Assignment 2

EECS325 Spring 2019
Yue Shu
Due: Tuesday, March 5, 2019

---

1. Under what circumstances can the HTTP response have empty body?

The HTTP response can have an empty body when there is no content to be sent in the response payload body.
For example, HTTP response with `HTTP status 204` could have an empty body.

---

2. Suppose Alice, with a Web-based e-mail account (such as Hotmail or Gmail), sends a message to Bob, who accesses his mail from his mail server using POP3. Discuss how the message gets from Alice's host to Bob's host. List the series of application-layer protocols that are used to move the message between the two hosts.

Only application-layer protocols are metioned in this problem, so the setup of TCP connection is not included in my answer.

- Alice's web-based email user agent uses HTTP to transfer the mail message from her user agent to her mail server
- The SMTP client side (Alice's mail server) transfers the mail message to the SMTP server side (Bob's mail server) with `SMTP protocol` over TCP connection
- Bob's user agent retrieves the message from his mail server using POP3

---

3. List and explain the two different approaches for DNS resolving.

- **iterated query**:
    - Local DNS server is majorly in charge of generating all the requests to different servers
    - Requesting host sends request to the local DNS server
    - Local DNS server keeps querying other level of servers to search for the IP address if it does not have the cache of the IP address of a certain layer following the sequence below:
        - root DNS server
        - TLD DNS server
        - authorative DNS server
    - more secured because when attacked by hacker, only one way of information is hacked
- **recursive query**:
    - The contacted name servers are in charge of looking for the IP address and sending queries to each other
    - Heavy load at upper level layers: root server, TLD DNS server
    - Not secure since when attacked by hacker, amplification effect will work

---

4. Is it possible for an organization's Web server and mail server to have exactly the same alias for a host name (e.g., foo.com)? What would be the types for the RRs that contain the hostnames of the web and the mail servers?

Yes, it is possible for organization's Web server and mail server to have exactly the same alias for a host name.

- type for RR that contains the host name of the **web server**:
    - `type=A`
- type for RR that contains the host name of the **mail server**:
    - `type=MX`

---

5. Suppose TCP operates over a 1 Gbps link.

a. Assume TCP could utilize the full bandwidth continuously and the average packet size is 60 bytes, how long does it take for the TCP sequence numbers to wrap around completely?

The sequence number field in the TCP header is 32 bits long, so we can have at most 2^{32} packets in total. And the total wraparound time T should be calculated as below:

$$T = \frac{2^{32}\ packets * 60\ byte/packet}{1 \times 10^9\ bit/s} = \frac{2^{32} * 60 * 8}{10^9} \approx 2062\ sec \approx 34\ min$$

b. Suppose an added 32-bit timestamp field increments 1000 times during the wraparound time you found above. How long would it take for the timestamp to wrap around?

Since the timestamp field incremetns 1000 times during the 2062-second wraparound time, the average time would be

$$2062/1000 = 2.062\ sec$$

And to wrap around the timestamp field completely, we may have the expression as below:

$$2^{32} * 2.062 \approx 8856222564\ sec \approx 281\ yr$$

Therefore, it will take around 281 years in total.

---

6. Suppose you have the following 8-bit bytes: 01010011 01100110 01110100.

a. What is the 1s complement of the sum of these 8-bit bytes? (Although TCP and UDP use 16-bit words in computing the checksum, you only need to consider 8-bit based checksum for this problem)

First of all we shall have the sum as below:

$$01010011 + 01100110 + 01110100 = 100101101$$

And to wrap around, we have:

$$100101101 = 00101110$$

And finally, we flip all the bits to have the 1s complement of the sum:

$$00101110 \rightarrow 11010001$$

b. Why use 1s complement of the sum instead of the sum itself as the checksum?

It saves time for the receiver to validate the checksum. By using 1s complement of the sum to get the checksum, the receiver can simply add up all the 16-bit words in the header along with the checksum together and then get the 1s complement of it and use it for validation. Given that there is no bit error in

the packet, the final result should just be a chunk of zero. So as long as the result is not zero, there must be an error.
By using the method described above, the receiver does not have to compare the sums bit by bit to find the error. Instead, it can compare two chunks of numbers all in one, which saves more time in nature.

