Project **II**

Content-Centric Networking construction

TA: *Jun-Bin Zhang, Sheng-Hui Peng, An-Fong Li*

Office Hour: 15:00 - 18:00 Monday

13:00 - 18:00 Wednesday

At 95521 CPS Lab or **LINE** or **Email**

p78083025@ncku.edu.tw

2021.05.05

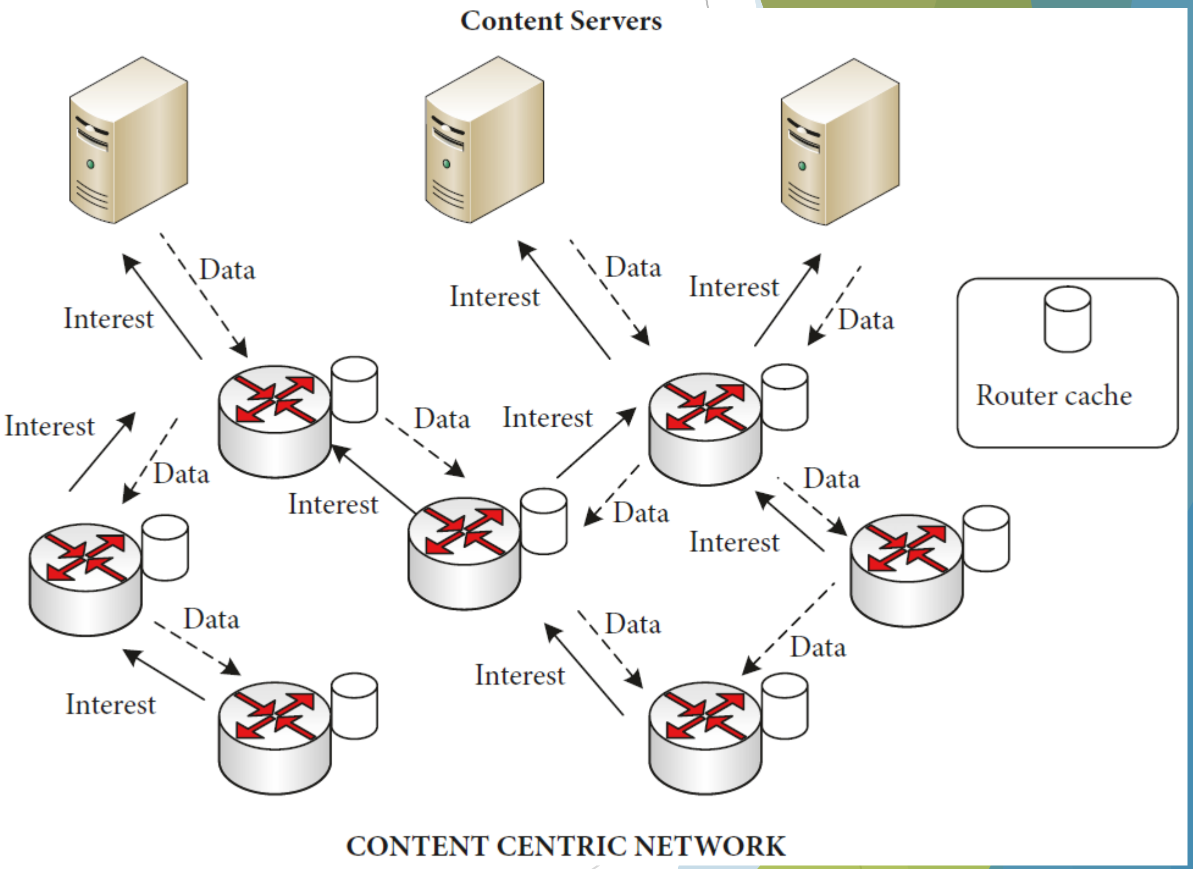
Outline

- **Introduction**
- **Project 2**
- **Network topology**
- **CCN construction**
- **Format**
- **Test data**
- **Output_interest and Output_data**
- **Process Flow**
- **Coding**

CPS LAB

1. Introduction

- **Content-Centric Networking (CCN)**
  - Use content name to replace IP addresses.
  - Focus on what users want.
  - Cache content on the router.
  - Get content responses from nearby routers without routing to servers or producers.



CPS LAB

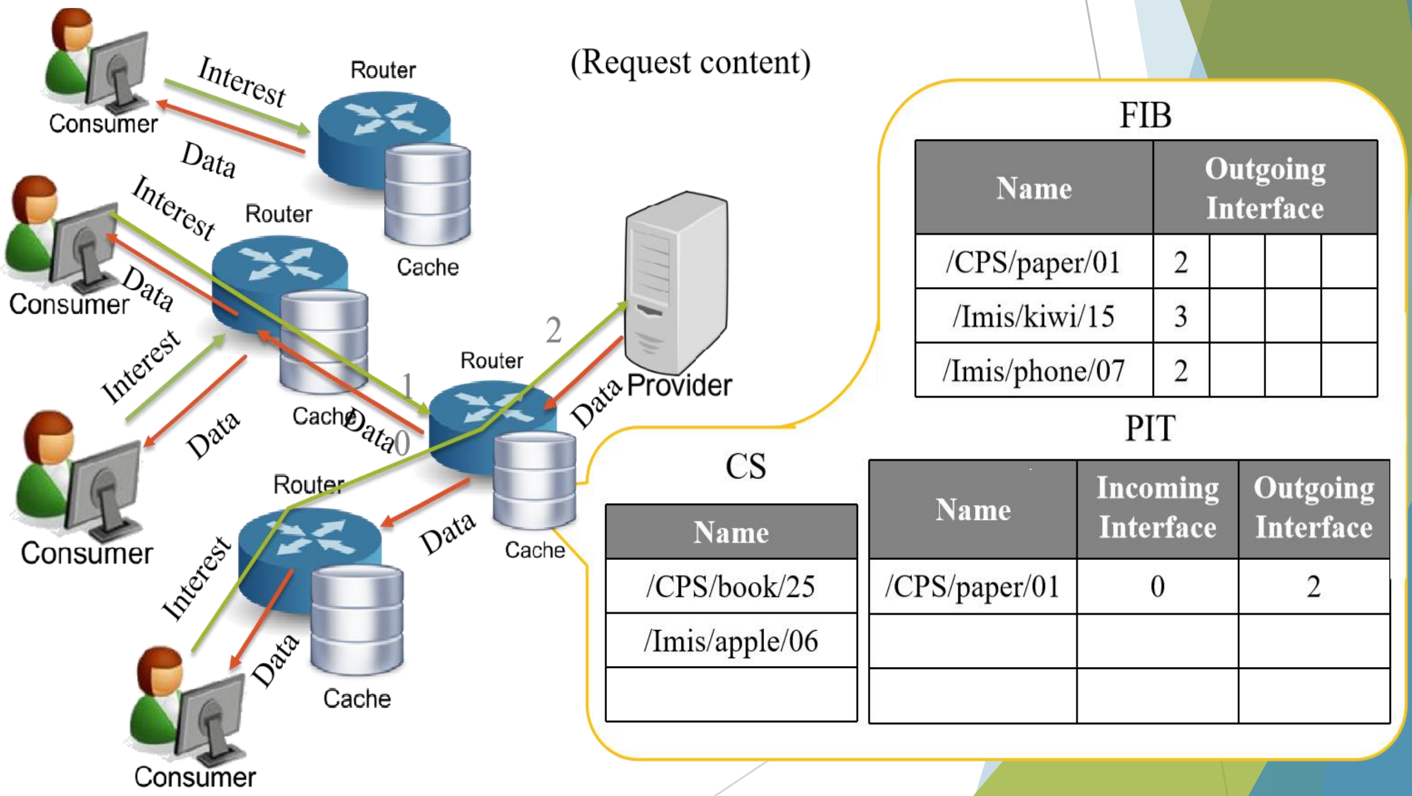
1. Introduction

- Interest Packet and Data Packet
- Content Store (**CS**), Pending Interest Table (**PIT**), Forwarding Information Base (**FIB**).
- Producer Store (PS)

Interest Packet
Name
Selector
Nonce

Data Packet
Name
Signature
Signed Info
Data

CCN packet format.



FIB

Name	Outgoing Interface
/CPS/paper/01	2
/Imis/kiwi/15	3
/Imis/phone/07	2

PIT

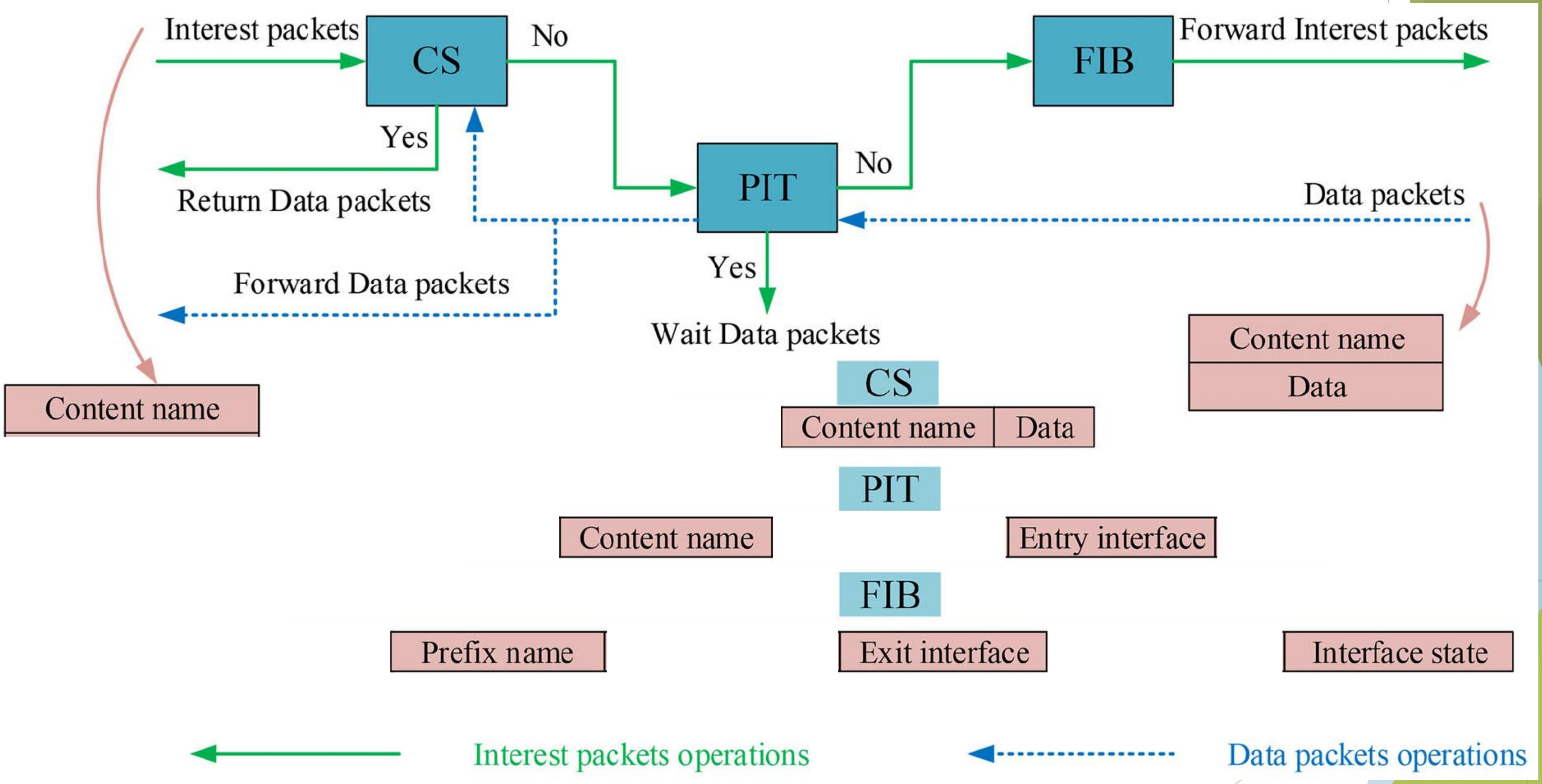
Name	Incoming Interface	Outgoing Interface
/CPS/paper/01	0	2

CS

Name
/CPS/book/25
/Imis/apple/06

Content-Centric Network architecture diagram.

1. Introduction



2019 Intelligent Information-Centric Networking routing scheme under imprecise information.pdf

2. Project 2

- You should create this CCN network based on **project 1**.
- **1. Construct a CCN simulator**

  interest.py、data.py、forward.py、ps.py、pit.py、cs.py、fib.py ......

- **2. Creat tables**

  Queue、PS、PIT、CS、FIB

Tip:

In Project 2, CS and FIB are not used, but in Project 3 they are optimized objects. That is to say, the interest packet only goes to the PS to find whether there is matching content data, not to the CS. CS also does not cache the content data in the data packet. Interest packets can only get content from the producer.

FIB is also empty and does not record routing information of data packets.

When the interest packet needs to be forwarded, it is sent to all its neighbors by broadcasting, and you can read the network to obtain neighbor connection information.

CPS LAB

2. Project 2

- **3. Test CCN simulator**

Input

read 'peremiters、networks、interests 、contents of producer' **json** files.

run your CCN simulator.

Output

real-time output packets information to save in csv files (**interest.csv** and **data.csv**).

- Interest Output.CSV format

= *[*"Time"=*str(int(time.time()-interest['run_start_time']),* "Type"=*interest['type'],* "Interest_ID"=*'I'+str(interest['interest_ID']),* "Consumer_ID"=*'C'+str(interest['consumer_ID']),* "Route_ID"=*'R'+str(interest['route_ID']),* "Content_name"=*interest['content_name'],* "Interest_hop"=*interest['interest_hop'],* "Path"=*interest['path'],* "Result"=*'Interest hit in PS',* "Hit"=*1,* "Miss"=*0]*

2. Project 2

- **3. Test CCN simulator**

Input

read 'peremiters、networks、interests、contents of producer' **json** files.

run your CCN simulator.

Output

real-time output packets information to save in csv files (**interest.csv** and **data.csv**).

- Data Output.CSV format

=["Time"=*str(times-data['run_start_time'])*, "Type"=*data['type']*, "Consumer_ID"=*'C'+str(data['consumer_ID'])*, "Route_ID"=*'R'+str(data['route_ID'])*, "Content_name"=*data['content_name']*, "Data_hop"=*data['data_hop']*, "Path"=*data['path']*, "Result"=*'Data miss in PIT'*, "Hit_consumer"=*0*, "Hit_PIT"=*0*, "Hit_Miss"=*1]*

CPS LAB

2. Project 2

- **Code**

  python

- **Upload**

  2021.06.02 10:00 am upload to moodle

- **Filename**

  project2_group1_V1.rar

2. Project 2

- **DEMO**
  - **Time:** 2021.06.02
  - **Location:** 95519 room

  - If you successfully establish CCN.

    Explain the idea of establishing a CCN simulator. Run the CCN simulator to output the interest packet and data packet information respectively.

  - If you try to establish a CCN, but it is not work.

    Explain your work and **answer** some of our questions about the PPT of Project 2.

  - If you haven't tried to establish CCN.

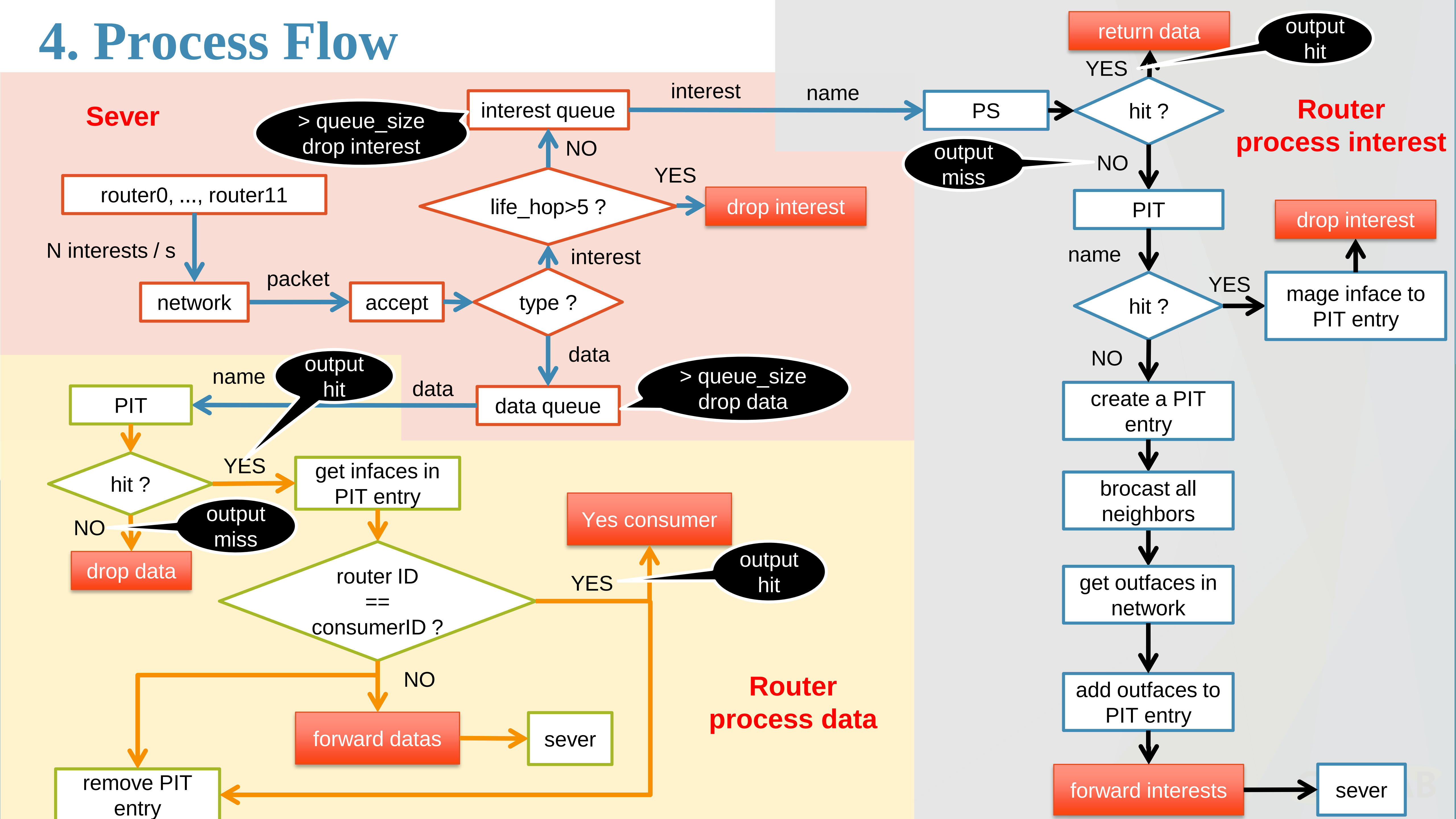
    Answer some of our questions about the PPT of Project 2.

