BRING COMMUNICATION TO THE STARS

AN IOCAFE SATELLITE PROJECT

ARCHITECTURE BY SOFIE LEHTIKOSKI, A CONTRIBUTOR

Overview

Goal

Create a software that enables easy communication with research satellites, by working with radio amateurs who already have preestablished equipment and communication lines

Main audience

Radio amateurs, satellite researchers, and programmers at large

Released under MIT License

Motivation

This is cool. Space is cool. So why not make something that ties together multiple groups in a practical way to make something happen? Enable some space research?

This powerpoint

Describes main components of project and their roles

Step-by-step how this can be feasibly done (on a high level)

CONCEPT

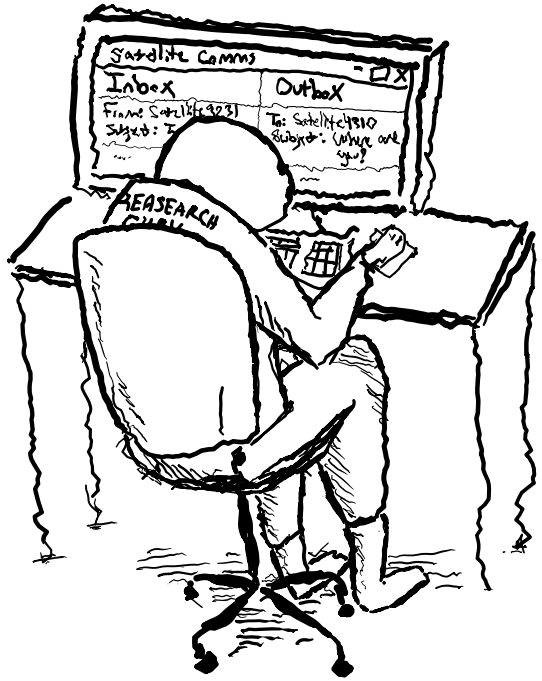
COMPONENTS, DESCRIPTIONS, AND END USER INTERFACE

Components

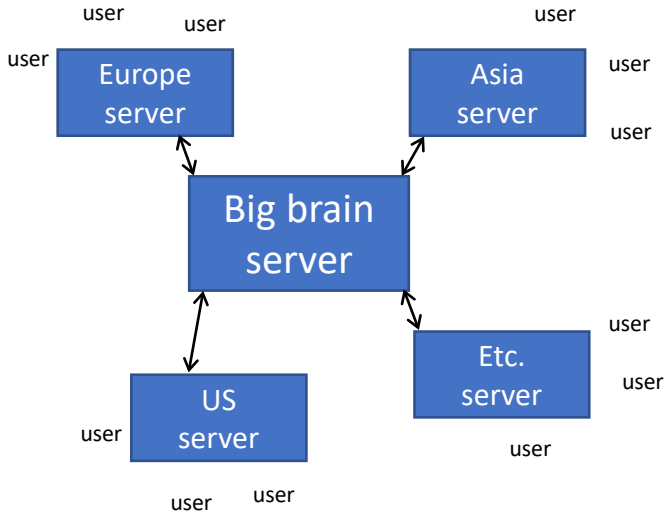
1. Server
   - Manages all calculations and package management
   - Communicates directly with the end user and the radio amateurs
2. Radio Amateurs
   - People with radio equipment in their backyards (or more professional radio hosts)
   - Actually sends and receives signals from satellites and routes them back to the server
   - Abbreviated “Radio AM” or “AM” in this powerpoint
3. Satellites
   - The cool tech we want to communicate with

Components (in reality)

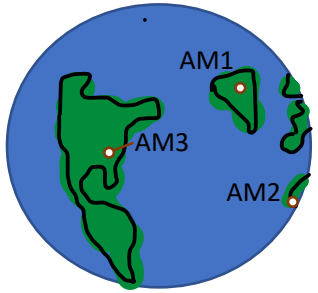
End Users
(typically researchers)



