ETH zürich

Networked Systems
ETH Zürich — seit 2015

D-ITET

Prof. Dr. Laurent Vanbever

August 2022

R. Jacob, E. Costa Molero, A. Dietmüller
A. Gran Alcoz, E. Kirci, R. Meier

Exam: Advanced Topics in Communication Networks

9 August 2022, 08:30–10:30, Room HG D 5.1

- ▷ Write your name and your ETH student number below on this front page and **sign it**.
- ▷ Put your **legitimation card** (legi) on the most accessible corner of your desk. Make sure that the side containing your name and **student number is visible**.
- ▷ Verify that you have received all task sheets (Pages 1 - 32).
- ▷ **Do not separate** the task sheets. We will collect the exams after you left the room.

- ▷ Write your answers directly on the task sheets.
- ▷ All answers fit within the allocated space—often in much less.
- ▷ If you need more space, use the **extra sheets** at the end of the exam. **Indicate the task** in the corresponding field, and add a **"see Extra Sheet X"** note in the original task space.
- ▷ **Read each task completely before you start solving it.**
- ▷ It is not required to score all points to get the best mark.

- ▷ Answer in **English**.
- ▷ **Write clearly** in blue or black ink (not red) using a **pen**, not a pencil.
- ▷ **Cancel** invalid parts of your solutions **clearly** (e.g., by crossing them out).
- ▷ At the end of the exam, **place the exam face up** on the most accessible corner of your desk. Then collect all your belongings and **exit the room** according to the given instructions.

- ▷ No written material nor calculator are allowed.

Family name: Student legi nr.:

First name: Signature:

Do not write in the table below (used by corrector only):

Task	Points
Programmable data planes	/23
Managing network traffic	/26
Optimizing network performance	/29
Design question	/42
Total	/120

leave blank

leave blank

Task 1: Programmable data planes — 23 Points

a) General Questions — (8 Points)

For each of the following statements, indicate whether they are *true* or *false*. There is always one correct answer. Each block of questions grants up to 4 points: 4 points for four correct answers, 2 points for three correct answers, and 0 points otherwise.

(i) P4 language, architecture, and programs (4 Points)

true	false	
☐	☐	All parsed headers must be emitted.
☐	☐	In general, P4 is intended to operate on packet headers.
☐	☐	All functions used in a P4 program must be 100% implemented with P4 code.
☐	☐	P4 cannot validate or modify checksums.

(ii) Data structures in P4 (4 Points)

true	false	
☐	☐	All P4 targets support the same stateful data structures.
☐	☐	With P4-compatible hardware, the same data structure cannot be accessed infinitely many times for the same packet going through the pipeline once.
☐	☐	Persistent data structures may provide different interfaces to the data plane and the control plane.
☐	☐	A counter marks specific packets if their rate exceeds a threshold.

b) Probabilistic data structures **(15 Points)**

leave blank

Let us consider a Bloom filter that uses ten cells and three hash functions. The hash functions for this task are defined as follows:

$$\begin{aligned} hash_1(x) &= x && \mod 10 \\ hash_2(x) &= 3 \cdot (x+1) && \mod 10 \\ hash_3(x) &= 5 \cdot (x+1) && \mod 10 \end{aligned}$$

where x is the numerical input value. In this task, we will input words and use the sum of the letters' ASCII values as x. For example, the word *AdvNet* has value 578. Table 1 contains the numerical values you need.

	x	$3\ (x+1)$	$5\ (x+1)$
AdvNet	578	1737	2895
lecture	756	2271	3785
great	531	1596	2660
boring	641	1926	3210

Table 1: Numerical word values.

