Homework 3

Submission date: 11-6-2021

Part 1

Before you begin:

1. Download *P4 VM* from here.
   Use your post.bgu account to gain access.
2. Install VirtualBox and run this VM.
3. Use this github classroom link: https://classroom.github.com/a/d13dkZYu
   Unlike previous homeworks, in this assignment you have a template code.
   Clone the repo to the vm that you've just downloaded.

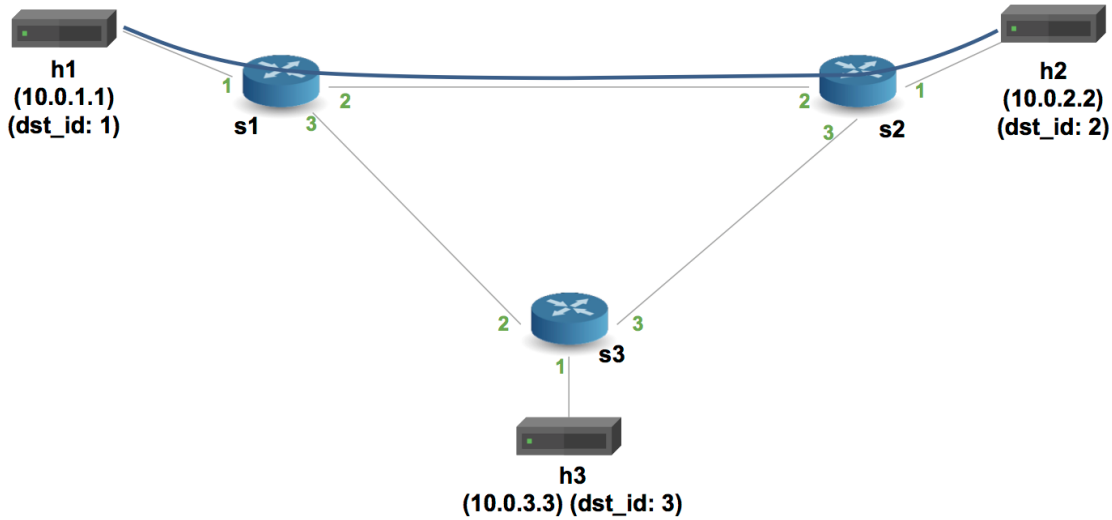
1. Basic tunneling

In this exercise, you will implement a basic tunneling protocol for the IP router that you completed in the tutorial. You will define a new header type to encapsulate the IP packet and modify the switch to perform routing using our new header.

The new header type will include:

- Protocol ID bit<16> proto_id, indicates the type of packet being encapsulated.
- Destination ID bit<16> dst_id, to be used for routing.

Basic environment check:



1. In your shell run –
   `make run`
   This will compile `basic_tunnel.p4`, start mininet with all the hosts and switches.
   Open terminals for h1 and h2 respectively -
   `mininet> xterm h1 h2`
2. On h2 run the server-
   `./receive.py`
3. On h1 run the client-
   `./send.py 10.0.2.2 "cloud computing is cool"`
   You should receive a packet at h2. You may observe that it contains headers of Ethernet, IP, and TCP.
4. exit the mininet environment, run-
   `make stop`

Note: A P4 program defines a packet-processing pipeline, but the rules within each table are inserted by the control plane. When a rule matches a packet, its action is invoked with parameters supplied by the control plane as part of the rule.

For this exercise, we have already added the necessary static control plane entries. As part of bringing up the Mininet instance, the `make run` command will install packet-processing rules in the routing tables of each switch. These are defined in the `sX-commands.txt` files, where X corresponds to the switch number.

Implementing basic tunneling:

The `basic_tunnel.p4` file contains an implementation of a basic IP router (from the tutorial). It also contains comments marked with `TODO` which indicate the functionality that you need to implement.