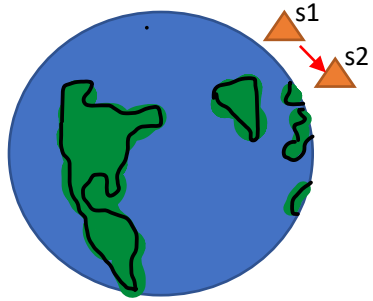
Server
(brain of operation)

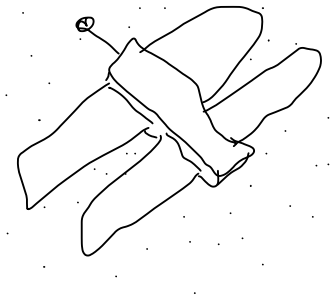


Radio AM
(glue of the operation)

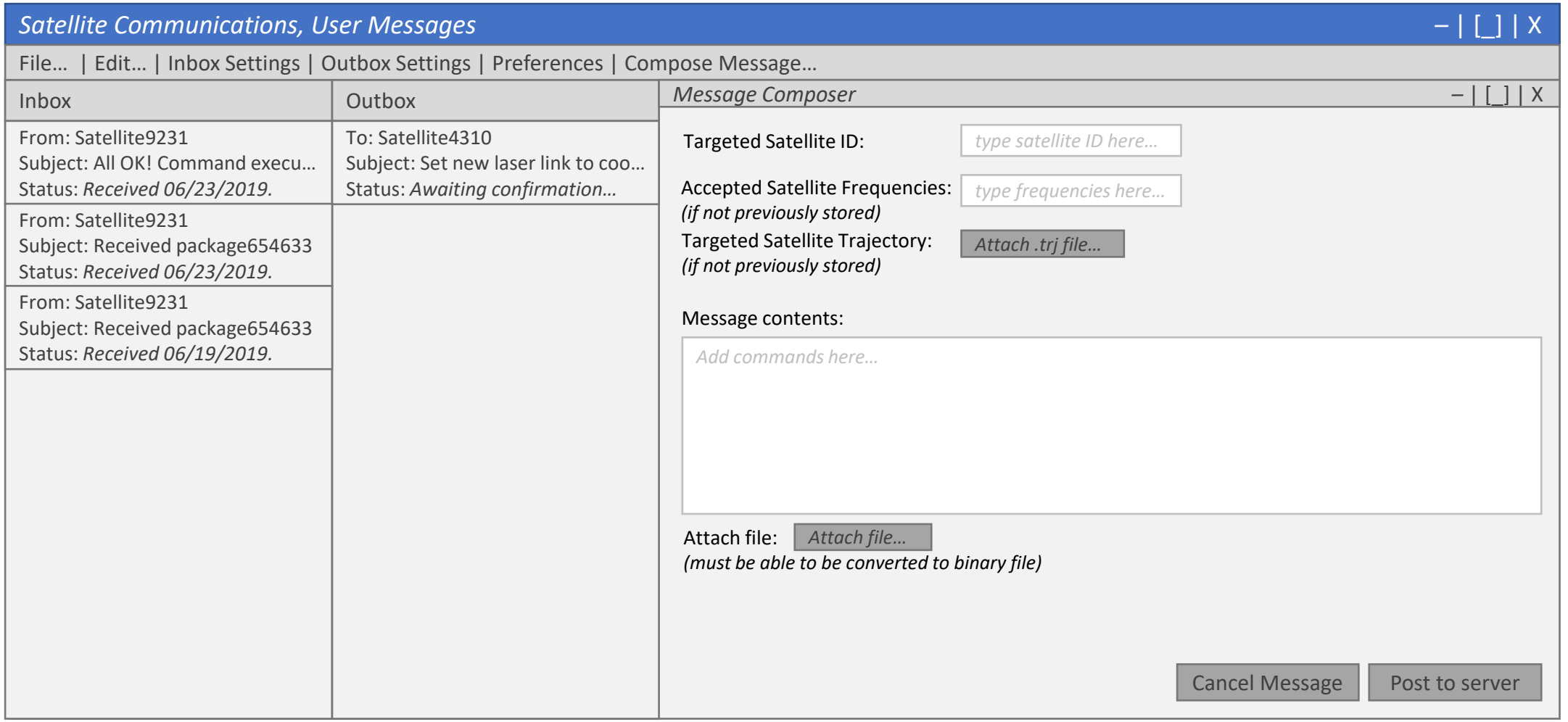


Satellites
(the cool tech we want to communicate with)





