```
EXT_PORT = 2
if received_on_port != EXT_PORT: # Heartbeat
    return ([],[])
```

```
from state import flow_emap, flow_id_to_backend_id, backends, backend_ip_emap, cht
EXP_TIME = 10 * 1000
BACKEND_EXP_TIME = 3600000000 * 1000
EXT_PORT = 2

if a_packet_received:
    flow_emap.expire_all(now - EXP_TIME)
    backend_ip_emap.expire_all(now - BACKEND_EXP_TIME)

h3 = pop_header(tcpudp, on_mismatch=([],[]))
h2 = pop_header(ipv4, on_mismatch=([],[]))
h1 = pop_header(ether, on_mismatch=([],[]))

assert a_packet_received
assert h1.type == 8 # 0x0800 == IPv4 in big endian
assert h2.npid == 6 or h2.npid == 17 # 6/17 -> TCP/UDP

if received_on_port == EXT_PORT: # Packet from the external network - client
    packet_flow = LoadBalancedFlowc(h2.saddr, h2.daddr, h3.src_port, h3.dst_port, h2.npid)
    alloc_flow_and_process_packet = False;
    if flow_emap.has(packet_flow):
        flow_id = flow_emap.get(packet_flow)
        backend_id = flow_id_to_backend_id.get(flow_id)
        if backend_ip_emap.has_idx(backend_id):
            flow_emap.refresh_idx(flow_emap.get(packet_flow), now)
            backend = backends.get(backend_id)
            return ([backend.nic],
                    [ether(h1, saddr=..., daddr=backend.mac),
                     ipv4(h2, cksum=..., daddr=backend.ip),
                     tcpudp(h3)])
        else:
            flow_emap.erase(packet_flow)
            alloc_flow_and_process_packet = True
    else:
        alloc_flow_and_process_packet = True
    if alloc_flow_and_process_packet:
        if backend_ip_emap.exists_with_cht(cht, _LoadBalancedFlow_hash(packet_flow)):
            bknd = backend_ip_emap.choose_with_cht(cht, _LoadBalancedFlow_hash(packet_flow))
            if not flow_emap.full():
                idx = the_index_allocated
                flow_emap.add(packet_flow, idx, now)
                flow_id_to_backend_id.set(idx, bknd)
            backend = backends.get(bknd)
            return ([backend.nic],
                    [ether(h1, saddr=..., daddr=backend.mac),
                     ipv4(h2, cksum=..., daddr=backend.ip),
                     tcpudp(h3)])
        else:
            return ([],[])
else: # A heartbeat from a backend
    bknd_addr = ip_addrc(h2.saddr)
    if backend_ip_emap.has(bknd_addr):
        backend_ip_emap.refresh_idx(backend_ip_emap.get(bknd_addr), now)
    else:
        if not backend_ip_emap.full():
            idx = the_index_allocated
            backend_ip_emap.add(bknd_addr, idx, now)
            backends.set(idx, LoadBalancedBackendc(received_on_port, h1.saddr, h2.saddr))
    return ([],[])
```