You need to implement the following:

1. A new header type called `myTunnel_t` that contains two 16-bit fields: `proto_id` and `dst_id`.
2. `Add myTunnel_t` header to the `headers` struct.
3. Update the parser to extract either the `myTunnel` header or `ipv4` header based on the `etherType` field in the Ethernet header. The etherType corresponding to the myTunnel header is `0x1212`. The parser should also extract the `ipv4` header after the `myTunnel` header if `proto_id` == `TYPE_IPV4` (i.e. 0x0800).
4. Define a new action called `myTunnel_forward` that simply sets the egress port (i.e. `egress_spec` field of the `standard_metadata` bus) to the port provided by the control plane.
5. Define a new table called `myTunnel_exact` that perfoms an exact match on the `dst_id` field of the `myTunnel` header. This table should invoke either the `myTunnel_forward` action if there is a match in the table, or it should invoke the `drop` action otherwise.
6. Update the `apply` statement in the `MyIngress` control block to apply your newly defined `myTunnel_exact` table if the `myTunnel` header is valid. Otherwise, invoke the `ipv4_lpm` table if the `ipv4` header is valid.
7. Update the deparser to emit the `ethernet`, then `myTunnel`, then `ipv4` header. Remember that the deparser will only emit a header if it is valid. A header's implicit validity bit is set by the parser upon extraction. So there is no need to check header validity here.
8. Add static rules for your newly defined table so that the switches will forward correctly for each possible value of `dst_id`. See the diagram below for the topology's port configuration as well as how we will assign IDs to hosts. For this step you will need to add your forwarding rules to the `sX-commands.txt` files.

Running your solution:

1. In your shell, run:
   `make run`
2. You should now see a Mininet command prompt. Open two terminals for `h1` and `h2`, respectively:
   `mininet> xterm h1 h2`
3. Each host includes a small Python-based messaging client and server. In `h2`'s xterm, start the server:
   `./receive.py`
4. First we will test without tunneling. In `h1`'s xterm, send a message to `h2`:
   `./send.py 10.0.2.2 "Cloud computing is cool"`
5. The packet should be received at `h2`. Examine the received packet and you should see that it consists of an Ethernet header, an IP header, a TCP header, and the message. If you change the destination IP address (e.g. try to send to `10.0.3.3`) then the message should not be received by `h2`, and will instead be received by `h3`.
6. Now let's test with tunneling. In `h1`'s xterm, send a message to `h2`:
   `./send.py 10.0.2.2 "Cloud computing is cool" --dst_id 2`
7. The packet should be received at `h2`. Examine the received packet and you should see that it consists of an Ethernet header, a tunnel header, an IP header, a TCP header, and the message.
8. In `h1`'s xterm, send a message:
   `./send.py 10.0.3.3 "Cloud computing is cool" --dst_id 2`
9. The packet should be received at `h2`, even though the IP address is the address of `h3`. This is because the switch is no longer using the IP header for routing when the `MyTunnel` header is in the packet.
10. Type `exit` or `Ctrl-D` to leave each xterm and the Mininet command line.

Extending basic tunneling:
To make things more realistic change your P4 code to have the switches add the `myTunnel` header to the IP upon ingress to the network and then remove the header as the packet leaves the network.
Call the new code `extending_tunneling.p4`

Submission:
Push your code to ***p4-git/P4-HomeWork/exercises/basic_tunnel/***

2. Load balancing as a service:

In this exercise, you will implement a load balancing based as a service. The switch will use two tables to forward packets to one of two destination hosts randomly.

The first table will use a hash function, applied to a 5-tuple consisting of the source and destination IP addresses, IP protocol, and source and destination TCP ports, to select one of two hosts. The second table will use the computed hash value to forward the packet to the selected host.

Basic environment check:

1. In your shell, run:
   `make`
2. You should now see a Mininet command prompt. Open three terminals for `h1`, `h2` and `h3`, respectively:
   `mininet> xterm h1 h2 h3`
3. Each host includes a small Python-based messaging client and server. In `h2` and `h3`'s XTerms, start the servers:
   `./receive.py`
4. In `h1`'s XTerm, send a message from the client:
   `./send.py 10.0.0.1 "cloud computing is cool"`
   The message will not be received.
5. Type `exit` to leave each XTerm and the Mininet command line.

The message was not received because each switch is programmed with `load_balance.p4`, which drops all packets on arrival. Your job is to extend this file.

Implementing load balancing:

The `load_balance.p4` file contains a skeleton P4 program with key pieces of logic replaced by `TODO` comments.

Your code should implement the following:

1. An action (called `set_ecmp_select`), which:
    1. Hashes the 5-tuple specified above using the `hash` extern
    2. Stores the result in the `meta.ecmp_select` field
2. A control that:
    1. Applies the `ecmp_group` table.
    2. Applies the `ecmp_nhop` table.

