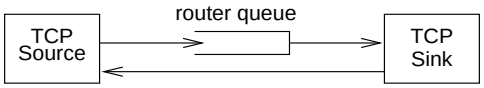
cmsc 417 S04 Exam 1 SOLUTION name: _____

• Total points 30. Total time 70 mins. 3 problems over 3 pages. **No book, no notes, no calculator.**

• Sign here _____ to have your exam scores listed on web by last five digits of your SID.



1. [10 pts] This problem deals with TCP congestion control using the *packet-level* model in Note 5 and the scenario shown above. Assume the following:

- The router link shown above on the source-to-sink path takes D seconds to transmit a packet and maximum buffer capacity of 3 packets. This link is the sole bottleneck of the TCP connection (i.e., the roundtrip time (rtt) equals the delay encountered in this link and overflow at the link buffer is the only cause of packet loss). No other traffic uses the link. The link buffer is initially empty.
- The source implements TCP Tahoe congestion control (slow-start, additive increase, multiplicative decrease), except that rto is constant at $4D$ seconds. It uses a single rto timer that restarts whenever it sends a packet. Packet payload is 1 KB. Source has initial cw (congestion window size) of 1 packet and initial sst (slow-start threshold) of 6 packets.
- The sink sends an ack for every packet received. It can buffer an unlimited number of packets.

Complete the table below for a 10 KB file transfer, assuming the file consists of chunks $0, 1, \cdots, 9$. The row at time jD, for $j = 0, 1, \cdots$, should indicate just after this time the values of nr, na, cw, sst, the 1KB chunks sent, and the chunks in the queue. The first two rows are filled in for your convenience. Also indicate the time when the transfer is completed (fully acked at the source).

SOLUTION

time	nr (bffrd)	na	cw	sst	chunks sent	queue	
$0D$	0	0	1	6	0	0	
$1D$	1	1	2	6	1,2	1,2	
$2D$	2	2	3	6	3,4	2,3,4	
$3D$	3	3	4	6	5,6	3,4,5	6 lost
$4D$	4	4	5	6	7,8	4,5,7	8 lost
$5D$	5	5	6	6	9	5,7,9	
$6D$	6	6	6+1/6	6	-	7,9	
$7D$	6 (7)	6	6+1/6	6	-	9	
$8D$	6 (7,9)	6	6+1/6	6	-	-	
$9D$	6 (7,9)	6	1	3	6	6	timeout
$10D$	8 (9)	8	2	3	8,9	8,9	
$11D$	10	10	3	3	-	9	

Grading:

- 6 points for pre-timeout evolution including timeout time, distributed roughly as follows: 2 for nr,na, cw,sst values; 2 for chunks sent and queue content; 2 for correct timeout instant.
- 3 points for post-timeout evolution, distributed roughly as follows: 1 for nr,na; 1 cw,sst values; 1 for chunks sent and queue content
- 1 point for completion time
- 0 points for using window-level model
- max 4 points if there is no timeout
- max 8 points if you used TCP Reno (fast recovery/retransmit).

2. [10 pts] Consider the same scenario as problem 1, but with the following changes:

- Use the *window-level* model from Note 5 (not the packet-level model).
- Rtt is constant at T seconds and rto is just slightly higher than rtt.
- A very large file of size F KB is to be transferred (so initial and final transients can be ignored).

Obtain the following as a function of F and T (the skeleton table below is for your convenience):

a. Transfer time (from start until everything is acked).

b. Goodput: number of packets delivered to sink user per second.

b. TxRate: number of packets transmitted by TCP source per second.

