

# Reliability & Congestion Control of UDP Streaming APP

---

네트워크프로젝트 12조

강남구, 이경돈, 이규민, 이원규, 장지호, 조하영



# Contents

01

Introduction



02

Codeworks



05

Conclusion



03

Experiment



04

Result



The slide features decorative geometric patterns in the corners, consisting of thin blue lines, dots, and concentric circles. The top-left corner has a cluster of lines and dots. The top-right corner has a circle with a dot inside and a line with a dot. The bottom-left corner has a circle with a dot inside and a line with a dot. The bottom-right corner has a circle with a dot inside and a line with a dot.

# 01

---

## Introduction

# Reliability & Congestion Control

- UDP Streaming Application
- **Reliability & Congestion Control**을 직접 구현
- 다양한 **Network Environment & Topology**  
시나리오에 대해 시뮬레이션을 진행



# Contribution

---



**조하영**

Streaming Code  
- Reliability



**이경돈**

Helper Code



**강남구**



**이원규**

Main Topology Code  
Experiment



**장지호**

Presentation



**이규민**

Streaming Code  
- Congestion Control

# Timeline



A horizontal timeline with five milestones. The title 'Timeline' is centered at the top. Below it, a horizontal line with dots at each milestone date is shown. The milestones are: 5/6 (Project Meeting), 5/17 (Streaming Code), 5/22 (Helper Code Main Code), 5/31 (Issue Tracking Experiment), and 6/1 (Feedback Presentation). The dates are in a large, bold font, and the descriptions are in a smaller font below them.

Date	Event
5/6	Project Meeting
5/17	Streaming Code
5/22	Helper Code Main Code
5/31	Issue Tracking Experiment
6/1	Feedback Presentation

5/6

Project Meeting

5/17

Streaming Code

5/22

Helper Code  
Main Code

5/31

Issue Tracking  
Experiment

6/1

Feedback  
Presentation

The background features decorative geometric patterns in the corners, consisting of thin blue lines, dots, and concentric circles.

# 02

---

## Codeworks

# Codeworks

ns3.29

```
├── src
│   ├── application
│   │   ├── model
│   │   │   ├── client.cc
│   │   │   ├── client.h
│   │   │   ├── server.cc
│   │   │   └── server.h
│   │   └── helper
│   │       ├── helper.cc
│   │       └── helper.h
└── scratch
    └── topology.cc
```

- Reliable & Congestion Control이 가능한 Application을 구현하기 위해 **Streaming Server & Client** 코드 구현
- Server & Client Application을 사용하기 위한 **Helper**
- Topology 구성 및 시뮬레이션을 위한 **Main Code**



# Server



# Client

Streamer()

Client()

StartApplication()

Bind

StartApplication()

Send()

Random Packet Loss

HandleRead(Ptr<Socket>)

PutFrameBuffer(uint32\_t frameN, uint32\_t seqN)

HandleRead(Ptr<Socket>)

SendCheck(Address from, Ptr<Socket>)

ReSend()

Flowcontrol()

◀ Congestion Control

▲ Reliable Communication

FrameConsumer()

StopApplication()

StopApplication()

~ Streamer()

~ Client()

# Reliable Communication



Client

```
SendCheck(Address from, Ptr<Socket> socket)
```

Missing Packet Number

Server



```
HandleRead()
```

```
ReSend()
```

Missing Packet

- **Client::SendCheck()**에서 Packet을 받으면 누락된 Frame/Packet이 있는지 확인
- 누락된 것이 있는 경우 Missing Packet Sequence Number를 Packet Header에 담아 Streamer로 전송
- Streamer는 Packet을 받으면 **Server::HandleRead()** Callback
- **Server::HandleRead()**에서 Sequence Number를 확인
- 해당 Frame Packet을 Client로 **ReSend()**

# Congestion Control

Server 

HandleRead()

FPS

FlowControl()

- **Server::Flowcontrol()**는 전송 상태에 따라 **FPS**를 조절
  - 전송이 원활하면 FPS를 증가시키지만,
  - Client가 ReSend()를 요청하면 FPS를 감소시키는 방식
- 두 가지 Congestion Control Mode를 구현
  1. **AIMD**
  2. **Slow Start**

# Congestion Control

## Congestion Control Mode

### 1. AIMD

- 1 Frame을 보내면 FPS 1 증가
- Client가 ReSend()를 요청하면 FPS를 절반으로 설정
- 단, 최소값 이하로는 내려가지 않는다.