(i) Start with an empty Bloom filter and update it by inserting the words *AdvNet*, *lecture*, and *great* one after another. Use the table below to indicate the state of the Bloom filter after each word. (3 Points)

	(empty)		**AdvNet**		**lecture**		**great**
0	0						
1	0						
2	0						
3	0						
4	0	→		→		→	
5	0						
6	0						
7	0						
8	0						
9	0						

(ii) After inserting the three words, query the Bloom filter for the word *boring*. Is this result correct? (1 Point)

Query result for *boring*: true ☐ false ☐

Is the result correct?: _____

(iii) Bloom filters may return incorrect results. What are these called? Explain what is incorrect about the result and under which circumstances these errors occur. (2 Points)

How are these errors called?

What is the problem?

When do these errors occur?

(iv) Additional hash functions may reduce the number of errors, but this is not always the case. Explain both possibilities. (2 Points)

How may additional hash functions reduce errors?

How may additional hash functions increase errors?

(v) You now want to implement the Bloom Filter above by starting from a CountMin Sketch that you have already written:

```
#define N_CELLS 10

struct metadata {
  bit<32> hash_one;
  bit<32> hash_two;
  bit<32> hash_three;
  bit<16> out_one;
  bit<16> out_two;
  bit<16> out_three;
  bit<16> query_result;
}
register<bit<16>>(N_CELLS) datastructure;

action compute_hashes() {
  // Compute hashes, abbreviated for simplicity.
  hash(meta.hash_one, ...);
  hash(meta.hash_two, ...);
  hash(meta.hash_three, ...);
}

action insert() {
  // Read.
  datastructure.read(meta.out_one, meta.hash_one);
  datastructure.read(meta.out_two, meta.hash_two);
  datastructure.read(meta.out_three, meta.hash_three);

  // Update.
  datastructure.write(meta.hash_one, meta.out_one + 1);
  datastructure.write(meta.hash_two, meta.out_two + 1);
  datastructure.write(meta.hash_three, meta.out_three + 1);
}

action query() {
  // Read.
  datastructure.read(meta.out_one, meta.hash_one);
  datastructure.read(meta.out_two, meta.hash_two);
  datastructure.read(meta.out_three, meta.hash_three);
}

control sketch {
  compute_hashes();
  insert();
  query();

  // Find minimum.
  if (meta.out_one <= meta.out_two) && (meta.out_one <= meta.out_three)
  {
    query_result = meta.out_one;
  } else if (meta.out_two <= meta.out_three) {
    query_result = meta.out_two;
  } else {
    query_result = meta.out_three;
  }
}

```

You could use this Sketch code as it is and get the same result as a Bloom Filter by interpreting the `query_result` differently. Explain how. (1 Point)

However, you realize that this wastes resources and is needlessly complex if one only desires a Bloom Filter. For each of the five sections of the program (metadata and register declaration, `compute_hashes`, `insert`, `query`, and `control`), explain:

1. which parts need to remain the same;
2. which parts can be simplified, and how.

Only explain. Do not write P4 code. (6 Points)

Metadata and register declaration:

`compute_hashes` action:

`insert` action:

`query` action:

`control` section:

leave blank

leave blank

Task 2: Managing network traffic **26 Points**

a) Label switching theory **(4 Points)**

For each of the following statements, indicate whether they are *true* or *false*. There is always one correct answer. Each block of questions grants up to 4 points: 4 points for four correct answers, 2 points for three correct answers, and 0 points otherwise.

true	false	
☐	☐	The path reservations made with RSVP must be refreshed periodically.
☐	☐	RSVP makes resource reservations for unidirectional data flows.
☐	☐	Only the Ingress Label Edge Routers can stack MPLS labels.
☐	☐	All routers can perform POP operations in an MPLS network.

b) Label switching application **(22 Points)**

leave blank

This sub-task is based on the topology shown in Figure 1.

