HOW TO RUN DEMO of ECMP-ER on Linux VM using BMv2

Extending ECMP toward A Practical Hardware Load Balancer

Kentaro Ebisawa

TOYOTA MOTOR CORPORATION

InfoTech, DC Infra Group

About

- This slide includes supplemental information about ECMP-ER demo using BMv2
- For the actual steps to run the demo, please refer to the instruction in the below document
- https://github.com/ToyotaInfoTech/ecmper/blob/main/HOWTO-ECMPER-DEMO.md

- ECMP-ER paper was published at IOTS 2022 on 9th Dec, 2022
  - IOTS 2022: IPSJ SIG IOT (Internet and Operation Technology) Symposium 2022
  - Program (Japanese): https://www.iot.ipsj.or.jp/symposium/iots2022-program/

CAUTION

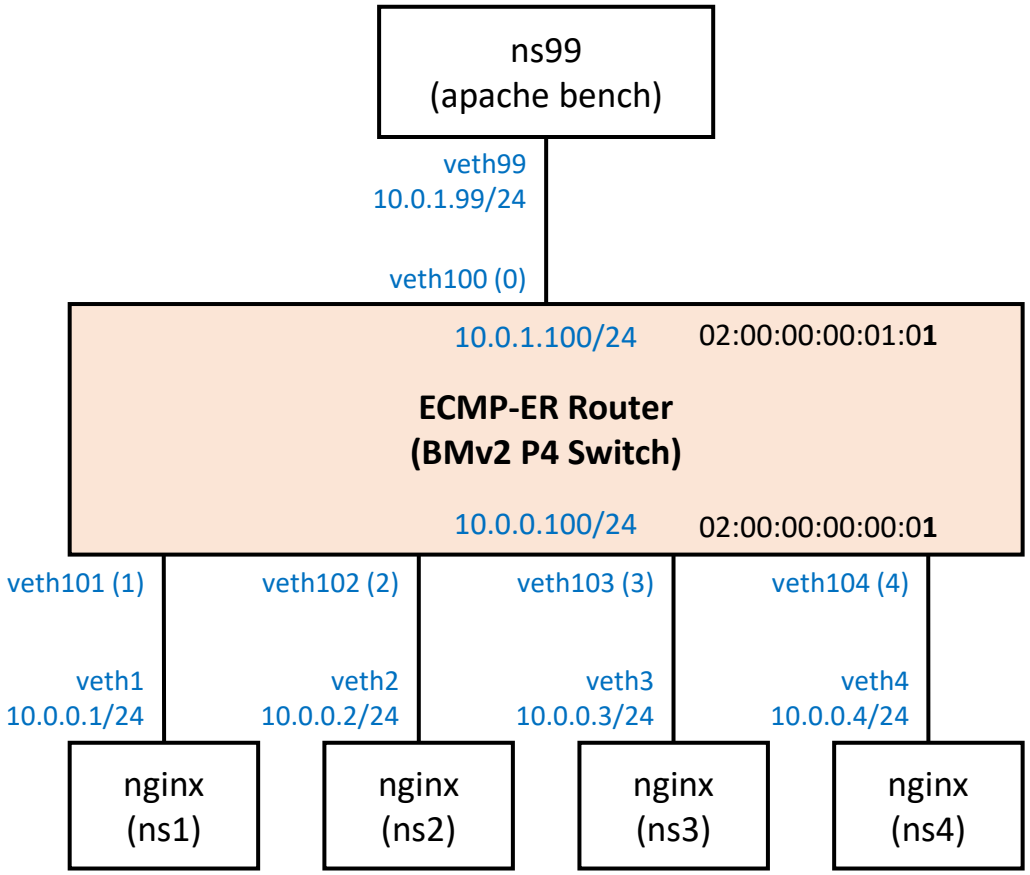
This is a demo implementation of ECMP-ER and should NOT be used in production.

There are room for improvements you should consider before using it in production.

e.g. controller P4 table update logic, XDP code on server, etc.