## FlowControl()

```
if(drop == 1) {  
    m_fps = m_fps / 2;  
    if(m_fps < 30)  
        m_fps = 30;  
}  
else {  
    m_fps++;  
}
```

# Congestion Control

## Congestion Control Mode

### 2. Slow Start

- threshold에 도달하지 않을 때 1 Frame을 보내면 FPS 2배 증가
- threshold에 도달했을 때 1 Frame을 보내면 FPS 1 증가
- Client가 ReSend()를 요청하면 FPS를 최소값으로 설정
  - 여기서 Streamer가 slowstart 상태이고, resend 요청이  $m\_seqN$ 보다 3 이상 낮으면 threshold도 감소시킴

## FlowControl()

```
if(drop == 1) {  
    if(m_slowstart) {  
        if(m_seqN - 3 > seqN) {  
            m_slowstart = false;  
            m_threshold = m_fps / 2;  
            m_fps = 30;  
        }  
        else  
            m_fps = 30;  
    }  
    else {  
        m_threshold = m_fps / 2;  
        m_fps = 30;  
    }  
}  
else {  
    if(m_seqN < m_threshold) //Slow start  
        m_fps = m_fps * 2;  
    else //Congestion avoidance  
        m_fps++;  
}
```

The background features decorative geometric patterns in the corners, consisting of thin blue lines, dots, and concentric circles. A horizontal line with dots at its ends is positioned below the number 03.

03

Experiment



# Experiment




여러 노드와 다양한 연결 방식을 가진 **Topology** 구성  
Argument Parsing을 사용해 Streaming Parameter 설정  
Streaming Server – Client의 **Status, Data Flow** 분석

## 1. Arguments for Streamer

- PacketSize : Packet Size
- PacketNip : Packet Per Frame
- Fps : Streaming FPS
- LossEn : Forced Packet Loss On/Off
- LossRate : Loss Probability
- Mode : Select Congestion Control Mode
- Threshold : Select Threshold

## 2. Arguments for Client

- PacketSize : Packet Size
  - PacketNip : Packet Per Frame
  - BufferSize : Frame Buffer Size
- 

# Topologies

— Background Flow  
— Streaming

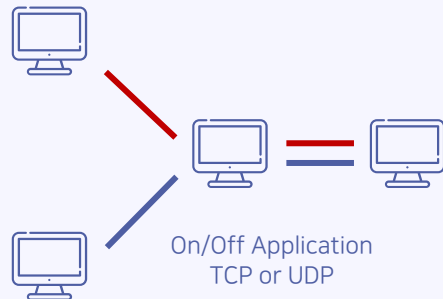
1



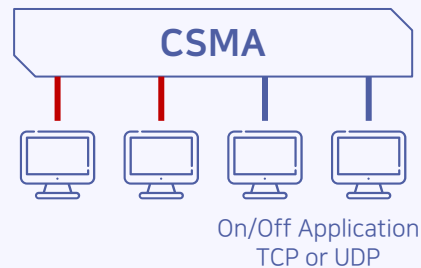
2



3



4





The background features decorative geometric patterns in the corners, consisting of thin blue lines, dots, and circles. In the top-left, there are several parallel lines and a small cluster of dots. In the top-right, a circle with a dot inside is connected to a line. In the bottom-left, there are more parallel lines and a small cluster of dots. In the bottom-right, there are several parallel lines and a circle with a dot inside.

