Drone Co-ordination Mini Project)

Overview)

I have created a program that will allow a user to input a drone ID (4 drones would have IDs from 1-4) as well as x, y and z co-ordinates and it will output appropriate responses to the drone drifting out of the defined axis or if it will collide with another drone if it moves there.

The program will tell the user whether the drone is going to drift out of the 20x20x20 co-ordinate space. The program has 7 possible outputs relating to the co-ordinates: 0, 1, 2, 3, 4, 5 and 6.

0: The drone is within a suitable margin (3-17) and it will not drift out of the co-ordinate space

1: The x co-ordinate is between 18-20 so it must move left to avoid drifting
2: The x co-ordinate is between 0-2 so it must move right to avoid drifting

3: The y co-ordinate is between 18-20 so it must move backwards to avoid drifting
4: The y co-ordinate is between 0-2 so it must move forwards to avoid drifting

5: The z co-ordinate is between 18-20 so it must move down to avoid drifting
6: The z co-ordinate is between 0-2 so it must move up to avoid drifting

The program also stores the position of the drones if they are within the safety margin. This allows the program to tell the user whether moving a drone to a position will cause there to be a collision due to a different drone already being there and it will not store the drone's position until it is chooses an empty position.

0: No collision and the drone can move there
1: There is a collision, and the drone must choose an empty position

Drone.p4)

Headers)

The ethernet header contains the standard information, the destination address, source address and ether type as 48 bits, 48 bits and 16 bits respectively.

The drone header information is all 32 bits and contains the drone ID, x, y and z position, x, y and z actions and whether there is a collision.

The drone ID is needed so that the program knows which drone is being moved.
The x, y and z positions are needed so that the program knows where it is moving.
The xact, yact and zact are needed so that the program can send back a packet telling the user whether the drone is drifting out of the safety margin and which direction the user must move the drone to correct this.
The coll is needed so the program can send back a packet telling the user whether the current drone is moving into a location where an existing drone is placed.

Structures)

The headers structures contain the ethernet_t and drone_t headers.

The metadata structure contains 2 triples of variables that store the x, y and z positions during the registerread() and registerwrite() actions so that the program has sufficient information to remove a drone's previous location when it moves to a new location.

Parser and Checksum)

The parser and checksum are the standard parser and checksum that was also used in calc.p4. The parser maps packets into headers and metadata. The checksum verifies that the incoming packet hasn't been corrupted.

Ingress Processing)

Registers)

There are 2 triples of registers that store the x, y and z coordinates of the drones and the other register stores the IDs at appropriate x, y and z co-ordinates. These are used to maintain an up-to-date record of only the drones current locations and the locations of where the drone is requesting to move to.

Actions)

The send_back() action sends the packet back exactly how calc.p4 did. It uses a temporary variable to swap the source and destination address and sends the packet back out the same port it came through.

There are 9 actions that are used in tables to change the xact, yact and zact according to where the drone is in the co-ordinate space.

The registerread() action is used to get the information from the xco, yco and zco registers and store it in the metadata so that it can be used in the if statements/

The registerwrite() action is used to empty the old position of the drone and update the xco, yco and zco registers with the new positions and then store the old positions in xco, yco and zco.

Tables)

The 3 tables apply the actions depending on where the drone is in the co-ordinate space. If the drone is between 0-2 or 18-20, it is too close to the edge so it must receive an action telling it to move closer to the centre. If it is between 3-17, the drone's position is ok.

Apply)

If statements)

The program first checks if the packet is valid and if not, it drops the packet. It then applies the 3 tables to check whether the drone is drifting. If the drone isn't drifting, it checks whether the drone's requested location is empty. If it isn't, it checks whether the drone occupying the space itself. If both these checks are false, the drone adds to the header coll=1, otherwise it says coll=0.

Egress Processing, Checksum Computation and Deparser)

These are identical to calc.p4's. The egress processing and checksum computation are empty for this program. The deparser sends the packet back.

Drone.py)

First, I created the drone header as a class containing the drone ID, x, y and z position, x, y and z actions and whether there is a collision as integer fields.

In the main loop, there is a while loop asking the user to input the id x, y and z co-ordinates with spaces in between.

This string is then split into a list containing these 4 items.

The packet is created using the destination address and converts the items in the list to integers and places them in the packet header.

The sent packet is then displayed so its easier to see the input co-ordinates but it can also be commented out.

The received packet is stored in the variable resp.

The 3 actions within resp are printed out as well as whether there is a collision. It also prints what each drone action means on the line below.

It also prints out error messages if the drone header is in the wrong format or if there is no packet sent by the p4 program.

Link to directory:
https://github.com/VKing15/CWM-ProgNets.git

(drone.p4, drone.py and Exercise 6.pdf are all within the main CWM-ProgNets folder)