Table of Contents

Demo Topology

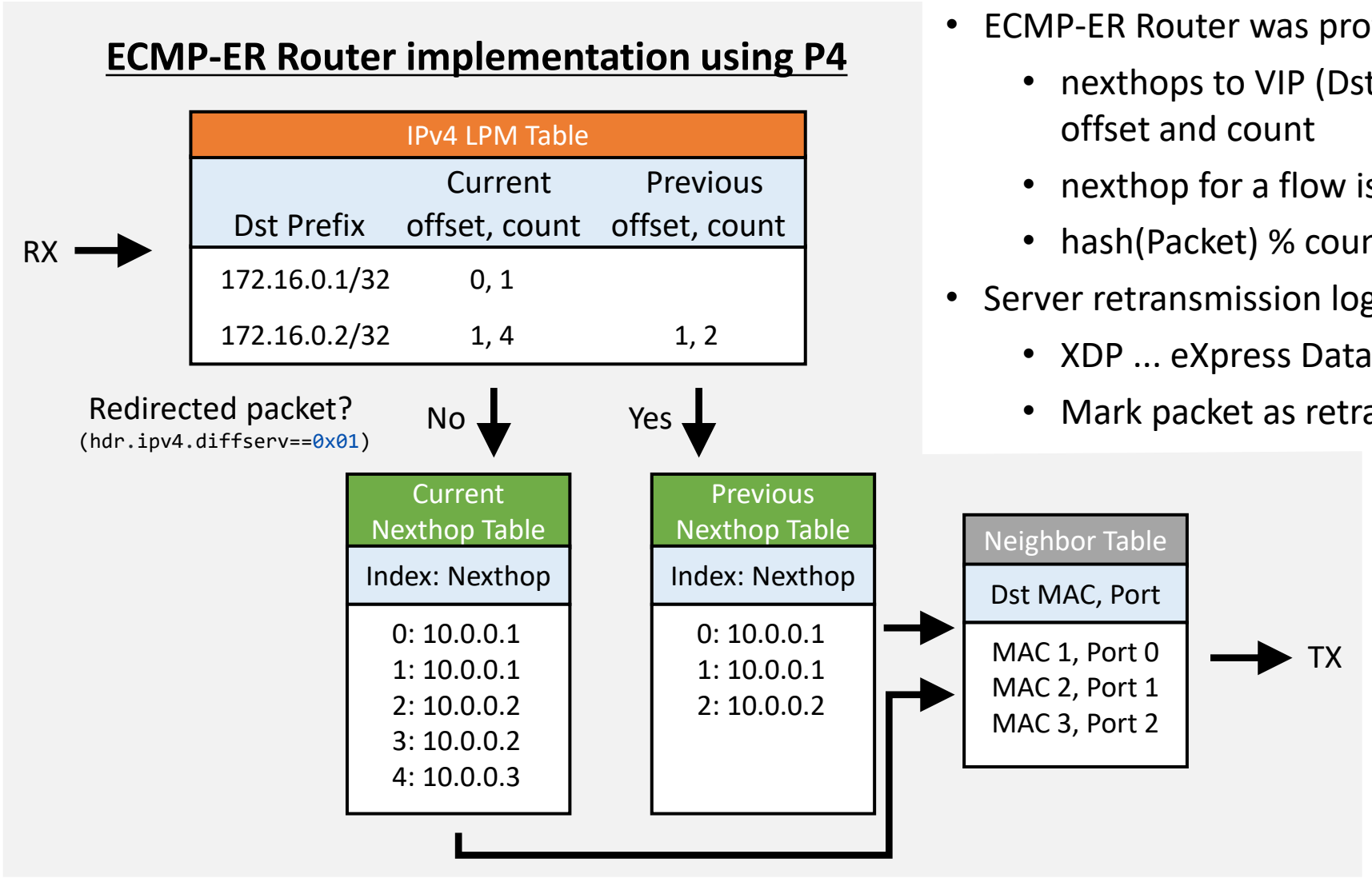


10.0.10.0/32 (VIP)
nginx listening to TCP port 80

mac addr of vethX : 02:03:04:05:06:0**X**

P4 Tables and Pipeline

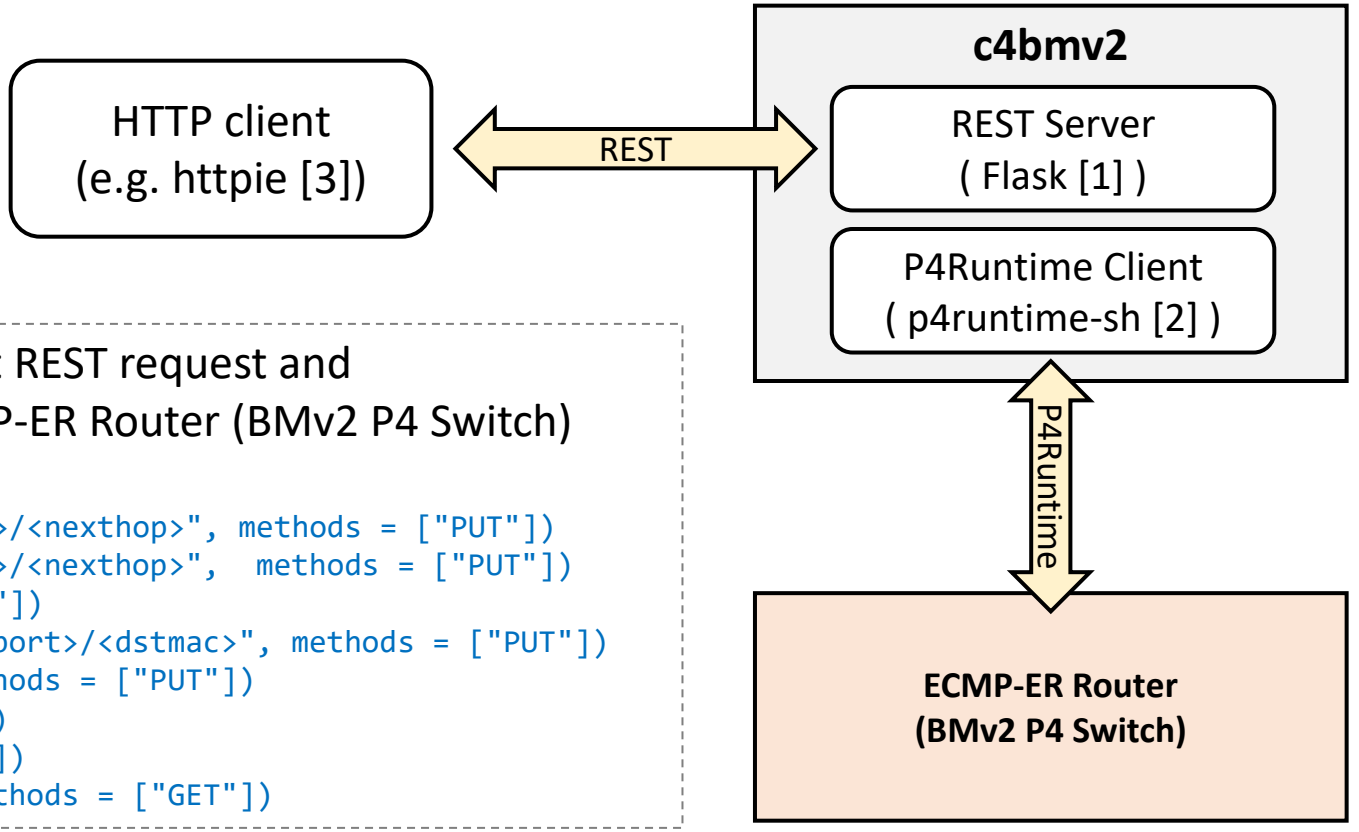
https://github.com/ToyotaInfoTech/ecmper/blob/main/p4src/ecmper.p4



- ECMP-ER Router was prototyped using P4
  - nexthops to VIP (Dst Prefix) are stored in a table using offset and count
  - nexthop for a flow is decided with below formula
  - hash(Packet) % count + offset
- Server retransmission logic was prototyped using XDP
  - XDP ... eXpress Data Path
  - Mark packet as retransmitted using ToS field