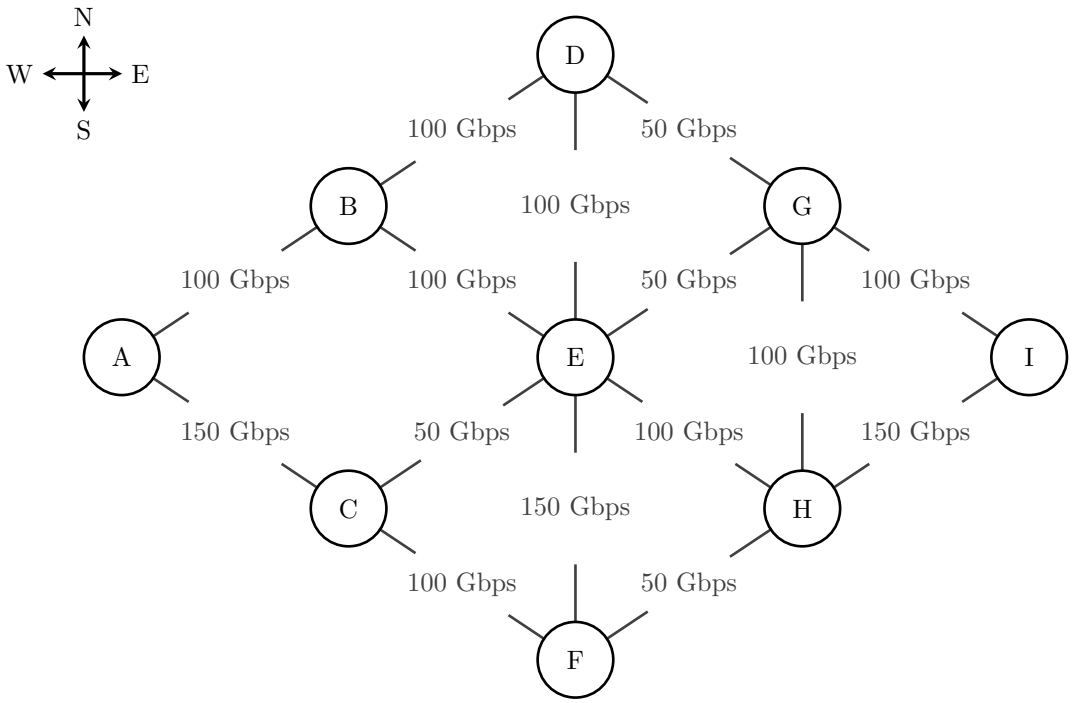

Figure 1: The topology of the network. Link annotations indicate the link's bidirectional capacity. All links have the same delay and unary cost.

(i) Consider the network topology in Figure 1. Link annotations indicate the link's bidirectional capacity. All links have the same delay and unary cost. Find the shortest constrained path(s) for an aggregated traffic demand of 75 Gbps from Router A to Router I. Describe the steps of your reasoning. (3 Points)

Path: _____

Algorithm steps: _____

Router A

Inlabel	Nexthop	Operation
L1	SE	Swap(L3)
L2	NE	Push(L5)
L4	SE	Swap(L1)
L6	NE	Push(L7)
L7	NE	Push(L5)
L8	NE	Push(L7)

Router B

Inlabel	Nexthop	Operation
L5	NE	Pop
L7	SE	Pop

Router C

Inlabel	Nexthop	Operation
L1	SE	Swap(L3)
L3	NE	Swap(L7)

Router D

Inlabel	Nexthop	Operation
L2	SE	Push(L7)
L7	S	Swap(L1)

Router E

Inlabel	Nexthop	Operation
L1	S	Swap(L4)
L6	NE	Push(L1)
L7	NE	Push(L1)
L8	SE	Swap(L1)

Router F

Inlabel	Nexthop	Operation
L3	NE	Push(L8)
L4	NE	Push(L8)

Router G

Inlabel	Nexthop	Operation
L1	SE	Pop
L7	SE	Pop

Router H

Inlabel	Nexthop	Operation
L1	NE	Swap(L5)
L8	NE	Pop

Router I

Inlabel	Nexthop	Operation
L2	Local	Pop
L3	Local	Pop
L4	Local	Pop
L5	Local	Pop
L6	Local	Pop
L7	Local	Pop

Table 2: Label forwarding tables for each router.