CPS LAB

3. Network topology

- Each node is both a **producer** and a **consumer**.
- Numbers represent **router ID** and **consumer ID**
- As a **producer**, router0 will publish the content name with its own router name prefix, for example *'r0/'*. Each producer publishes 100 pieces of content, such as *'r0/00-99'*.
- As a **consumer**, router6 can send out an interest packet to request any content name with a router name prefix, such as *'r6/88'*.

4. Process Flow



5. CCN construction

- **main.py**
- **sever.py**
- **network.py**

- **ps.py**
- **pit.py**
- cs.py
- fib.py

- **interest.py**
- **data.py**
- **forward.py**

6. Format

- Route_ID = *0 1 … 11*
- Consumer_ID = *0 1 … 11*
- Interest_ID = *routerID + 0-99*   ex. 0000, 0001, 1000... 1100
- Content_name = *'route_ID/0-99'*   ex. 'r0/00'

Tip:

When output packet informaton to CVS, please convert router ID, consumer ID, interest ID into string type, and add ‘R’, ‘C’,’I’ in front.

ex. ‘R1’, ‘C10’, ’I1003’

6. Format

- network = *{'r'+'route_ID': [route_ID, ...], ... }*

  ex. network = *{'r1': [0, 2], 'r2': [1, 3, 4], ... }*

- producer_contents = *{'r'+'route_ID': [content_name, ...], ... }*

  ex. producer_contents = *{'r0': ['r0/00', 'r0/01', ..., 'r0/99'], 'r1': ['r1/00', 'r1/01', ..., 'r1/99'], ... }*

Tip:

You can read directly from the json file we provide. Then read each router's own part according to the *router ID*.

6. Format

- Interest packets sent by consumer

start_packets = *[{'type': 'interest', 'interest_ID': '0000', 'consumer_ID': 0, 'route_ID': 0, 'content_name': 'r4/01', 'interest_hop': 0, 'life_hop': 5, 'run_start_time': 1615357629, 'path': '0'}, ...*

{'type': 'interest', 'interest_ID': '0004', 'consumer_ID': 0, 'route_ID': 0, 'content_name': 'r7/08', 'interest_hop': 0, 'life_hop': 5, 'run_start_time': 1615357629, 'path': '0'}]

Tip:

Consumers send new *interests* to the network at a certain *frequency*. You can read by *router ID* and order from the interest json file.

Then, the *interests* you read from the json file only contains the *interest ID* and *content name*, so you need to add other information about the *interest* in the above format.

run_start_time is the time when the simulator starts to run. It is used to calculate how many seconds the current simulator has executed.

6. Format

- Interest packets received by router

interest = {*'type': 'interest', 'interest_ID': 0000, 'consumer_ID': 0, 'route_ID': 0, 'content_name': 'r0/00', 'interest_hop': 0, 'life_hop': 5, 'run_start_time': 0, 'path': 'p0/'}*

- *'route_ID':* The *router ID* of the previous router that forwarded the *interest*, not the current *router_ID.*
- *'interest_hop'* : Add 1 every time *interest* are forwarded
- *'life_hop'*: The maximum number of times an *interest* is forwarded
- *'run_start_time'*: The time when the simulator started running
- *'path'*: The current *router_ID* is added every time the *interest* is forwarded, and "/" is added.

Tip:

The router forwards an *interest* once, and the number of *interest_hop + 1*. The same is true for the number of *data_hop* in the *data packet*.

When the router receives an *interest*, it will read *life_hop* and determine whether it is > 5. If yes, the *interest* droped and not be stored in the *queue*.

6. Format

- Data packets received by router

data = *{'type': 'data', 'consumer_ID': 0000, 'route_ID': 0, 'content_name': 'r0/00', 'content_data': 'r0/000', 'data_hop': 0, 'run_start_time': 0, 'path': 'p0/'}*

- *'route_ID':* The *router ID* of the previous router that forwarded the *data packet*, not the current *router_ID*.
- *'content_data': = 'content_name'* + *str(current time)*
- *'data_hop'* : Add 1 every time *data packet* is forwarded
- *'run_start_time'*: The time when the simulator started running
- *'path'*: The current *router_ID* is added every time the *data packet* is forwarded, and "/" is added.

Tip:

The router forwards an *data packet* once, and the number of *data_hop* + *1*.

6. Format

- **ps** = *[content_name, ...]*
- **pit** = *{'content_name': [[inface, ...], [outface, ...]], ... }*
- **pit_entry** = *[[inface, ...], [outface, ...]]*

- cs = *[[content_name, data, time, cost], ...]*
- cs_entry = *[content_name, data, time, cost]*
- fib = *{'content_name': [[outface, cost, time], ...], ... }*
- fib_entry = *[[outface, cost, time], ...]*

Tip:

CS and FIB will not be used in project 2, but you can write the program first. Each router has these tables independently.

6. Format

- interest hit in PS

Create a *data packet*, the format is as follows

data packet =

[[5, {'type': 'data', 'consumer_ID': 0, 'route_ID': 4, 'content_name': 'r4/01', 'content_data': 'r4/011615357625', 'data_hop': 1, 'run_start_time': 1615357618, 'path': 'p4/'}]]

- 'content_data': = 'content_name' + str(current time)

6. Format

- interest miss in PS

Create forwarding *interest packets*, which may need to be forwarded to multiple *outgoing interfaces*. The first number of each list is the *outgoing interface* (*router ID*). The format is as follows

interest packets =

*[[**3**, {'type': 'interest', 'interest_ID': '0001', 'consumer_ID': 0, 'route_ID': 5, 'content_name': 'r7/07', 'interest_hop': 3, 'life_hop': 5, 'run_start_time': 1615357599, 'path': 'p0/3/5'}],*

*[**4**, {'type': 'interest', 'interest_ID': '0001', 'consumer_ID': 0, 'route_ID': 5, 'content_name': 'r7/07', 'interest_hop': 3, 'life_hop': 5, 'run_start_time': 1615357599, 'path': 'p0/3/5'}],*

*[**5**, {'type': 'interest', 'interest_ID': '0001', 'consumer_ID': 0, 'route_ID': 5, 'content_name': 'r7/07', 'interest_hop': 3, 'life_hop': 5, 'run_start_time': 1615357599, 'path': 'p0/3/5'}]]*

6. Format

- data hit in PIT

Create forwarding *data packets*, which may need to be forwarded to multiple *incoming interfaces*. The first number of each list is the *incoming interface* (*router ID*). The format is as follows

data packets =

[[2, {'type': 'data', 'consumer_ID': 0, 'route_ID': 4, 'content_name': 'r5/09', 'content_data': 'r5/091615357621', 'data_hop': 2, 'start_time': 1615357615, 'path': 'p5/4/'}],

[6, {'type': 'data', 'consumer_ID': 0, 'route_ID': 4, 'content_name': 'r5/09', 'content_data': 'r5/091615357621', 'data_hop': 2, 'start_time': 1615357615, 'path': 'p5/4/'}]]

- 'content_data': = 'content_name' + str(current time)

7. Test data

- **networks**
- json={

```
    "r0":[1, 3],
    "r1": [0, 2, 3],
    "r2": [1, 4],
    …
    "r11": [7, 10]
  }
```

Tip:

The key is the *router ID* of **string type**. The value is a list, and the numbers in the list are the **numeric type** of the *router ID*.

7. Test data

- **parameters**
- json={

```
 "route_num": 12,          # number of routers
 "frequency": 2,           # 2 new interest packet / second sent by each router to network
 "content_num": 100,       # number of content published by each producer
 "run_time": 100,          # Simulator running time
 "queue_size": 10,         # number of packet stored in the queue
 "cache_size": 10,         # number of content data stored in cs
 "FIB_size": 50            # number of entry stored in the fib
}
```

Tip:

In project 2, when the queue is **full**, the newly received packet is **droped**.

7. Test data

- **producer_contents**

  12 note *100 content =1200 total

- json={

  "r0": ["r0/0", "r0/1", "r0/2", ..., "r0/98", "r0/99"],

  "r1": ["r1/0", "r1/1", "r1/2", ..., "r1/98", "r1/99"],

  "r2": ["r2/0", "r2/1", "r2/2", ..., "r2/98", "r2/99"],

  ...

  "r11": ["r11/0", "r11/1", "r11/2", ..., "r11/98", "r11/99"]

  }

Tip:

The key is the *router ID* of **string type**. The value is a list containing one **100** ***content names*** prefixed with its own *router ID*.

7. Test data

- **interests**
- 5 interest/s*12 = 60 interest/s   12*100 interest=1200 interests
- json={

```
 "r0": [{'interest_ID': 0000,'content_name':'r1/3'}, ...],
 "r1": [{'interest_ID': 1000,'content_name':'r9/2'}, ...],
 "r2": [{'interest_ID': 2000,'content_name':'r11/8'}, ...],
 ...
 "r11": [{'interest_ID': 1100,'content_name':'r6/7'}, ...]
}
```

Tip:

The key is the *router ID* of **string type**. The value is a list containing one **100** *interest packets*. Each *interest packet* is a **dictionary** containing *interest_ID* and *content_name*.

7. Test data

- **interests**
- 5 interest/s*12 = 60 interest/s   12*100 interest=1200 interests
- json={

"r0": [{'interest_ID': 0000,'content_name':'r1/3'}, ...],

"r1": [{'interest_ID': 1000,'content_name':'r9/2'}, ...],

"r2": [{'interest_ID': 2000,'content_name':'r11/8'}, ...],

...

"r11": [{'interest_ID': 1100,'content_name':'r6/7'}, ...]

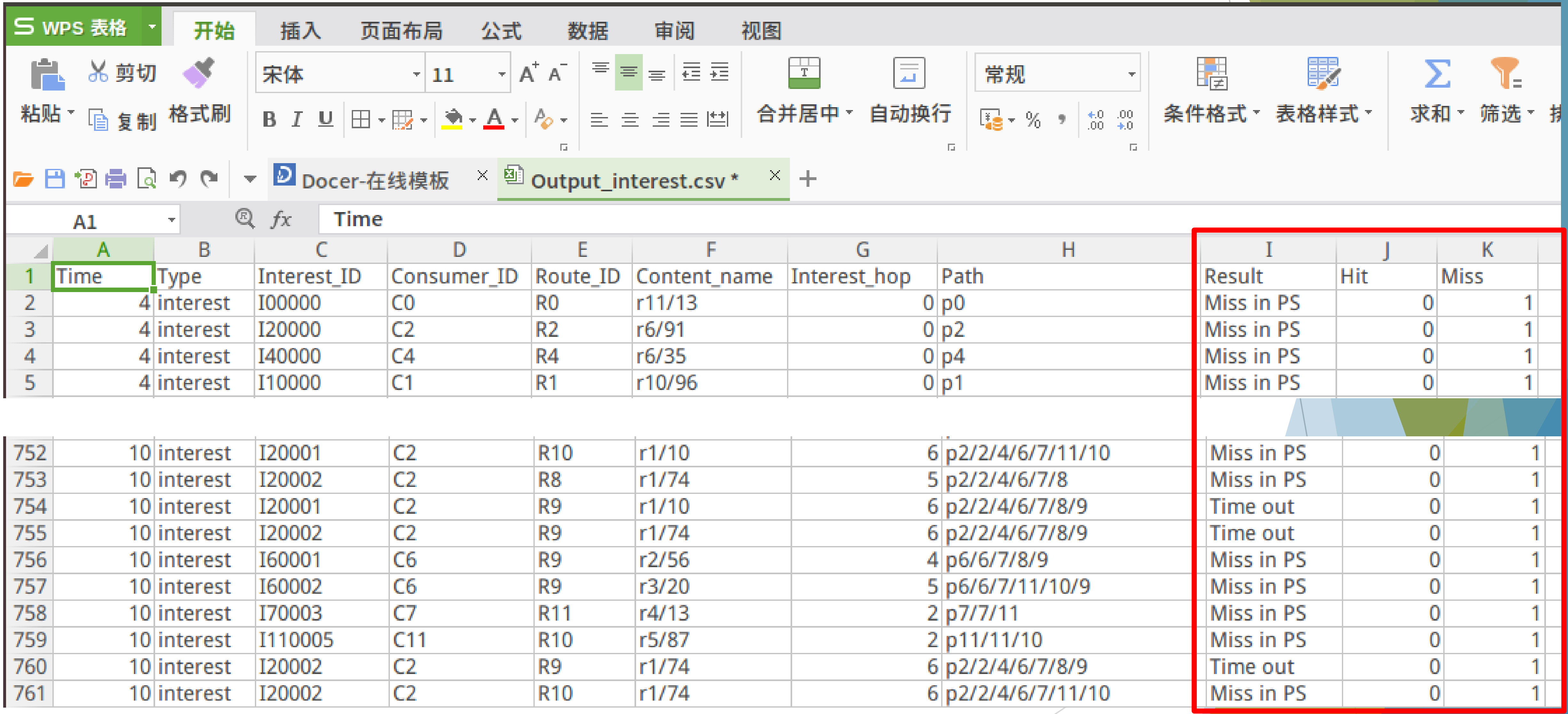
}

Tip:

interest_ID = routerID + 00-99

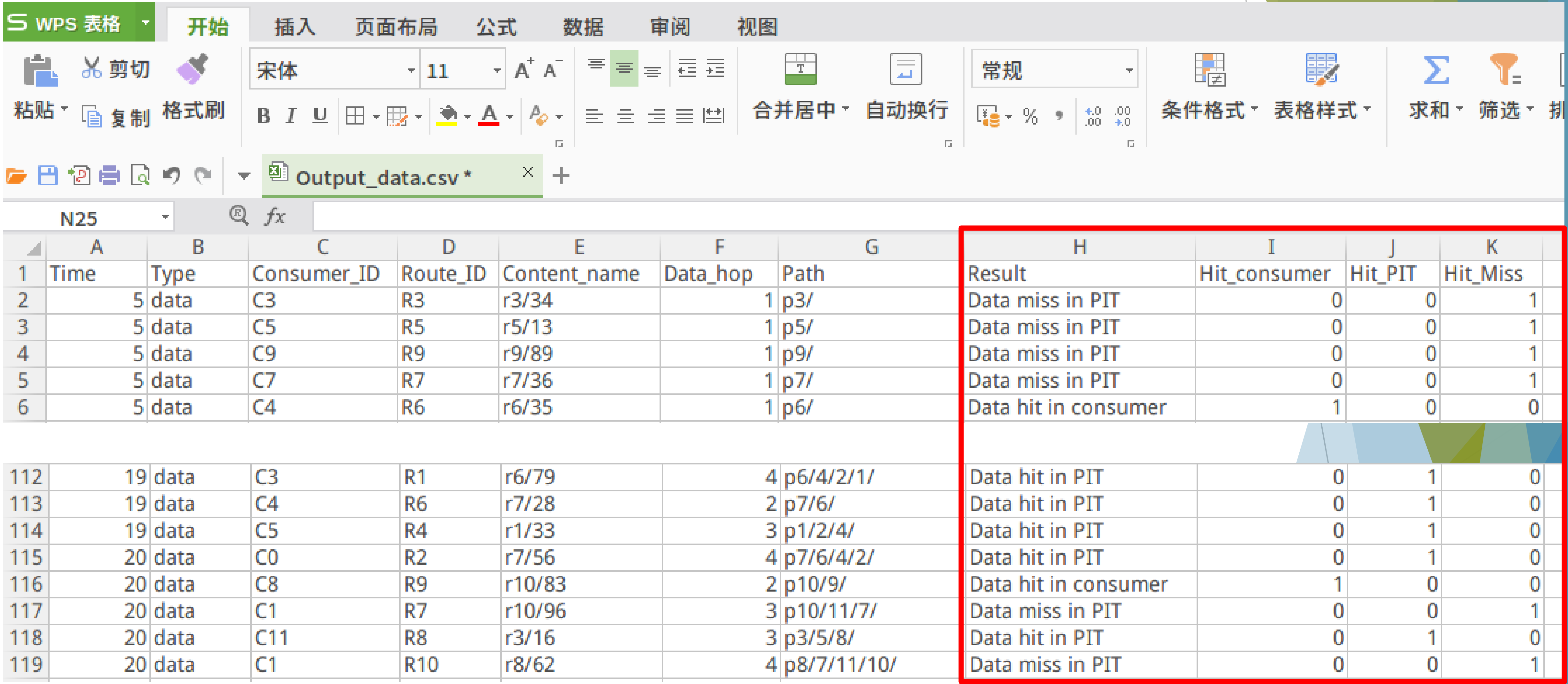
content_name is randomly selected from 1200 *content* published by 12 producers. Therefore, some *content names* may be repeatedly selected.

