Samantha Rack
CSE 30264
HW 9

1. When a video application runs UDP and does not implement a congestion control algorithm, then its sending rate is not at all adjusted based on feedback of lost packets as is the case for TCP with its congestion control. Since this application does not follow a congestion control algorithm, the UDP packets are sent out at the same rate despite the amount of congestion that may be on the network. So as the network gets busy, though packets may be lost, the application continues to use the same bandwidth. Any TCP traffic using the same links as this UDP traffic have congestion control implemented with the protocol. When the network begins to become congested, the UDP packets continue to be sent at the same rate, and the TCP traffic is affected by the congestion control. The TCP traffic will be overrun by the UDP traffic.

   The RTCP reports can be used in the same way that the feedback of ACKs is used for TCP congestion control. Whenever an RTCP packet is received at the source, the host can evaluate the congestion of the network and adjust the amount of data that should be sent without congesting the network. With this implementation, the video application will have an effective congestion control that will prevent it from using all of the bandwidth and overrunning the TCP.

2. Bucket is initially full at size B. For each second, B must be greater than or equal to zero at each second of the transmission schedule. The maximum size is B.

   (a) Token rate: 2 packets/s

   0: B – 8
   1: (B – 8) + 2 – 4 = B – 10
   2: (B – 10) + 2– 1 = B – 9
   3: (B – 9) + 2 – 0 = B – 7
   4: (B – 7) + 2 – 6 = B – 11
   5: (B – 11) + 2 - 1 = B – 10
   minimum with (B-11) in the bucket, **minimum bucket size = 11**

   (b) Token rate: 4 packets/s

   0: B – 8
   1: (B – 8) + 4 – 4 = B – 8
   2: (B – 8) + 4 – 1 = B – 5
   3: (B – 5) + 4 – 0 = B – 1
   4: (B – 1) + 4 – 6 = B – 6 //note that the first part of the operation (B – 1 + 4)
   5: (B – 6) + 4 – 1 = B - 3 //cannot exceed B (the maximum bucket size)
   minimum with (B – 8) in the bucket, **minimum bucket size = 8**

3. employee0.name = “RICHARD” // size = 7
   employee0.ssn = 4367
   employee0.hireday->month = “DECEMBER” //size = 8

employee0.hireday->day = 2
employee0.hireday->year = 1998
employee0.salary_history = {80000, 85000, 90000, 0, 0}
employee0.num_raises = 2

7 RICHARD 4267 8 DECEMBER 2 1998 3 80000 85000 90000 2

This response assumes that only the nonzero salaries should be sent, so salary_history is a variable length array (therefore we send 3 before the values in the array)

4. Each is a 32 bit encoding (4 bytes). ASN.1 is tagged with a triple of (type, size in bytes, data)

(a) 101: (**int, 4, 101)**

(b) 10120: **(int, 4, 10120)**

(c) 16909060: **(int, 4, 16909060)**

5. Using gzip for compression.
   - C File: ftp_server_lib.h
     Original size: 8230
     After compression: 2512
     3.28 : 1
   - Compiled Binary: myftp
     Original size: 16685
     After compression: 6354
     2.63 : 1
   - Bitmap image file: dk.bmp
     Original size: 540054
     After compression: 163116
     3.31 : 1
   - Text file generated with a csh script: file.txt
     Original size: 49000
     After compression: 233
     210.3 : 1
     The csh script uses echo in a for loop to insert the same string into a file 1000 times. Because there is so much redundancy in the file, it is able to be compressed very well.

6. (a) Encodings:

a – 0	1
b – 10	2
c – 110	3
d – 111	3

When the data is being read, if the first bit is a 0, then this constitutes the end of the letter and it is determined to be an a. If the first bit in a 1, then it is a b, c, or d. In this case, if the second bit is a 0, then this constitutes the end of the letter (at 2 bits) and it is a b. If the second bit is a 1, then it is either c or d, and this is determined based on the third bit. Once a pattern is matched, then the process begins again, decoding the next letter.

(b) N is the number of letters being encoded.

Original = 2*N

Compressed = 1*(.5*N) + 2*(.3*N) + 3*(.2*N) = 1.7*N

2/1.7

Compression achieved is 1.18 to 1.

(c) Encodings:

a – 0	1
b – 10	2
c – 110	3
d – 111	3

Though this is the same encoding as above, there is no further way to optimize it. a and b have the same usage percentage, but they both cannot be encoded with one bit because we must be able to encode the other two letters as well.

Original = 2N

Compressed = 1(.4N) + 2(.4N) + 3(.15N) + 3(.5N) = 1.8N

2/1.8

Compression achieved is 1.11 to 1.