(ii) Consider the label forwarding tables shown in Table 2. In this forwarding state, at least two non-overlapping paths exist from Router A to Router I; i.e., the LSP tunnels do not share any edge. For a packet entering in Router A, find two labels (one for each path) that deliver packets to Router I over non-overlapping paths. Indicate the path taken in each case and the associated label headers at each hop. (6 Points)

Hint: For example, a packet initially labeled with L1 entering in Router A terminates at Router I with L7. The path taken in this case is:
$L1 \rightarrow (A) \rightarrow L3 \rightarrow (C) \rightarrow L7 \rightarrow (E) \rightarrow L1, L7 \rightarrow (G) \rightarrow L7 \rightarrow (I)$

Path 1:

Path 2:

(iii) Explain one purpose of label stacking. (2 Points)

In Table 2, find two different cases of label stacking. On which links are label stacking used? (2 Points)

Link 1: _____

Link 2: _____

How do the stacked labels on these links help with the explained purpose? (1 Point)

(iv) Consider that Router A wants to reserve a path towards Router I. Describe the RSVP message Router A sends to Router I to establish its LSP with explicit routing. What message type does Router A send, and what state does it create in intermediate routers? (4 Points)

(v) Describe the RSVP message that Router I sends back to Router A. How is the message routed, and what type of state does it create in intermediate routers? (4 Points)

leave blank

leave blank

Task 3: Optimizing network performance **29 Points**

a) Quality of Service **(17 Points)**

(i) Consider a token-bucket scheduler in policing mode (i.e., whenever there are no tokens available, incoming packets are dropped) with a bucket size of 10 tokens. The bucket starts full at $t = 1$ and re-fills by one token instantaneously at the end of each time slot. Table 3 describes four scenarios in which an incoming traffic pattern needs to be scheduled by the token bucket. Each column indicates how many packets arrive at the token bucket input during a time slot.

- For each scenario, indicate whether the token bucket will drop some of the incoming traffic (*packet drops = yes*) or it will allow all packets to be sent (*packet drops = no*). Circle the answer in the rightmost column in Table 3.

- For the scenario(s) with packet drops, if any, circle the *first* time slot at which packet drops will occur. (4 Points)

time slot / scenarios	1	2	3	4	5	6	7	8	9	10	Packet drops?
Traffic pattern 1	1	1	1	1	1	1	1	1	1	1	yes / no
Traffic pattern 2	3	3	3	3	3	3	3	3	3	3	yes / no
Traffic pattern 3	10	1	10	1	10	1	10	1	10	1	yes / no
Traffic pattern 4	10	0	0	0	0	1	1	1	2	2	yes / no

Table 3: The traffic patterns of different applications.

(ii) Consider a link with a total capacity of 22 units. This link is shared by 5 sources which, respectively, demand $R1 = 2$, $R2 = 3$, $R3 = 5$, $R4 = 7$, $R5 = 12$ units. What is the max-min fair allocation for each source? Describe your computation.

(5 Points)

(iii) What is the difference between work-conserving and non-work-conserving scheduling algorithms? (2 Points)

(iv) Indicate whether each of the following scheduling algorithms is *work-conserving* or *non-work-conserving*. (2 Points)

Token Bucket:

Fair Queuing:

Priority Queuing:

(v) Why isn't Fair Queuing used on the entire Internet? Does Token Bucket have the same problem? (2 Points)

Limitation:

Does it apply to Token Bucket?

(vi) Describe one drawback of Priority Queueing and one way to mitigate it. (2 Points)

Drawback:

Mitigation:

b) Fast convergence **(12 Points)**

leave blank

Consider the network topology in Figure 2. Each link is annotated with its IGP weight. In the following, we are interested in identifying possible Loop-Free Alternates (LFAs).