8. Output_interest



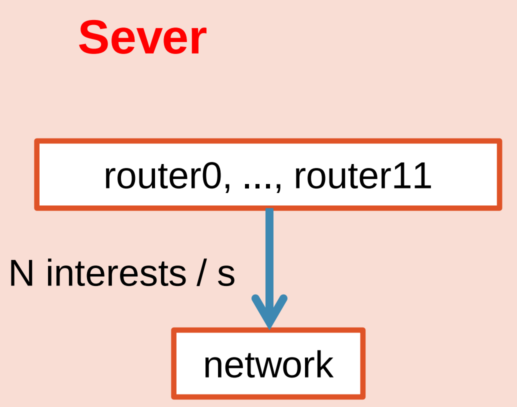
	A	B	C	D	E	F	G	H	I	J	K
1	Time	Type	Interest_ID	Consumer_ID	Route_ID	Content_name	Interest_hop	Path	Result	Hit	Miss
2	4	interest	I00000	C0	R0	r11/13	0	p0	Miss in PS	0	1
3	4	interest	I20000	C2	R2	r6/91	0	p2	Miss in PS	0	1
4	4	interest	I40000	C4	R4	r6/35	0	p4	Miss in PS	0	1
5	4	interest	I10000	C1	R1	r10/96	0	p1	Miss in PS	0	1
752	10	interest	I20001	C2	R10	r1/10	6	p2/2/4/6/7/11/10	Miss in PS	0	1
753	10	interest	I20002	C2	R8	r1/74	5	p2/2/4/6/7/8	Miss in PS	0	1
754	10	interest	I20001	C2	R9	r1/10	6	p2/2/4/6/7/8/9	Time out	0	1
755	10	interest	I20002	C2	R9	r1/74	6	p2/2/4/6/7/8/9	Time out	0	1
756	10	interest	I60001	C6	R9	r2/56	4	p6/6/7/8/9	Miss in PS	0	1
757	10	interest	I60002	C6	R9	r3/20	5	p6/6/7/11/10/9	Miss in PS	0	1
758	10	interest	I70003	C7	R11	r4/13	2	p7/7/11	Miss in PS	0	1
759	10	interest	I110005	C11	R10	r5/87	2	p11/11/10	Miss in PS	0	1
760	10	interest	I20002	C2	R9	r1/74	6	p2/2/4/6/7/8/9	Time out	0	1
761	10	interest	I20002	C2	R10	r1/74	6	p2/2/4/6/7/11/10	Miss in PS	0	1

8. Output_data



	A	B	C	D	E	F	G	H	I	J	K
1	Time	Type	Consumer_ID	Route_ID	Content_name	Data_hop	Path	Result	Hit_consumer	Hit_PIT	Hit_Miss
2	5	data	C3	R3	r3/34	1	p3/	Data miss in PIT	0	0	1
3	5	data	C5	R5	r5/13	1	p5/	Data miss in PIT	0	0	1
4	5	data	C9	R9	r9/89	1	p9/	Data miss in PIT	0	0	1
5	5	data	C7	R7	r7/36	1	p7/	Data miss in PIT	0	0	1
6	5	data	C4	R6	r6/35	1	p6/	Data hit in consumer	1	0	0
112	19	data	C3	R1	r6/79	4	p6/4/2/1/	Data hit in PIT	0	1	0
113	19	data	C4	R6	r7/28	2	p7/6/	Data hit in PIT	0	1	0
114	19	data	C5	R4	r1/33	3	p1/2/4/	Data hit in PIT	0	1	0
115	20	data	C0	R2	r7/56	4	p7/6/4/2/	Data hit in PIT	0	1	0
116	20	data	C8	R9	r10/83	2	p10/9/	Data hit in consumer	1	0	0
117	20	data	C1	R7	r10/96	3	p10/11/7/	Data miss in PIT	0	0	1
118	20	data	C11	R8	r3/16	3	p3/5/8/	Data hit in PIT	0	1	0
119	20	data	C1	R10	r8/62	4	p8/7/11/10/	Data miss in PIT	0	0	1

9. Process Flow



- ## Sever process
- The network has 12 node. You can read from the **networks** JSON file .
- networks format

= *{'r'+'route_ID': [route_ID, ...], ... }*

= *{'r1': [0, 2], 'r2': [1, 3, 4], ... }*

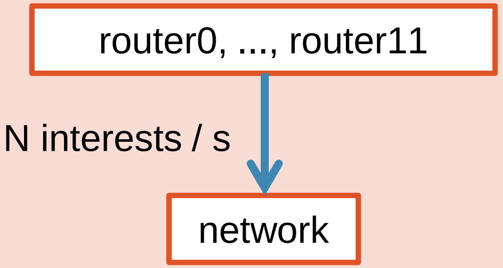
- Each node is both a **producer** and a **consumer**.
- **router ID** = **consumer ID =** *0 1 ... 11*
- As a **producer**, router0 will publish the *content name* with its own router name prefix, for example *'r0/'*. Each producer publishes 100 pieces of content, such as *'r0/00-99'*. You can read from the **producer_contents** JSON file .
- producer_contents format

= *{'r'+'route_ID': [content_name, ...], ... }*

= *{'r0': ['r0/00', 'r0/01', ..., 'r0/99'], 'r1': ['r1/00', 'r1/01', ..., 'r1/99'], ... }*

9. Process Flow

Sever



router0, ..., router11

N interests / s

network

- ## Sever process

- As a **consumer**, router1can send out *interest packet* to request any content name with a router name prefix, such as *'r1/88'*.

- All routers simultaneously send out ***N* new interests / s** to the network.

- *N* = *frequency*. You can read from the **parameters** and **interests** JSON file based on the *router name*. *interests* has only *interest_ID* and *content name*. You need to fill in other information according to the complete format of the *interest*.

- interests format

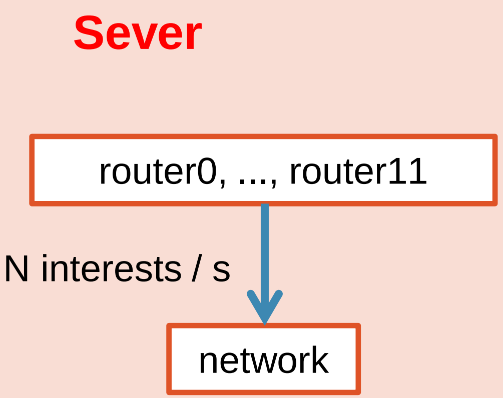
={"r0": [{'interest_ID': 0000,'content_name':'r1/3'}, ...], "r1": [{'interest_ID': 1000,'content_name':'r9/2'}, ...], ... "r11": [{'interest_ID': 1100,'content_name':'r6/7'}, ...]}

- interest format

= {'type': 'interest', 'interest_ID': 0000, 'consumer_ID': 0, 'route_ID': 0, 'content_name': 'r0/00', 'interest_hop': 0, 'life_hop': 5, 'run_start_time': 0.0, 'path': 'p0/'}

9. Process Flow

- ## Sever process
- Get N in turn from the *interests* belonging to this router. And package them *start_packets*, and then hand them to the server to send them to the network.



- Interest packets format sent by consumer
- start_packets

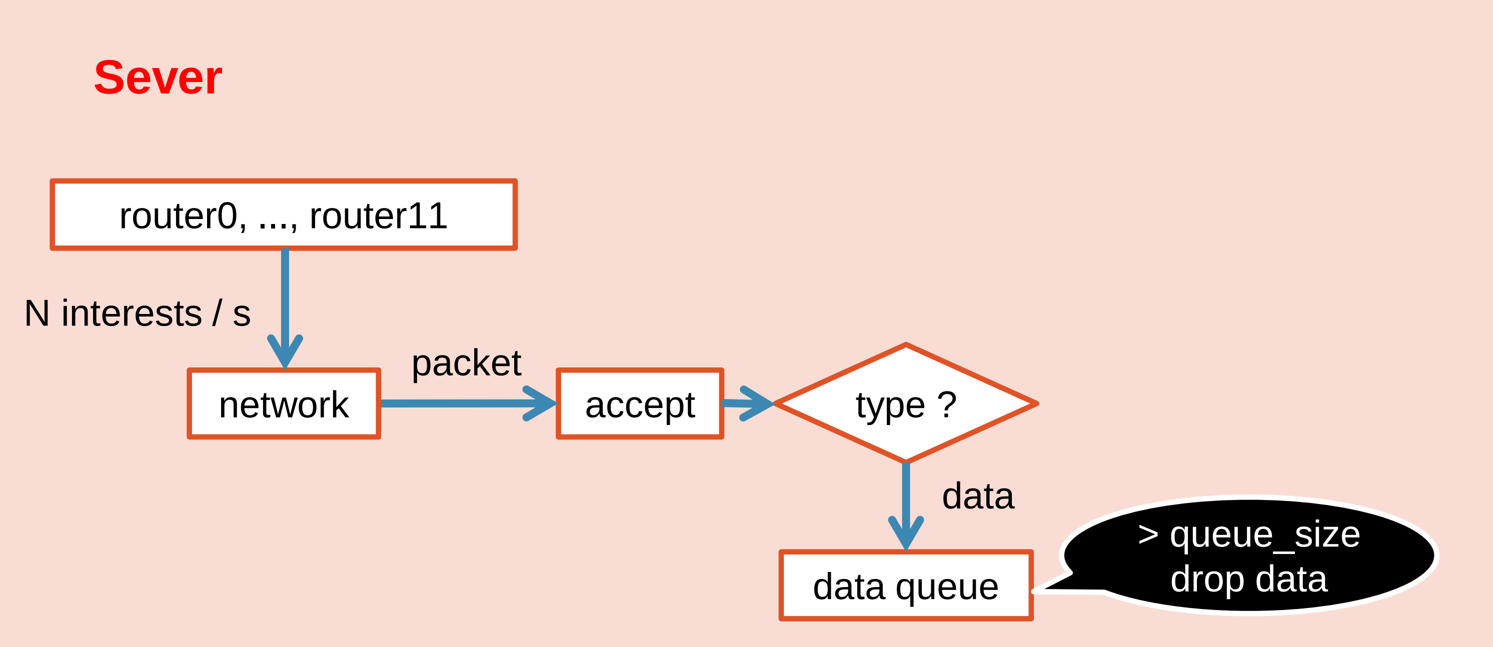
= [{'type': 'interest', 'interest_ID': '0300', 'consumer_ID': 3, 'route_ID': 3, 'content_name': 'r7/01', 'interest_hop': 0, 'life_hop': 5, 'run_start_time': 1615357629, 'path': 'p3/'},

.....

{'type': 'interest', 'interest_ID': '0304', 'consumer_ID': 3, 'route_ID': 3, 'content_name': 'r11/8', 'interest_hop': 0, 'life_hop': 5, 'run_start_time': 1615357629, 'path': 'p3/'}]

9. Process Flow

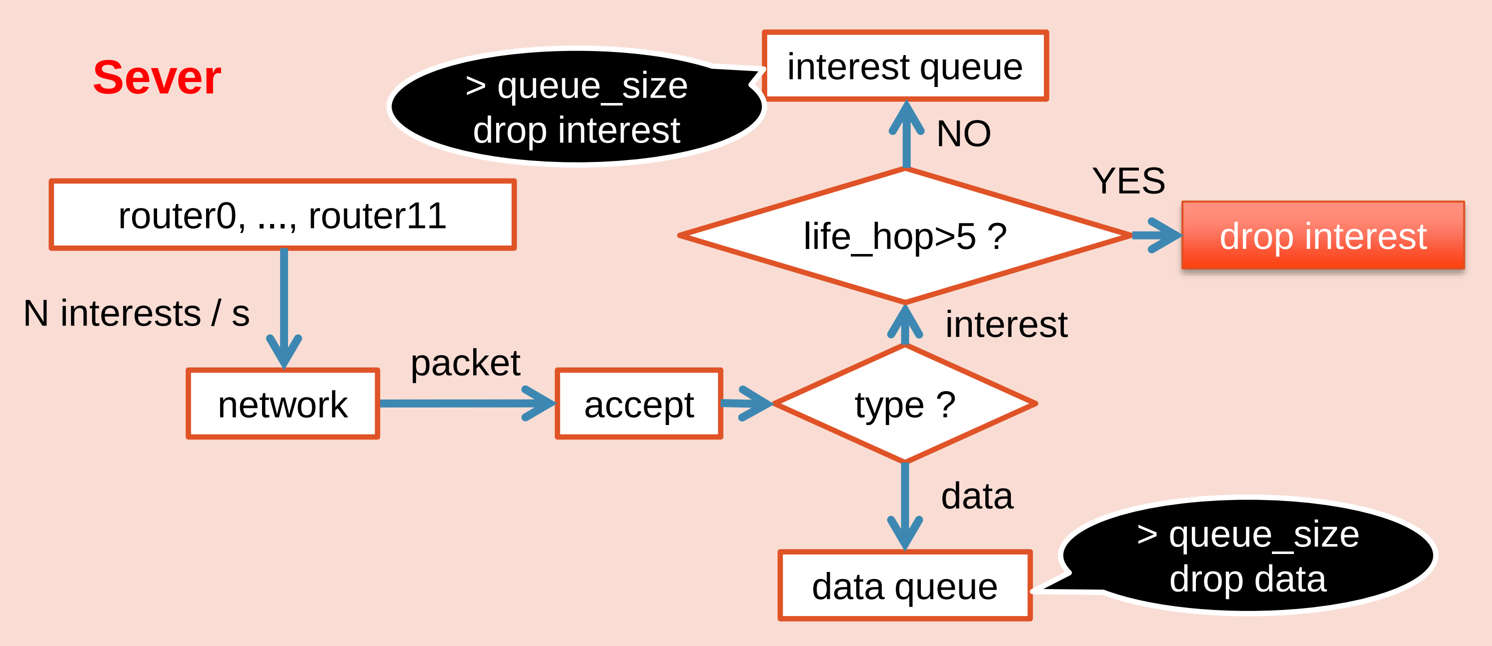
- **Sever process**
- When the accept receives a *packet*, it needs to read the *type* of *packet* and determine whether it is *interest* or *data*.



- If it is a *data packet*, you need to determine whether the *data_queue* size is greater than the *queue_size*. You can read the *queue_size* from the **parameter** JSON file.
- If it is YES, it indicates that the *data_queue* is full, and the received *packet* is droped and not stored in the *data_queue*.
- If NO, then store the received *packet* in the *data_queue*.

