ERRS: Project
Mini-Amazon / Mini-UPS

For this project, you will either be doing “mini-Amazon” (an online store) or “mini-UPS” (a shipping website). If you are doing Amazon, you will have to make your system work with the UPS systems in your interoperability group (IG)—2 groups doing Amazon and 2 groups doing UPS.

1 The “World”

Since you won’t have access to real warehouses and trucks, your code will interact with a simulated world that I wrote. You will connect to the simulation server (port 12345 for UPS, port 23456 for Amazon), and send commands and receive notifications.

The server supports different worlds (identified by a 64-bit number). You may create as many worlds as you want. There is presently no authentication on the worlds, so please only use your own. You can use the init-world program to create a basic world to get started. Later, you can create your own worlds if you want.

Each world is comprised of a Cartesian coordinate grid where “addresses” are integer coordinates (so you will deliver a package to e.g., (2, 4)). The world contains trucks (controlled by UPS) and warehouses (controlled by Amazon). These have to work together to deliver packages.

The messages you can send and receive are in the .proto files (amazon.proto and ups.proto) that I have provided. Notice that all messages either start with A or U to indicate which part they belong to.

The basic flow is that you send an A/UConnect message with the worldid that you want, and receive an A/UConnected response. Once you have received this response, you may send A/UCommands and receive A/Responses. You should not send any other message, nor expect to receive any—all of the details are embedded in the A/UCommands/Responses.

A/UCommands include two common options: simspeed and disconnect. You can adjust the simulation speed (higher numbers make things happen more quickly in the world). Note that the simulation speed is per connection, non-persistent, and only affects future events. If you set disconnect to true in a command, the server will finish processing whatever it is currently working on, then send a response with finished = true, and close the connection. Things will only happen in the world while you are connected to it (so you don’t miss anything), however, if you just close the connection without asking, you might lose an in-flight message.

Amazon Commands details:

buy You can ask for more of a some products to be delivered to a warehouse. Specify item id, description (any text) and the quantity you want. If this product has never been seen before, it will be created. If the product has been seen before, you SHOULD provide the same description (if you use different descriptions for the same product id, the behavior is undefined). NOTE: buying new stock does not involve UPS.
**topack** Pack a shipment for delivery. You will be notified when it is ready. The warehouse that you request to pack the shipment MUST have sufficient inventory (and the inventory will be reduced accordingly).

**load** Load a shipment on to a truck. In order for this to succeed, the shipment MUST be packed (and you must have received a ready notification) AND the truck MUST be at the warehouse, ready to receive the shipment (the shipper must have sent them to pickup and they must have received notification of completion).

**Amazon Response details:**

**arrived** When you buy, you will later get a notification that your orders have arrived. At this time, you should update your records of what is in stock, and may use the goods described in this message to fulfill orders.

**ready** Notification that packing is complete

**loaded** Notification that you have finished loading a shipment onto a truck

**UPS Command details:**

**deliveries** Once a package has been loaded, you can issue this command to send the truck to deliver it to a particular location. Note that you MAY pickup other packages before making deliveries. Once the truck is sent to perform deliveries, it is busy until it completes them all, and cannot be given other commands (but other idle truck can). If you specify multiple deliveries at once, they will be performed in the order you list them in the command.

**pickups** Send a truck to a warehouse to pickup a package. The package need not be ready to issue this command. While the truck is en route, it is busy and cannot be given other commands.

**UPS Response details:**

**completions** You will receive this notification when either (a) a truck reaches the warehouse you sent it to (with a pickup command) and is ready to load a package or (b) a truck has finished all of its deliveries (that you sent it to make with a deliveries command). At this point the truck may be given other instructions. Note that the completion tells you the current location of the truck. If you initialize a world yourself, you will also receive a completion notification with the initial position of each truck as soon as they finish initialization.

**delivered** You will receive this notification when each package is delivered.
2 Bare Minimum Functionality (40 pts)

The first piece of functionality you should aim for is to be able to purchase an item, and have it go all the way through to delivery. Figure 1 illustrates.

For Amazon, this means that you need a web interface on which someone can buy a product (you don’t need a catalog of products yet—you can start with just one “Buy” button and a fixed ‘address’ to deliver to). You then need to go through all the steps to get the package delivered (tell your warehouse to pack it, then when it is ready and a truck has arrived, load it).

For UPS, this means that you need a web interface which will display the shipments that exist, and their status (e.g., created, truck en route to warehouse, truck waiting for package, out for delivery). Note that you will need to create a packageid (aka Tracking Number). These MUST be unique for the world. If you reuse a packageid in the same world, things will go wrong. You need to go through all the steps to deliver the package (send truck to warehouse, wait for Amazon to say its loaded, send it for delivery).
3 Actually Useful (40 pts)

Now that you have the basic functionality, it's time to make it useful.