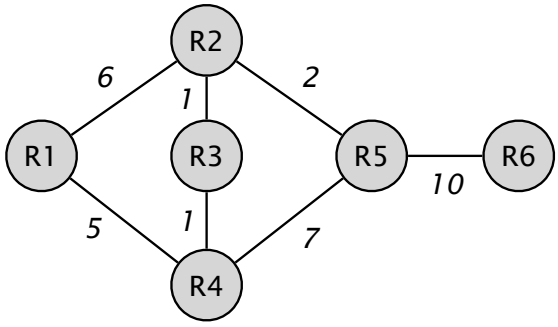

Figure 2: Network topology.

(i) Compute the shortest path from each router towards R6 and their cost. (2 Points)

Path between R1→R6: _____

Cost of R1→R6: _____

Path between R2→R6: _____

Cost of R2→R6: _____

Path between R3→R6: _____

Cost of R3→R6: _____

Path between R4→R6: _____

Cost of R4→R6: _____

Path between R5→R6: _____

Cost of R5→R6: _____

(ii) Which router(s) can be considered a per-prefix LFA of R1 towards R6? Justify your answer. (3 Points)

(iii) Does R1 have any *remote* LFA (RLFA) towards R6? Justify your answer. (3 Points)

(iv) Explain what Bidirectional Forwarding Detection (BFD) is and why it is beneficial for fast convergence. (4 Points)

leave blank

Task 4: Design question **42 Points**

In this task, you are hired to deploy a new link-layer protocol in the network shown in Figure 3. The protocol is an alternative to `Ethernet` called `Advnet`. The `Advnet` header consists of the following fields:

- `dstPath` (18 bits): the path through the network (more below)
- `advType` (16 bits): indicates the protocol used in the payload (e.g., IPv4)
- `options` (6 bits): can be used for custom applications. Packets coming from hosts always have all options set to `0`.

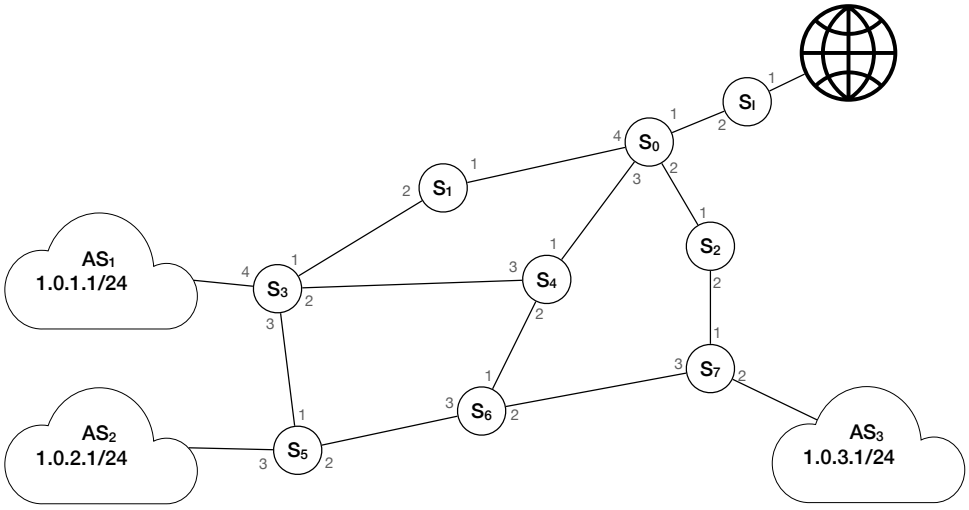

Figure 3: Network topology

`Advnet` is designed specifically to work in the topology shown in Figure 3. Note the following properties of the topology and `Advnet`:

- The network serves as the backbone which interconnects three ASes (AS_1, AS_2, AS_3) and the Internet.
- Incoming packets from the ASes already have a valid `Advnet` header.
- Packets from and to the Internet use `Ethernet`. All other links in Figure 3 (i.e., all except the one between S_I and the Internet) use `Advnet`.
- S_I serves as the gateway to the Internet.
- Each switch has between 2 and 4 active ports.

