to the case of decreased delay, and the conclusions in Section 5.2.2.2 hold.

5.2.4.3. Netblt

If this scheme is in a steady-state and is transmitting a single packet every s_b time units, it will never detect the changed parameter, since by Proposition 1, the inter-arrival time of the packets will not change at all! The extra bandwidth can be detected only by the probing action rate increases until there is no further improvement. Depending on the rate increase policy, there will be a loss of available bandwidth while the scheme moves upwards to the correct rate. (In all fairness to NETBLT, the burst operation could easily be modified to detect this situation.)

5.2.4.4. 2P

Assume that a pair of packets has just finished service at the bottleneck when the change occurs. The next pair will arrive after $2s_b$-$2s_{new}$ time, will take $2s_{new}$ time to get service and will return to the sender after a delay D_2. The source will start sending at the new rate at this time. Let $T = 2(s_b - snew) + 2s_{new} + D_2$. Then during this time, T/s_b packets will be transmitted, when the number sent should have been T/s_{new}. Thus the loss of bandwidth is

$$L = T(1/s_b - 1/s_{new}) \text{ packets.}$$

6. Summary and Conclusions

We have modeled a virtual circuit in a computer network as a closed loop of D/D/1 queues, and then simplified it further to one D/D/1 queue and delay-boxes in a loop. Because the model is purely deterministic, we have been able to analyze the transient behavior of the system, which is what determines perfor-

mance in situations such as start-up, packet-loss etc. In this paper we have analyzed three schemes for dealing with packet loss and given exact analyses of their performance, where loss of throughput is considered to be the idle time at the bottleneck. In addition, we have looked at the buffering requirements of the schemes. Our analysis shows that a rate-based transmission scheme, with selective retransmission, performs better than the traditional window-based schemes.

We analyze some congestion-avoidance schemes that adapt to changes in the state of the VC. We present a scheme, 2P, which is an enhancement of the rate-based scheme. The scheme monitors both R and s_b. It also uses a pulsed or burst mode of transmission to probe the state of the VC, rather than a systematic increase in rate.

We develop an approach to transient analysis by considering a VC and its associated flow control scheme in their steady state of operation at the time a sudden change in some parameter of the VC occurs. Using the basic criterion that the system must remain close to the 'knee' in its operation, we analyse the performance of various schemes. If they are pushed above the knee, there is a danger of congestion, and if they are pushed below, some of the available bandwidth is lost. Thus the extent to which they deviate from the desired operating point, and the speed with which they recover are important qualities. We again find that our scheme, which uses both, R and s_b, performs better than schemes that use only one or the other of these parameters.

With the usual caveats regarding the simplicity of the model and the many assumptions made, we believe a number of important lessons have emerged from this analysis, which should be of help in determining the transmission control policies at the network and higher layers. First, windows are really ways of ensuring that buffer space at the sender and receiver are properly matched. As such, they are completely adequate at the datalink layer of the network, where communication can be considered as point-to-point. At higher layers, we need other mechanisms to ensure that intermediate switches or nodes, which are often transparent to the sender as well as the receiver, are not overloaded.

The second lesson is that the entire VC can be characterized by at least two important parameters, R and s_b, and we must attempt to find out both of these if we are to operate efficiently i.e. avoid congestion, and at the same time utilize as much of the available bandwidth as possible. A scheme that pays proper attention to both is likely to be better than one that relies entirely on one or the other.

Third, although the state of the VC can be probed by changing the transmission rate (or the window-size), many schemes that use such probing either approach the cliff or lose bandwidth. Our model and results suggest a way to avoid these problems: make changes in rate that last only for a small time compared with R (small bursts), and then compensate for these changes so that the overall rate of transmission in one round-trip period is relatively undisturbed. Thus, we avoid having to approach the cliff, or to lose much bandwidth. At the same time, as a bonus, it turns out that such an approach gives us an estimate of both, R as well as s_b.

Fourth, timers are important. A protocol such as NETBLT would work best with the easy availability of low-cost high-resolution timers [ClLa87]. Because current systems lack these, many complications have been introduced into the protocol design. For the efficient working and testing of future protocols, it may be necessary to ensure that good quality timer services are available cheaply and easily.