End User Interface (mock-up only, simplified)

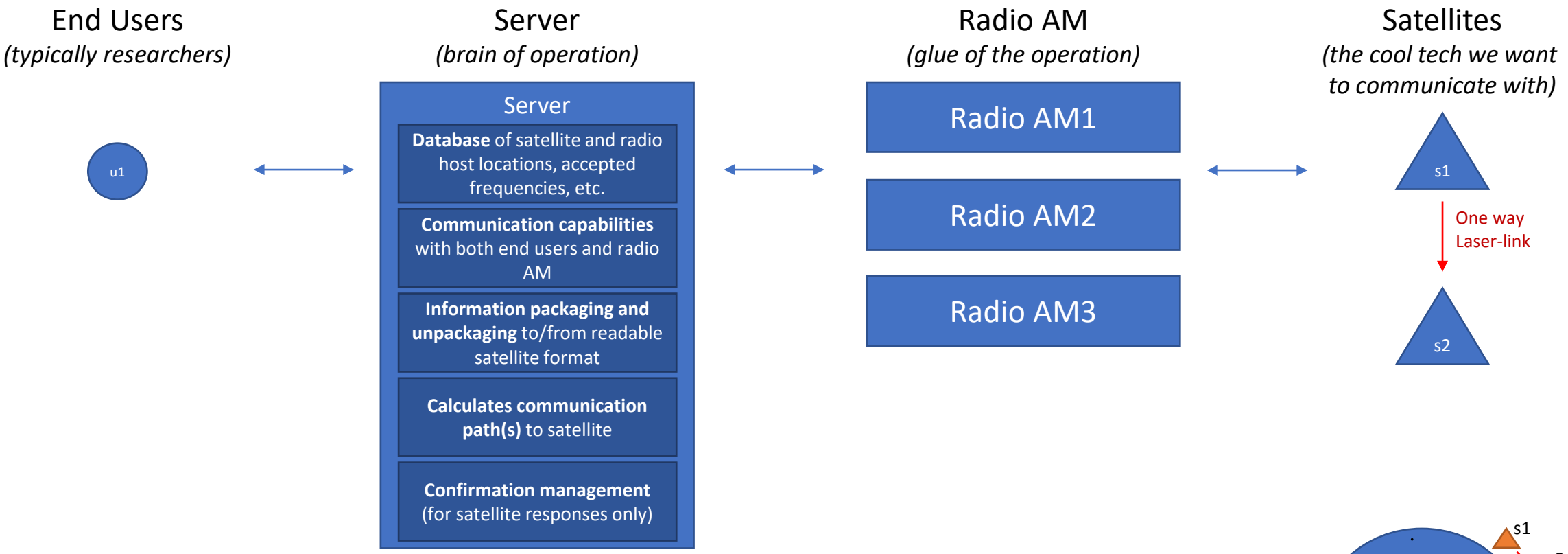


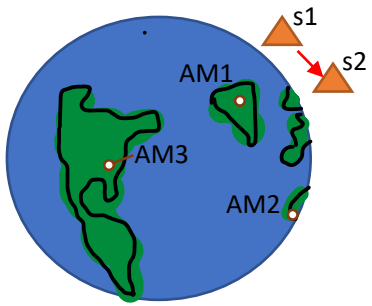
Note: *This mock-up is intended only to show the basic satellite communication idea and necessary inputs for the system. In reality, the GUI will likely look very different.*

