

## Problem 1

Suppose Bob joins a BitTorrent torrent, but he does not want to upload any data to any other peers (so called free-riding).

- (a) Bob claims that he can receive a complete copy of the file that is shared by the swarm. Is Bob's claim possible? Why or why not?
- (b) Bob further claims that he can further make his "free-riding" more efficient by using a collection of multiple computers (with distinct IP addresses) in the computer lab in his department. How can he do that?

Write your solution to Problem 1 in this box

1

- a. Yes, Bob's claim is possible, he can still receive a complete copy of the file as long as there is still enough people in the swarm to share data with Bob. The reason is because in the implementation of BitTorrent, you don't have to upload data for others, but it is recommended to share with others as well, as you can get a couple benefit from doing so, such as higher upload rate will find better partner, and get the file faster.
- b. Yes, Bob can do free-riding for all of his computer and can access set of chunk from each of client by unchocking their neighboring peers, and even he can choose his file subset by making the host ask about the request.

## Problem 2

Suppose you have a new computer just set up. `dig` is one of the most useful DNS lookup tool. You can check out the manual of `dig` at <http://linux.die.net/man/1/dig>. A typical invocation of `dig` looks like:  
`dig @server name type`.

Suppose that on April 19, 2017 at 15:35:21, you have issued “`dig google.com a`” to get an IPv4 address for `google.com` domain from your caching resolver and got the following result:

```
; <<>> DiG 9.8.3-P1 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17779
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 4

;; QUESTION SECTION:
;google.com.                IN      A

;; ANSWER SECTION:
google.com.                 239     IN      A      172.217.4.142

;; AUTHORITY SECTION:
google.com.                 55414   IN      NS      ns4.google.com.
google.com.                 55414   IN      NS      ns2.google.com.
google.com.                 55414   IN      NS      ns1.google.com.
google.com.                 55414   IN      NS      ns3.google.com.


;; ADDITIONAL SECTION:
ns1.google.com.            145521  IN      A      216.239.32.10
ns2.google.com.            215983  IN      A      216.239.34.10
ns3.google.com.            215983  IN      A      216.239.36.10
ns4.google.com.            215983  IN      A      216.239.38.10

;; Query time: 81 msec
;; SERVER: 128.97.128.1#53(128.97.128.1)
;; WHEN: Wed Apr 19 15:35:21 2017
;; MSG SIZE rcvd: 180
```

- What is the discovered IPv4 address of `google.com` domain?
- If you issue the same command 1 minute later, how would “ANSWER SECTION” look like?
- When would be the earliest (absolute) time the caching resolver would contact one of the `google.com` name servers again?
- If the client keeps issuing `dig google.com A` every second, when would be the earliest (absolute) time the caching resolver would contact one of the `.com` name servers?

Write your solution to Problem 2 in this box

2.

- a. 172.217.4.142
- b. google.com.            179    IN    A    172.217.4.142
- c. The number next to google.com in ANSWER SECTION is the TTL, which if we keep use dig command, the number will decrease. Thus, when TTL reach 0, then it needs to contact the server again. After 239 seconds or April 19 15:39:20 2017
- d.  Until the cache cleans, which is 5514 seconds, or April 20 06:58:55 2017

!

### Problem 3

The sender side of *rdt3.0* simply ignores (that is, takes no action on) all received packets that are either in error or have the wrong value in the acknum field of an acknowledged packet. Suppose that in such circumstances, *rdt3.0* were simply to retransmit the current data packet. Would the protocol still work? (Hint: Consider what would happen if there were only bit errors; there are no packet losses but premature timeouts can occur. Consider how many times the *n*th packet is sent, in the limit as *n* approaches infinity).

Write your solution to Problem 3 in this box

3. When *rtd3.0* sends a package, it will have an acknowledgement whether or not the receiver receive the package. When the receiver sends an error message or packet lost, the sender will retransmit the file again and again until it is received by the receiver. By doing a retransmission again and again, it will lead to premature timeout, which is sending too many unnecessary packages, and make the congestion on the network traffic. It may still work sometimes, but it is not efficient and sometimes the package that sent may be wrong due to the collision on the retransmission of several packages.

## Problem 4

Consider a reliable data transfer protocol that uses only negative acknowledgments. Suppose the sender sends data only infrequently. Would a NAK-only protocol be preferable to a protocol that uses ACKs? Why? Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

Write your solution to Problem 4 in this box

4. In NAK, it will only recognize the data lost when the subsequent file is lost. For example, suppose the sender sent package a,b, and c (must be in order). Then receiver only receives a and c, then it will notify the sender that package b is gone. In the first case, where data is sent infrequently, NAK only approach will not be suitable because it will realize a data is lost when it receives the next data (in the example, we will know that the data is lost when c is received). Then what if C is not being sent immediately after B? Then we will have a long delay to realize that the package b has already lost.

In the second case, where sender has a lot of data to send, and experiences few losses, then NAK would be more preferable than ACK because since the data is sent frequently, then we will recognize the lost data faster, and we even barely send NAK to the sender, as we experience a few loss only. Moreover, we don't need to send a notification (ACK) every time data receives.

## Problem 5

Suppose an application uses *rdt3.0* as its transport protocol. As the stop-and-wait protocol has very low channel utilization (shown in the lecture), the designers of this application let the receiver keep sending back a number (more than two) of alternating ACK 0 and ACK 1 even if the corresponding data have not arrived at the receiver. Would this application design increase the channel utilization? Why? Are there potential problems with this approach? Explain.

Write your solution to Problem 5 in this box

5. Yes, the channel utilization will increase. As, the receiver is keep sending ACK 0 or ACK 1 alternatively, the sender may keep sending data even though the data has not yet received by the receiver. However, in rdt3.0 the channel utilization increases as it used as pipeline data in channel.

Yes! There is a potential problem with this approach, such as data loss. For example, when the sender send the data, whether or not the receiver receive it, it will send the acknowledgement to the sender, which may cause problems. Such as, the sender think that the data received correctly and won't retransmit the data. In addition, when the sender sends the next data, the receiver will miss the previous data. Unless the design handle the sequence number correctly, then this design should be okay.