The approach of using deterministic analysis can be extended in several directions. One way is to allow the service times at the servers to be probabilistic. We believe that some of our simpler results can be extended in that direction quite easily. The second extension would be to study the effect of interference between two virtual circuits that share a common server or node, taking explicitly into account the kinds of mechanisms that operate within them. This would further illuminate the phenomena that lead to congestion. Finally, we would like to explore 2P in greater detail. A companion paper describes the implementation and simulation of 2P [KeAg90].

In conclusion, we claim that our deterministic model of a virtual circuit is both simple and powerful. Our analysis leads us naturally to a new scheme, 2P that has several desirable features. We hope to carry out a deeper analysis of 2P on the lines of this work in the future.

References

[BoPl88]

J. Bolot, B.D. Plateau and A.U. Shankar, "Performance Analysis of Transport Protocols over Congestive Channels", Tech. Report, CS-TR-2004.1, Dept. of Computer Science, University of Maryland, College Park, August, 1988.

[Clar82]

D. D. Clark, "Window and Acknowledgement Strategy in TCP", NIC RFC-813, July 1982.

[ClLa87]

D. D. Clark, M. L. Lambert and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol" NIC RFC-998, March 1987.

[DeKe89]

A. Demers, S. Keshav, and S. Shenker,"Analysis and Simulation of a Fair-queueing Algorithm", Proc. ACM SigComm 89, pp 1-12, also to appear in Journal of Internetworking, Vol. 1, No. 1, 1990.

[GeKl80]

M. Gerla and L. Kleinrock, "Flow Control : A Comparative Survey", IEEE Trans. on Communication, Vol. COM-28, No. 4, pp 553-574, April 1980.

[Hahn86]

E. Hahne, "Round Robin Scheduling for Fair Flow Control in Data Communication Networks", Report LIDS-TH-1631, Laboratory for Information and Decision Systems, M.I.T., Cambridge, Mass., Dec. 1986.

[Jaco88]

V. Jacobson "Congestion Avoidance and Control" Proc. ACM SigCOMM, Communication Architectures and Protocols, Vol. 18, No.4, August 1988.

[Jain86]

R. K. Jain "A timeout-based congestion control scheme for window flow-controlled networks" IEEE Journal on Selected Areas in Comm. Vol. SAC-4, No.7, Oct. 1986, pp 1162-1167.

[Jain88]

R.K. Jain and K.K Ramakrishnan, "Congestion Avoidance in Computer Networks with a Connectionless Network Layer : Concepts, Goals and Methodology", Proc. IEEE 1988 Computer Communication Conference, August, 1988

[Jain89]

R.K. Jain, "A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks", DEC Technical Report, DEC-TR-566, April 1989.

[KaPa87]

P. Karn and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", Proc. ACM SIGCOMM, pp. 2-7, 1987.

[KeAg90]

S. Keshav, A. Agrawala and S. Singh, "Design and Analysis of Flow Control Algorithm for Networks of Rate Allocating Servers", in preparation.

[Morg89]

S.P. Morgan, "Queueing Disciplines and Passive Congestion Control in Byte-Stream Networks", Proc. IEEE INFOCOM ’89, pp 711-729, 1989.

[Nagl84]

J. Nagle, "Congestion Control in TCP/IP Internetworks", Comp. Comm. Review, pp 11-17, Jan. 1984.

[Nagl87]

J. Nagle, "On Packet Switches with infinite Storage", IEEE Trans. Comm., Vol. 35, pp 435-438,

1987.

[SaSu88]

D. Sanghi, M. Subramanian, A.U. Shankar, O. Gudmundson and P. Jalote, "Instrumenting a TCP implementation", CS-TR-2061, Dept. of Computer Science, University of Maryland, College Park, July 1988.

[WaAg89]

J.G. Waclawsky and A.K. Agrawala "Dynamic behavior of data flow within Virtual Circuits" University of Maryland Technical Report, CS-TR-2250, May, 1989.

[WoMa89]

F. Wong and J.R.B. De Marca "Fairness in Window Flow- controlled Computer Networks", IEEE Trans. Comm. Vol. 37, No.5, May 1989.

[Zhan86]

L. Zhang,"Why TCP timers don’t work well" in Proc. ACM SIGCOMM ’86, pp 397-405, August 1986.

[Zhan89]

L. Zhang, "A New Architecture for Packet Switching Network Protocols", PhD thesis, Massachusetts Institute of Technology, July 17, 1989.