# 04

---

## Results



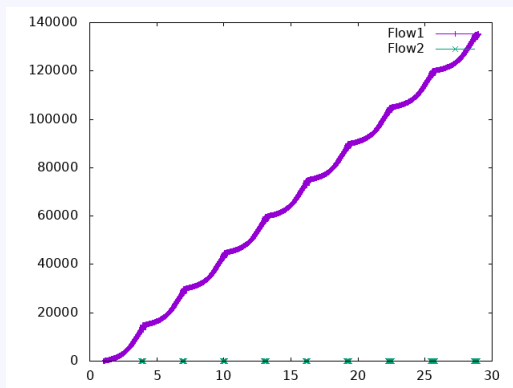
## Topology 1



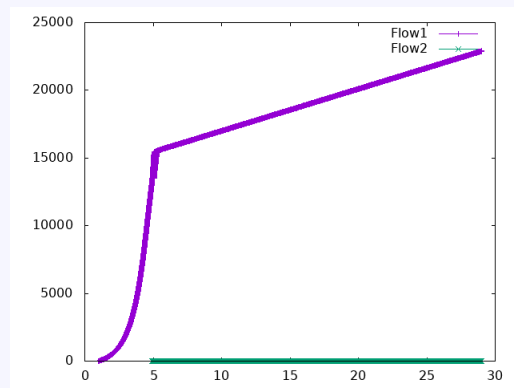
# Result (1-1)

- **Packet NIP**에 따른 Sequence Number & Resend Count 측정 (packet size = 30, mode 0, buffer size = 40)  
→ 한 Frame에서 다음 Frame Number인 Packet이 도착할 때 Resend 요청이 발생하기 때문에 Packet NIP가 작을수록 재전송 주기가 짧아지는 결과를 확인할 수 있음.

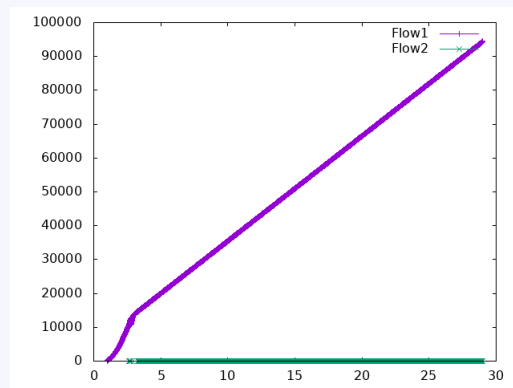
Flow1: Sequence Number, Flow2: Resend Count



Packet NIP = 10



Packet NIP = 60



Packet NIP = 100



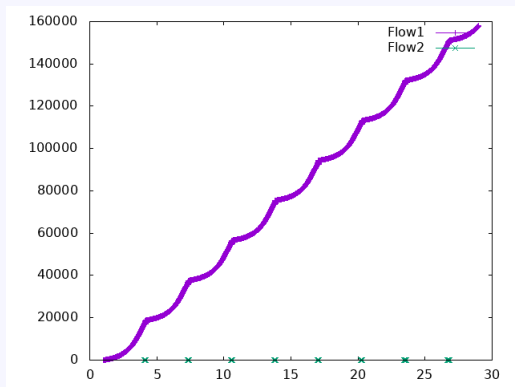
## Topology 1



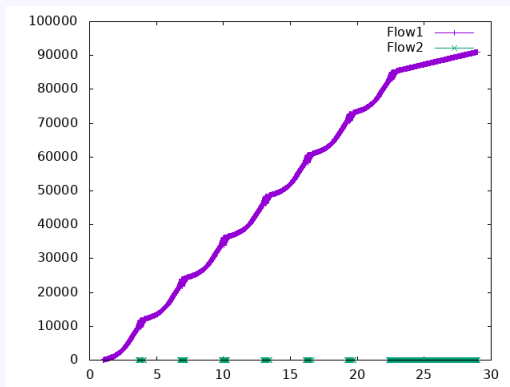
# Result (1-2)

- **Packet Size**에 따른 Sequence Number & Resend Count 측정 (packet nip = 30, mode 0, buffer size = 40)  
→ 한 Frame에서 다음 Frame Number인 Packet이 도착할 때 Resend 요청이 발생하기 때문에 Packet Size가 작을수록 Streaming Rate가 증가하고, 재전송 주기가 짧아지는 결과를 확인할 수 있음.