Run your implementation as shown above.
This time, your message from `h1` should be delivered to `h2` or `h3`. If you send several messages, some should be received by each server.

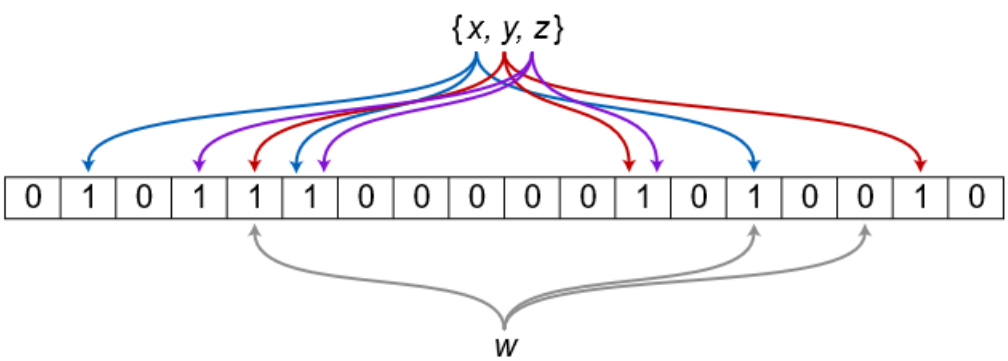
Submission:
Push your code to ***p4-git/P4-HomeWork/exercises/load_balance/***

3. Firewall - NON MANDATORY:

In this exercise you need to write a P4 program that implements a simple stateful firewall. You will use a bloom filter.
Basics of bloom filter:
An empty Bloom filter is a bit array of m bits, all set to 0. There are also k different hash functions, each of which maps a set element to one of the m bit positions.

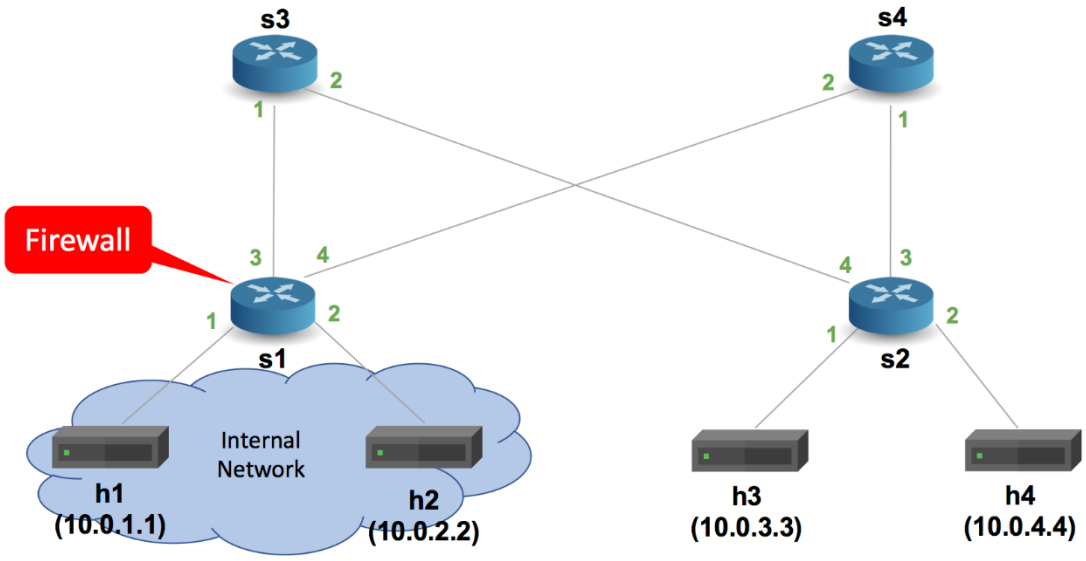
- To add an element, feed it to the hash functions to get k bit positions, and set the bits at these positions to 1.
- To test if an element is in the set, feed it to the hash functions to get k bit positions.
    - If any of the bits at these positions is 0, the element **definitely isn't** the set.
    - If all are 1, then the element **may be** in the set.