The main difference between `Ethernet` and `Advnet` is that `Advnet` does not have a destination address. Instead, it contains the precise path that the packet should take through the network. The path is encoded as a sequence of port numbers that specify on which port each switch has to send the packet. Each port number consists of 3 bits, and the `path` field can specify a path with up to 6 switches—hence its length of 18 bits.

For example, let us consider a packet sent from AS_1 with the following `path` value:

$$\underbrace{010}_{2}\,\underbrace{010}_{2}\,\underbrace{011}_{3}\,\underbrace{011}_{3}\,\underbrace{000}_{0}\,\underbrace{000}_{0}$$

This packet would be sent along the path $S_3 \rightarrow S_4 \rightarrow S_6 \rightarrow S_5 \rightarrow AS_2$. If a block (at any position) is 0, then it is ignored.

In this task, you will develop a P4 program that runs on all switches in this network (**except** S_I): switches may have different table entries but run the same P4 code.

Figure 4 contains a code skeleton that you will extend and modify during this task. S_I runs a different program that you will develop later in this task.

```
1  /* -*- P4_16 -*- */
2  #include <core.p4>
3  #include <v1model.p4>
4
5  const bit<16> TYPE_IPV4 = 0x0800;
6
7  header advnet_t {
8      bit<18>  dstPath;
9      bit<16>  advType;
10     bit<6>   options;
11 }
12
13 header ipv4_t {
14     bit<4>  version;     bit<4>  ihl;            bit<8>    diffserv;
15     bit<16> totalLen;    bit<16> identification; bit<3>    flags;
16     bit<13> fragOffset;  bit<8>  ttl;            bit<8>    protocol;
17     bit<16> hdrChecksum; bit<32> srcAddr;        bit<32>   dstAddr;
18 }
19
20 struct metadata { }
21
22 struct headers {
23     advnet_t   advnet;
24     ipv4_t     ipv4;
25 }
26
27 parser MyParser(packet_in packet, out headers hdr, inout metadata meta,
28                 inout standard_metadata_t standard_metadata) {
29
30     state start {
31         transition parse_advnet;
32     }
33
34     // ======================
35     // ToDo: parse Advnet and IPv4
36     // ======================
37 }
38
39 control MyVerifyChecksum(...) { apply {  } }
40
41 control MyIngress(inout headers hdr, inout metadata meta,
42                   inout standard_metadata_t standard_metadata) {
43
44     action send_to_port(bit<9> port, bit<18> x) {
45         standard_metadata.egress_spec = port;
46         hdr.advnet.dstPath = hdr.advnet.dstPath & x;
47     }
48
```

```
49      table l2_forward {
50          key = {
51              hdr.advnet.dstPath: ternary;
52          }
53          actions = {
54              send_to_port;
55              NoAction;
56          }
57          default_action = NoAction();
58          size = 1024;
59      }
60
61      apply {
62         l2_forward.apply();
63      }
64  }
65
66
67  control MyEgress(...) { apply {  } }
68
69  control MyComputeChecksum(...) { apply { } }
70
71  control MyDeparser(packet_out packet, in headers hdr) {
72      apply {
73          // ======================
74          // ToDo
75          // ======================
76      }
77  }
78
79  V1Switch(MyParser(), MyVerifyChecksum(), MyIngress(), MyEgress(),
        MyComputeChecksum(), MyDeparser()) main;
```

Figure 4: P4 code skeleton

leave blank

a) Getting familiar with the network **(5 Points)**

Before you start implementing `Advnet`, the customer wants to make sure that you understand how the protocol and the topology work.

