IP Multicast in Python

- Sending multicast packets:
  - Create a UDP socket as usual:

```
my_socket =
socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
```

  - Set the **`IP_MULTICAST_TTL`** protocol option to multicast, e.g.,

```
ttl = struct.pack('B', 1)
(or: ttl = (1).to_bytes(1, byteorder='big')
 or: ttl = b'01')

my_socket.setsockopt(socket.IPPROTO_IP,
socket.IP_MULTICAST_TTL, ttl)
```

  - ttl controls the number of subnetworks that the multicast packets can traverse (ttl = 1 restricts it to the local subnetwork), i.e., via the IP TTL field.

IP Multicast in Python

- TTL
  - There are both IP_MULTICAST_TTL and IP_TTL so that unicast and multicast TTL can be set differently.)
  - Used to control scope of multicast transmission

0	Restricted to the same host. It will not be transmitted on any interface.
1	Restricted to the same subnet. It will not be forwarded by any router.
32	Restricted to same site, organization or department.
64	Restricted to the same region.
128	Restricted to the same continent.
255	Global.

  - Values between these ranges can be used.

IP Multicast in Python

- Sending multicast packets:
  - Then do a normal UDP send with multicast (address, port) tuple:

    ```
    my_socket.sendto(b'Hello Everyone!',
    ("230.0.0.10", 2000))
    ```

- You can also bind the socket to a specific interface to ensure that multicast packets are transmitted on the correct interface. See Python code example.

  e.g., `self.socket.bind(("192.168.1.22", 30000))`

IP Multicast in Python

- Receiving multicast packets:
  - Create a normal UDP socket bound to an address and port:

```
my_socket = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)

my_socket.bind(("0.0.0.0", 2000))
```

  - Alternately, you can bind to the actual multicast address and multicast port, e.g.,

```
my_socket.bind(("230.0.0.10", 2000))
```

  - In both cases, the bind address is used for filtering the arriving multicast packets. (This does not seem to work under Windows any more.)

IP Multicast in Python

- Receiving multicast packets:
  - We need to issue an add group membership request to the local multicast router. The request argument is 8 bytes long, consisting of the multicast address followed by the local interface address to receive on:

```
multicast_group_bytes =
socket.inet_aton("239.0.0.10")

multicast_if_bytes = socket.inet_aton("0.0.0.0")
```

    Using all zeros, the system will choose a default interface. For more control you can specify the specific interface, e.g.,

```
... = socket.inet_aton("192.168.1.10")
```

IP Multicast in Python

- Receiving multicast packets:
  - Then prepare and make the multicast request:

```
multicast_request = multicast_group_bytes +
multicast_if_bytes


my_socket.setsockopt(socket.IPPROTO_IP,
socket.IP_ADD_MEMBERSHIP, multicast_request)
```

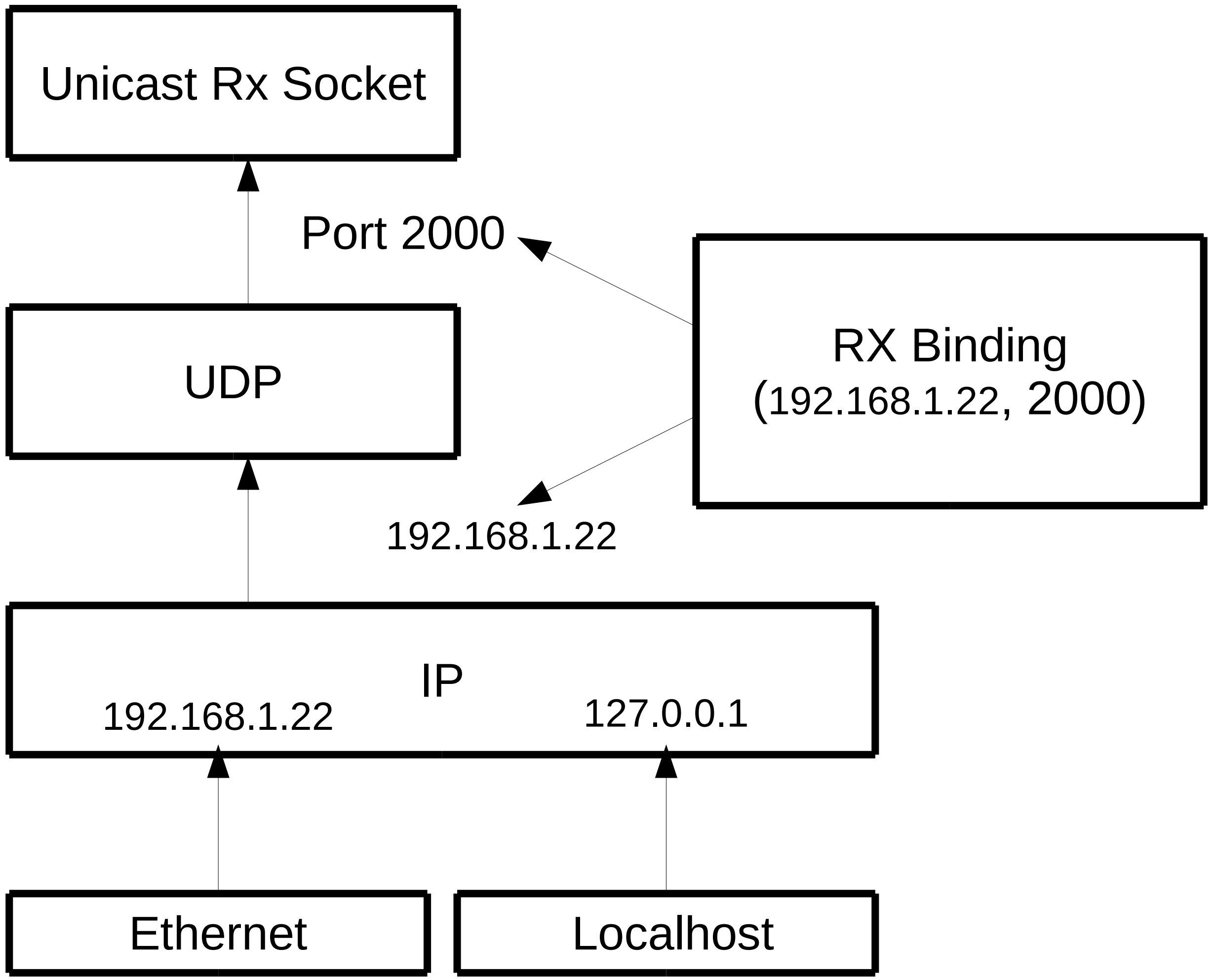
  - Then do a normal UDP receive:

```
my_socket.recvfrom(Receiver.RECV_SIZE)
```