Example:



The Bloom filter above is with three elements *x*, *y* and *z*. It consists of 18 bits and uses 3 hash functions. The colored arrows point to the bits that the elements of the set are mapped to.
The element *w* definitely isn't in the set, since it hashes to a bit position containing 0.
For a fixed error rate, adding a new element and testing for membership are both constant time operations, and a filter with room for *n* elements requires $O(n)$ space.
(Credit to youbasic.org)

The pod-topology for this exercise consists of four hosts connected to four switches, which are wired up as they would be in a single pod of a fat tree topology.



Switch s1 will be configured with a P4 program that implements a simple stateful firewall (`firewall.p4`), the .rest of the switches will run the basic IPv4 router program (`basic.p4`) from the tutorial

:The firewall on s1 should have the following functionality

- Hosts h1 and h2 are on the internal network and can always connect to one another.
- Hosts h1 and h2 can freely connect to h3 and h4 on the external network.
- Hosts h3 and h4 can only reply to connections once they have been established from either h1 or h2, but cannot initiate new connections to hosts on the internal network.

Note: This stateful firewall is implemented 100% in the dataplane using a simple bloom filter. Thus, there is some probability of hash collisions that would let unwanted flows to pass through.

<u>Basic environment check:</u>

1. In your shell, run:
   make run
2. You should now see a Mininet command prompt. Try to run some iperf TCP flows between the hosts. TCP flows within the internal network should work:

   ```
   mininet> iperf h1 h2
   ```
3. TCP flows from hosts in the internal network to the outside hosts should also work:

   ```
   mininet> iperf h1 h3
   ```
4. TCP flows from the outside hosts to hosts inside the internal network should NOT work. However, since the firewall is not implemented yet, the following should work:

   ```
   mininet> iperf h3 h1
   ```
5. Type `exit` to leave the Mininet command line. Then, to stop mininet:
   `make stop`,And to delete all pcaps, build files, and logs: `make clean`

implementing Firewall:

1) Header type definitions for Ethernet (`ethernet_t`), IPv4 (`ipv4_t`) and TCP (`tcp_t`).
2) Parsers for Ethernet, IPv4 and TCP that populate `ethernet_t`, `ipv4_t` and `tcp_t` fields.
3) An action to drop a packet, using `mark_to_drop()`.
4) An action (called `compute_hashes`) to compute the bloom filter's two hashes using hash algorithms `crc16` and `crc32`. The hashes will be computed on the packet 5-tuple consisting of IPv4 source and destination addresses, source and destination port numbers and the IPv4 protocol type.
5) An action (`ipv4_forward`) and a table (`ipv4_lpm`) that will perform basic IPv4 forwarding (adopted from `basic.p4`).
6) An action (called `set_direction`) that will simply set a one-bit direction variable as per the action's parameter.
7) A table (called `check_ports`) that will read the ingress and egress port of a packet (after IPv4 forwarding) and invoke `set_direction`. The direction will be set to `1`, if the packet is incoming into the internal network. Otherwise, the direction will be set to `0`. To achieve this, the file `pod-topo/s1-runtime.json` contains the appropriate control plane entries for the `check_ports` table.
8) A control that will:
   a) First apply the table `ipv4_lpm` if the packet has a valid IPv4 header.
   b) Then if the TCP header is valid, apply the `check_ports` table to determine the direction.
   c) Apply the `compute_hashes` action to compute the two hash values which are the bit positions in the two register arrays of the bloom filter (`reg_pos_one` and `reg_pos_two`). When the direction is `1` i.e. the packet is incoming into the internal network, `compute_hashes` will be invoked by swapping the source and destination IPv4 addresses and the source and destination ports. This is to check against bloom filter's set bits when the TCP connection was initially made from the internal network.
   d) If the TCP packet is going out of the internal network and is a SYN packet, set both the bloom filter arrays at the computed bit positions (`reg_pos_one` and `reg_pos_two`). Else, if the TCP packet is entering the internal network, read both the bloom filter arrays at the computed bit positions and drop the packet if either is not set.
9) A deparser that emits the Ethernet, IPv4 and TCP headers in the right order.
10) A `package` instantiation supplied with the parser, control, and deparser
11) Run your code as shown above, for example run `iperf` flow between two hosts, between which of the pair it works, which are blocked? You can add more tests of your own.