"controller 4 bmv2" a.k.a c4bmv2 design

c4bmv2.py [0] ... a controller made of "REST server + P4Runtime Client" written in python3



c4bmv2 will accept REST request and
configure (get info) from ECMP-ER Router (BMv2 P4 Switch)

```
@app.route("/add/<prefix>/<int:preflen>/<nexthop>", methods = ["PUT"])
@app.route("/del/<prefix>/<int:preflen>/<nexthop>",  methods = ["PUT"])
@app.route("/install", methods = ["PUT"])
@app.route("/neigh/add/<nh_addr>/<int:port>/<dstmac>", methods = ["PUT"])
@app.route("/neigh/del/<nh_addr>", methods = ["PUT"])
@app.route("/clear", methods = ["PUT"])
@app.route("/tables", methods = ["GET"])
@app.route("/tables/<path_string>", methods = ["GET"])
```

[0] https://github.com/ToyotaInfoTech/ecmper/blob/main/tools/c4bmv2.py

[1] https://flask.palletsprojects.com/ [2] https://github.com/p4lang/p4runtime-shell [3] https://httpie.io/

Server side XDP client logic

Server retransmission logic operates autonomously

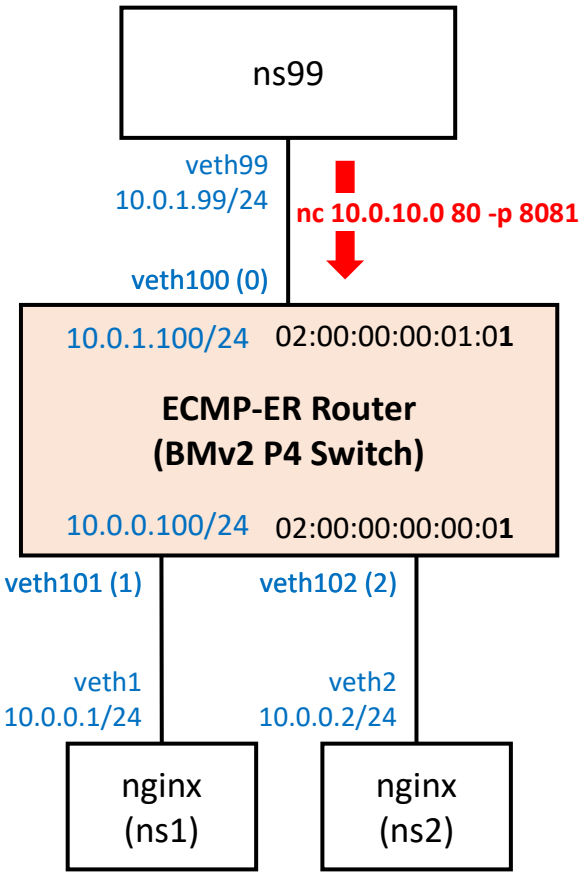
Control Plane is NOT required

1. If received packet is TCP and not a SYN packet (== part of existing Flow), and if the server does not own established socket for the packet, server sends the packet to ECMP-ER router
2. Else, it will receive packet

```
if packet is TCP and packet is not SYN then
    if find_socket(packet) is None then
        return retransmit(packet)
return receive(packet)
```

Source Code: https://github.com/ToyotaInfoTech/ecmper/tree/main/xdp

Demo 1: Packet flow when ECMP-ER is disabled (XDP detached)



10.0.10.0/32 (VIP)
nginx listening to TCP port 80

mac addr of vethX : 02:03:04:05:06:0**X**

1. 3 way hand shake: veth100(1,2,3) & veth101 (1,2,3)
(add server ... hash table change)
2. “GET /10B.txt HTTP/1.0” from ns99: veth100(4), veth102(1)
3. ns2 send back RST: veth102(2), veth100(5)