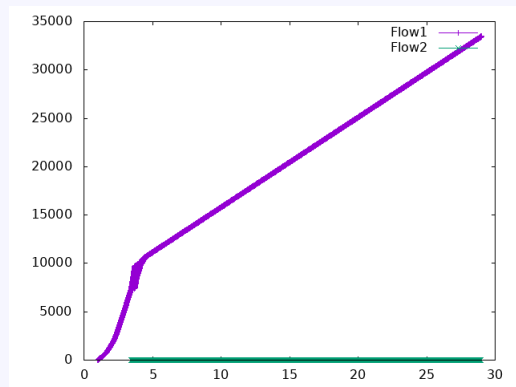
Flow1: Sequence Number, Flow2: Resend Count



Packet Size = 10



Packet Size = 60



Packet Size = 100



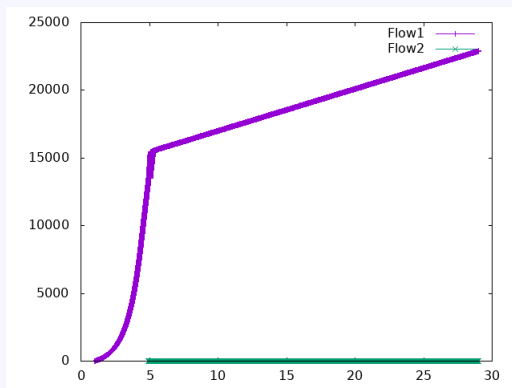
## Topology 1



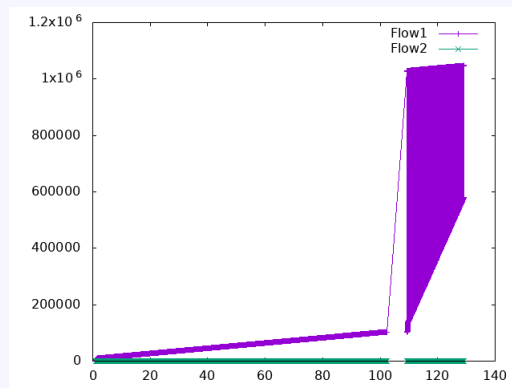
# Result (1-3)

- Congestion Control Mode에 따른 Sequence Number & Resend Count 측정 (size = 30, packet ntp = 30, buffer size = 40)
  - AIMD에서는 Drop이 나지 않으면 FPS가 1, Drop이 나면 절반이 되기 때문에 그래프가 선형으로 변화하게 됨.
  - Slow Start에서는 Drop이 나지 않으면 FPS가 2배가 되어 급격히 증가하게 되고, 손실 된 Packet들의 Segment가 면적으로 나타나게 됨.

Flow1: Sequence Number, Flow2: Resend Count



AIMD



Slow Start



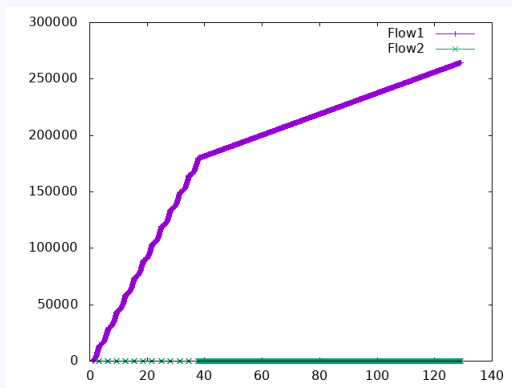
## Topology 1



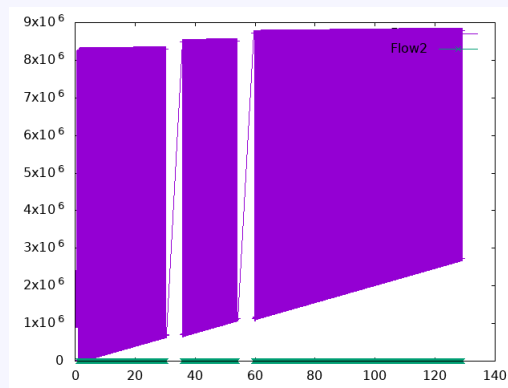
# Result (1-4)

- **Threshold**에 따른 Sequence Number & Resend Count 측정 (packet size = 30, packet ntp = 30, mode 0, buffer size = 40)  
→ Threshold가 증가하면 fps가 두배 씩 증가하는 주기가 길어져 Sending Rate가 급격히 증가한다. 따라서 Drop되는 Packet이 증가하게 되며, Resend 할 때마다 Packet Check에서 손실 된 Packet들의 Segment가 면적으로 나타나게 됨.