Submission:

Push your code to ***p4-git/P4-HomeWork/firewall/exercises/firewall/***

Part 2

1. **רשות -** בשאלה זו נבדוק את ביצועי כמה אלגוריתמים לבעיית ה-BinPacking.
   1. הריצו את האלגוריתמים NextFit, BestFit, ו-WorstFit על סדרת הקלטים הבאה σ(משמאל לימין), והראו את השיבוצים המתקבלים:

σ =0.6, 0.5, 0.4, 0.3, 0.2, 0.6, 0.6, 0.1, 0.8, 0.1, 0.1, 0.2, 0.4, 0.7, 0.2, 0.9, 0.4, 0.5, 0.8

   2. הוכיחו/הפריכו: קיים קלט שעבורו WorstFit משיג את הביצועים הטובים ביותר מבין שלוש האלגוריתמים בסעיף א'.
   שימו לב: אם אתם מוכיחים זאת, עליכם להראות את הקלט, ולהראות ש-WorstFit משתמש במספר קטן (ממש) של מכונות ביחס לאלגוריתמים האחרים. אם אתם מפריכים זאת, עליכם להסביר מדוע לא יתכן קלט כזה.
   3. הריצו את האלגוריתמים BFD (BestFitDecreasing) ו-FFD (FirstFitDecreasing) על סדרת הקלטים σ מסעיף א' והראו את השיבוצים המתקבלים.

2. בשאלה זו נוכיח חסמים על ביצועי האלגוריתם המקוון NextFit.
   1. הראו כי האלגוריתם NextFit הוא 2-תחרותי.
   רמז: הסתכלו על סה"כ העומס על זוגות מכונות עוקבות, והסיקו מכך חסם תחתון על סך העומס הקיים במערכת.
   2. הראו כי לכל קבוע $\epsilon > 0$, לא ניתן להוכיח כי NextFit הוא $(2 - \epsilon)$-תחרותי.
   רמז: מומלץ להראות זאת בשלילה, ע"י בניית קלט מתאים המוביל לסתירה.

3. **רשות -** בשאלה זו ננתח את ביצועי האלגוריתם FFD (FirstFitDecreasing).
   1. בסעיף זה נוכיח כי FFD הוא $\frac{3}{2}$-קירוב.
   נסמן ב-$\sigma = p_1, \ldots, p_n$ את סדרת (גדלי) המשימות אותן משבץ האלגוריתם.
   נסמן ב-m את מספר המכונות בה עושה האלגוריתם שימוש, ויהא $m_0 = \lceil \frac{2}{3}m \rceil$.
      i. הראו כי אם במכונה ה-m_0 (על פי סדר פתיחת המכונות) משובצת משימה p_i עבורה $p_i > 0.5$, אז $OPT \geq \frac{2}{3}m$.
      רמז: הראו כי ניתן להבטיח במקרה זה כי בכל מכונה $j < m_0$ משובצת משימה בגודל לפחות 0.5. מה ניתן להסיק מכך על מספר המכונות אותן נדרש OPT לפתוח?
      ii. הראו כי אם במכונה ה-m_0 (על פי סדר פתיחת המכונות) לא משובצת אף משימה p_i עבורה $p_i > 0.5$, אז $OPT \geq \frac{2}{3}m$.
      רמז: כמה משימות לפחות משובצות במכונות $m_0, \ldots, m$? האם יש משימה כזו שניתן לשבץ במכונה $j < m_0$? הסיקו כי סך גדלי המשימות ב-σ הוא לפחות $\left\{m_0 - 1, 2\left(m - m_0\right) + 1\right\}$, וכי ביטוי זה הוא לפחות $\left(\lceil \frac{2}{3}m \rceil - 1\right)$.
      iii. הסיקו כי FFD הוא $\frac{3}{2}$-קירוב.
   2. ראש הצוות שלכם בחברת WePackBetter ביקש מכם להכין מסמך המוכיח ש-FFD הוא $\frac{4}{3}$-קירוב. מה תנסו לעשות בכדי להוכיח זאת?

4. **רשות -** בשאלה זו עליכם להשוות בין שתי טכנולוגיות לביצוע encapsulation:
   1. פרוטוקול VXLAN (אותו ראיתם בכיתה)
   2. פרוטוקול NVGRE

במענה על השאלה יש לזהות 2 מאפיינים שבהם VXLAN עולה על NVGRE, ו-2 מאפיינים שבהם NVGRE עולה על VXLAN. שימו לב -- המאפיינים צריכים להיות קשורים להבטי רשתות (ולא הבטים מסחריים, וותק, וכד').