CPS LAB

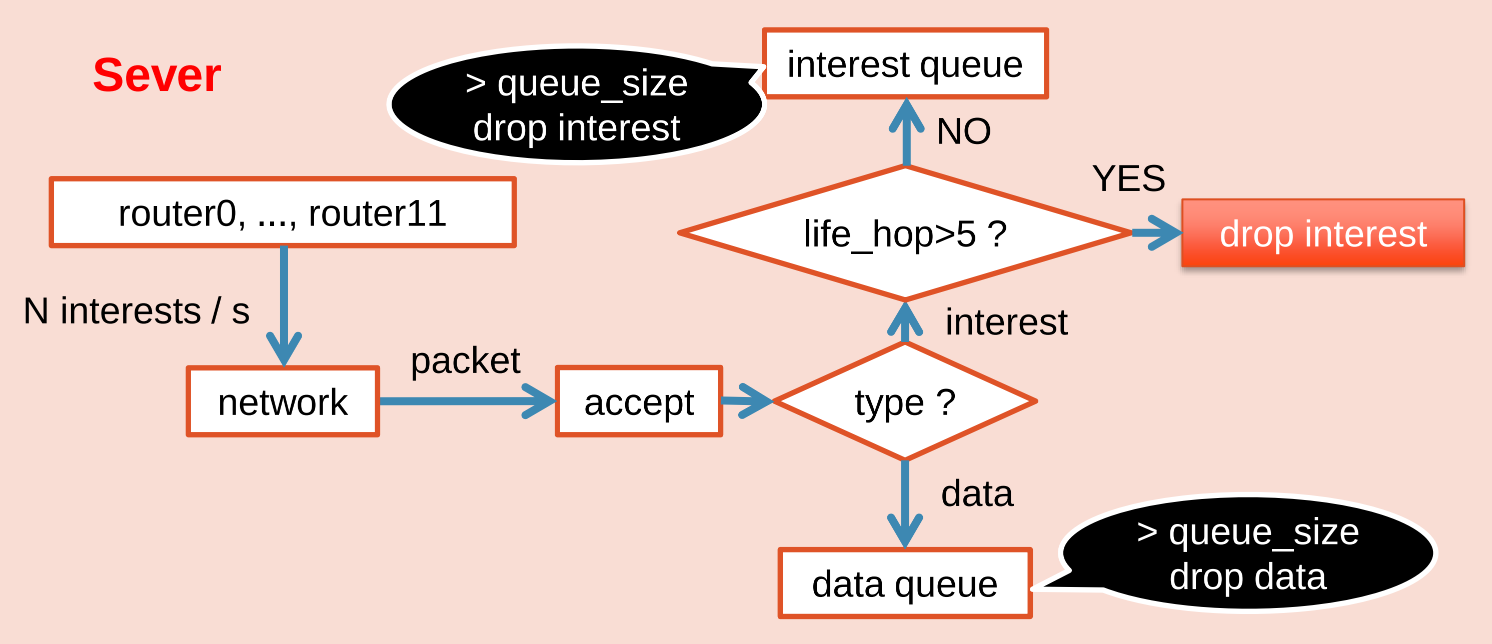
9. Process Flow



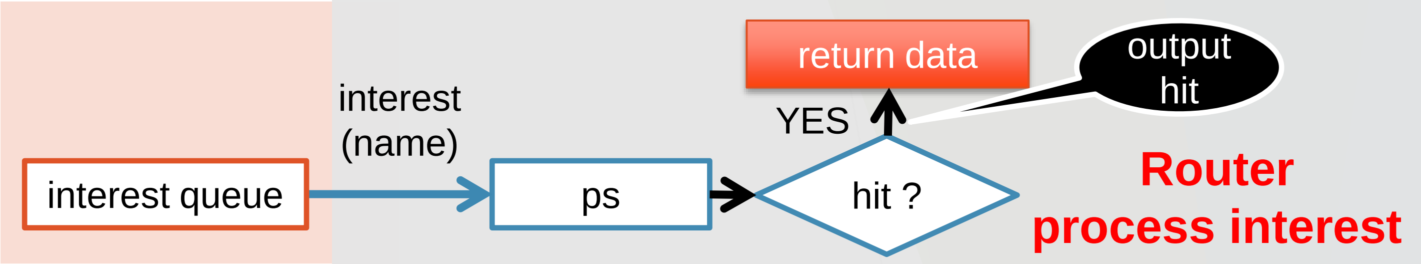
- ## Sever process
- If it is a *interest packet*, you need to read the *life_hop* from the *packet*.
- Then see if it is greater than 5. If YES, then drop the *interest packet*. Because we set an *interest packet* to have a life cycle of only 5 hops in the network.
- If NO, you need to determine whether the *interest_queue* size is greater than the *queue_size*. You can read the *queue_size* from the **parameter** JSON file.
- If it is YES, it indicates that the *interest_queue* is full, and the received *packet* is droped and not stored in the *interest_queue*.
- If NO, then store the received *packet* in the *interest_queue*.

9. Process Flow

- ## Sever process
- After the simulator runs for a fixed period of time, it exits. You can read *run_time* from the **parameters** JSON file.
- The server monitors whether a *packet* is received in real time.
- The router takes the first *packet* from the *interest_queue* and *data_queue* for processing each time.



9. Process Flow



Router process interest

- ## Interest process

- The router takes the first *packet* from the *interest_queue* for processing each time.

- The router gets *name* from packet. Go to ***ps*** *(producer store)* to find if there is matching *data*.

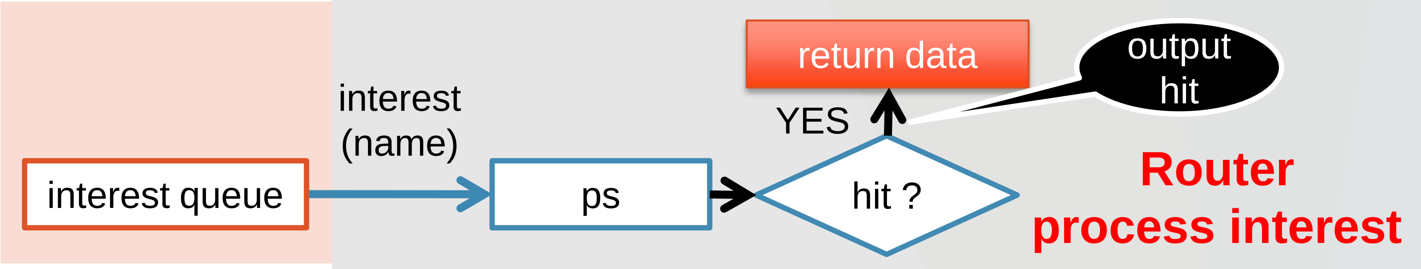
- ***ps*** stores 100 content published by each producer, such as *'r0/00-99'*. You can read from the **producer_contents** JSON file.

- ***ps*** format

= *[content_name, ...]*

- If it hits, *data packet* is returned.

9. Process Flow



Router process interest

- **Interest process**
- Create a data packet.
- *data packet* format

= *[[5, {'type': 'data', 'consumer_ID': 0, 'route_ID': 9, 'content_name': 'r9/01', 'content_data': 'content_name' + str(current time), 'data_hop': 1, 'run_start_time': 1615357618, 'path': 'p9/'}]]*

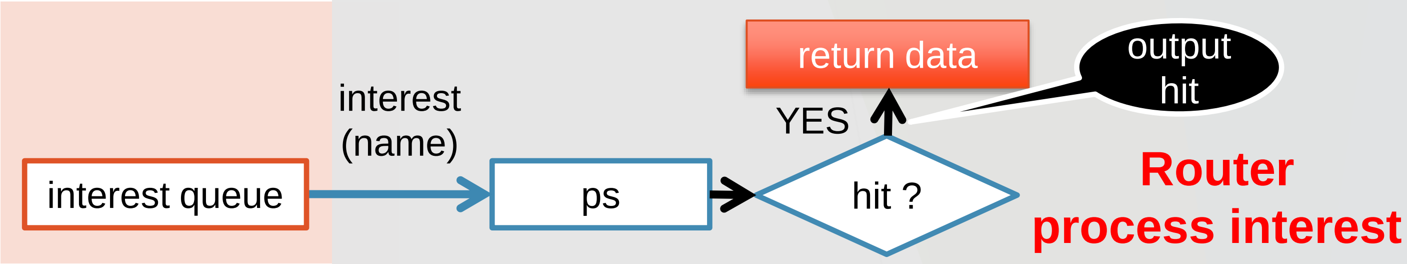
☆ The first element 5 of the list represents the *incoming interface* (it is *router_ID* in *interest*) of the *interest.*

☆ The second element of the list is a dictionary, which represents a *data packet.*

☆ *content_data* format

= *'content_name' + str(current time)*

9. Process Flow



Router
process interest

- **Interest process**
- Output 'Interest hit in PS' to CSV.
- Interest Output.CSV format

```
= [
"Time"=str(int(time.time()-interest['run_start_time']),
"Type"=interest['type'],
"Interest_ID"='I'+str(interest['interest_ID']),
"Consumer_ID"='C'+str(interest['consumer_ID']),
"Route_ID"='R'+str(interest['route_ID']),
"Content_name"=interest['content_name'],
"Interest_hop"=interest['interest_hop'],
"Path"=interest['path'],
"Result"='Interest hit in PS',
"Hit"=1,
"Miss"=0
]
```

9. Process Flow

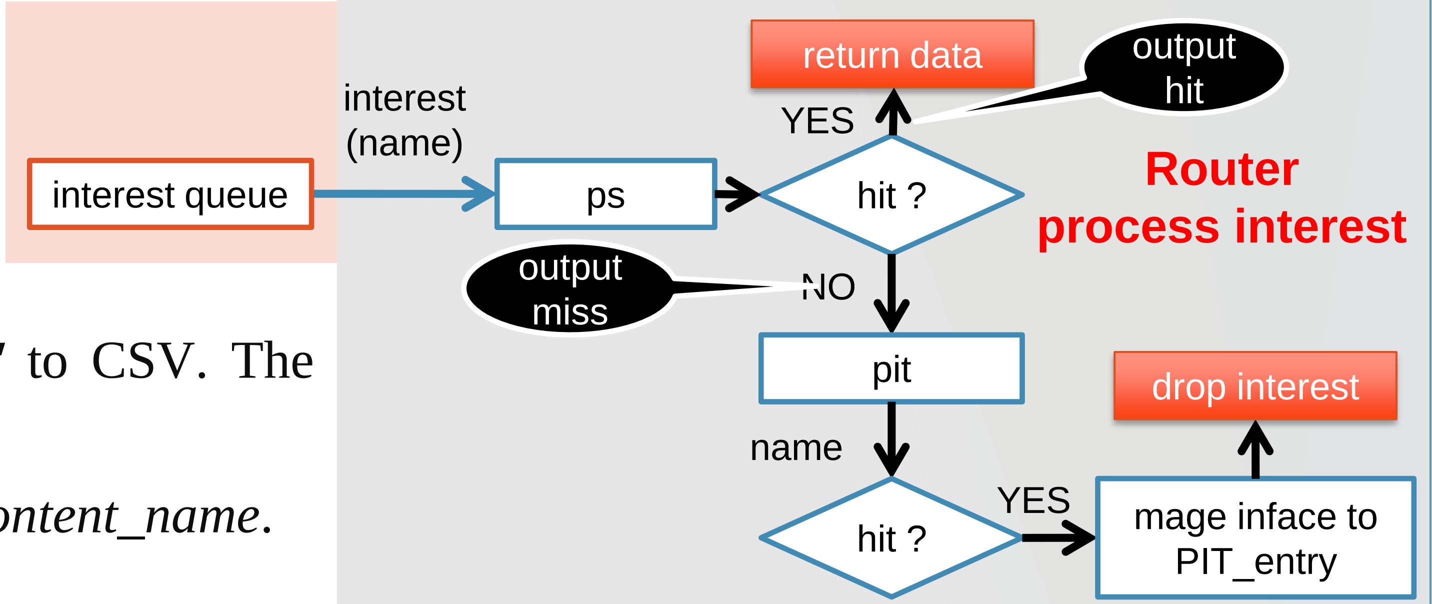
- **Interest process**

- If miss in *ps*, output *'Interest miss in PS'* to CSV. The format following above.
- And go to *pit* to find a matching *entry* by *content_name*.
- ***pit*** format

= *{'content_name': [[inface, ...], [outface, ...]], ... }*

- ***PIT_entry*** format

= *[[inface, ...], [outface, ...]]*

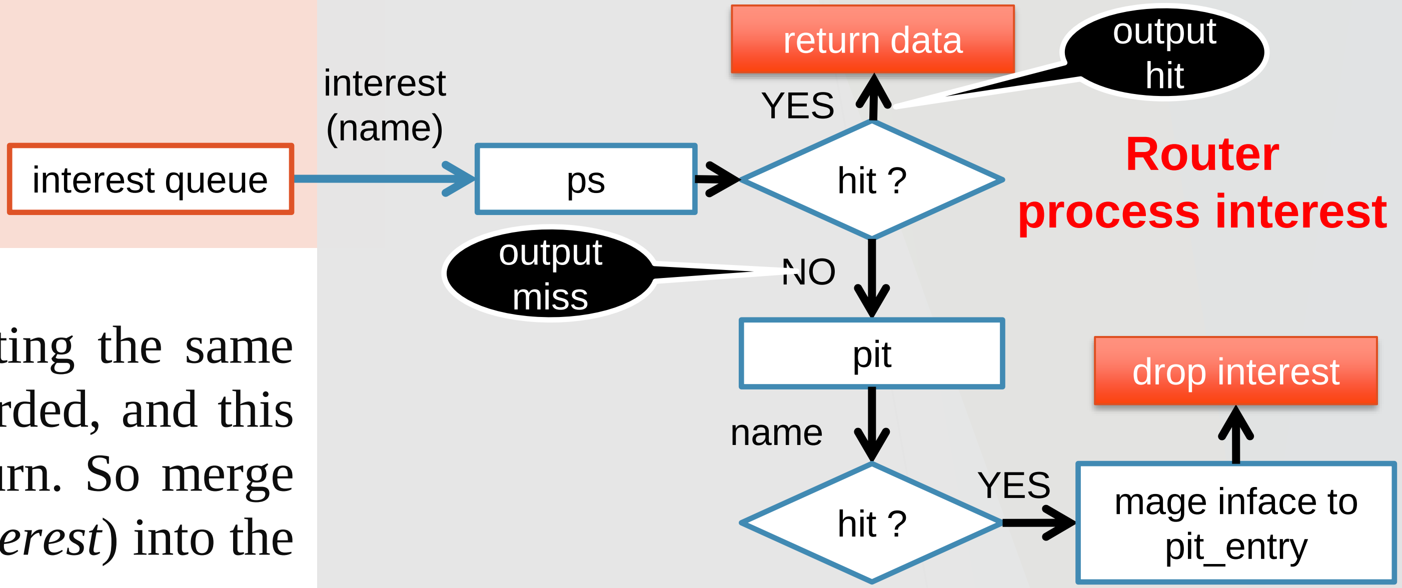


9. Process Flow

- **Interest process**
- If it hits, it means that the *interest* requesting the same *content name* has been received and forwarded, and this *interest* only needs to wait for *data* to return. So merge the *incoming interface* (it is *router_ID* in *interest*) into the *inface* of entry.
- ***PIT_entry*** format

= *[[inface, ...], [outface, ...]]*

- You may want to check to make sure that there is no duplicate *inface* in *PIT_entry* .
- And then drop the *interest*. The router takes a *data packet* from the *data_queue* for processing.



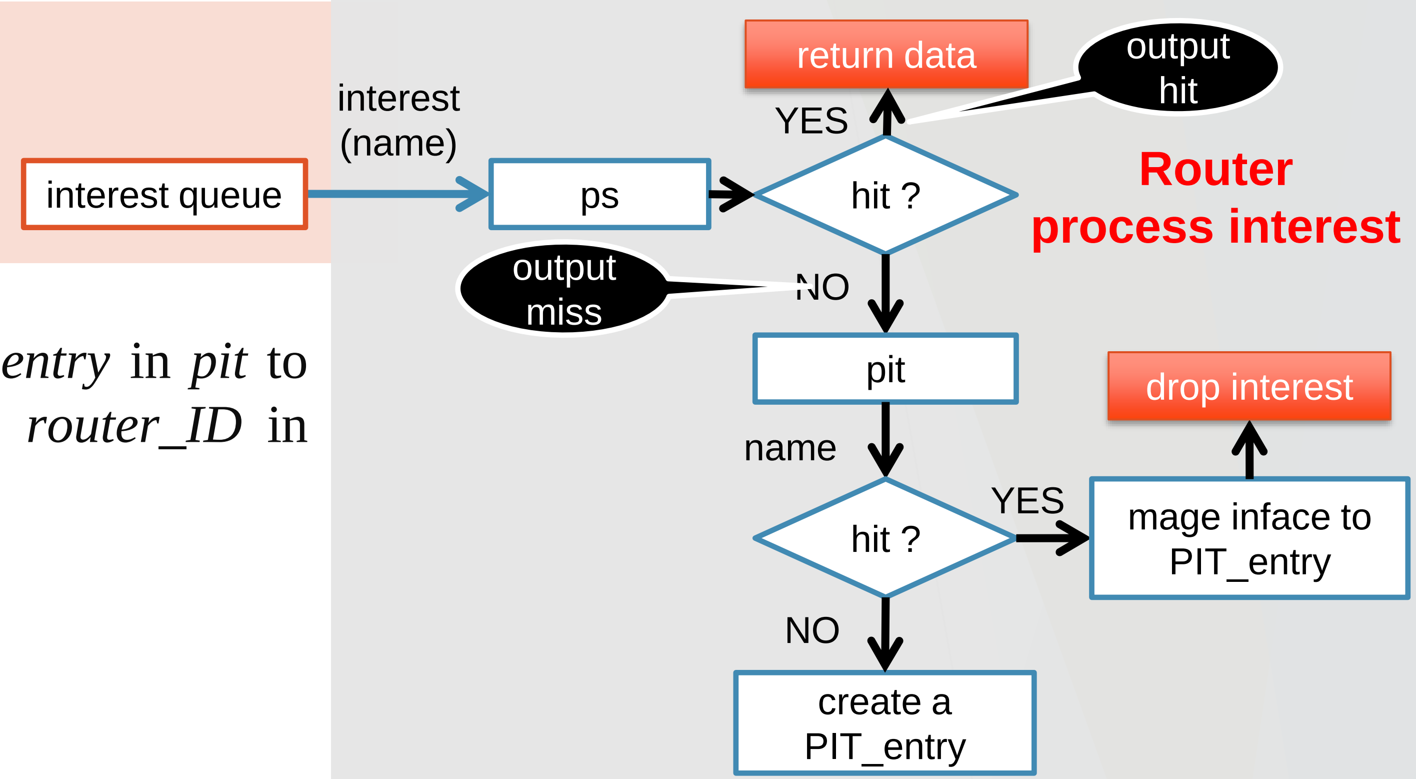
9. Process Flow

- **Interest process**
- If *content name* miss in *pit*, create a new *entry* in *pit* to record the *content name* and *inface* (it is *router_ID* in *interest*).
- ***PIT_entry***