ARCHITECTURE

ARCHITECTURE AND HIGH-LEVEL COMMUNICATION STEPS

Architecture





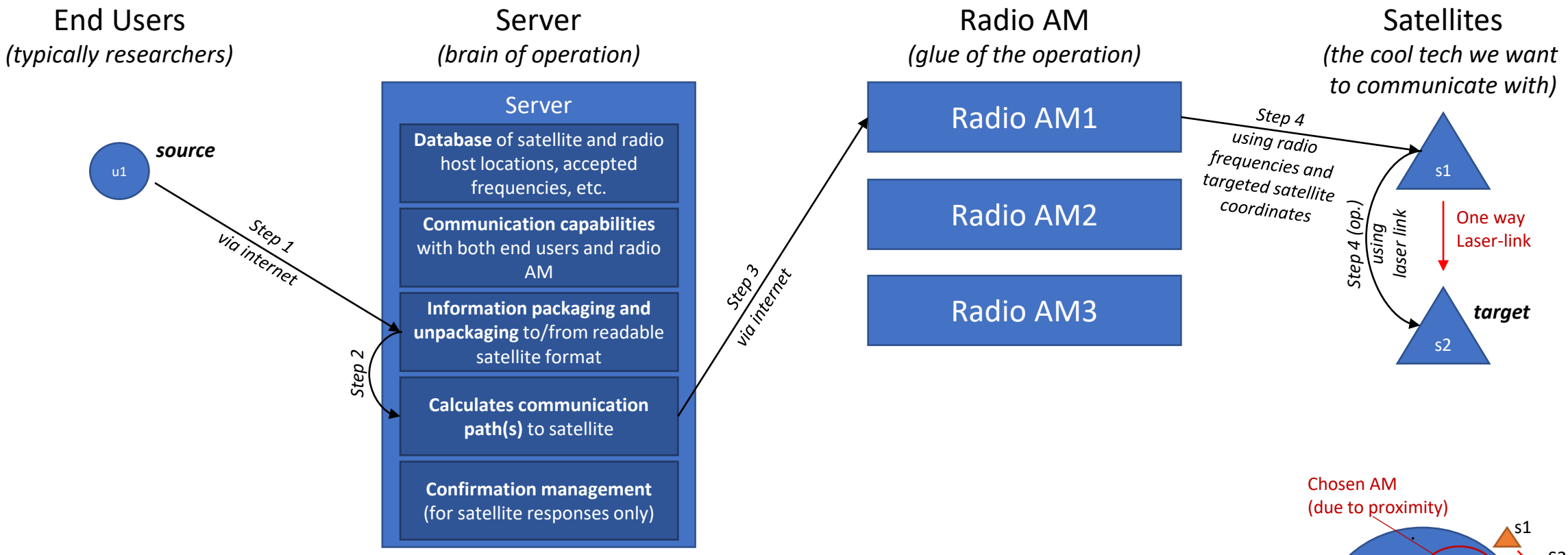
Two Types of Communication

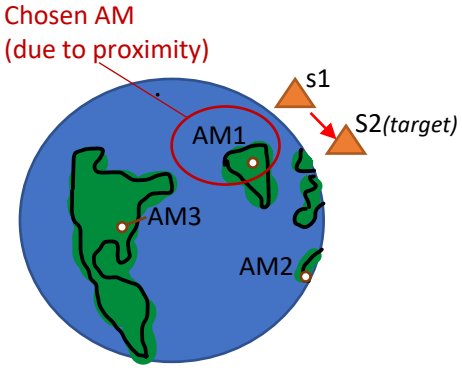
1. User -> Satellite
   - When the user requests the satellite for a status report or to do other commands
   - Message will be handled however the satellite is implemented to handle the given message
     - Not that this software will do no given checks on the message, other than just passing it onwards as shown. See security discussion at end
2. Satellite -> User
   - When the satellite wants to give an automated on on-event status report to the user

User -> Satellite Communication Steps

1. User posts message to server
2. Server packages message and uses database to calculate best path to user-specified satellite
3. Sever sends package to appropriate radio AM, with instructions for any following laser-linked path
4. Radio AM sends package to satellite
5. Satellite, upon receipt of package, begins broadcasting response until a Radio AM tells it the response was received
6. Server repeats #2 and #3 until some radio receives confirmation from the satellite that the package was received
7. Server sends unpackaged confirmation and response back to the end user
8. End user is happy