(i) Which path does a packet with `path=011 001 011 100 000 000` sent from AS_3 take? (1 Point)

(ii) Which egress port does S_4 use if it receives a packet with `path=000 000 010 011 011 000` on port 3? (1 Point)

(iii) One advantage of `Advnet` compared to `Ethernet` is that its header is shorter (the `Ethernet` header is 112 bits long). How long does it take to send 1'500'000 bytes of IP *payload* over an `Advnet` and an `Ethernet` link with the following properties:

- 100 Gbps line rate (12.5×10^9 bytes per second)
- no packet loss or congestion
- maximum 1500 bytes IP *payload* per packet

Your final result can be a fractional number (there is no need to compute the division). (3 Points)

with `Ethernet`

with `Advnet`

b) Parsing packets **(5 Points)**

leave blank

It is time to extend the program such that it parses and deparses `Advnet` packets correctly.

(i) The parser states for `Advnet` and `IPv4` are missing in the skeleton. Write their code below. Note that the skeleton already contains the header definitions, and if a packet is not `IPv4`, it is enough to parse the `Advnet` header. (4 Points)

```
state parse_advnet {

}

state parse_ipv4 {

}
```

(ii) Write the code for the deparser. (1 Point)

```
apply {

}
```

leave blank

c) Basic forwarding **(9 Points)**

The code skeleton already defines and applies a table called `l2_forward`. In this task, you need to fill the table with entries.

Note that the table uses ternary matches. In table entries, you can write ternary matches using "*" for bits/digits that should be ignored. For example, you can write

```
1****************1
```

to match on `Advnet` paths with the first and the last bit equal to 1. Since multiple entries can match on the same packet, list the rules with decreasing priority—the first entry has the highest priority.

(i) As part of the forwarding action, the program executes (line 44 in Figure 4)

```
hdr.advnet.dstPath = hdr.advnet.dstPath & x;
```

What is the purpose of this line? (2 Points)

(ii) Give 5 entries that **switch S_1** must maintain in the `l2_forward` table. (More entries are needed, but we ask you to list only 5 of them.) Abbreviate the action names as follows: "SP" for `send_to_port` and "NA" for `NoAction`. (5 Points)

Match	Action	
`dstPath`	name	parameters

(iii) Currently, the table reserves space for 1024 entries. But the number of entries required is much smaller. How many entries need to fit in the table at most in this network? No changes to the code are allowed. Explain your answer. (2 Points)

leave blank

d) A more scalable solution **(13 Points)**

The current version of `Advnet` only allows paths of a maximum length of 6. This works in the existing network, but it is not very flexible. In this task, you will develop `AdvnetV2` which works for longer paths.

(i) How would you extend `Advnet` such that it allows paths of a length of up to 32 switches without creating unnecessary overhead? The following constraints need to be satisfied:

- If a packet's path has length ≤ 6, the length of the entire header must not be longer than 40 bits.
- Each switch has at most 5 ports.
- The new protocol does not need to be compatible with the original protocol or the code in Figure 4.
- The `options` field can be removed from the header.

Describe your proposal for `AdvnetV2` below. It is enough to describe your high-level idea here. You will be asked for more details later. (2 Points)

(ii) Explain how one would need to change the code in Figure 4 such that it works with `AdvnetV2`. Your description should be detailed enough to allow a knowledgeable P4 programmer to implement it. You do not need to write P4 code. (8 Points)

(iii) Write down the contents of your `AdvnetV2` header for a packet that a host in AS_2 wants to send to AS_3 (you can select a path). List all fields in your `AdvnetV2` header and specify their contents. (3 Points)