= *[[inface], []]*

- ***pit***

= *{... , 'content_name': [[inface], []]}*



Router process interest

9. Process Flow

- ## Interest process

- Then forward the *interest* to all neighbors. You can obtain the *outgoing interfaces* with neighbors from the *network* by *router_ID* (Current router, not *router_ID* in *interest*).
- **networks**

= *{"r0":[1, 3], "r1": [0, 2, 3], "r2": [1, 4], ... , "r11": [7, 10]}*

- *outfaces*

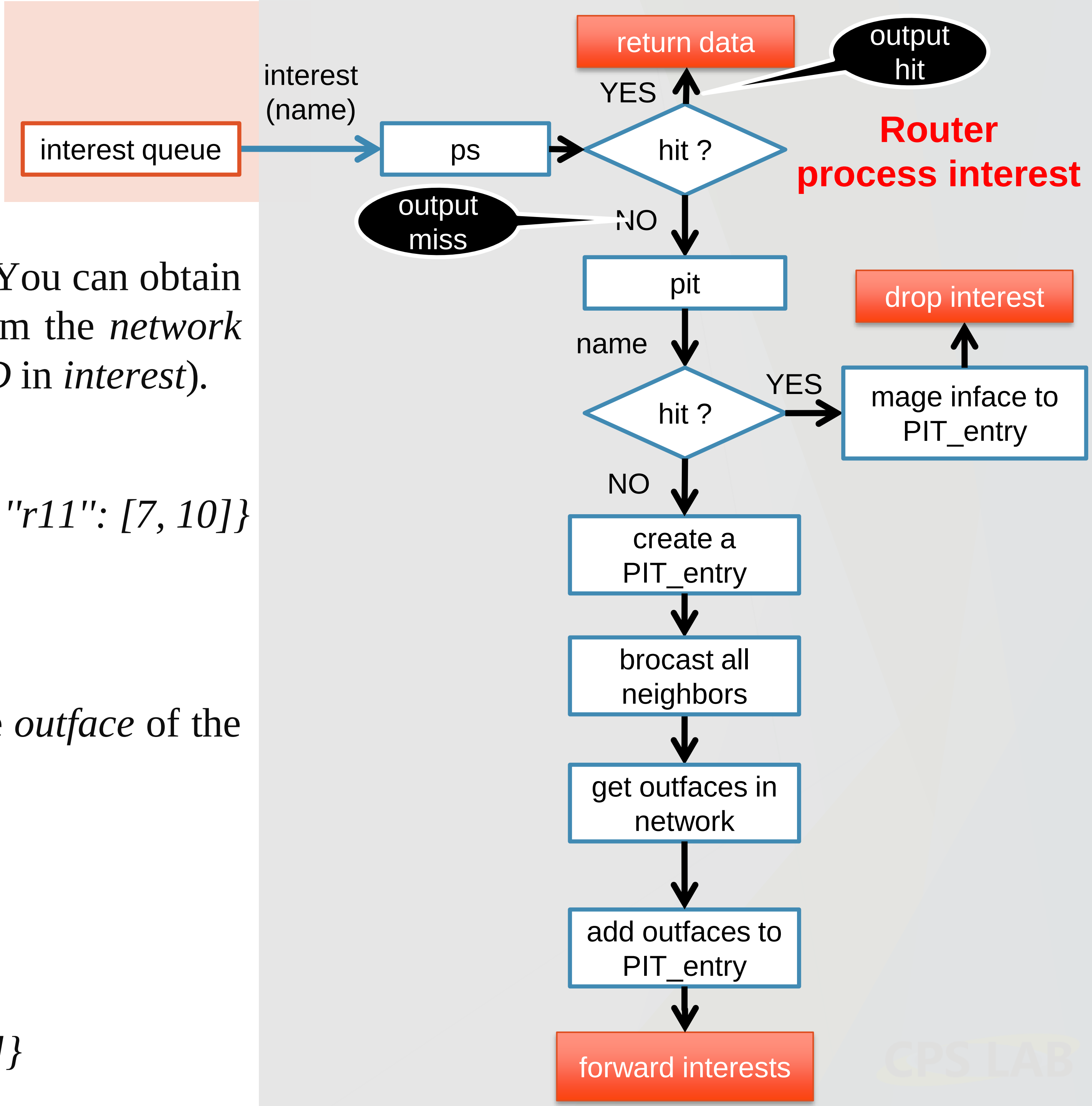
= *[0, 2, 3]*

- And record these *outgoing interfaces* in the *outface* of the matching *entry* in *pit*.
- ***PIT_entry***

= *[[inface], [outface, ...]]*

- ***pit***

= *{... , 'content_name': [[inface], [outface, ...]]}*



9. Process Flow

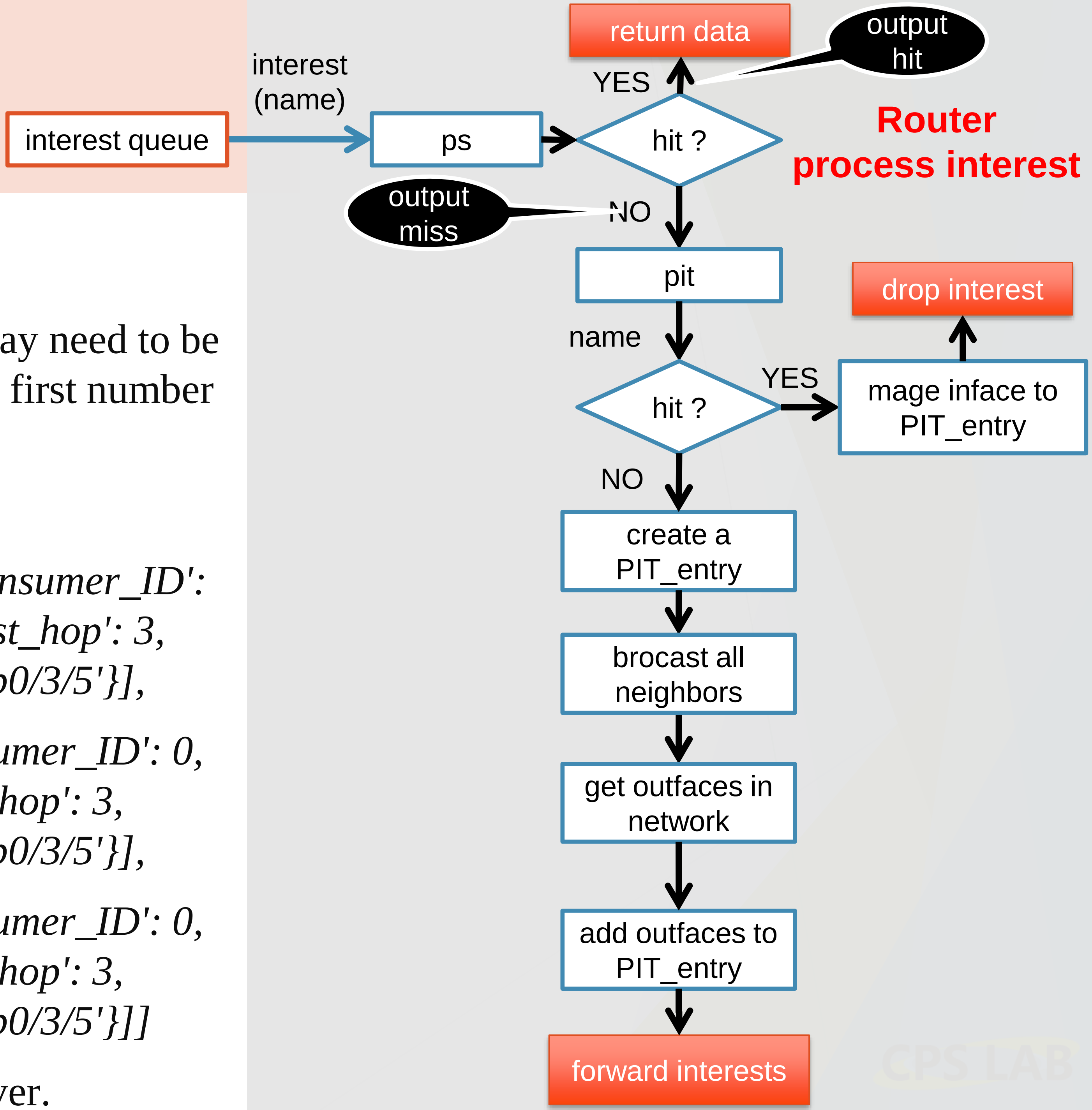
- ## Interest process
- *interest* miss in *ps*
- Create forwarding *interest packets*, which may need to be forwarded to multiple *output interfaces*. The first number of each list is the *outface* (router ID).
- interest packets format

*=[[**7**, {'type': 'interest', 'interest_ID': '0001', 'consumer_ID': 0, 'route_ID': 5, 'content_name': 'r7/07', 'interest_hop': 3, 'life_hop': 5, 'start_time': 1615357599, 'path': 'p0/3/5'}],*

*[**4**, {'type': 'interest', 'interest_ID': '0001', 'consumer_ID': 0, 'route_ID': 5, 'content_name': 'r7/07', 'interest_hop': 3, 'life_hop': 5, 'start_time': 1615357599, 'path': 'p0/3/5'}],*

*[**8**, {'type': 'interest', 'interest_ID': '0001', 'consumer_ID': 0, 'route_ID': 5, 'content_name': 'r7/07', 'interest_hop': 3, 'life_hop': 5, 'start_time': 1615357599, 'path': 'p0/3/5'}]]*

- The *interest packets* is forwarded by the server.



Router
process interest

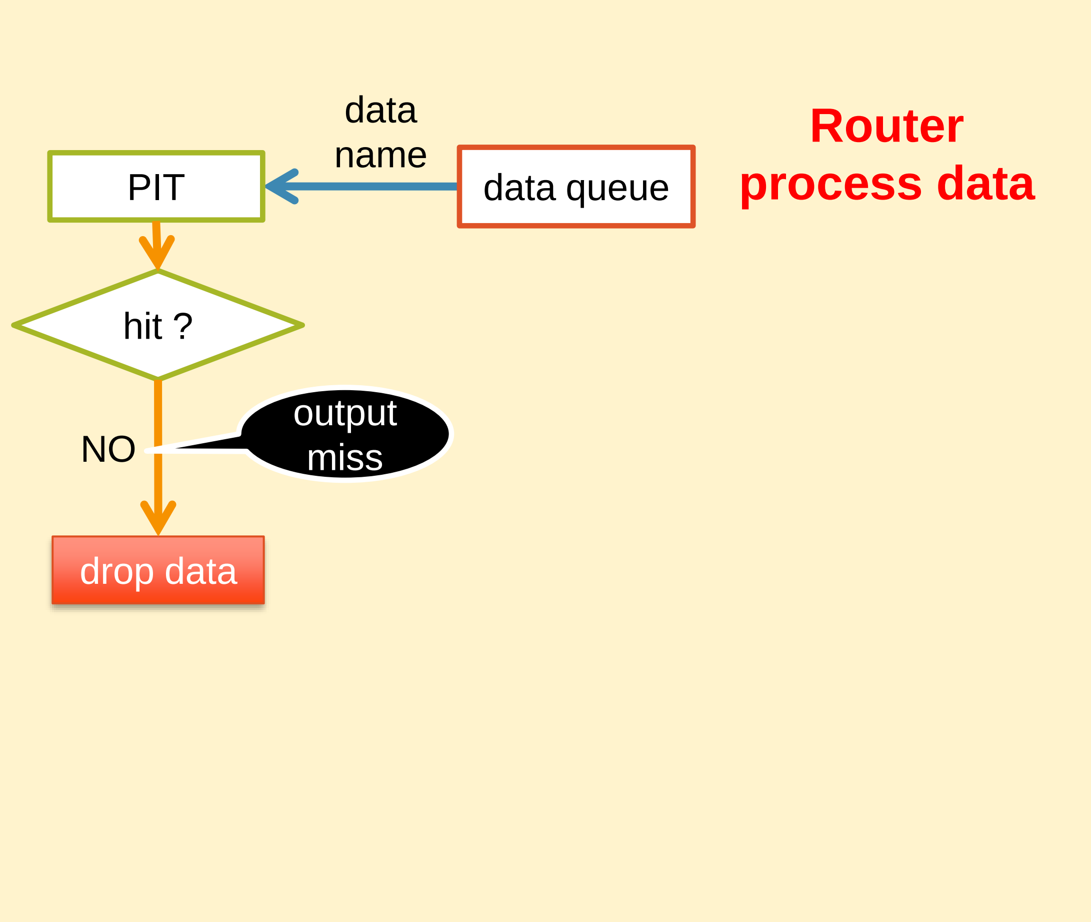
9. Process Flow

- ## Data process

- The router takes the first *packet* from the *data_queue* for processing each time.
- The router gets *name* from packet. Go to *pit* to find if there is matching *name*.
- ***pit*** format

= {'*content_name*': [[*inface*, ...], [*outface*, ...]], ... }

- If it miss, *data* is droped. The router takes a *interest packet* from the *interest_queue* for processing.
- Output '*Data miss in PIT*' to CSV.



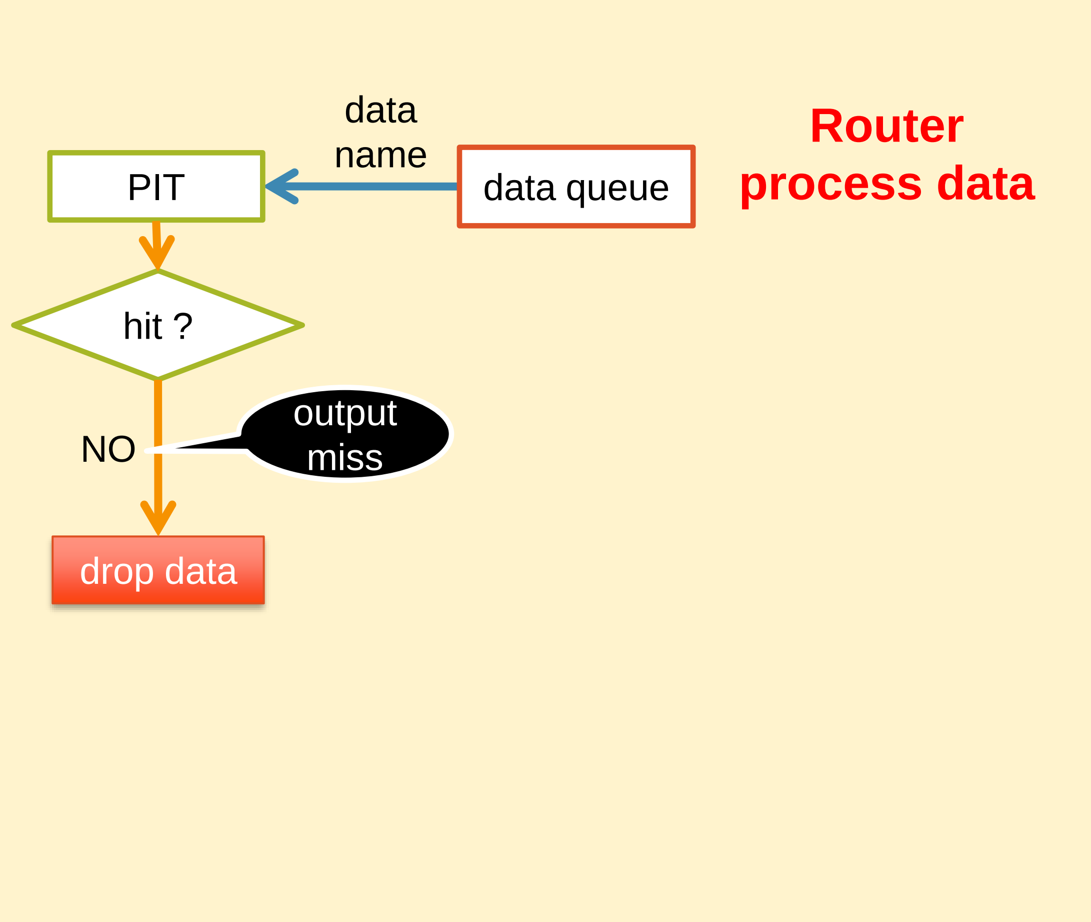
9. Process Flow

- **Data process**
- Output *'Data miss in PIT'* to CSV.
- Data Output.CSV format

=[

"Time"=str(times-data['run_start_time']),
"Type"=data['type'],
"Consumer_ID"='C'+str(data['consumer_ID']),
"Route_ID"='R'+str(data['route_ID']),
"Content_name"=data['content_name'],
"Data_hop"=data['data_hop'],
"Path"=data['path'],
"Result"='Data miss in PIT',
"Hit_consumer"=0,
"Hit_PIT"=0,
"Hit_Miss"=1

]