Steps 1-4. User -> Satellite Communication

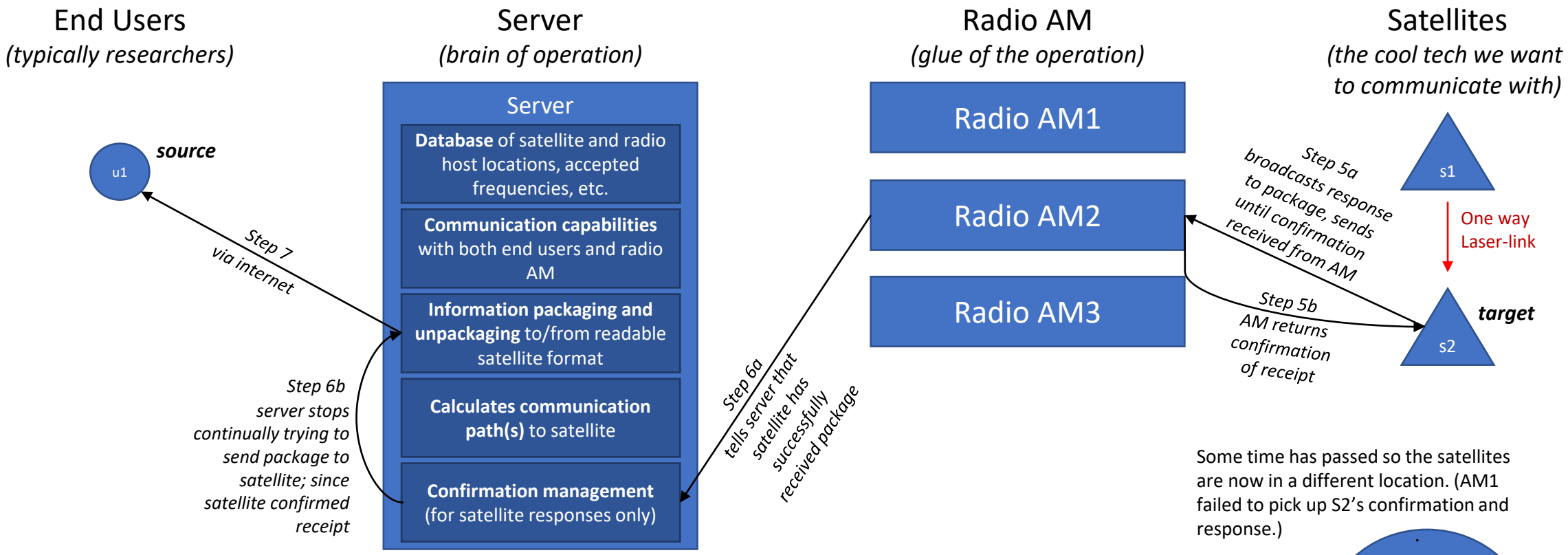




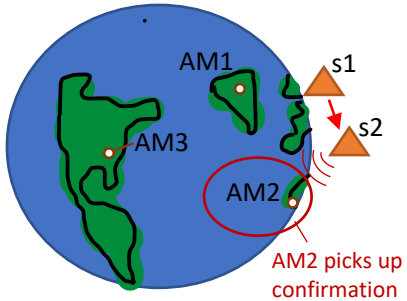
Legend
→ arrows describe data flow

Note: User1 and Satellite2 chosen as source and target, respectively, for example purposes only. (Any user can send information, and any satellite can be chosen – as long as the user has permission/access for that satellite.)

Steps 5-8. User -> Satellite Communication



Some time has passed so the satellites are now in a different location. (AM1 failed to pick up S2's confirmation and response.)