Flow1: Sequence Number, Flow2: Resend Count



Threshold = 50



Threshold = 500



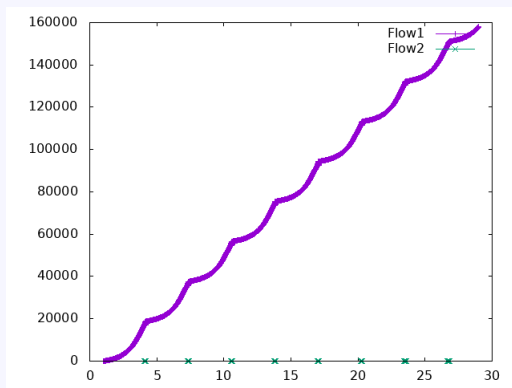
## Topology 1



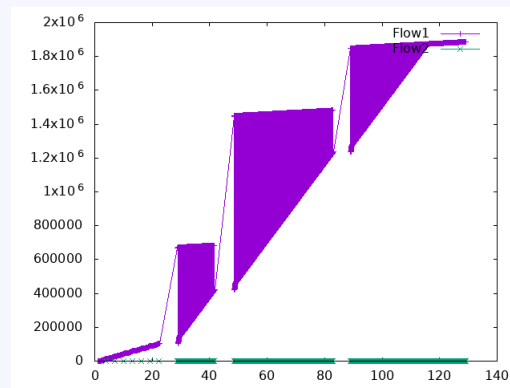
# Result (1-5)

- **Loss Rate**에 따른 Sequence Number & Resend Count 측정 (packet size = 30, packet nip = 30, mode 0, buffer size = 40)  
→ Loss Rate가 증가하면 Drop 되는 패킷도 증가하고, Resend 할 때마다 Packet Check에서 손실 된 Packet들의 Segment가 면적으로 나타나게 됨.

Flow1: Sequence Number, Flow2: Resend Count



Loss Rate = 0.01



Loss Rate = 0.1



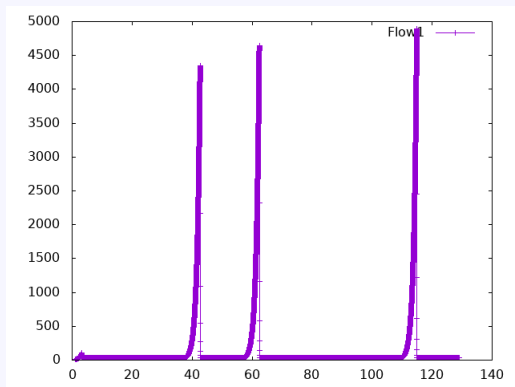
## Topology 1



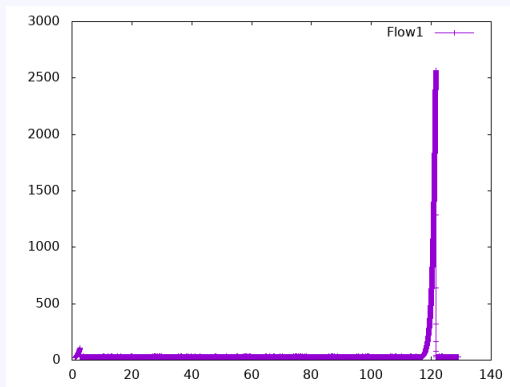
# Result (2-1)

- 초기 FPS에 따른 FPS 측정 (mode 0, packetSize = 100, packetNip = 100, threshOld = 200, bufferSize = 40)
- FPS가 클 수록 서버가 전송하는 Packet/Frame 개수가 증가하고, 마찬가지로 클라이언트가 받는 Packet/Frame의 개수도 증가한다.
- 이때 Packet Sequence를 확인하는 과정에서 손실된 Packet을 더 많이 발견하게 되고, 최종적으로 FPS가 감소하는 경향이 나타난다.