CPS LAB

9. Process Flow

Data process

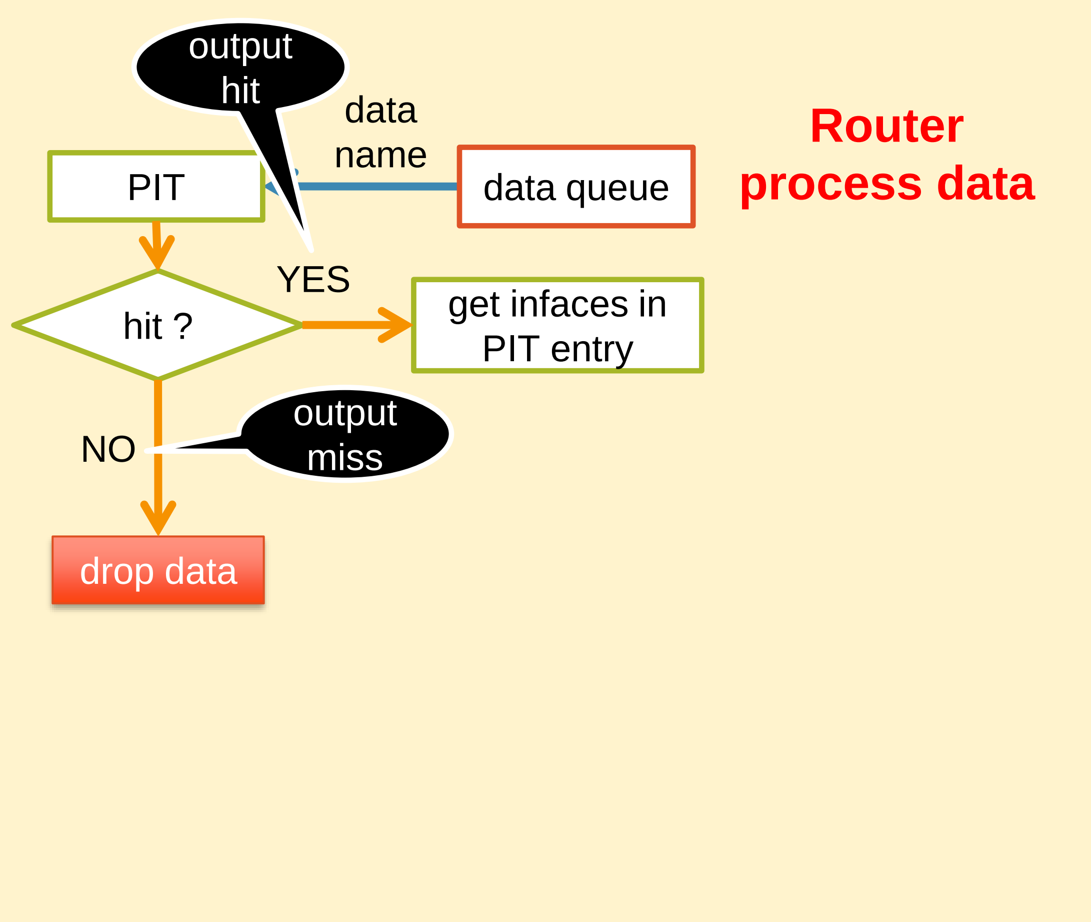
- Otherwise, it means that this is the first *data* returned for this *name*. Get the *incoming interfaces* from the *PIT_entry*.
- ***pit*** format

= {*'content_name': [[inface, ...], [outface, ...]], ... }*

- ***PIT_entry*** format

= *[[inface, ...], [outface, ...]]*

- Output '*Data hit in PIT*' to CSV. The format following above.



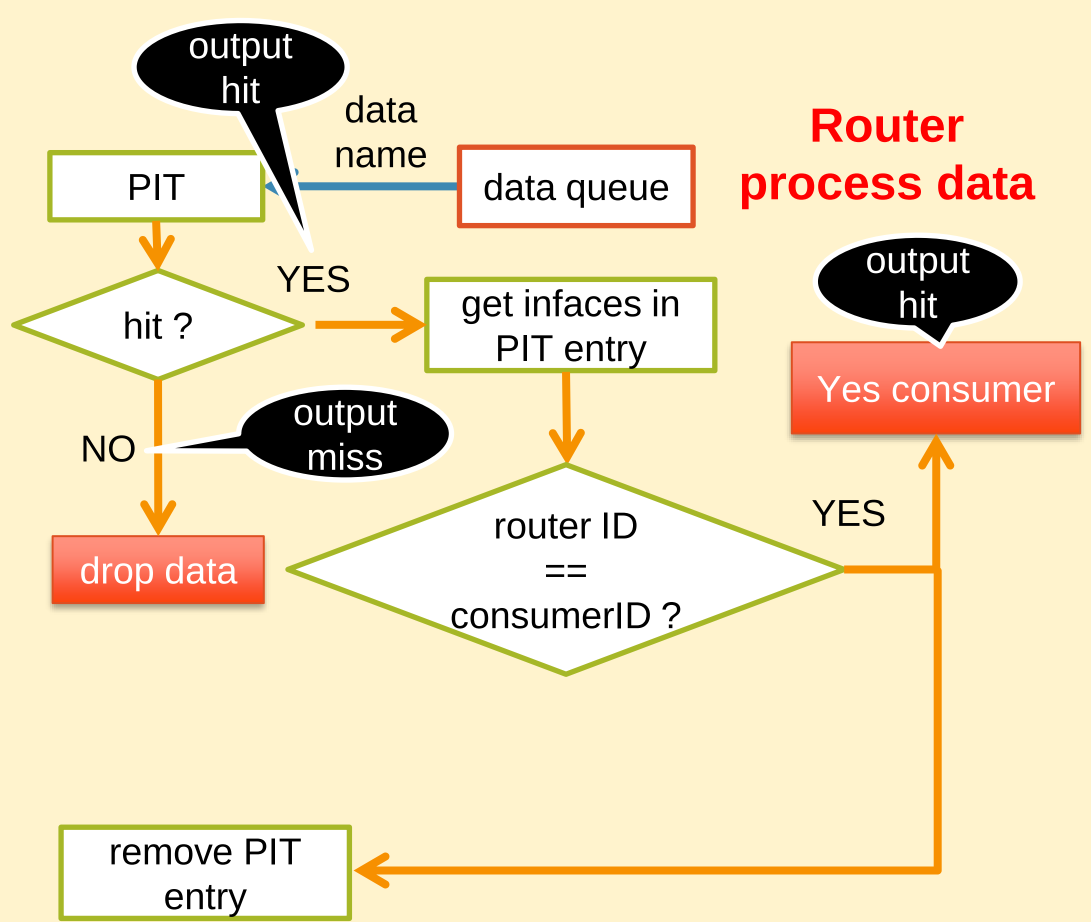
9. Process Flow

- ## Data process

- Determine whether the current *router_ID* is equal to the *consumer_ID* in the *data packet*. If yes, it means that the consumer has received a *data* response. Output *'Data hit in consumer'* to CSV.

- Data Output.CSV format

=["Time"=str(times-data['run_start_time']),
"Type"=data['type'],
"Consumer_ID"='C'+str(data['consumer_ID']),
"Route_ID"='R'+str(data['route_ID']),
"Content_name"=data['content_name'],
"Data_hop"=data['data_hop'],
"Path"=data['path'],
"Result"='Data hit in consumer',
"Hit_consumer"=1,
"Hit_PIT"=1,
"Hit_Miss"=0]

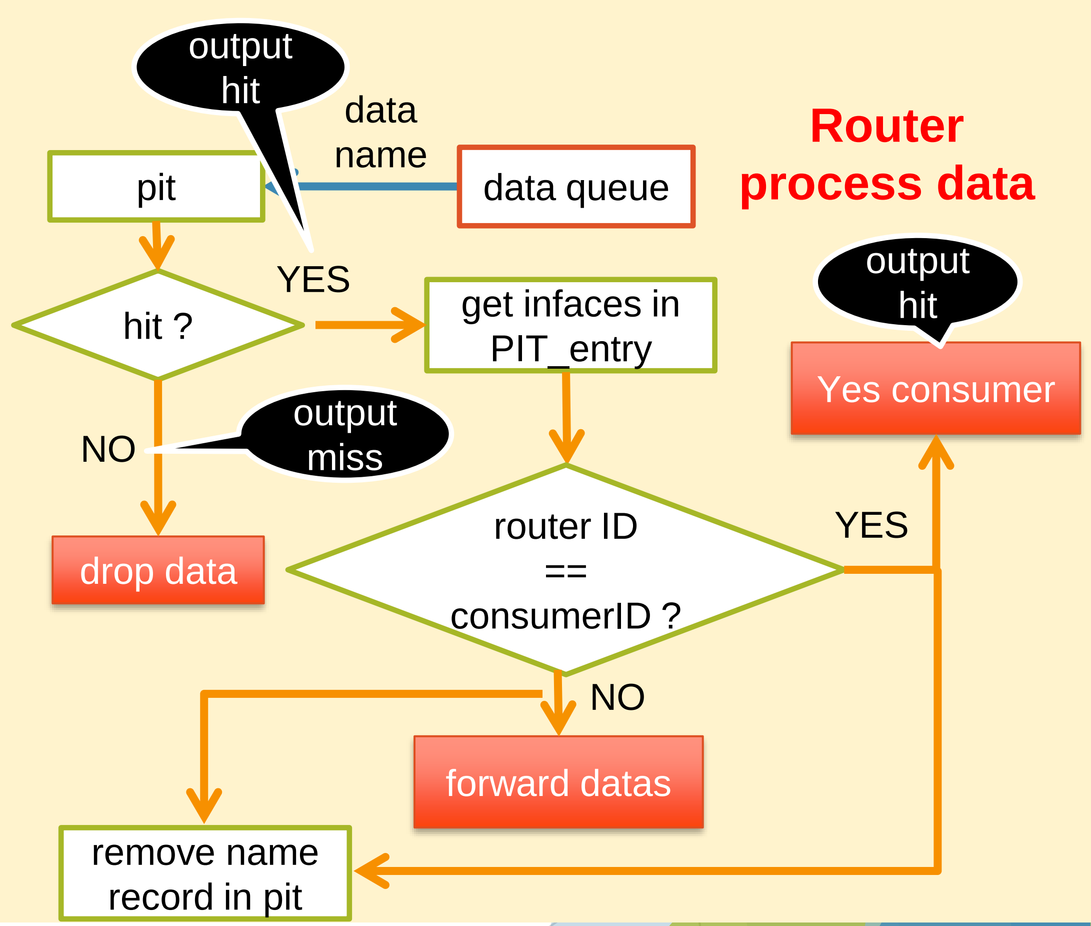


CPS LAB

9. Process Flow

- ## Data process
- Remove this *content_name* record from *pit*.
- Otherwise, forward the *data packet* to all routers on the *incoming interface* and remove this *content_name* record from *pit*.
- ***pit*** format

= {'*content_name*': [[*inface*, ...], [*outface*, ...]], ... }



9. Process Flow

- ## Data process

- *data* hit in *pit*

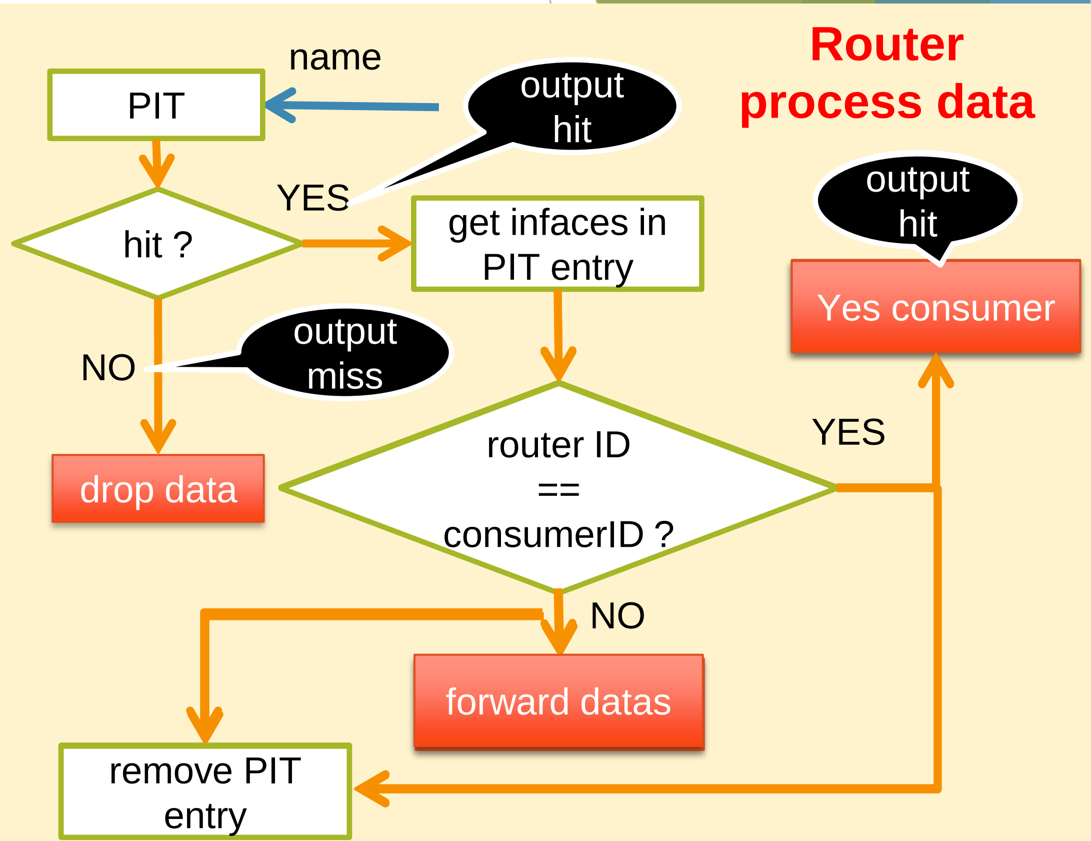
- Create forwarding *data packets*, which may need to be forwarded to multiple *incoming interfaces*. The first number of each list is the *incoming interface* (next *router ID*).

- *data packets* format

= [[2, {'type': 'data', 'consumer_ID': 0, 'route_ID': 4, 'content_name': 'r5/09', 'content_data': '', 'data_hop': 2, 'start_time': 1615357615, 'path': 'p5/4/'}],

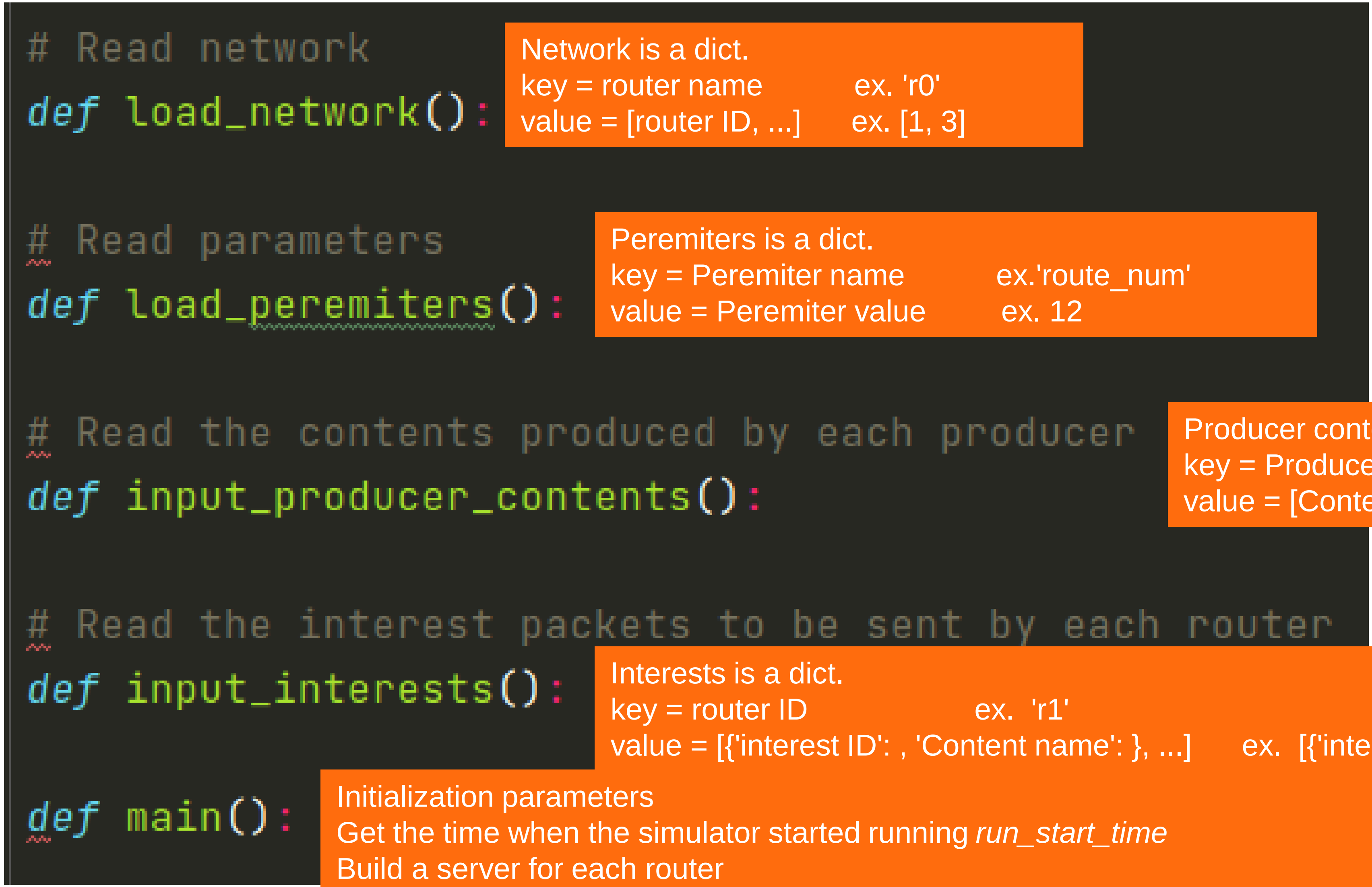
[6, {'type': 'data', 'consumer_ID': 0, 'route_ID': 4, 'content_name': 'r5/09', 'content_data': '', 'data_hop': 2, 'start_time': 1615357615, 'path': 'p5/4/'}]]

- The *data packets* is forwarded by the server.



