

**Internetworking and End-to-End Protocols**

Assigned reading: Peterson and Davie: Chapters 4, 5 and 9.1. All problems carry equal weight. For full credit, show all work.

**1. TCP RTT Estimation**

One difficulty with the original TCP SRTT estimator is the choice of an initial value. In the absence of any special knowledge of network conditions, the typical approach is to pick an arbitrary value, such as 3 seconds, and hope this will converge quickly to an accurate value. If this estimate is too small, TCP will perform unnecessary retransmissions. If it is too large, TCP will wait a long time before retransmitting if the first segment is lost. Also, the convergence might be slow.

- Choose  $\alpha = 0.85$  and  $SRTT(0) = 3$  seconds, and assume all measured RTT values = 1 second with no packet loss. What is  $SRTT(19)$ ? Recall,  $SRTT(k + 1) = \alpha * SRTT(k) + (1 - \alpha) * RTT(k + 1)$ .
- Now let  $SRTT(0) = 1$  second and assume RTT values = 3 seconds and no packet loss. What is  $SRTT(19)$ ?
- Now consider  $\alpha = 0.95$ . Repeat parts a. and b. and comment on the effect of a larger or smaller  $\alpha$ .

sol

- Based on the formula give and  $\alpha = 0.85$ , we can calculate the nth SRTT value using:

$$SRTT(n) = \alpha^n * SRTT(0) + (1 - \alpha) * RTT * (\alpha^{n-1} + \alpha^{n-2} + \dots + \alpha + 1)$$

So with the given initial conditions, we get  $SRTT(19) = 1.0912$  seconds

- If we instead use the new initial conditions and  $\alpha = 0.85$ , we get  $SRTT(19) = 2.9088$  seconds
- If  $\alpha = 0.95$ , the assumptions in part a result in 1.7547 (as compared to 1.0912), and the assumptions in part b result in 2.2453 (as compared to 2.9088). As we can see, the larger  $\alpha$  puts more weight put on the initial value and so results in a slower convergence time to the real RTT.

**2. TCP Slow Start**

Although slow start with congestion avoidance is an effective technique for coping with congestion, it can result in long recovery times in high-speed networks.

- Assume a roundtrip time delay of 125 ms (about what might occur across a continent) and a link with an available bandwidth of 1.2 Gbps and a segment size of 512 octets. Determine the window size needed to keep the pipe full and the time it will take to reach that window size after a time out using the Jacobson Algorithm.
- Repeat for a segment size of 32 Kbytes.

Sol

- The size of the window (in bits) is:  $125 \text{ ms} \times 1.2 \text{ Mbps} = 150 \times 10^6$  bits. Each packet is 512 octets (4096 bits), so the window size in packets is:  $\lceil (150 \times 10^6 \text{ bits} / 4096 \text{ bits/packet}) \rceil = 36622$  packets.

If we assume that the sender was transmitting at the full window before the timeout, then our congestion threshold is half the full window size (18311 packets). TCP will use exponential growth until the congestion threshold is passed, at which point additive increase begins. Therefore, we can use exponential increase to a window size of  $2^{15} = 32768$  (the first power of 2 bigger than 18311), which will take 15 RTT. Now we do additive increase to get to the full window. This will take  $36622 - 32768 = 3854$  RTT using additive increase (one packet increase in the window per RTT). So, the total time to reach the full window is  $(15 + 3854) \text{ RTT} = 3869 * (80 \text{ ms}) = 483.625$  seconds (more than 8 minutes!).

- The window size (in bits) does not change, but we need to recompute the window size for packets of 32KB. Each packet is 262,144 bits, so the window is:  $\lceil (150 \times 10^6 \text{ bits} / 262144 \text{ bits/packet}) \rceil = 573$  packets.

This makes the congestion threshold 287 packets, so we can use exponential growth up to 512 ( $2^9$ ) packets. Then we begin additive increase for  $(573 - 512) = 62$  RTT. Therefore, the total time to reach the maximum send window size is  $(9 + 62) \text{ RTT} = 71 * (125\text{ms}) = 8.875$  seconds.

### 3. TCP Congestion Performance

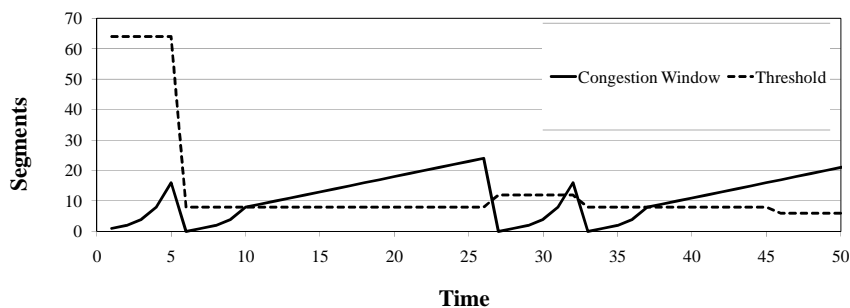
Consider a TCP system implementing slow start and congestion avoidance with fast retransmit and fast recovery. When a connection is setup the congestion window is initialized to one segment and the slow start threshold to 64 segments. To simplify the problem, assume that the timeout is equal to the RTT (an exact estimate) and specify time in units of RTT, such that one time slot is one RTT.