Legend

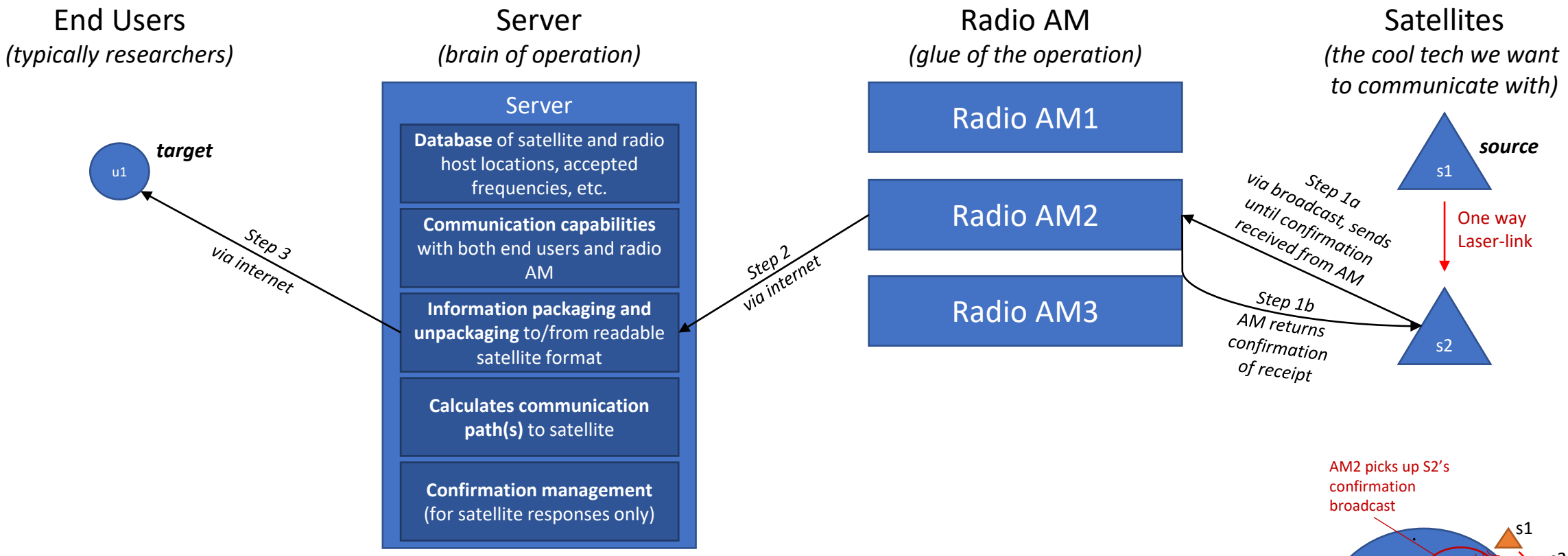
→ arrows describe data flow

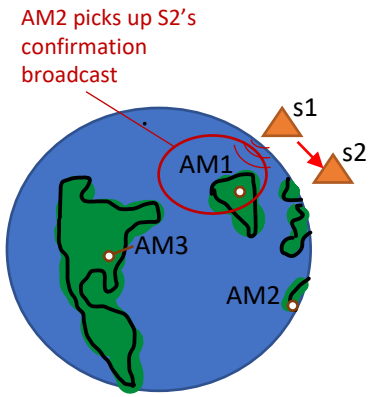
Note: Now Satellite2 is sending confirmation back to User1 that the message was received successfully (target sends confirmation back to the source). This follows the general same process as satellite -> user communication.

Satellite -> User Communication Steps

1. Satellite decided it needs to send a message back to a user, and so broadcasts a package and the recipient's user ID
2. Some radio AM picks up the package and the user ID, sends it back to the server, and gives the satellite confirmation that the package was received
3. The server sends the package, unpackaged, back to the user's inbox as understandable text or binary files
4. The end user is happy, and can reconstruct these files appropriately however they'd like to

Satellite -> User Communication Steps





Legend
→ arrows describe data flow

Note: Satellite1 and User1 chosen as source and target, respectively, for example purposes only. (Any user can send information, and any satellite can be chosen – as long as the user has permission/access for that satellite.)

Messages Not Received:
The Purpose of Confirmation of Receipt

- Confirmation is set up so that if:
  - radio AM -> satellite communication fails (and the Radio AM did try to send out the package), the server will keep sending the package via various radio AMs until the satellite sends back a response
  - satellite -> radio AM communication fails, the satellite will keep broadcasting the package until confirmation of receipt is given from some Radio AM
- Therefore, a Radio AM is always the last one to give confirmation to the satellite before a communication ends
- This allows interfering conditions like clouds, sunspots, etc., any other interfering factors, to slow down a communication, but not halt it entirely