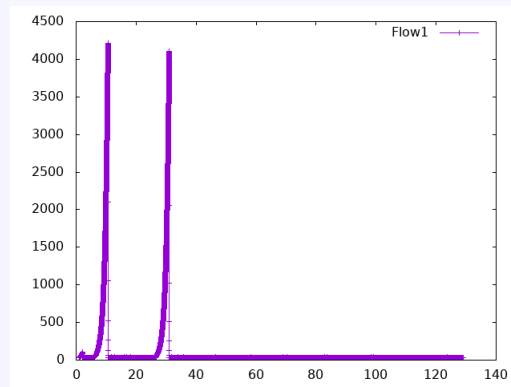
Flow1: FPS



FPS = 10



FPS = 20



FPS = 30



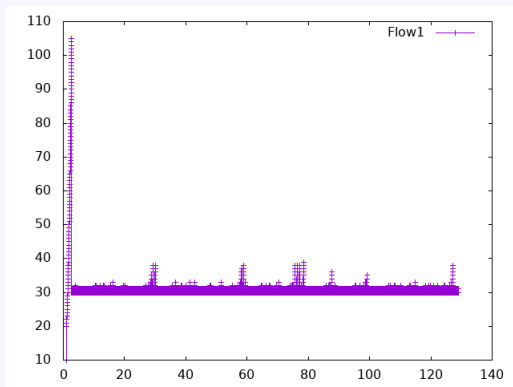
## Topology 1



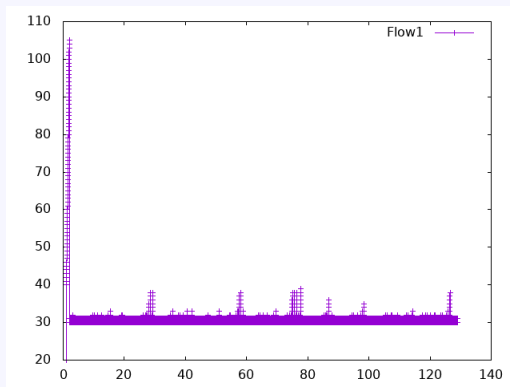
# Result (2-2)

- 초기 FPS에 따른 FPS 측정 (mode 1, packetSize = 100, packetNip = 100, threshOld = 200, bufferSize = 40)
- FPS가 클 수록 서버가 전송하는 Packet/Frame 개수가 증가하고, 마찬가지로 클라이언트가 받는 Packet/Frame의 개수도 증가한다.
- 이때 Packet Sequence를 확인하는 과정에서 손실된 Packet을 더 많이 발견하게 되어 최종적으로 FPS가 감소하는 경향이 나타난다.

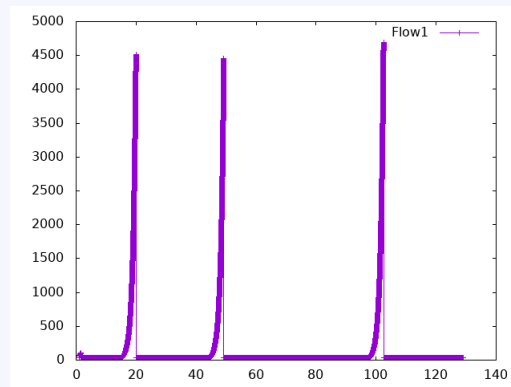
Flow1: FPS



FPS = 10



FPS = 20



FPS = 30





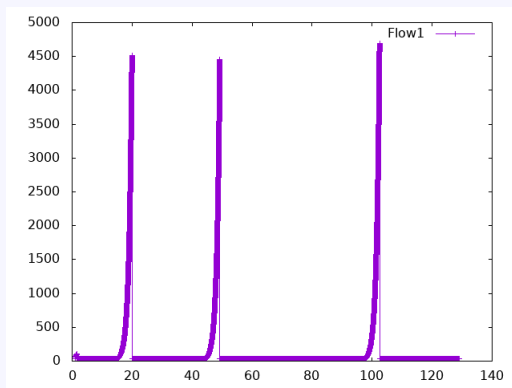
## Topology 1



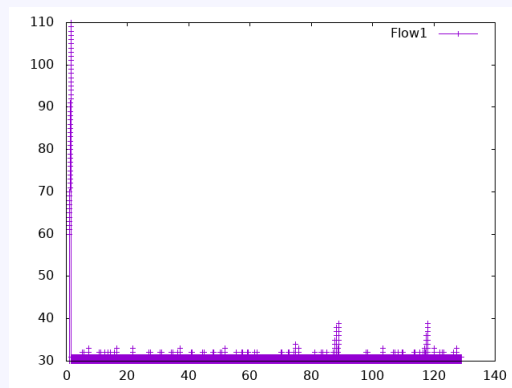
# Result (2-3)