SOLUTION

time	nr	na	cw	sst	chunks sent	queue	
$0T$	0	0	1	6	0	0	
$1T$	1	1	2	6	1,2	1,2	
$2T$	3	3	4	6	3,4,5,6	3,4,5	6 lost
$3T$	6	6	1	2	6	6	
$4T$	7	7	2	2	7,8	7,8	
$5T$	9	9	3	2	9,10,11	9,10,11	
$6T$	12	12	4	2	12,13,14,15	12,13,14	15 lost
$7T$	15	15	1	2	15	15	
⋮							

Rows $3T$, $4T$, $5T$, $6T$, repeat every $4T$ seconds. In this duration, nr increases by $9(= 15 - 6)$ and 10 packets are transmitted by the source.

So goodput is $9/(4T)$ packets/second.

Transfer time is $(4FT)/9$ seconds.

TxRate is $10/(4T)$ packets/second.

Grading:

- 7 points for evolution, distributed roughly as follows: 2 for nr/na; 2 for chunks sent/queue content; 2 for timeout.
- 3 points for formulas of goodput, transfer time, TxRate.
- 0 points for using packet-level model.
- 0 points for having no timeouts.
- -3 points if rtt is based on number of packets sent.
- -3 points if cw increases by fractions (e.g., 1/3, 2/3).
- -2 points if queue is not empty at start of each row.

3. [10 pts] This problem deals with the sliding-window data transfer protocol in Note 4. Assume the following:

- Cyclic sequence numbers take values from $0, \cdots, N-1$. Datablock size is one byte. A data messsage has exactly one byte.
- The sink entity implements the receive buffer with a circular array `rbuff` of `RW` bytes and another circular array `rfull` of `RW` booleans (to mark the non-empty entries of `rbuff`).

Specifically, the sink has the following variables:

```
int nr ;                // (as usual) unbounded seq num of next byte to be received in sequence
int nd ;              // (as usual) unbounded seq num of next byte to be delivered to sink user
byte[RW] rbuff ;                                    // array of bytes representing receive buffer
boolean[RW] rfull ;                                 // rfull[j] is true iff rbuff[j] is not empty
Initially, nr = nd = 0  and  rfull[j] = false for j = 0, ..., RW - 1.
```

"Circular" buffer means that the sink stores byte `k` in `rbuff[k mod RW]`. So, for example, `rbuff[nd mod RW]` holds the byte to be next delivered to the user (if `nd` < `nr` and `rfull[nd mod RW]` is true).

Give the code for the receive (`D`, `data`, `cn`) event, where `data` is the received data byte and `cn` is the accompanying cyclic sequence number. Your code must not exceed 15 lines in total. Program elegance counts.

SOLUTION

```
Rec(D, data, cn) {
  int j := (cn-nr) mod N ;                                    3 points
  if j < RW - (nr - nd) and (not rfull[nr+j mod RW]){         3 points
      rbuff[nr+j mod RW] := data ;                            2 points
      rfull[nr+j mod RW] := true ;
      if j = 0 then {                                         2 points
          while (nr < nd + RW and rfull[nr mod RW]) do {
              nr := nr+1 ;
          }
      }
  }
  Send(ACK, (nr mod N))
}
```

Grading:

- 3 points for obtaining the offset wrt `nr` (i.e., `j` := (`cn` − `nr`) `mod N`). Max 1 point if modulo taken wrt `RW`.
- 3 points for determining whether data is in receive window (i.e., the test `j` < `RW` − (`nr` − `nd`)), distributed as follows: 2 points for noting that the current receive window size `rw` equals `RW` − (`nr` − `nd`); 1 point for the check `j` < `rw`; you do not lose points for not checking whether data is already buffered (i.e., omitting (`not rfull[nr + j mod RW]`)).
- 2 points for storing data and updating rfull, distributed as follows: 1 point for storing to the correct location (i.e., `nr` + `j mod RW`, or equivalently, `cn mod RW`); 1 point for storing data in a location of `rbuff` and setting the corresponding `rfull` location to true.
- 2 points for updating `nr`, distributed as follows: 1 point for checking that `rfull[nr mod RW]` is true; 1 point for checking that `nr` < `nd` + `RW` (not stepping outside `rw`).