HOW THIS CAN BE DONE IN PRACTICE

IMPLEMENTATION DETAILS

Implementation Overview

- Still in speculation/planning stage, so not much info here
- Server needs to basically be built
  - IOCafe can handle the communication packaging and sending, probably.
  - Some graph theory for finding the simplest/quickest path for sending the message
  - Consistent way of storing the radio AM locations, satellite trajectories, available frequencies for each, etc. in a database
  - Big brain server talks to sub-brain servers which are more locally hosted (for maintainability and faster comms)
- Needs some implementation on Radio AM side to automatically accept and send packages to server
  - Also need to actually talk to radio amateurs too to get their approval and work with them
  - See next slide for more details
- Some abstract support should also be added for the satellite-end communication
  - This support will basically involve two abstract ideas:
    - sending out a broadcast to radio AMs (no actual handling?), upon receipt of a package
    - sending out a readable package, repeatedly, until a confirmation code is received in return from some radio AM
  - Ultimately, however, implementing message handling, hardware-software connection for actual message deployment/receipt, and so on and such is up to whoever deployed the satellite

Server Architecture Psuedocode

Note: *This pseudocode is intended only to get across the idea of what the architecture is doing and a more specific definition of the role of each component. In reality, the code will likely look very different.*

Server

main()

```
while True:
    userSatPackage = getPckgFromAMQueue()
    if userSatPackage.type == 'RESPONSE':
        # Stop trying to send the given package to the
        # satellite; response has been recieved
        stopPckgSending(userSatPackage.packageID)
    sendFromSatToUser(recievedSatPackage)

    sendUserPackage = getPckgFromUserQueue()
    sendFromUserToSatellite(sendToUser)
```

sendFromUserToSatellite(userPackage)

```
# Sends the package to appropriate radioAM, to send
to satellite
satCoords = userPackage.satCoords
satFreqs = userPackage.satFreqs
path = calcRadioAMPath(satCoords,satFreqs)
rAMInstruct = wrapIntoInstruct(
                        path,
                        satCoords, satFreqs,
                        userPackage)
sendInstructionsToRadioAM(rAMInstruct)
```

sendFromSatToUser(satPackage)

```
# Sends the package to the encoded user ID, after
unpacking it
...
```

Radio AM

main()

```
while True:
    satPackage = checkForSatPackages()
    if satPackage is not None:
        confirmSatReceipt(
                satPackage.ID,
                satPackage.coords)
        uploadResponseToServerQueue(satPckg)

    if recievedServerInstructions:
            sendPackageToSatellite(serverInstruct)
```

sendPackageToSatellite(serverInstruct)

```
# Uses server instructions, which contains the
# satCoords at this moment, satFreqs, any laser-linked
# path instructions, and the actual package, to actually
# send the package to the satellite
...
```

confirmSatReceipt(satPackageID,satCoords)

```
# Sends satellite confirmation of receipt once, based
# on the satCoords embedded into the satPackage
...
```

uploadSatPackageToServerQueue(satPckg)

```
# Blindly uploads the sat package, whatever it is, to
# the server to handle
...
```

Satellite

main()

```
while True:
    recievedPackage = checkForPackages()
    if recievedPackage:
        if recievedPackage.type == 'confirmation':
            # stop sending whatever package is being sent
            stopBroadcasting(recievedPackage.pckgID)
        else:
            # send response back
            beginBroadcastingData(
                    userID,satCoords,msg,freq,
                    addMsgType='RESPONSE'
                )
    if sendTimedOrOnEventPackageToUser:
            beginBroadcastingData(
                userID,coords,msg,freq,addMsgType=None
                )
```

beginBroadcastingData(userID,satCoords,msg,freq,addMsgType=None)

```
# Initiates thread that starts broadcasting the data
# that should be sent to the stated userID
...
```

broadcastData(packageID)

```
# Closes the broadcast thread corresponding to the
# packageID
...
```

Security Note

- All returned messages from the satellite will necessarily be visible to all radio AM’s
- Only messages sent to the satellite will be protected (?)

Interested?

- Great. Awesome. Fantastic. Good job, being interested in space.
- I have no idea how you can get involved so I’m passing this up to Markku and Pekka. If you are Markku or Pekka, hi. I’m Sofie. (You should probably also edit this bullet point out and replace it will real info.)