For Amazon, you should add the following features:

- A searchable catalog of products (you don't need a large quantity, nor real products).
- The ability to specify an address (i.e., (x,y) coordinates) for delivery.
- The ability to specify a UPS account name to associate the order with (optional).
- Provide the Tracking Number for the shipment.

Note: there is no “payment” system—so you can just take a “credit card number” and pretend it is ok or just pretend that everything is free.

For UPS, you should add the following features:

- The ability to enter a tracking number and see the status of the shipment.
- User accounts (with user ids and passwords). If you are logged in, you should be able to do the following to packages you own (your user id was supplied to Amazon when the purchase was made):
  - See a list of all packages that belong to them.
  - See the details of the package (e.g., items inside it)
  - If the package is not yet out for delivery, redirect it to a different address. (Note: if the user loses a race and the package goes out for delivery before you can update it, that is OK, but you need to tell them this).

4 Niceties (20+)

The last 20 points are flexible (you don't have to do all of these), and depend on what you do and how well you do it. You can also make up your own features. Points for all of these (including your own features) will be based on technical sophistication of what you are doing. It is possible to exceed 100 points by having a rich set of well-executed features. (However, it becomes exponentially more difficult to earn points the further about 100 you are).

For Amazon:

- Add use accounts (with user ids and passwords).
  - Remember purchase history.
  - Remember/autofill information (address, UPS id, etc).

- Support an efficient delivery strategy for your UPS partners (e.g., give them the choice of warehouses for orders, etc).
• Ratings and recommendations (maybe personalized by account)?
• Gift Cards
• Any other features you want.

For UPS:

• Allow users to purchase purchase priority delivery (meaning that they “pay” a fee and you pickup their packages sooner/deliver them sooner even if others were first).

• Implement an efficient delivery strategy. Perform some sort of planning on which trucks go to which warehouses for which deliveries, coordinate with Amazon on which warehouse is best for the pickup, etc.

• Create an “admin” view, showing a map of your trucks, and what they are doing, etc.

• Any other features you want.

5 Other Notes

A few other notes:

• You are not graded on UI/UX, however, we it must be at least usable. We recommend making it nice so you can show off your project (but save that for some final polishing).

• Communication between Amazon and UPS is entirely up to everyone in your IG. We make no requirements on the technologies/protocols/specifics used.

• You may use any language or combination of languages you want. However, you will probably want to use C, C++, Python, or Java (or Scala) to deal with the truck/warehouse protocol, as those are the languages supported by Google Protocol Buffers.

• You should have at most ONE Amazon connection and at most ONE UPS connection to a given world at a time.

• I’m giving you the binary of the world sim to run on your own server. This lets you poke around in the database and see what went wrong. To set it up:

```
createdb packagesim
psql packagesim < tables.sql
```

then you can run `sim` (which will sit there quietly if all is well). You can run `init-world` to setup a world. If you look in the database, you will see the following tables (all have wid for world id):

```
deliveries These are the deliveries that trucks are currently doing (have been given a deliveries command, but not yet finished).

shipments Shows all shipments that have been created, and whether or not they are completed. You MUST get to completed = true on your shipments before they are done.

truckhas Shows the which shipments have been picked up by which trucks.

trucks Shows the states of the trucks. 1 = idle, 2 = en route to a warehouse, 3 = invalid, 4= delivering, 5 = waiting for pickup. If statechange is not infinity then the truck will finish what it is doing at the given time.

warehouse The position of each warehouse (by warehouse id: hid)

whincoming When you purchase more items for a warehouse, they get listed in this table until they are delivered (artime shows when they will arrive).

whready Shows what is ready at which warehouse

whstock How much of which item each warehouse has in stock.

- You can use the rst.sh and restock.sh scripts to reset the state of a world and restock it. Resetting the state will make all trucks idle, and clear all shipments, deliveries, warehouse ready state, etc. It will NOT clear warehouse stock or warehouse incoming stock. You can use this if you get your world in a messed up state. Restocking will add 10000 to every item in each warehouse that has fewer than 20000 (you can adjust these in the script).

- I wrote this in a day. It isn’t so friendly when you do things wrong. It will give you an error message, but no emphasis was placed on error message friendliness. This got a morning of testing, so bugs are entirely possible—please let me know (and give me a test case to reproduce it) if you find one. I’ll fix things and release new versions.