Packet transmissions are such that at each time slot the sender sends all packets in the congestion window. If ACK's are received in the next time slot, there is no timeout. In addition, to simplify matters either the entire window is acknowledged or none of its segments are acknowledged.

For a particular connection, ACK's are received in time slots 1-7, 9-30, 32-40, and 42-50. Timeouts occur in slots 8 and 31; in slot 42, three duplicate ACK's are received for the packets sent in time slot 41.

For the system described, plot both the congestion window and the slow start threshold (on the same graph) versus time (slots). Remember to consider the differences between slow start and congestion avoidance with fast retransmit and fast recovery when changing the congestion window.

Sol



In slot 6, the congestion window has expanded to 16 segments. Due to the timeout, the next slot will have a congestion threshold of  $16/2 = 8$  and the congestion window will be reset to 1. The congestion window grows exponentially until slot 10, when it equals the congestion threshold. It then grows linearly until the timeout in slot 27, when the congestion threshold is set to  $24/2 = 12$ , and the congestion window is reset to 1 again. The congestion window grows exponentially until slot 32, when it equals the congestion threshold of 16. It would then grow linearly except for the timeout in slot 33, when the congestion threshold is set to  $16/2 = 12$ , and the congestion window is reset to 1 again. In slot 45, three duplicate ACKs are received. The congestion window is then halved to  $30/2 = 15$  and begins to linearly again.

### 4. Adaptive Retransmission, Part I

- What is the retransmission ambiguity problem addressed by the Karn-Partridge algorithm? How does the algorithm avoid the ambiguity?
- Write a C program to estimate the frequency of unnecessary retransmission with the original TCP adaptive retransmission algorithm. In particular, write a one-million-iteration loop that calls a function `double get_rtt ();` to measure RTT, calculates an estimated RTT, and counts the number of times that the next measured RTT exceeds the current estimate. Use a damping factor of 0.15 ( $\alpha$  on P&D p. 404 is between 0.8 and 0.9) and double-precision floating-point variables. Programs longer than 50 lines will receive no credit.

- c. Repeat part (b) for Jacobson's algorithm (Jacobson-Karels in P&D, defined on p. 405-406), using  $\delta = 0.15$ .

Sol

- a. An acknowledgement does not necessarily acknowledge the most recent transmission of a packet. Therefore, any ACKs that are received for a packet that was transmitted more than once are not used in calculating the estimated RTT.
- b. A program that calculates the number of retransmissions for either the original TCP timeout algorithm or Jacobson's algorithm is:

```
#define ALPHA 0.85
#define DELTA 0.15

int main(void)
{
    double estimate = 0.0;
    double lastRTT, deviation;
    int retranscount = 0;
    int i;

    srandom (time(NULL));
    estimate = get_rtt();
    deviation = fabs(get_rtt() - estimate);

    for(i = 0; i < 1000000; i++) {
        lastRTT = get_rtt();
#ifdef ORIG
        if(lastRTT > 2.0*estimate)
            retranscount++;
        estimate = ALPHA*estimate + (1 - ALPHA)*lastRTT;
#else
        if(lastRtt > estimate + 4.0*deviation)
            retranscount++;
        deviation = (1 - DELTA)*deviation + DELTA*(fabs(lastRTT - estimate));
        estimate = DELTA*estimate + (1 - DELTA)*lastRTT;
#endif
    }
    printf("%d retransmissions\n", retranscount);
}
```

## 5. Adaptive Retransmission, Part II

Execute your code from problem 4 parts (b) and (c) using the `get_rtt()` function given below and report the frequency of unnecessary retransmissions as a percentage of packets (rounded off to an integer).