Router process data

10. Coding----main.py



```
# Read network
def load_network():

# Read parameters
def load_peremiters():

# Read the contents produced by each producer
def input_producer_contents():

# Read the interest packets to be sent by each router
def input_interests():

def main():
```

Network is a dict.
key = router name   ex. 'r0'
value = [router ID, ...]   ex. [1, 3]

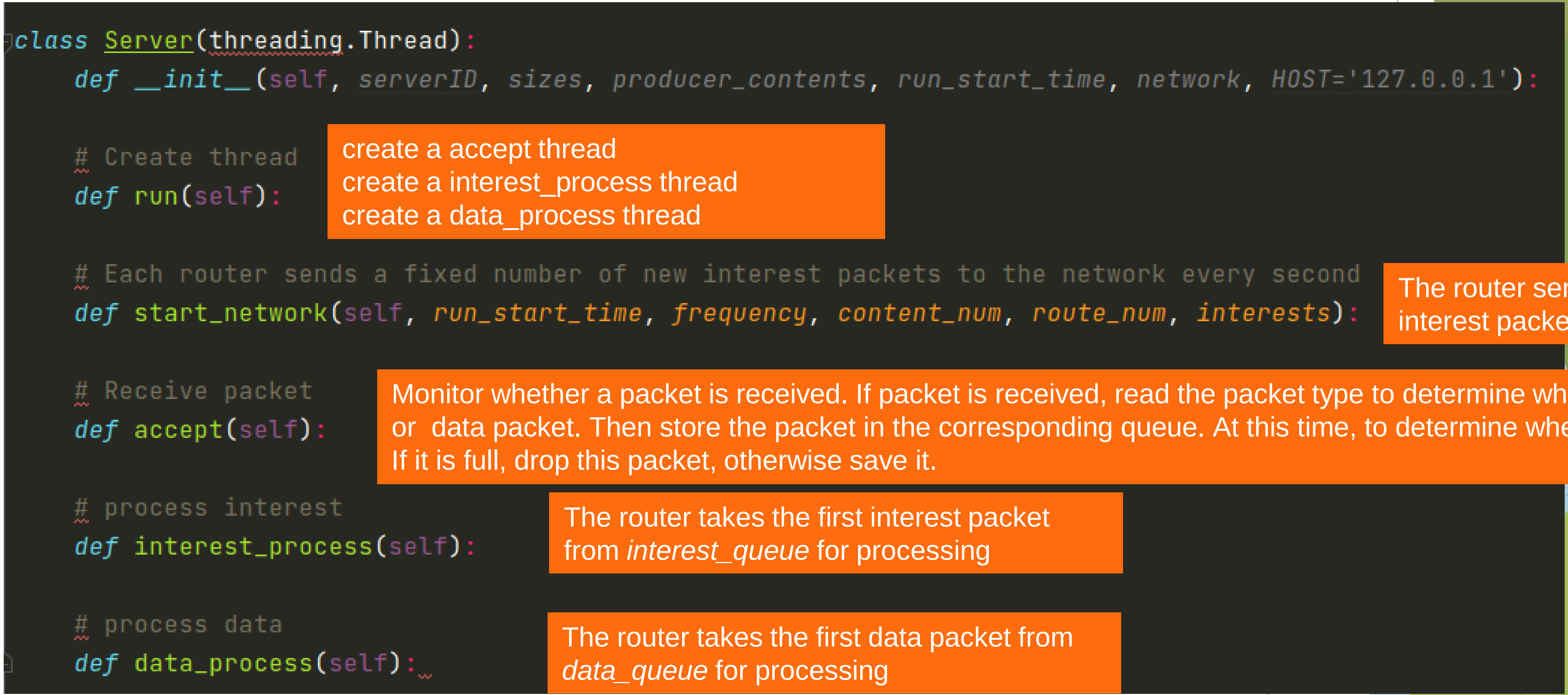
Peremiters is a dict.
key = Peremiter name   ex.'route_num'
value = Peremiter value   ex. 12

Producer contents is a dict.
key = Producer ID   ex. 'p1'
value = [Content name, ...]   ex. ['r1/0', 'r1/99']

Interests is a dict.
key = router ID   ex. 'r1'
value = [{'interest ID': , 'Content name': }, ...]   ex. [{'interest ID': 0001, 'Content name': 'r1/99'}]

Initialization parameters
Get the time when the simulator started running *run_start_time*
Build a server for each router
At regular intervals（frequency） in the while loop, all routers send out new interest packets to the network. At the same time, it keeps checking whether the simulator end time is up.

10. Coding----server.py



```
class Server(threading.Thread):
    def __init__(self, serverID, sizes, producer_contents, run_start_time, network, HOST='127.0.0.1'):

    # Create thread
    def run(self):

    # Each router sends a fixed number of new interest packets to the network every second
    def start_network(self, run_start_time, frequency, content_num, route_num, interests):

    # Receive packet
    def accept(self):

    # process interest
    def interest_process(self):

    # process data
    def data_process(self):
```

create a accept thread
create a interest_process thread
create a data_process thread

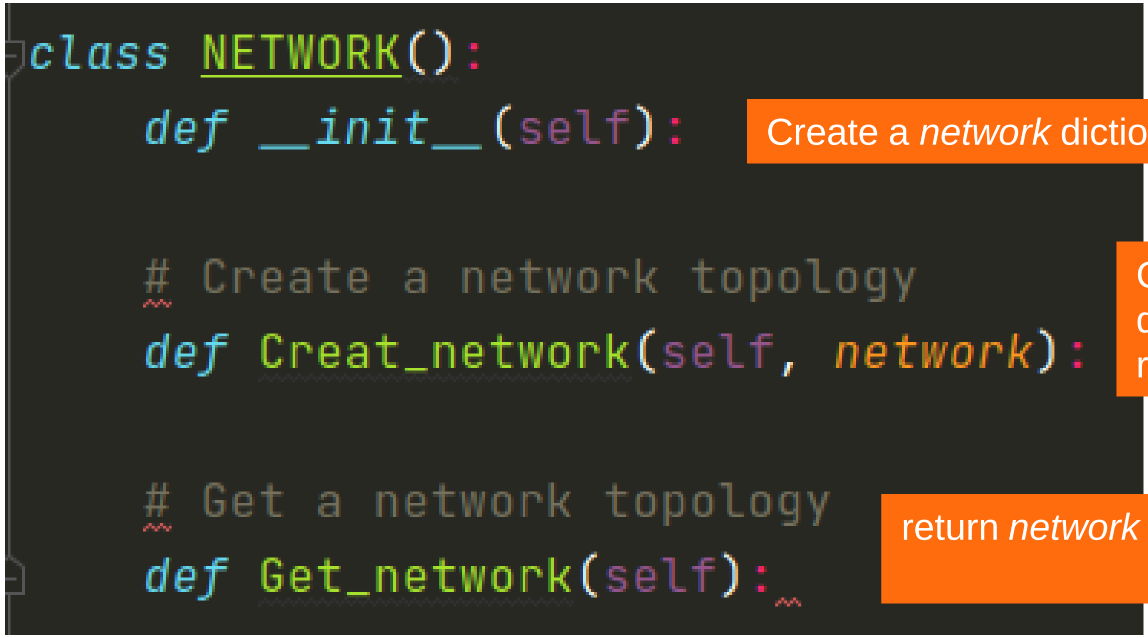
The router sends a new interest packet to the network.

Monitor whether a packet is received. If packet is received, read the packet type to determine whether it is interest or data packet. Then store the packet in the corresponding queue. At this time, to determine whether queue is full. If it is full, drop this packet, otherwise save it.

The router takes the first interest packet from *interest_queue* for processing

The router takes the first data packet from *data_queue* for processing

10. Coding----network.py



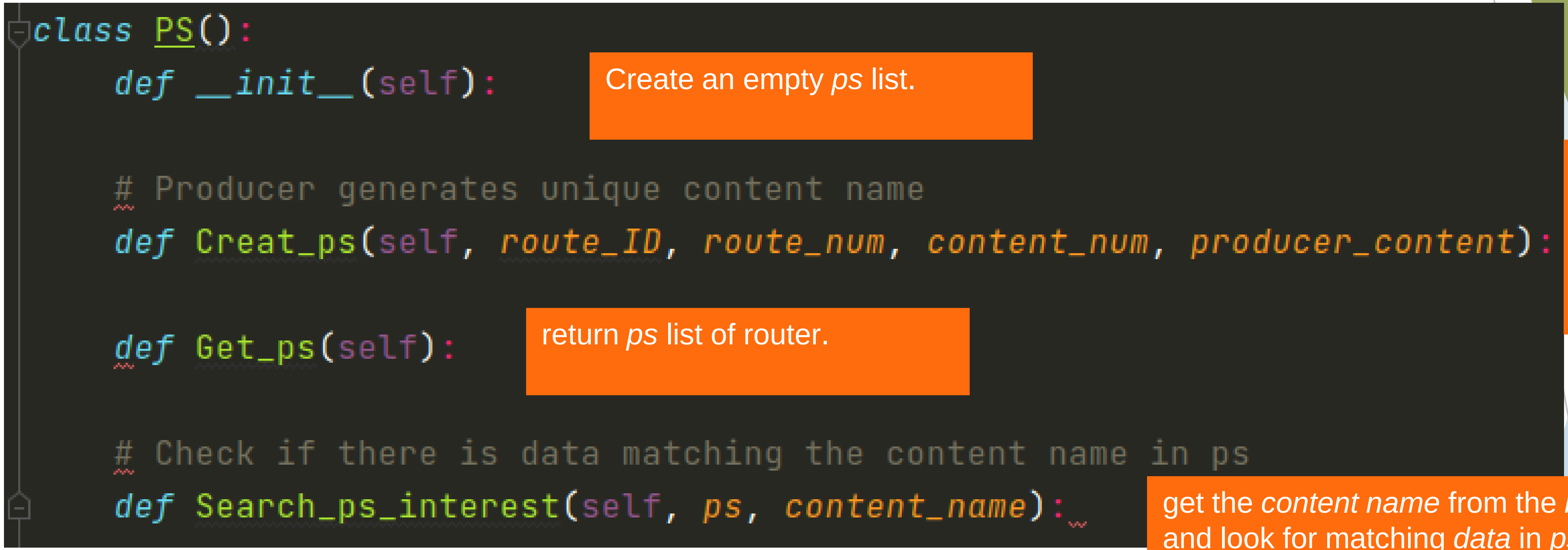
```python
class NETWORK():
    def __init__(self):

    # Create a network topology
    def Creat_network(self, network):

    # Get a network topology
    def Get_network(self):
```

Create a *network* dictionary.

Create a *network* dictionary, assign the *network* dictionary read from the networks JSON file to it, and return this dictionary.

return *network* dictionary.

10. Coding----ps.py



```
class PS():
    def __init__(self):

    # Producer generates unique content name
    def Creat_ps(self, route_ID, route_num, content_num, producer_content):

    def Get_ps(self):

    # Check if there is data matching the content name in ps
    def Search_ps_interest(self, ps, content_name):
```

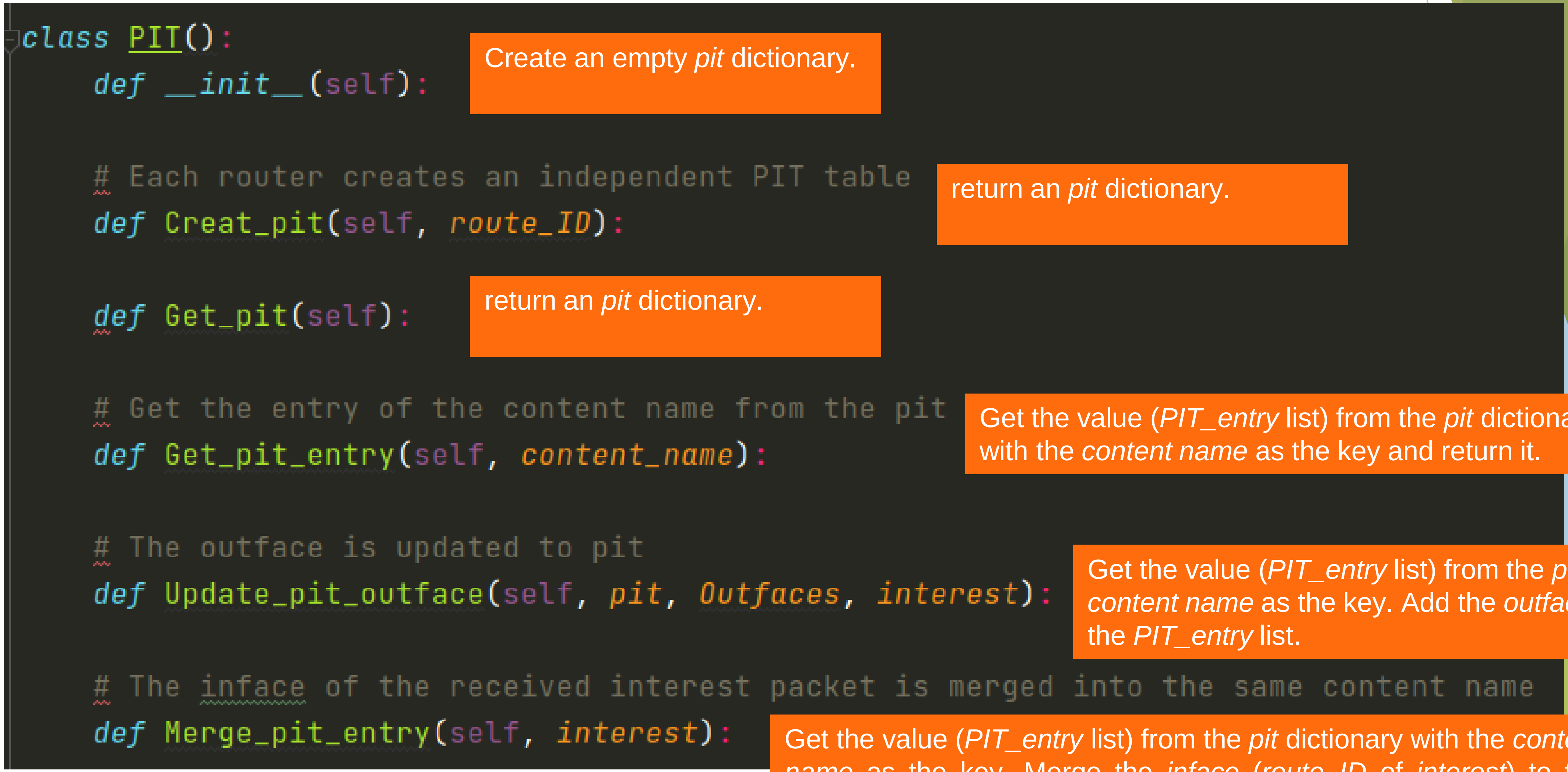
Create an empty *ps* list.

Store the producer contents of the current router read from the producer content JSON file into *ps* list of router.

return *ps* list of router.

get the *content name* from the *interest* packet, and look for matching *data* in *ps*. If it matches, return TRUE, otherwise return FALSE.

10. Coding----pit.py



```
class PIT():
    def __init__(self):

    # Each router creates an independent PIT table
    def Creat_pit(self, route_ID):

    def Get_pit(self):

    # Get the entry of the content name from the pit
    def Get_pit_entry(self, content_name):

    # The outface is updated to pit
    def Update_pit_outface(self, pit, Outfaces, interest):

    # The inface of the received interest packet is merged into the same content name
    def Merge_pit_entry(self, interest):
```

Create an empty *pit* dictionary.

return an *pit* dictionary.

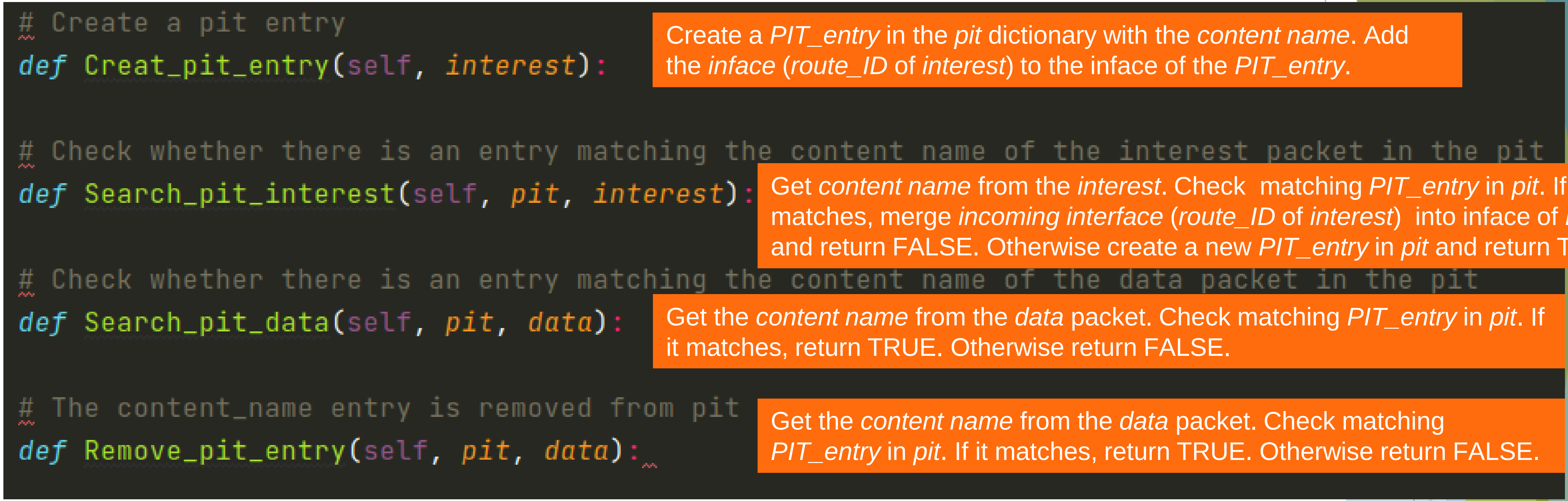
return an *pit* dictionary.