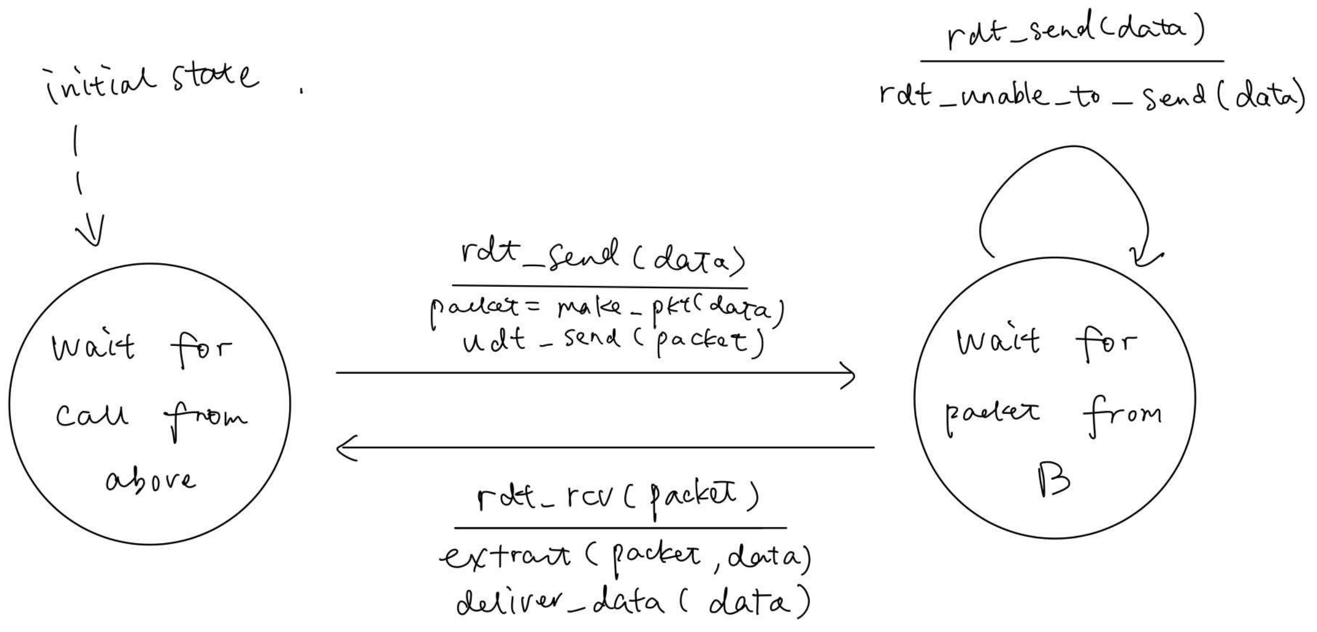
---

7. Finite state machine problem

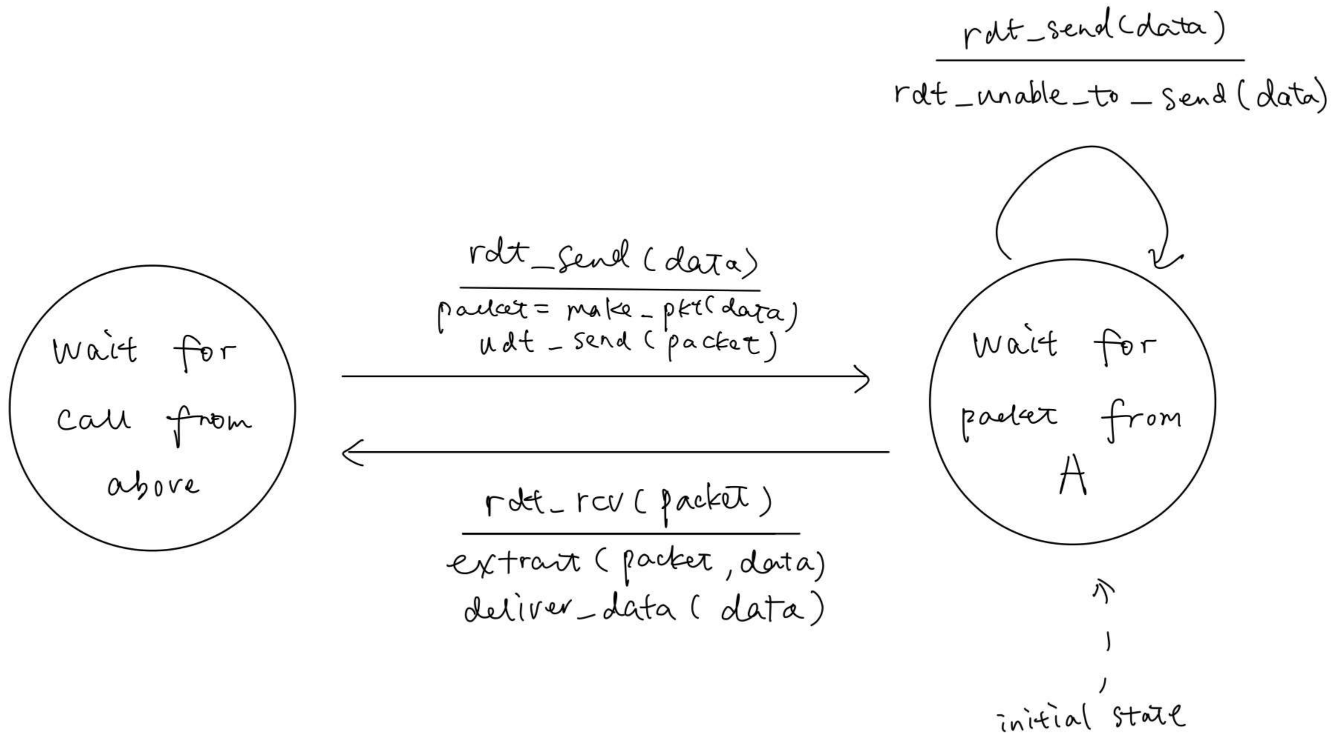
Consider two network entities, A and B, which are connected by a perfect bidirectional channel (i.e., any message sent will be received correctly; the channel will not corrupt, lose, or reorder packets). A and B are to deliver data messages to each other in an alternating manner: First, A must deliver a message to B, then B must deliver a message to A, then A must deliver a message to B and so on. If an entity is in a state where it should not attempt to deliver a message to the other side, and there is an event like `rdt_send(data)` call from above that attempts to pass data down for transmission to the other side, this call from above can simply be ignored with a call to `rdt_unable_to_send(data)`, which informs the higher layer that it is currently not able to send data. [Note: This simplifying assumption is made so you don't have to worry about buffering data.]

Draw a FSM specification for this protocol (one FSM for A, and one FSM for B). Note that you do not have to worry about a reliability mechanism here; the main point of this question is to create a FSM specification that reflects the synchronized behavior of the two entities. You should use the following events and actions that have the same meaning as protocol rdt1.0 in Figure 3.9: `rdt_send(data)`, `packet = make_pkt(data)`, `udt_send(packet)`, `rdt_rcv(packet)`, `extract(packet,data)`, `deliver_data(data)`. Make sure your protocol reflects the strict alternation of sending between A and B. Also, make sure to indicate the initial states for A and B in your FSM descriptions.

- **FSM for A**:



- **FSM for B**:



- What worth our attention is that the initial state for A is the "wait for call from above" state since A should always deliver a message B first.
- For the same reason as above, the initial state for B is the "wait for packet from A" state.
- Also, we don't need to worry about the idle state transition of the "wait for call from above" state since the mechanism of the transition between A and B guarantees that A will never receive a message from B when it shouldn't, and vice versa.