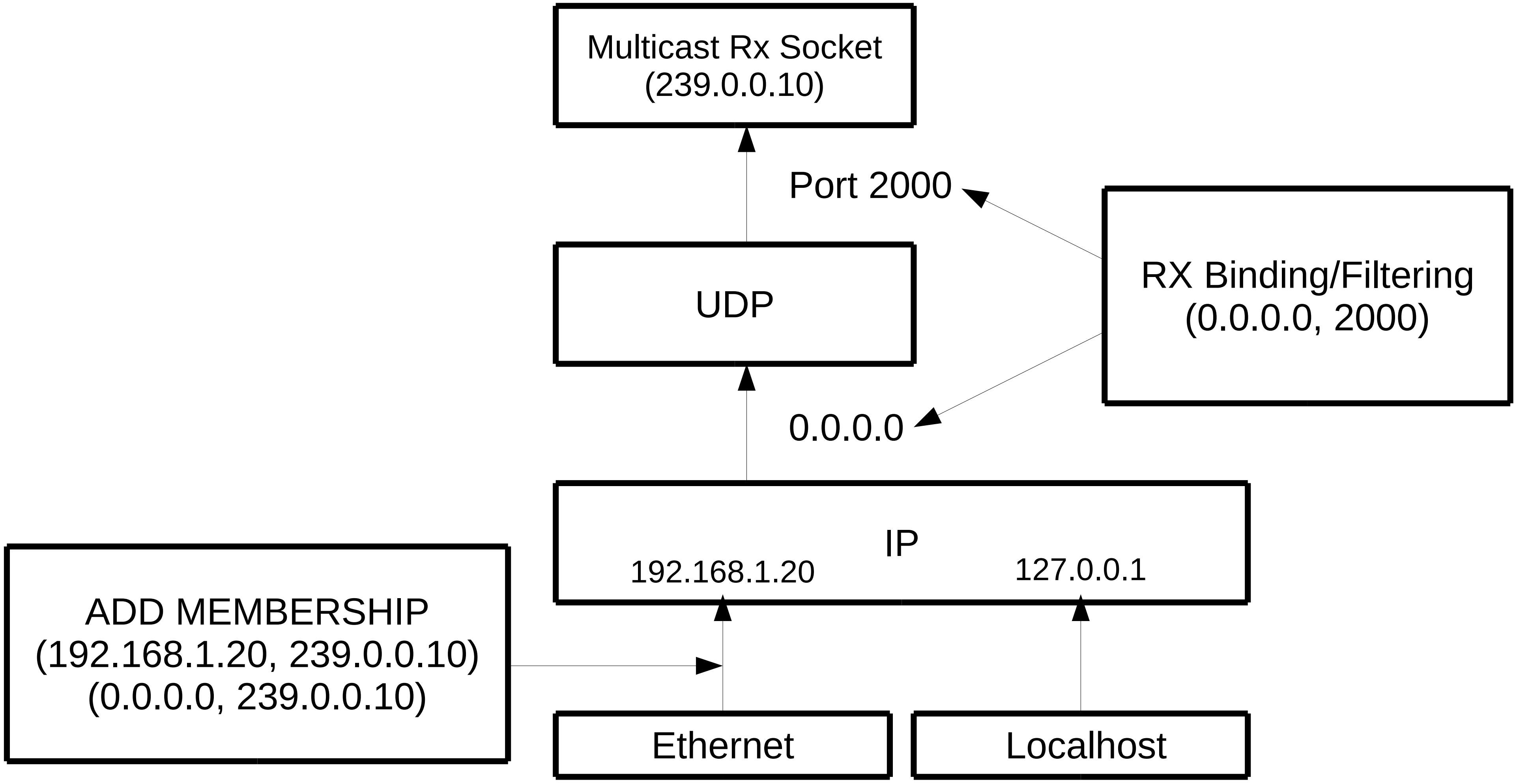
  - You can also stop receiving multicast packets:

```
my_socket.setsockopt(socket.IPPROTO_IP,
socket.IP_DROP_MEMBERSHIP, multicast_request)
```

Unicast Rx Socket Binding



Multicast Rx Socket Binding


Multicast Rx Socket
(239.0.0.10)
Port 2000
RX Binding/Filtering
(0.0.0.0, 2000)
UDP
0.0.0.0
IP
192.168.1.20
127.0.0.1
ADD MEMBERSHIP
(192.168.1.20, 239.0.0.10)
(0.0.0.0, 239.0.0.10)
Ethernet
Localhost