Homework #1 Student ID**: 2018280070,** Name**: Peter Garamvoelgyi**

1. In a MAC-based L2 switch, how is the MAC-port list set up and maintained?

In Ethernet, each network interface is assigned a globally unique 6-byte MAC[1] address. FF:FF:FF:FF:FF:FF is the broadcast address.

L3 devices, such as routers, need a way to find the MAC address corresponding to a host with a given IP address. IPv4 uses ARP[2] to achieve this. The IP-MAC pairs are cached locally in the *ARP-table*. IPv6 replaces ARP with Neighbor Discovery Protocol (NDP).

L2 devices, however, are not aware of L3 concepts such as IP addresses, only MAC addresses. Instead of an ARP-table, L2 switches maintain a *forwarding table* that assign MAC addresses to ports[3].

Setting up the forwarding table: Every time an L2 switch receives a frame on port *x*, it assigns the source MAC address of the frame to *x* in its forwarding table. Unlike ARP, this process has no dynamic MAC discovery; a switch can only assign a MAC address to one of its ports once it has received at least one frame originating from that address.

Forwarding frames: When an L2 switch receives a frame on port *y*, it finds the entry belonging to the destination MAC in its forwarding table and forwards the packet on the corresponding port. If there is no corresponding entry in the forwarding table, it forwards the packet to all ports except *y*[4].

MAC address	Port
`01:23:45:67:89:AB`	2
`78:A2:45:47:1B:1A`	1
…	…

Ethernet nodes use STP[5] to build a loop-free logical topology (a *spanning tree*) in an Ethernet network. This protocol ensures that there is only one active path between two nodes. As a result, it can be expected that all frames from a given source MAC address arrive at the same port. Receiving a frame on a different port suggests that either the network topology or the active paths (the spanning tree) changed; in this case, the switch should update the forwarding table to use the new port.

[1] MAC: Medium Access Control
[2] ARP: Address Resolution Protocol
[3] Port here means the physical port of the switch; not to be confused with TCP ports.
[4] This process is termed *flooding*.
[5] STP: Spanning Tree Protocol

2. Where fragmented packets should be reassembled? At the end node (destination host), or at an intermediate node (such as router)? Why?

The relevant section from RFC791:

> *„The basic internet service is datagram oriented and provides for the fragmentation of datagrams at gateways, with reassembly taking place at the destination internet protocol module in the destination host. Of course, fragmentation and reassembly of datagrams within a network or by private agreement between the gateways of a network is also allowed since this is transparent to the internet protocols and the higher-level protocols."*[6]

Fragmentation can happen at the source node or any intermediate nodes. Nodes along a given path might need to fragment packets depending on their MTU. If DF[7] is set but the packet does not fit the MTU, the node must drop it and send back the appropriate ICMP message.[8]

Alternatively, source nodes might use Path MTU discovery to detect the minimal MTU along a path, and proceed to send packets of this size using fragmentation on the source node. This way the other nodes along the path will probably not need to fragment the packets anymore.[9]

Place of reassembly: As routers under high load have very little time for each packet, it is best to leave the resource-intensive reassembly process to the destination node. However, nodes wishing to inspect the contents of a packet might need to reassemble it first. An example is NAT, where the NAT-enabled router must inspect the TCP header to extract the port. Other examples include advanced firewall techniques and deep packet inspection.

[6] RFC791 Internet Protocol https://tools.ietf.org/html/rfc791

[7] DF: Don't Fragment flag in the IP header. For IPv6 packets, this is always set (implicitly).

[8] *Destination Unreachable* or *Datagram Too Big* for ICMP, *Packet Too Big* for ICMPv6

[9] However, the path MTU might change between discovery and data transfer, due to changes in routing or network topology. These changes can be detected by setting the DF flag.