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
double
get_rtt () {
    static int init = 0;
    if (!init) {
        init = 1;
        srandom (time (NULL));
    }
    if ((random () % 100) < 18)
        return 500.0;
    return 100.0;
}
```

sol

The frequency of retransmissions for Jacobson's algorithm is much less as compared to Karn-Partridge algorithm (typically 15-20% as compared to 1-5% for karn-partridge).

## 6. Fair Queueing

A router implements fair queueing of four incoming flows over a single outgoing channel. In this problem, you will determine the order of packets sent out from the router and calculate statistics from your results. Assume for simplicity that the problem starts at time  $t = 0$  and that sending a packet of length  $N$  requires  $N$  time units. Packets arrive on the four flows, named A, B, C, and D, as follows:

- A packets of length 10 arrive at times 0, 5, 10, 15
- B packets of length 8 arrive at 25, 45, 65, 85, 105
- C packets of length 5 arrive at 10, 20, 30, 40, 50, 60, 70, 80
- D packets of length 4 arrive at 1, 2, 30, 31, 32, 33, 80, 81, 82, 83

All service counters start at 0 at time 0.

- a. For each packet sent on the outgoing link, record the start time, the service counter for each flow at that time, and the name of the packet (*e.g.*, write *B3* for flow B's third packet).

Packet	A1	C1	D1	D2	C2	B1	D3	A2	B2	C3	D4	C4	B3	D5	A3	C5	D6	B4	D7	C6	D8	A4	C7	B5	D9	C8	D10
Time	0	10	15	19	23	28	36	40	50	58	63	67	72	80	84	94	99	103	111	115	120	124	134	139	147	151	155
Si	10	5	4	8	10	8	12	20	16	15	16	20	24	20	30	25	24	32	28	30	32	40	35	40	36	40	40
Delay	0	0	14	17	3	3	6	35	5	28	32	27	7	48	74	44	66	18	31	55	39	109	64	34	65	71	72

- b. Calculate the percentage of the link used by each flow up to time 100. Was the link fairly distributed? Why or why not?  
**A: 30%, B: 24%, C: 25%, D: 21%. Yes, the link was fairly distributed, though D was hurt by its burstiness.**
- c. Calculate the average packet delay for each flow—the average difference between the arrival time of the packets and the time that they begin to be sent on the outgoing link. Explain the differences in the averages in terms of the burstiness of the arrivals.  
**A: 54.5, B: 13.4, C: 36.5, D: 38.9. B seems to have done the best because of lack of burstiness, long separations between packets, and medium packet size. A was hurt by big packets, and slightly bursty arrival. D was also hurt by a large amount of burstiness.**
- d. Repeat parts (a), (b), and (c) using round-robin scheduling. Skip queues that are empty on their turn. Explain the differences in utilization and delay in comparison with fair queueing.
- a.

Packet	A1	C1	D1	A2	B1	C2	D2	A3	B2	C3	D3	A4	B3	C4	D4	B4	C5	D5	B5	C6	D6	C7	D7	C8	D8	D9	D10
Time	0	10	15	19	29	37	42	46	56	64	69	73	83	91	96	100	108	113	117	125	130	134	139	143	148	152	156
Si	10	5	4	20	8	10	8	30	16	15	12	40	24	20	16	32	35	20	40	30	24	35	28	40	32	36	40
Delay	0	0	14	14	4	17	40	36	11	34	39	58	18	51	65	15	73	81	12	65	97	64	59	63	67	70	73

- b. **A: 40%, B: 24%, C: 20%, D: 16%. No, the link was not fairly distributed. A had all of its arrivals very early and they were all fairly long. The round-robin algorithm wasn't able to take fairness into account.**
- c. **Once again, D was hurt by the burstiness of its arrivals. A and C were slightly less bursty, but A went first and only had four packets.**

## 7. nslookup Network Utility

Use the ews machines for this problem. Show the commands that you use to solve the problem and the output you get. No credit if you don't show your work. The nslookup utility allows you to query domain name servers for the internet. Review the section of the text on domain name servers and read the man page for nslookup. You can enter the interactive mode of nslookup by simply typing nslookup, and from there you can type ? for a list of commands.

- a. Find the names of the nameservers known to cs.uiuc.edu.
- b. Find the canonical name and IP address for the machine with alias www.cs.uiuc.edu.
- c. Find the names of the mailserver(s) for the machine with alias www.cs.uiuc.edu

Sol

```
a. > nslookup
nslookup
Server:          130.126.161.184
Address:         130.126.161.184#53
> set type=NS
> cs.uiuc.edu
Server:          130.126.161.184
Address:         130.126.161.184#53

Non-authoritative answer:
cs.uiuc.edu      nameserver = dns1.cso.uiuc.edu.
cs.uiuc.edu      nameserver = dns2.cso.uiuc.edu.
cs.uiuc.edu      nameserver = dcs-ns1.cs.uiuc.edu.
cs.uiuc.edu      nameserver = dcs-ns2.cs.uiuc.edu.
cs.uiuc.edu      nameserver = dns1.iu.edu.

Authoritative answers can be found from:
dns1.iu.edu      internet address = 134.68.220.9
dns1.cso.uiuc.edu internet address = 128.174.5.103
dns2.cso.uiuc.edu internet address = 128.174.5.104
dcs-ns1.cs.uiuc.edu internet address = 128.174.252.5
dcs-ns2.cs.uiuc.edu internet address = 128.174.252.4
```

```
b. > set type=CNAME
> www.cs.uiuc.edu
Server:          130.126.161.184
Address:         130.126.161.184#53

Non-authoritative answer:
*** Can't find www.cs.uiuc.edu: No answer

Authoritative answers can be found from:
cs.uiuc.edu
    origin = dcs-ns2.cs.uiuc.edu
    mail addr = hostmaster.cs.uiuc.edu
    serial = 2009040702
    refresh = 7200
    retry = 300
    expire = 604800
    minimum = 7200
```

```
c. > set type=MX
> www.cs.uiuc.edu
Server:          130.126.161.184
Address:         130.126.161.184#53

Non-authoritative answer:
*** Can't find www.cs.uiuc.edu: No answer

Authoritative answers can be found from:
cs.uiuc.edu
    origin = dcs-ns2.cs.uiuc.edu
    mail addr = hostmaster.cs.uiuc.edu
    serial = 2009040702
```

```
refresh = 7200  
retry = 300  
expire = 604800  
minimum = 7200
```