הגשה:

בפורמט הבא: part2_ID1_ID2.pdf

כאשר ID1 ו- ID2 אלה תעודות הזהות של המגישים.

Part 3

1. **רשות -** אתם עובדים בחברה הנותנת שירותי אחסון אובייקטים - object storage. החברה שלכם משתמשת בסביבת דיסקים הטרוגנית, שבה גדלי הדיסקים הם 256GB, 512GB, ו-1TB, ואופן פיזור המידע בדיסקים הוא בהתאם למנגנון Consistent Hashing - CH, **כפי שהוצג בהרצאה**.
   1.1. בתשתית שלכם יש:
   1000 דיסקים של 256GB
   1000 דיסקים של 512GB
   2000 דיסקים של 1TB
   מהו ה-partition power המומלץ לשימוש במערכת שלכם לצורך השגת איזון עומסים - load balancing - מיטבי?
   1.2. המנהל שלכם בחברה טוען כי מנגנון ה-CH הוא גרוע, וכי העומסים במערכת שלכם מאוד לא אחידים. בפרט הוא טוען כי הדיסקים של 1TB כמעט ריקים, בעוד הדיסקים של 256GB כמעט מלאים. הסבירו כיצד הדבר ייתכן כאשר משתמשים ב-CH כפי שהוצג בכיתה?
   1.3. הציעו גרסא משופרת של CH שבה ניתן להגיע לאיזון עומסים מיטבי מבחינת **אחוז** הדיסק המלא. הסבירו/הוכיחו את תשובתכם. שימו לב -- בתשובתכם לשאלה זו עליכם להציג גרסא משופרת כללית, לאו דווקא לסביבה הספציפית המתוארת בשאלה זו.
   1.4. לאור תשובתכם על סעיף 1.3 לעיל, כיצד תגדירו את הפרמטרים השונים במערכת CH המשופרת, העובדת בסביבה הספציפית המתוארת בשאלה זו?

הגשה:

בפורמט הבא: part3_ID1_ID2.pdf

כאשר ID1 ו- ID2 אלה תעודות הזהות של המגישים.

Part 4

בחלק זה תדרשו להשתמש בידע שלכם בסכימת MapReduce בסביבת Hadoop בכדי לפתור בעיית BigData של אתר מסחר מקוון (online retail).

תאור הבעיה:

אתם עובדים בחברה המספקת שירותי אנליזה ו-BigData למפעיל אתר מסחר מקוון גדול, עם מאות אלפי משתמשים, ועשרות אלפי פריטים העומדים למכירה באתר.
שירות שרת ה-web של הלקוח שלכם (מבוזר, כמובן, ומבוסס על עותקים רבים של השרת המטפלים בבקשות המשתמשים השונים באתר), מתחזק קבצי לוג של ביקורי משתמשים בעמודי המוצר השונים באתר, ואינדיקציה האם התבצעה רכישה במהלך הביקור באותו עמוד.
מטעמי אבטחת מידע, הפרטים המזהים של המשתמשים לא מועברים אליכם (כנותני שירות חיצוניים), וקבצי הלוג שברשותכם עברו מידה מסויימת של אנונימיזציה (ראו פרטים בהמשך).

בפרט, קבצי הלוג נשמרים בפורמט CSV, כך שכל שורה מכילה רשומה על פי פירוט השדות הבא (משמאל לימין):

Timestamp (TS), SessionID (SID), ProductPageID (PPID), PurchaseIndicator (PI)

כל רשומה כזו תהווה אירוע (event) שארע בשרת.

פירוש השדות:

1. **Timestamp - TS**:
   הזמן שבו ביקר המשתמש בעמוד.
   פורמט: YYYYMMDDhhmmss.milli
   דוגמא: 20200531195512.8776 מייצג ביקור בעמוד בתאריך 31 במאי 2020, בשעה 19:55 12 שניות, ו-8776 מילישניות.