ecmper-veth100.trc

	Time	Source	Destination	Protocol	Length	Info
1	06:36:12.496096	10.0.1.99	10.0.10.0	TCP	74	8081 → 80 [SYN] Seq=0 Win=62244 Le
2	06:36:12.499155	10.0.10.0	10.0.1.99	TCP	74	80 → 8081 [SYN, ACK] Seq=0 Ack=1 W
3	06:36:12.499216	10.0.1.99	10.0.10.0	TCP	66	8081 → 80 [ACK] Seq=1 Ack=1 Win=62
4	06:36:25.804998	10.0.1.99	10.0.10.0	TCP	88	8081 → 80 [PSH, ACK] Seq=1 Ack=1 W
5	06:36:25.807533	10.0.10.0	10.0.1.99	TCP	54	80 → 8081 [RST] Seq=1 Win=0 Len=0

ecmper-veth101.trc

	Time	Source	Destination	Protocol	Length	Info
1	06:36:12.497978	10.0.1.99	10.0.10.0	TCP	74	8081 → 80 [SYN] Seq=0 Win=62244 Len=0 MSS=34
2	06:36:12.498344	10.0.10.0	10.0.1.99	TCP	74	80 → 8081 [SYN, ACK] Seq=0 Ack=1 Win=65474 L
3	06:36:12.499869	10.0.1.99	10.0.10.0	TCP	66	8081 → 80 [ACK] Seq=1 Ack=1 Win=62336 Len=0

ecmper-veth102.trc

	Time	Source	Destination	Protocol	Length	Info
1	06:36:25.806475	10.0.1.99	10.0.10.0	TCP	88	8081 → 80 [PSH, ACK] Seq=1 Ack=1 W
2	06:36:25.806778	10.0.10.0	10.0.1.99	TCP	54	80 → 8081 [RST] Seq=1 Win=0 Len=0

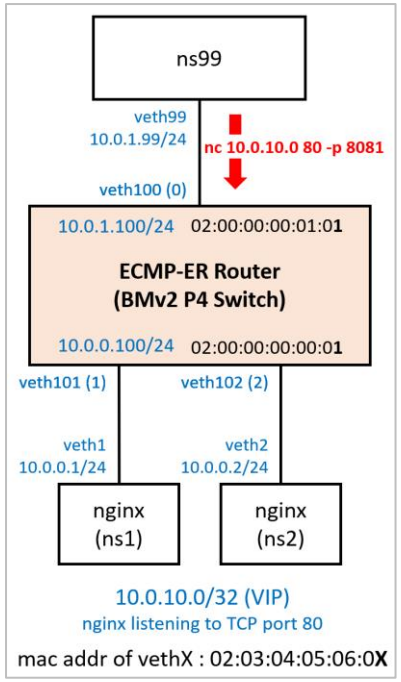
Demo 1: Packet flow when ECMP-ER is enabled (XDP attached)

1. 3 way hand shake: veth100(1,2,3) & veth101 (1,2,3)

(add server ... hash table change)

2. “GET /10B.txt HTTP/1.0” from ns99: veth100(4), veth102(1)
3. ns2 redirect packet to Router: veth102(2)
4. Router send packet to ns1 using prv_nh table: veth101(4) and ns1 ACK: veth101(5), veth100(5)
5. "¥n" from ns99: veth100(6), veth102(3)
6. ns2 redirect packet to Router: veth102(4)
7. Router send packet to ns1 using prv_nh table: veth101(6) and ns1 ACK: veth101(7), veth100(7)
8. ns1 send "HTTP/1.1 200 OK" to Router: veth101(8) which is forwarded to ns99: veth100(6)

... packet flow of ns99 -> ns2 -> Router -> ns1 -> ns99 continues until this session ends ...