- **Loss Rate**에 따른 FPS 측정 (mode 1, packetSize = 100, packetNip = 100, threshHold = 200, bufferSize = 40, fps = 30)  
→ Loss Rate가 증가하면 Drop 되는 패킷도 증가하고, Packet Sequence를 확인하는 과정에서 손실된 Packet을 더 많이 발견하게 되어 최종적으로 FPS가 감소하는 경향이 나타난다.

Flow1: FPS



Loss Rate = 0.01



Loss Rate = 0.1



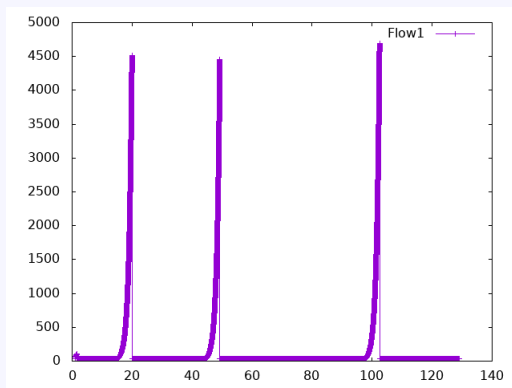
## Topology 1



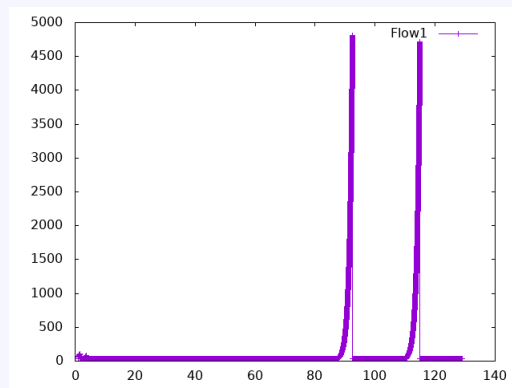
# Result (2-4)

- **Buffer Size**에 따른 FPS 측정 (mode 1, packetSize = 100, packetNip = 100, threshHold = 200, fps = 30)  
→ Buffer Size가 클 수록 Buffer Full로 인해 Packet Loss가 될 확률이 감소한다. 따라서 클라이언트가 계속 패킷을 받을 수 있기 때문에 FPS가 증가경향이 나타난다.

Flow1: FPS



Buffer Size = 40



Loss Rate = 100



## Topology 1

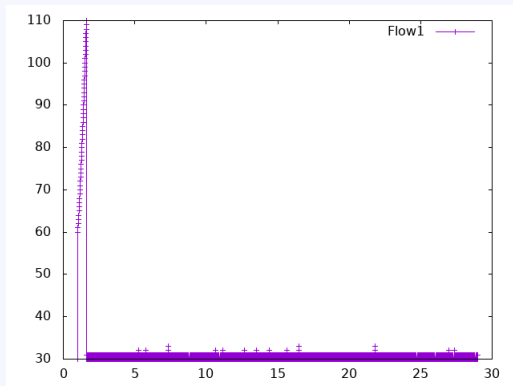


# Result (2-5)

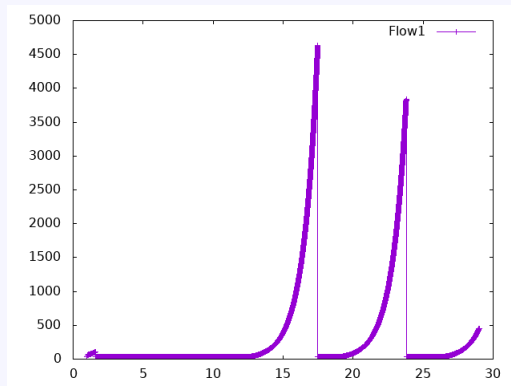
- Loss Rate에 따른 FPS 측정

base parameter: fps = 30 packetSize = 100, packetNip = 100, mode = 1, threshold = 200, bufferSize = 40