Get the value (*PIT_entry* list) from the *pit* dictionary with the *content name* as the key and return it.

Get the value (*PIT_entry* list) from the *pit* dictionary with the *content name* as the key. Add the *outfaces* to the outface of the *PIT_entry* list.

Get the value (*PIT_entry* list) from the *pit* dictionary with the *content name* as the key. Merge the *inface* (*route_ID* of *interest*) to the inface of the *PIT_entry* list. At the same time, make sure that the *inface* is not repeated in inface of the *PIT_entry* list.

10. Coding----pit.py



```
# Create a pit entry
def Creat_pit_entry(self, interest):
```

Create a *PIT_entry* in the *pit* dictionary with the *content name*. Add the *inface* (*route_ID* of *interest*) to the inface of the *PIT_entry*.

```
# Check whether there is an entry matching the content name of the interest packet in the pit
def Search_pit_interest(self, pit, interest):
```

Get *content name* from the *interest*. Check matching *PIT_entry* in *pit*. If it matches, merge *incoming interface* (*route_ID* of *interest*) into inface of *PIT_entry* and return FALSE. Otherwise create a new *PIT_entry* in *pit* and return TRUE.

```
# Check whether there is an entry matching the content name of the data packet in the pit
def Search_pit_data(self, pit, data):
```

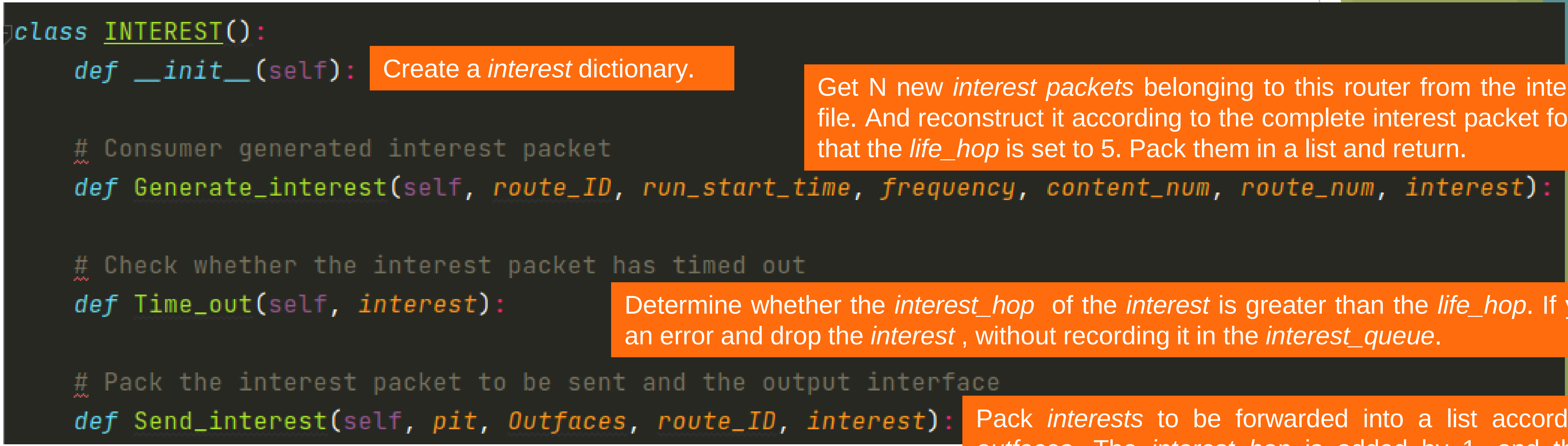
Get the *content name* from the *data* packet. Check matching *PIT_entry* in *pit*. If it matches, return TRUE. Otherwise return FALSE.

```
# The content_name entry is removed from pit
def Remove_pit_entry(self, pit, data):
```

Get the *content name* from the *data* packet. Check matching *PIT_entry* in *pit*. If it matches, return TRUE. Otherwise return FALSE.

CPS LAB

10. Coding----interest.py



```
class INTEREST():
    def __init__(self):

    # Consumer generated interest packet
    def Generate_interest(self, route_ID, run_start_time, frequency, content_num, route_num, interest):

    # Check whether the interest packet has timed out
    def Time_out(self, interest):

    # Pack the interest packet to be sent and the output interface
    def Send_interest(self, pit, Outfaces, route_ID, interest):
```

Create a *interest* dictionary.

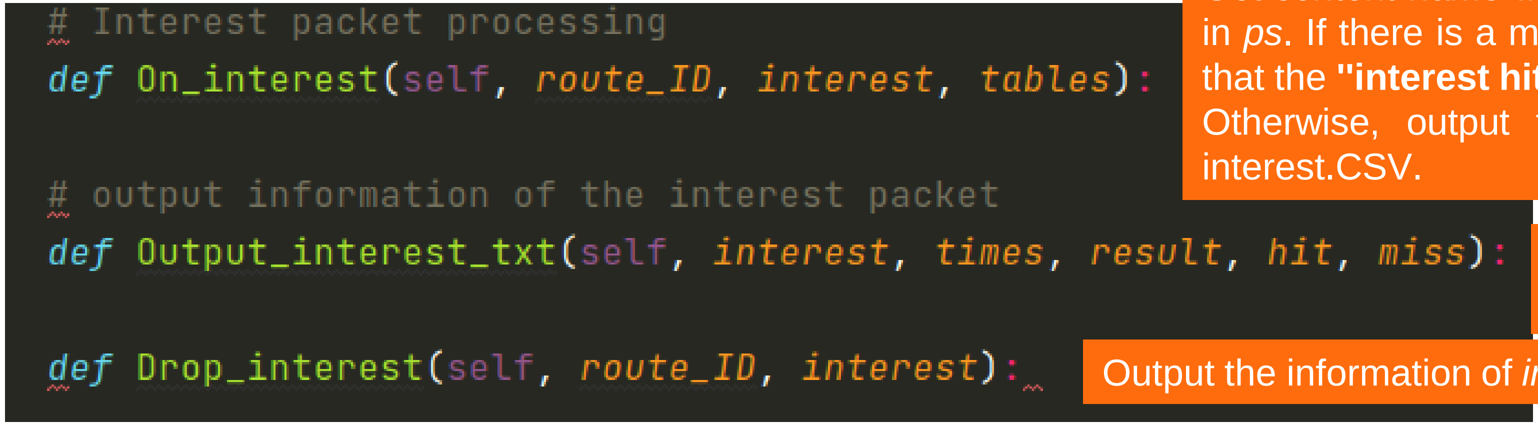
Get N new *interest packets* belonging to this router from the interest JSON file. And reconstruct it according to the complete interest packet format. Note that the *life_hop* is set to 5. Pack them in a list and return.

Determine whether the *interest_hop* of the *interest* is greater than the *life_hop*. If yes, return an error and drop the *interest* , without recording it in the *interest_queue*.

Pack *interests* to be forwarded into a list according to the *outfaces*. The *interest_hop* is added by 1, and the *path* of interest is added to the current *router ID*. At the same time, the *outfaces* is updated to the *PIT_entry* matching *content_name*.

CPS LAB

10. Coding----interest.py



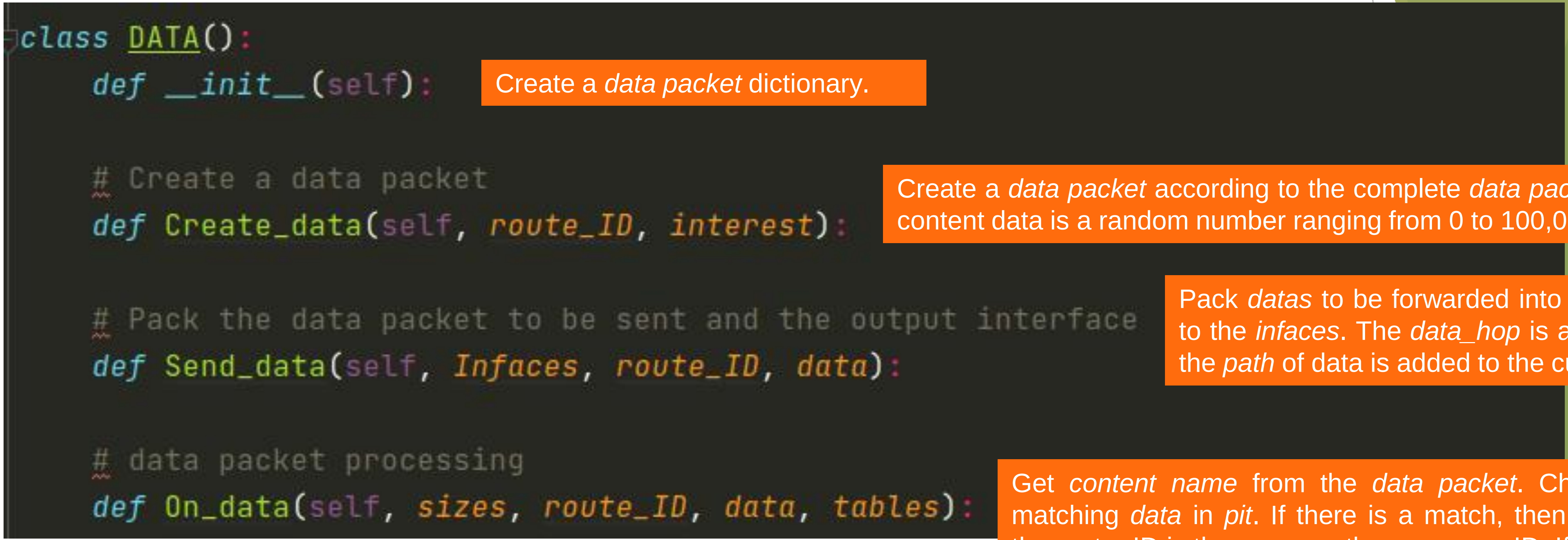
```
# Interest packet processing
def On_interest(self, route_ID, interest, tables):

# output information of the interest packet
def Output_interest_txt(self, interest, times, result, hit, miss):

def Drop_interest(self, route_ID, interest):
```

Get *content name* from the *interest* packet. Check if there is matching *data* in *ps*. If there is a match, create a *data packet*. Then output the information that the **"interest hits in ps"** to CSV and return the data packet. Otherwise, output the information that the **“interest miss in ps”** to interest.CSV.

Output *interest packet* information and write to interest.CSV

Output the information of *interest packet* and write it to interest.CSV

10. Coding----data.py



```
class DATA():
    def __init__(self):

    # Create a data packet
    def Create_data(self, route_ID, interest):

    # Pack the data packet to be sent and the output interface
    def Send_data(self, Infaces, route_ID, data):

    # data packet processing
    def On_data(self, sizes, route_ID, data, tables):
```

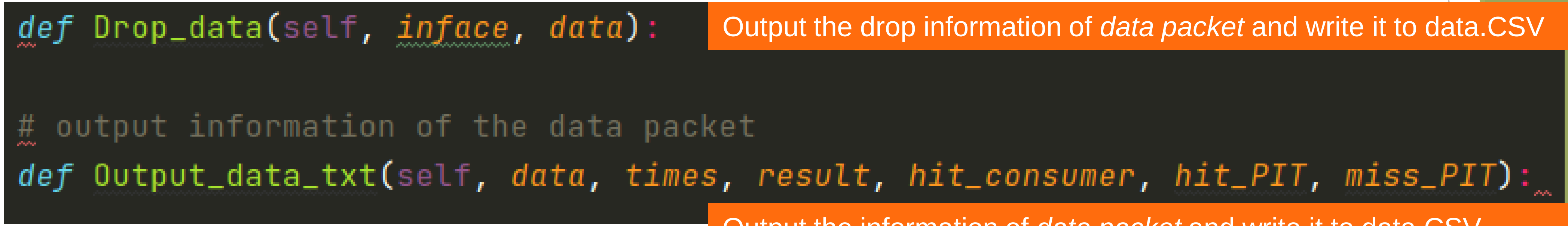
Create a *data packet* dictionary.

Create a *data packet* according to the complete *data packet* format. The content data is a random number ranging from 0 to 100,000.

Pack *datas* to be forwarded into a list according to the *infaces*. The *data_hop* is added by 1, and the *path* of data is added to the current *router ID*.

Get *content name* from the *data packet*. Check if there is matching *data* in *pit*. If there is a match, then check whether the router ID is the same as the consumer ID. If yes, output the information that the ***"data hit in consumer"*** to data.CSV. Otherwise, output the information that the **"data hit in pit"** to data.CSV and forward *data packet*.
If not match in pit, output the information that the **"data miss in pit"** to data.CSV.

10. Coding----data.py

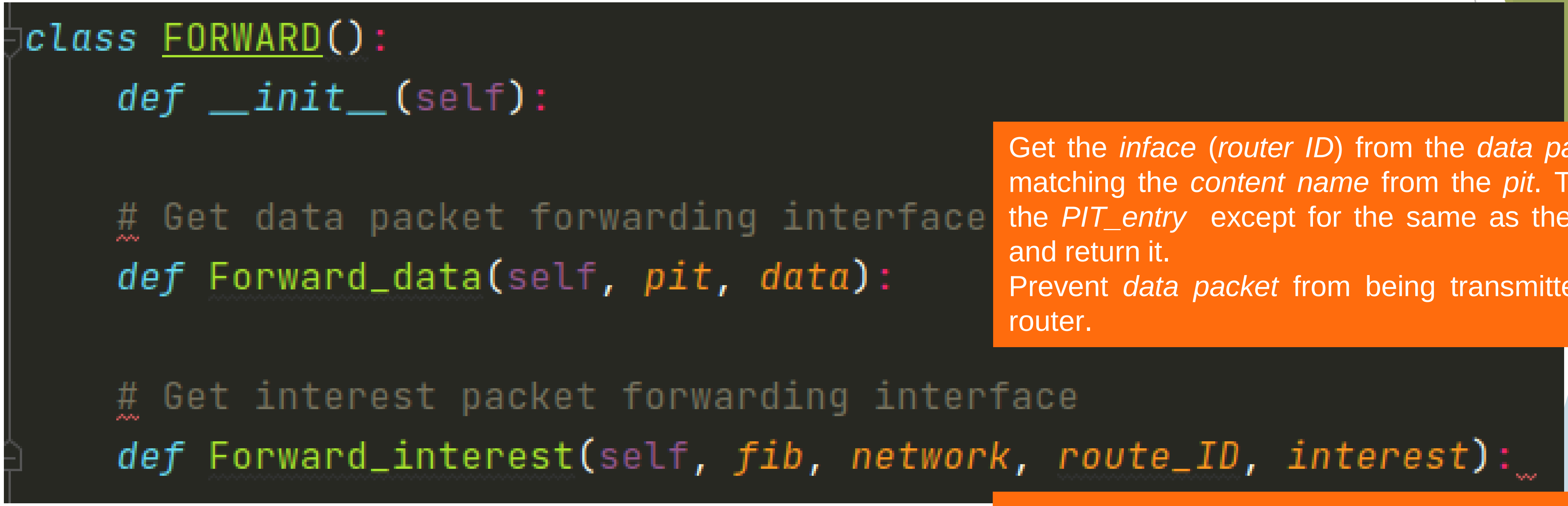


```python
def Drop_data(self, inface, data):

# output information of the data packet
def Output_data_txt(self, data, times, result, hit_consumer, hit_PIT, miss_PIT):
```

Output the drop information of *data packet* and write it to data.CSV

Output the information of *data packet* and write it to data.CSV

10. Coding----forward.py



```
class FORWARD():
    def __init__(self):

    # Get data packet forwarding interface
    def Forward_data(self, pit, data):

    # Get interest packet forwarding interface
    def Forward_interest(self, fib, network, route_ID, interest):
```

Get the *inface* (*router ID*) from the *data packet*. Get the *PIT_entry* matching the *content name* from the *pit*. Then pack the infaces of the *PIT_entry* except for the same as the *inface* into the *Infaces*, and return it.
Prevent *data packet* from being transmitted back to the previous router.

Get the *inface* (*router ID*) from the *inteseat packet*. Get the *FIB_entry* matching the *router ID* from the *network*. Then pack the outfaces of the *FIB_entry* except for the same as the *inface* into the *Outfaces*, and return it.
Prevent *interest packet* from being transmitted back to the previous router.

Thank You !

Q&A

Email： P780083025@ncku.edu.tw

CPS LAB