ecmper-xdp-veth100.trc

	Time	Source	Destination	Protocol	Length	Info
1	05:27:48.292768	10.0.1.99	10.0.10.0	TCP	74	8081 → 80 [SYN] Seq=0 Win=62244 Len=0
2	05:27:48.294341	10.0.10.0	10.0.1.99	TCP	74	80 → 8081 [SYN, ACK] Seq=0 Ack=1 Win=
3	05:27:48.294378	10.0.1.99	10.0.10.0	TCP	66	8081 → 80 [ACK] Seq=1 Ack=1 Win=62336
4	05:28:09.447740	10.0.1.99	10.0.10.0	TCP	88	8081 → 80 [PSH, ACK] Seq=1 Ack=1 Win=
5	05:28:09.451664	10.0.10.0	10.0.1.99	TCP	66	80 → 8081 [ACK] Seq=1 Ack=23 Win=6553
6	05:28:10.679866	10.0.1.99	10.0.10.0	HTTP	67	GET /10B.txt HTTP/1.0
7	05:28:10.682920	10.0.10.0	10.0.1.99	TCP	66	80 → 8081 [ACK] Seq=1 Ack=24 Win=6553
8	05:28:10.683470	10.0.10.0	10.0.1.99	HTTP	317	HTTP/1.1 200 OK (text/plain)
9	05:28:10.725536	10.0.1.99	10.0.10.0	TCP	66	8081 → 80 [ACK] Seq=24 Ack=253 Win=62
10	05:28:11.939793	10.0.1.99	10.0.10.0	HTTP	67	Continuation
11	05:28:11.942469	10.0.10.0	10.0.1.99	TCP	54	80 → 8081 [RST] Seq=253 Win=0 Len=0

ecmper-xdp-veth101.trc

	Time	Source	Destination	Protocol	Length	Info
1	05:27:48.293695	10.0.1.99	10.0.10.0	TCP	74	8081 → 80 [SYN] Seq=0 Win=62244 Len=0
2	05:27:48.293819	10.0.10.0	10.0.1.99	TCP	74	80 → 8081 [SYN, ACK] Seq=0 Ack=1 Win=6
3	05:27:48.294766	10.0.1.99	10.0.10.0	TCP	66	8081 → 80 [ACK] Seq=1 Ack=1 Win=62336
4	05:28:09.450787	10.0.1.99	10.0.10.0	TCP	88	8081 → 80 [PSH, ACK] Seq=1 Ack=1 Win=6
5	05:28:09.450865	10.0.10.0	10.0.1.99	TCP	66	80 → 8081 [ACK] Seq=1 Ack=23 Win=65536
6	05:28:10.681760	10.0.1.99	10.0.10.0	HTTP	67	GET /10B.txt HTTP/1.0
7	05:28:10.681797	10.0.10.0	10.0.1.99	TCP	66	80 → 8081 [ACK] Seq=1 Ack=24 Win=65536
8	05:28:10.682685	10.0.10.0	10.0.1.99	HTTP	317	HTTP/1.1 200 OK (text/plain)
9	05:28:10.727061	10.0.1.99	10.0.10.0	TCP	66	8081 → 80 [ACK] Seq=24 Ack=253 Win=622
10	05:28:11.941641	10.0.1.99	10.0.10.0	HTTP	67	Continuation
11	05:28:11.941705	10.0.10.0	10.0.1.99	TCP	54	80 → 8081 [RST] Seq=253 Win=0 Len=0

ecmper-xdp-veth102.trc

	Time	Source	Destination	Protocol	Length	Info
1	05:28:09.449898	10.0.1.99	10.0.10.0	TCP	88	8081 → 80 [PSH, ACK] Seq=1 Ack=1 Win=487
2	05:28:09.449927	10.0.1.99	10.0.10.0	TCP	88	[TCP Retransmission] 8081 → 80 [PSH, ACK
3	05:28:10.680904	10.0.1.99	10.0.10.0	HTTP	67	GET /10B.txt HTTP/1.0
4	05:28:10.680917	10.0.1.99	10.0.10.0	TCP	67	[TCP Keep-Alive] 8081 → 80 [PSH, ACK] Se
5	05:28:10.726280	10.0.1.99	10.0.10.0	TCP	66	8081 → 80 [ACK] Seq=24 Ack=253 Win=486 L
6	05:28:10.726285	10.0.1.99	10.0.10.0	TCP	66	[TCP Dup ACK 5#1] 8081 → 80 [ACK] Seq=24
7	05:28:11.940805	10.0.1.99	10.0.10.0	HTTP	67	Continuation
8	05:28:11.940817	10.0.1.99	10.0.10.0	TCP	67	[TCP Keep-Alive] 8081 → 80 [PSH, ACK] Se