2. **SessionID - SID:**
   מזהה ה-session של המשתמש. ניתן להתייחס לערך זה כמזהה המשתמש. למרות שאין פרטים מזהים של המשתמשים, כן ניתן לשייך רשומות שונות לאותו משתמש על פי ה-SessionID של הרשומה.
   פורמט: מספר בין 1 ל- 100,000
   דוגמא: 94211

3. **ProductPageID - PPID:**
   עמוד המוצר שבו ביקר המשתמש (שמזוהה ב-SessionID) באותו זמן (מזוהה ב-Timestamp). גם כאן, למרות שאין פרטים מזהים של המוצר, כן ניתן לשייך רשומות שונות לאותו עמוד מוצר על פי ה-ProductPageID של הרשומה.
   פורמט: מספר בין 1 ל- 10,000
   דוגמא: 8007

4. **PurchaseIndicator - PI:**
מציין האם התבצעה רכישה של המוצר (שמזוהה ב-ProductPageID) במהלך הביקור של המשתמש (שמזוהה ב-SessionID) באותו זמן (מזוהה ב-Timestamp).
פורמט: ביט 0 (אם לא התבצעה רכישה) או ביט 1 (אם התבצעה רכישה)
דוגמא: 1

הלקוח שלכם מעוניין לזהות תבניות פעילות המובילות לרכישת מוצר. בפרט, הלקוח מעוניין לזהות סדרות של 2 ביקורים של משתמש בדפים שונים בזה אחר זה, כך שבדף השלישי שהמשתמש מבקר בו מיד לאחר מכן, מתבצעת רכישה.
לדוגמא, אם לאחר איגוד כל הרשומות של משתמש מסויים (נאמר, SID1), ומיונן על פי שדה ה-Timestamp בסדר עולה נראה סדרה של רשומות:

TS1,SID1,PPID1,0
TS2,SID1,PPID2,0
TS3,SID1,PPID3,1

הלקוח ירצה לדעת כי סדרת 2 הביקורים [(PPID1,0), (PPID2,0)] הובילה לרכישה בביקור בעמוד הבא.
כדוגמא נוספת, אם באיגוד כל הרשומות של משתמש אחר (נאמר, SID2), ומיונן על פי שדה ה-Timestamp בסדר עולה נראה סדרה של רשומות:

TS4,SID2,PPID4,1
TS5,SID2,PPID5,0
TS6,SID2,PPID6,1

הלקוח ירצה לדעת כי גם סדרת 2 הביקורים [(PPID4,1), (PPID5,0)] הובילה לרכישה בביקור בעמוד הבא.
שימו לב כי הלקוח שלכם לא מתעניין בזהות הלקוח שביצע את הרכישה, וגם לא בזמן הספציפי שבו התבצעו הביקורים, אלא רק בתבנית הביקור בסדרת העמודים שהובילה לרכישה. הוא אפילו לא מתעניין במהו הפריט שנרכש בסופו של דבר.
הלקוח שלכם הסביר לכם כי אם יוכל לזהות תבניות כאלו, הוא יוכל לוודא כי הבקשה הבאה של המשתמש (ללא קשר לאיזה עמוד הוא יבקש) תופנה לעותק של שרת ה-web שיכול לתת זמן תגובה קצר ביותר (ע"י קינפוג מתאים של ה-load balancer שלו), ועל ידי כך לשפר את חווית המשתמש של אותו רוכש פוטנציאלי, ובשאיפה להגדיל את הסיכוי שתתבצע רכישה בעמוד הבא שבו יבקר.

הגדרת המשימה:

קלט: תיקיה ובה קובץ לוג אחד או יותר מן העותקים השונים של שרתי ה-web של הלקוח שלכם.
זהו הארגומנט הראשון של התוכנית, כפי שהוצג בדוגמא של MapReduce בתרגול - [0]args ב- Java
פלט: רשימה של זוגות של ביקורים בעמודים (כפי שמוסבר בדוגמא לעיל), ממויינת (בסדר יורד) על פי מספר הרכישות שהתבצעו לאחריהן. מיקום התוצאה יינתן כארגומנט השני של התוכנית, כפי שהוצג בדוגמא של MapReduce בתרגול - [1]args ב- Java

אופן הפתרון:

עליכם לפתור את הבעיה תוך שימוש בסכימת **MapReduce** בסביבת **Hadoop**. פתרונות שלא בסכימת **MapReduce** לא יתקבלו.
את הפתרון צריך לממש ב- **Java**, בצורה של chained batch, בדומה לדוגמא MapReduceSortedChain, שהוצגה בתרגול.