3. How does IPv6 take care of MTU and path MTU?

RFC1981 describes the process of PMTU[10] discovery as follows:

> *„[...] a source node initially assumes that the PMTU of a path is the (known) MTU of the first hop in the path. If any of the packets sent on that path are too large to be forwarded by some node along the path, that node will discard them and return ICMPv6 Packet Too Big messages [ICMPv6]. Upon receipt of such a message, the source node reduces its assumed PMTU for the path based on the MTU of the constricting hop as reported in the Packet Too Big message."* [11]

This description shows that IPv6-enabled nodes are capable of

1. using ICMPv6 messages to signal their MTU towards the source and
2. iteratively discovering the PMTU based on such ICMP responses.

RFC1883 highlights an important difference between fragmentation in IPv4 and IPv6:

> *„unlike IPv4, fragmentation in IPv6 is performed only by source nodes, not by routers along a packet's delivery path [...] It is strongly recommended that IPv6 nodes implement Path MTU Discovery, in order to discover and take advantage of paths with MTU greater than 576 octets."* [12]

... 576 octets being the minimum required MTU for IPv6 links. Thus, we could say that IPv6 has an implicit DF flag set for every packet.

RFC1981 also suggests that nodes periodically (infrequently) increase their assumed PMTU. This way that can detect increases in PMTU due to changes in network topology.

[10] PMTU: Path MTU
[11] RFC1981 Path MTU Discovery for IP version 6 https://tools.ietf.org/html/rfc1981
[12] RFC1883 Internet Protocol, Version 6 (IPv6) Specification https://tools.ietf.org/html/rfc1883

4. **(Bonus) If there is only one external IP address but more than 2^{16} TCP sessions need to go out, is there a way to extend PAT so that we can accommodate the requirement?**

Example scenario: Let us assume that hosts in a LAN share the single external IP address 1.2.3.4. A host with the local IP 192.168.0.6 wants to open a TCP connection with 8.8.8.8 using source port 1000 and destination port 53.[13]

During PAT[14], the gateway rewrites the source port of the TCP segment to 2345[15]. It also rewrites the source address in all corresponding IP packets to 1.2.3.4. The router also remembers that incoming TCP segments to port 2345 should be forwarded to 192.168.0.6:1000.

Destination port	Forward to
`2345`	`192.168.0.6:1000`
`7899`	`192.168.5.9:2500`
…	…

Problem statement: As the source port field of the TCP segment header is only 2 bytes long, the maximum number of parallel TCP connections this mechanism can accommodate is 2^{16}. To extend this, some connections need to share the same external port (2345 in the example above), and the gateway needs a way to distinguish the corresponding TCP segments.

Idea 1: Assign a unique ID to each TCP connection and store this in the *urgent pointer*[16] field in the TCP header or the TCP payload. The problem with this solution is that we cannot expect the remote party (e.g. 8.8.8.8) to follow our scheme and include the ID in the segments it sends.

Idea 2: Store additional information in the PAT table. TCP is connection-oriented and the source/destination ports and addresses do not change within a connection. We could store the remote port along with the destination port in our PAT table:

Destination + remote port	Forward to
`(2345, 53)`	`192.168.0.6:1000`
`(2345, 80)`	`192.168.5.9:2500`
…	…

This way, segments between `192.168.0.6:1000` and `8.8.8.8:53` and segments between `192.168.5.9:2500` and `104.16.124.127:80` can share the same external port `2345`. The extra storage overhead is 2 bytes per PAT entry. Theoretically, 2^{16} connections could share a port with this method, making our router able to serve $2^{16} \cdot 2^{16} = 2^{32}$ TCP connections simultaneously. However, this number is much smaller in reality, as connection with the same remote port cannot share. Checking this also puts a larger burden on the gateway.

A more general way is to store both the remote port and the address in the PAT table. This way we can reduce the number of such *collisions*, and we can serve much more connections. However, this method has a significant storage overhead of 2+4 bytes per PAT entry (for IPv4).

[13] This could be a DNS request to Google's public DNS server, although DNS typically uses UDP.
[14] PAT: Port Address Translation, a technique usually employed in NAT-enabled routers.
[15] This is an example; the actual port is implementation-specific.
[16] This field is normally not used.