ANSWERS

A12.10 Project: a reliable link

Networks and the web · A level · OCR H446 1.3.3, AQA 7517 4.9.3.1, Eduqas A500QS 2.2 · about 35 min

BugBotLab

What this lesson is about

Receive a route as packets out of order, reject the damaged one, and drive it.

- What the project uses
- The packet format
- How a single bit is caught
- Designing the program

Questions

5 marks in all

1. What does this program print? [1 mark]

```
def checksum(text):
    return format(sum(ord(c) for c in text) % 256, '02X')
print(checksum('1/3:forward 30'))
```

Answer:

```
45
```

The character codes add up to 1093, and 1093 modulo 256 is 69, which is 45 in hexadecimal.

2. A receiver recalculates a packet's checksum and it does not match. What should it do? [1 mark]

- ☐ **A** Discard the packet and ask for it to be sent again
- ☐ **B** Use the packet, because the header is still readable
- ☐ **C** Change the checksum so it matches
- ☐ **D** Stop the whole transfer permanently

Answer: A. A mismatch means the packet was damaged; the protocol recovers by getting a fresh copy, here with a NAK.

3. Which error would the checksum 'sum of character codes modulo 256' fail to detect? [1 mark]

- ☐ **A** Two characters swapped
- ☐ **B** One bit flipped in one character
- ☐ **C** A character missing
- ☐ **D** An extra character added

Answer: A. Swapping characters leaves the sum unchanged; stronger checks such as a CRC detect it.

4. Why does the receiver send an ACK for packets that arrive intact? [1 mark]

- ☐ **A** So the sender can tell a delivered packet from one that was lost and needs sending again
- ☐ **B** So the packet's checksum can be recalculated
- ☐ **C** Because routers need ACKs to find a route
- ☐ **D** To encrypt the next packet

> **Answer:** A. A sender that hears no ACK within its timeout resends, which recovers lost packets as well as damaged ones.

5. Which layer of the TCP/IP stack numbers segments, acknowledges them and resends missing ones? [1 mark]

- ☐ **A** Transport
- ☐ **B** Application
- ☐ **C** Network
- ☐ **D** Link

> **Answer:** A. TCP, at the transport layer, gives reliable delivery over the unreliable packet switched network below.

The task: a reliable link

The robot starts facing forward. The Base sends a route of three packets in the format above, one every half second, out of order. One packet arrives damaged. When the Base hears `NAK <number>`, it sends that packet again. 1. Receive: check `messages()` every 0.1 s. For each packet, recalculate its checksum. If it matches, store the command under the packet's number and send `ACK <number>`. If it does not, print `packet <number> corrupt` and send `NAK <number>`. Keep going until you hold a good copy of every packet, as many as the total in the header says. 2. Act: for each number from 1 to the total, in order, print `run: <command>`, then drive it at speed 50. A command is a direction (`forward`, `backward`, `left` or `right`, where left and right slide sideways) and a whole number of centimetres. 3. Send `DONE`. Do not drive until every packet is held, and read the route from the packets rather than typing it. If you act on the damaged packet, you will end up in the red zone.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

def checksum(text):
    total = 0
    for ch in text:
        total = total + ord(ch)
    return format(total % 256, "02X")

wait(3)
for sender, text in messages():
    print(text)
```

The hint students can ask for: Only a packet whose checksum matches may be stored or acknowledged; a damaged one gets a NAK with its number and is thrown away. Keep listening until you hold every packet the header's total asks for. Do not move until then, and run the commands in sequence number order, not arrival order.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

def checksum(text):
    total = 0
    for ch in text:
        total = total + ord(ch)
    return format(total % 256, "02X")

good = {}
total = None
while total is None or len(good) < total:
    wait(0.1)
    for sender, text in messages():
        if "*" not in text:
            continue
        body, check = text.rsplit("*", 1)
        header, command = body.split(":", 1)
        number, count = header.split("/")
        if checksum(body) == check:
            good[int(number)] = command
            total = int(count)
```

```
            send("ACK " + number)
        else:
            print("packet", number, "corrupt")
            send("NAK " + number)

for n in range(1, total + 1):
    word, cm = good[n].split()
    print("run:", good[n])
    if word == "forward":
        forward(50, distance=int(cm))
    elif word == "backward":
        backward(50, distance=int(cm))
    elif word == "left":
        left(50, distance=int(cm))
    elif word == "right":
        right(50, distance=int(cm))
send("DONE")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.