e) A gateway to the Internet **(10 Points)**

leave blank

Now we focus on S_I, which serves as a gateway between the `Advnet` network and the Internet. A colleague has already implemented the gateway; the code is shown in Figure 5. However, it is not working yet, and your task is to fix it.

```
1  /* -*- P4_16 -*- */
2  #include <core.p4>
3  #include <v1model.p4>
4
5  const bit<16> TYPE_IPV4 = 0x0800;
6
7  header advnet_t {
8      bit<18>  dstPath;
9      bit<16>  advType;
10     bit<6>   options;
11 }
12
13 header ethernet_t {
14     bit<48> dstAddr;
15     bit<48> srcAddr;
16     bit<16> etherType;
17 }
18
19 header ipv4_t {
20     bit<4>  version;     bit<4>  ihl;            bit<8>    diffserv;
21     bit<16> totalLen;    bit<16> identification; bit<3>    flags;
22     bit<13> fragOffset;  bit<8>  ttl;            bit<8>    protocol;
23     bit<16> hdrChecksum; bit<32> srcAddr;        bit<32>   dstAddr;
24 }
25
26 struct metadata { }
27
28 struct headers {
29     advnet_t   advnet;
30     ethernet_t ethernet;
31     ipv4_t     ipv4;
32 }
33
34 parser MyParser(...) {
35     // [hidden]
36 }
37
38 control MyVerifyChecksum(...) {
39     apply {  }
40 }
41
42 control MyIngress(inout headers hdr, inout metadata meta,
43                   inout standard_metadata_t standard_metadata) {
44
45     action advnet_to_ethernet(bit<9> port) {
46         standard_metadata.egress_spec = port;
47         hdr.ethernet.setValid();
48         hdr.ethernet.srcAddr = 0x010203040506;
49         hdr.ethernet.dstAddr = 0x020304050607;
50         hdr.advnet.setInvalid();
51     }
52
53     action ethernet_to_advnet(bit<9> port, bit<32> path) {
54         standard_metadata.egress_spec = port;
55         hdr.advnet.setValid();
56         hdr.advnet.dstPath = path;
57         hdr.ethernet.setInvalid();
58     }
```

```
59
60      table transform_to_ethernet {
61          key = {
62              hdr.ipv4.dstAddr: lpm;
63          }
64          actions = {
65              advnet_to_ethernet;
66              NoAction;
67          }
68          default_action = NoAction();
69          size = 1024;
70      }
71
72      table transform_to_advnet {
73          key = {
74              ipv4.dstAddr: lpm;
75          }
76          actions = {
77              ethernet_to_advnet;
78              NoAction;
79          }
80          default_action = NoAction();
81          size = 1024;
82      }
83
84      if (hdr.ipv4.isValid()) {
85          if (hdr.advnet.isValid()) {
86              transform_to_ethernet();
87          }
88          else if (hdr.ethernet.isValid()) {
89              transform_to_advnet();
90          }
91      }
92  }
93
94  control MyEgress(...) { apply {  } }
95
96  control MyComputeChecksum(...) { apply { } }
97
98  control MyDeparser(packet_out packet, in headers hdr) {
99      apply {
100         // [hidden]
101   }
102 }
103
104 V1Switch(MyParser(), MyVerifyChecksum(), MyIngress(), MyEgress(),
        MyComputeChecksum(), MyDeparser()) main;
```

Figure 5: Gateway code

(i) The programmer who wrote this code made some mistakes which caused the compilation to fail. Use the table below to list the lines that contain errors and write down the fixed version (rewrite the entire line). If you need to add a new line (say, after line 123), write "after 123" as the line number. (6 Points)

Hints: All errors are in the `MyIngress` *control block, and six lines contain at least one error. We only look for errors that the compiler would detect.*

Line	Correct code

(ii) Now that you fixed all the syntactical errors, the program compiles and—after you added the correct table entries—it converts `Ethernet` to `Advnet` and vice-versa. Unfortunately, something is not quite right yet with the converted packets.

What is the problem, and how can you fix it? First, explain the problem, then use the table below to write the code you need to add to fix it and the line after which to insert it. (4 Points)

Hint: You need to add only two lines of code.

Problem: _____

_____

_____

Code that needs to be added:

Insert after line	Code to insert

Extra Sheet 1

In case you need more space, use the following pages. Make sure to always indicate the task to which the answer belongs (e.g., *3 d) (ii)*).

Task: _____

Extra Sheet 2

Task: _____