Backwards learning
Assignment 2
Computer Networks TI2406
Delft University of Technology

Jesse Donkervliet

February 2018

1 About the assignment

This assignment consists of a written part and a programming part. The goal of the written part is to help you complete the programming part of the assignment. Please format your code in a TA-legible manner and make sure to use plenty of comments to explain the logic behind what you're doing. You can check the correctness of your implementation using the test case provided on Brightspace. The case provided on Brightspace might not consider all possible edge-cases. You are encouraged to create your own test cases to confirm the correctness of your implementation.

This assignment is checked by a teacher or TA as a pass/fail during an interview. To pass the assignment, both team members should be able to concisely answer the questions, and explain how and why the submitted code works. If you have any questions regarding the assignment or your solution, or if you are done and would like the student assistants to check your work, you can use the queue, available via: `https://queue.ewi.tudelft.nl/`.

Before registering for sign-off, please make sure that you have submitted your deliverable on CPM. Your submission must have the following properties:

1. The file format of the report must be MD (Markdown) or PDF (Portable Document Format). Clear, concise answers are preferred over verbose ones.
2. Hand in a `.zip` or `.tar.gz` archive containing both the code and the report.
3. The code is written in Java, C, C++, or Python.[1]
4. The root of your archive should contain a single directory. Within this directory you can make separate subdirectories for your code and other files.

The deadline for the assignment can be found on Brightspace and CPM. If you have any questions which you cannot figure out on your own, do not hesitate to contact us. You can post questions on the Brightspace discussion forum, or send them to cn-cs-ewi@tudelft.nl.

2 Introduction

In this assignment, you implement a backwards-learning algorithm to be used in network switches. These switches use ALPv2. The protocol is identical to ALP, except for two important details:

1. The length field is now only 2 bytes large.
2. The checksum is only computed over the payload. The source, destination, and length fields are not included in the checksum calculation.

For your convenience, the ALPv2 frame layout is shown in Figure 1.

[1] If you really want to use another language, you can ask a teacher or TA for permission.

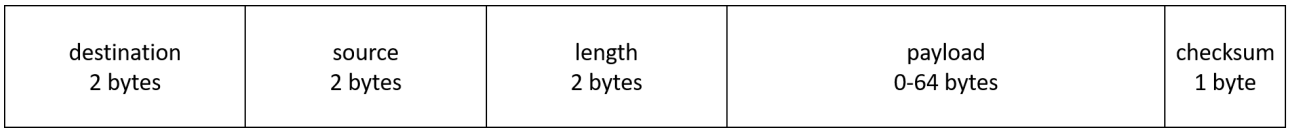

Figure 1: Schematic overview of an ALPv2 frame.

2.1 The Medium Access Control sublayer

The Medium Access Control layer is responsible for deciding who transmits data on the channel and when it is transmitted. In classic Ethernet, this is a large responsibility, because machines shared a single channel. If multiple machines send at the same time on the same channel, a collision occurs and the transmitted data is distorted.

With modern Ethernet, this task is often no longer required, because machines use point-to-point links.[2] Every channel only has two devices connected to it. These devices can be regular computers, but they can also be routers or switches. Switches are what make the point-to-point link architecture possible. Without it, a machine would need to have a separate cable for each other machine it is connected to. Instead, switches are used. These switches can be compared to a telephone switch board; it receives frames on one link, and then transmits it on another link, depending on where the frame needs to go. Because these links are not directly connected to each other, the throughput of the network improves significantly. Every machine can send frames to the switch at the same time, without the possibility of collisions.

3 Assignment

The core task of a switch is to forward incoming frames on the correct links. To make the use of switches practical, they are in charge of their own setup. That is, the switch will figure out on which links to forward frames without the user having to configure it.

3.1 Exploration

In this part of the assignment you answer a number of questions about switches, how they configure themselves, and what to do with the frames they receive. Answering these questions simplifies the programming assignment in the next section.

1. Which fields from the ALPv2 frame can the switch use to build a topological map of the network?
2. What happens with the frames addressed to a machine x, if this machine never transmits any frames itself?
3. What happens with the frames addressed to a machine x, if this machine is disconnected from the switch and then reconnected on a different link?
4. Describe how a switch could reconfigure itself to solve this problem.
5. If a hub is used in the network, it is possible that machines x and y share a single link on the switch. Once the switch knows that x and y are connected on the same link, what should it do with frames that are sent by x and are addressed to y?
6. The frames received by the switch contain a 1-byte checksum. Is checking for data errors on the switch a good idea? Give both an advantage and a disadvantage of doing error detection on the switch.

[2] In other communication channels, such as wireless channels, the MAC sublayer remains crucial.

3.2 Programming

In this part of the assignment, you implement the software that runs on the ALPv2 switch. This switch receives frames on multiple links. Your code needs to forward these frames on the appropriate links. You may assume that machines never change the link to which they are connected. You may also assume that you are always allowed to transmit on a link, even if multiple machines are connected to it; you can ignore potential collisions. Frames are forwarded to their destination, even if their CRC checksum indicates a transmission error.

Similar to the previous assignment, your code should read input from stdin, and print the output to stdout. You can find an example `problem.in` and `problem.out` file on Brightspace.

Your switch should support the following input:

1. One line with an integer l, indicating the number of links on the switch.
2. One line with an integer m, indicating the number of messages that the switch will receive.
3. m lines that correspond to incoming messages. Each line contains a decimal number that specifies the incoming link and an ALPv2 frame, separated by a space. The bytes that are part of the frame are encoded in hexadecimal notation, which means every two characters correspond to one byte.

For example, consider the following input:

```
5
2
0 00aa00bb000800000000000000000000
3 00bb00aa0005000000000000
```

This input shows that a frame, sent by a machine with MAC address `00bb`, is received on link 0. The machine wants to send 8 bytes to another machine with address `00aa`. Machine `00aa` replies by sending 5 bytes in return. This frame comes in at the switch on port 3. Because the payload is all-zeroes, the ALPv2 checksum is also all-zeroes.

The output should have the following format:

1. m lines, each with the original frame, followed by all ports on which the frame is sent out by your switch. All values are space separated. If the frame is not sent out on any links, the line only contains the original frame.

Initially, it is unknown how the machines are connected to the different ports. In the backward-learning algorithm, when the location of a machine is unknown, the message is flooded. If a message is sent out on multiple links, these links should appear in **ascending order** in the output. This means that, for the input given above, the following output is expected:

```
00aa00bb000800000000000000000000 1 2 3 4
00bb00aa0005000000000000 0
```

Your program should also take into account machines connected to the switch via a hub. From the perspective of the switch, this means these machines are connected on the same port.