Full-stack

Pay-as-you-go

Push-button

5

Support for partial specifications

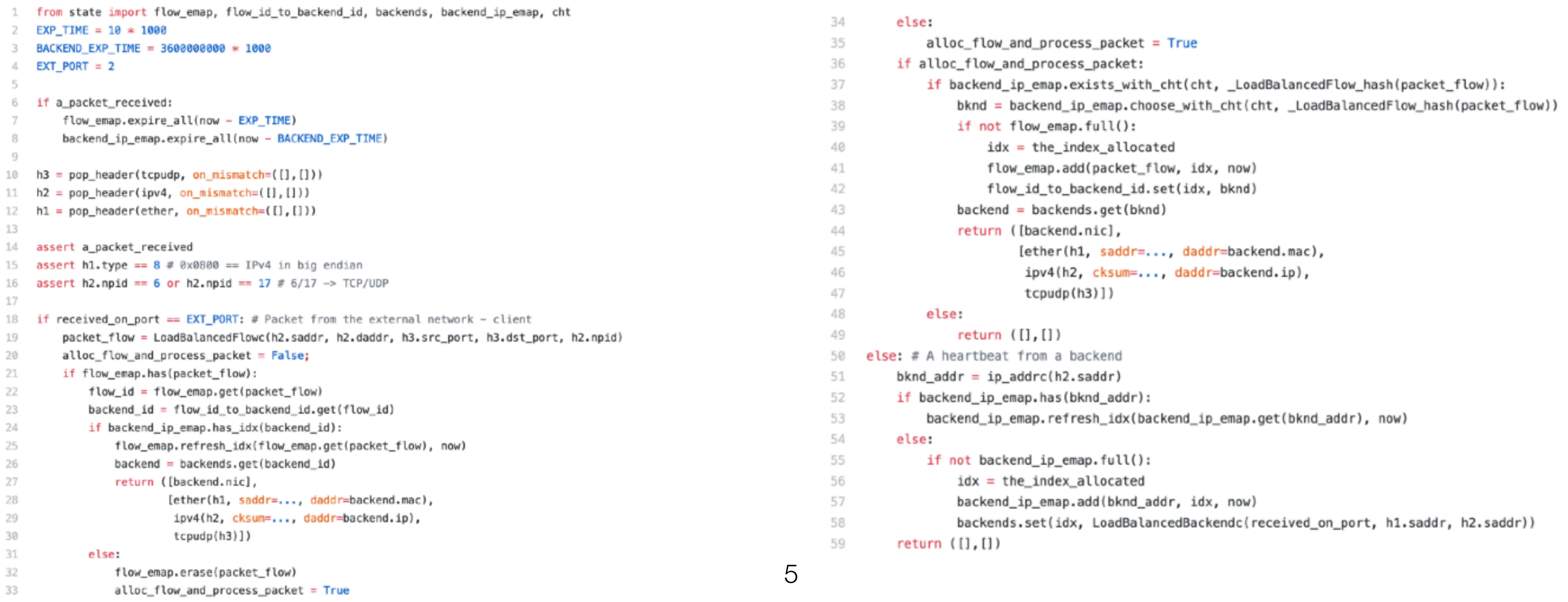
Push-button

Pay-as-you-go

Full-stack

Support for partial specifications

```
EXT_PORT = 2
if received_on_port != EXT_PORT: # Heartbeat
    return ([],[])
```



```
from state import flow_emap, flow_id_to_backend_id, backends, backend_ip_emap, cht
EXP_TIME = 10 * 1000
BACKEND_EXP_TIME = 3600000000 * 1000
EXT_PORT = 2

if a_packet_received:
    flow_emap.expire_all(now - EXP_TIME)
    backend_ip_emap.expire_all(now - BACKEND_EXP_TIME)

h3 = pop_header(tcpudp, on_mismatch=([],[]))
h2 = pop_header(ipv4, on_mismatch=([],[]))
h1 = pop_header(ether, on_mismatch=([],[]))

assert a_packet_received
assert h1.type == 8 # 0x0800 == IPv4 in big endian
assert h2.npid == 6 or h2.npid == 17 # 6/17 -> TCP/UDP

if received_on_port == EXT_PORT: # Packet from the external network - client
    packet_flow = LoadBalancedFlowc(h2.saddr, h2.daddr, h3.src_port, h3.dst_port, h2.npid)
    alloc_flow_and_process_packet = False;
    if flow_emap.has(packet_flow):
        flow_id = flow_emap.get(packet_flow)
        backend_id = flow_id_to_backend_id.get(flow_id)
        if backend_ip_emap.has_idx(backend_id):
            flow_emap.refresh_idx(flow_emap.get(packet_flow), now)
            backend = backends.get(backend_id)
            return ([backend.nic],
                    [ether(h1, saddr=..., daddr=backend.mac),
                     ipv4(h2, cksum=..., daddr=backend.ip),
                     tcpudp(h3)])
        else:
            flow_emap.erase(packet_flow)
            alloc_flow_and_process_packet = True
    else:
        alloc_flow_and_process_packet = True
    if alloc_flow_and_process_packet:
        if backend_ip_emap.exists_with_cht(cht, _LoadBalancedFlow_hash(packet_flow)):
            bknd = backend_ip_emap.choose_with_cht(cht, _LoadBalancedFlow_hash(packet_flow))
            if not flow_emap.full():
                idx = the_index_allocated
                flow_emap.add(packet_flow, idx, now)
                flow_id_to_backend_id.set(idx, bknd)
            backend = backends.get(bknd)
            return ([backend.nic],
                    [ether(h1, saddr=..., daddr=backend.mac),
                     ipv4(h2, cksum=..., daddr=backend.ip),
                     tcpudp(h3)])
        else:
            return ([],[])
else: # A heartbeat from a backend
    bknd_addr = ip_addrc(h2.saddr)
    if backend_ip_emap.has(bknd_addr):
        backend_ip_emap.refresh_idx(backend_ip_emap.get(bknd_addr), now)
    else:
        if not backend_ip_emap.full():
            idx = the_index_allocated
            backend_ip_emap.add(bknd_addr, idx, now)
            backends.set(idx, LoadBalancedBackendc(received_on_port, h1.saddr, h2.saddr))
    return ([],[])
```

Push-button

Pay-as-you-go

Full-stack

Whole SW stack: framework, OS, drivers

NF: 3 KLOC

Framework, driver: 85 KLOC