מה זמין לכם לצורך ביצוע המשימה:

1. קובץ מכווץ ובו הלוג של שרתי ה-web, בהתאם לפורמט שתואר לעיל.
   בנוסף קובץ מכווץ קטן. אין קשר בין הקבצים, פשוט הרבה יותר נוח לעבוד עם קובץ קטן.
2. דוגמאות קוד שהודגמו בתרגול - https://github.com/ariksa/2020_CCV/tree/master/hw4
3. כל זוג יקבל מכונה, או image של מכונה, בה מותקנים הדוף וסביבת פיתוח IntelliJ
4. כדי להפעיל את סביבת הדוף, יש להריץ את קובץ **hadoop-start.sh** בתיקיית הבית.
5. כדי להריץ את IntelliJ, יש להריץ את הקובץ **intellij-start.sh** בתיקיית הבית.

להלן מוצעים לכם מספר רמזים/שלבים מומלצים בבואכם לפתור את הבעיה. כל אחד מן השלבים ניתן (וצריך) לפתור באמצעות סכימת MapReduce על ידי הגדרת Mapper מתאים, ו-Reducer מתאים.

שלב 1:

קלט: תיקיה ובה קבצי הלוג של שרתי ה-web של הלקוח (args[0]). אתם יכולים להניח כי הקבצים תקינים ונתונים בפורמט המתואר לעיל.
פלט: סדרה של רשומות מן הצורה:

SID, EventList

כאשר SID הוא מזהה של משתמש, ו-EventList הוא רשימה של כל האירועים שמיוחסים למשתמש SID המופיעים בקבצי הלוג השונים, ממויינים על פי ה-Timestamp שלהם בסדר עולה.
הערה: שימו לב כי כשאתם מטפלים בארועים של משתמש, עדיף לתחזק מערך ממויין של ארועים, ולהכניס כל ארוע חדש למקום הנכון. (רמז - TreeSet)

שלב 2:

קלט: הפלט של שלב 1
פלט: סדרה של רשומות מן הצורה

(PPID1,bit1), (PPID2, bit2), (PPID3, bit3)

כך שכל רשומה מתאימה לסדרה של 3 אירועים של אותו משתמש (נאמר, SID) אשר קרו בזה אחר זה, כאשר bitX הוא ה-PI המתאים לאותו אירוע. שימו לב כי הפלט איננו בנוי מ-3 הרשומות המלאות של האירועים, אלא כולל רק חלק מן המאפיינים של האירועים.
לצורך המחשה, עבור הדוגמא שהובאה קודם לכן עבור משתמש SID2, אחת הרשומות בפלט צריכה להיות

(PPID4,1), (PPID5,0), (PPID6,1)

שימו לב: אם יש משתמשים שמיוחסים להם פחות מ-3 אירועים, יש להפוך את האירועים המיחסים להם לשלשה מנוונת, כך שהאירוע האחרון המיוחס למשתמש משוכפל כך שישלים ל-3 אירועים, ובכל האירועים מתייחסים ל-PI כבעל ערך 0.

שלב 3:

קלט: הפלט של שלב 2
פלט: סדרה של רשומות מן הצורה

(PPID1, bit1), (PPID2, bit2), count

כך ש-count הוא מספר הפעמים שרצף שני האירועים (PPID1, bit1) ואחריו (PPID2, bit2) הוביל לרכישה באירוע שבא אחריו (על פני כל המשתמשים).

שלב 4:

קלט: הפלט של שלב 3
פלט: גרסא ממויינת של הקלט, בסדר יורד על פי ה-count. שבירת שוויון לקסיקוגרפית על פי (PPID1, PPID2)

הפלט של שלב זה צריך להשמר לתיקיה ([1]args).

הגשה:

1. צרו **תיקיה** חדשה: **mapreduce**
2. הגישו את כל הקוד שלכם לתיקיה הנ"ל (תיקיה אחת עם כל קבצי java)

הערות:

בבדיקת המשימה, הקוד שלכם ירוץ **בסביבת הדופ**, גם על קבצי קלט אחרים (באותו פורמט). בדיקת המשימה תתבסס בין היתר על:

1. הפלט הסופי של שלב 4 ביחס לקובץ הקלט שקיבלתם.
2. הפלט של שלב 4 עבור קבצים אחרים עליהם ירוץ הקוד שלכם. הפלט עבור קבצים אלו יצטרך להיות בהתאם לנדרש.