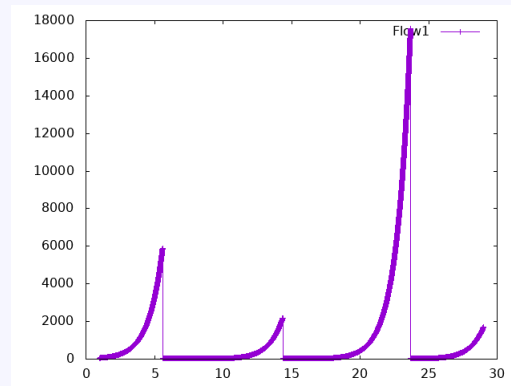
Flow1: FPS



Loss Rate = 0



Loss Rate = 0.1



Loss Rate = 0.3



Topology2

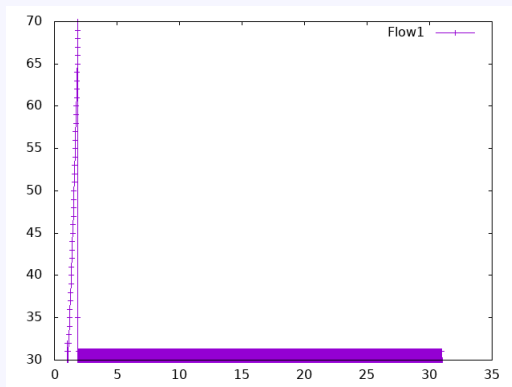


# Result (3-1)

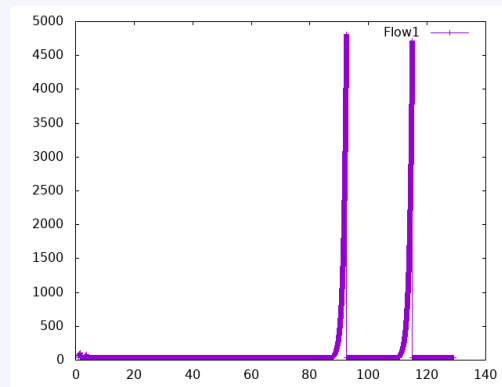
- Loss Rate에 따른 Server Streaming FPS 측정 (mode 0)

base parameter: packetSize = 100, packetNip = 100, threshold = 200, bufferSize = 40

Flow1: FPS



Loss Rate = 0



Loss Rate = 0.1



## Topology2

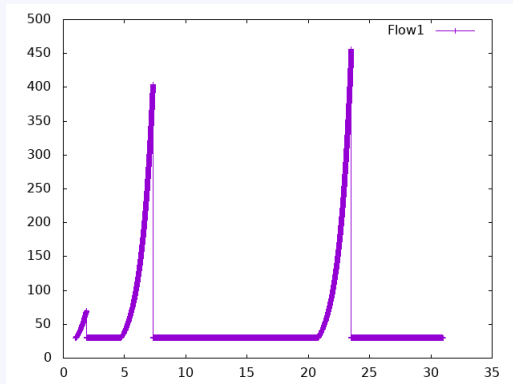


# Result (3-2)

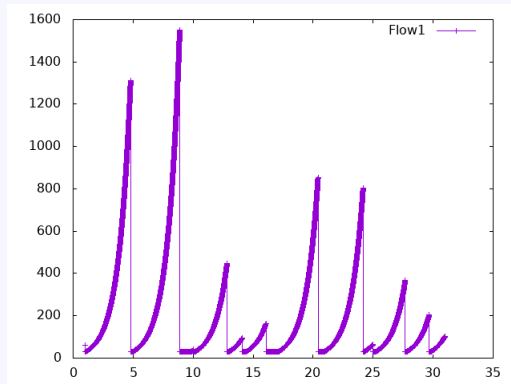
- Loss Rate에 따른 Server Streaming FPS 측정 (mode 1)

base parameter: packetSize = 100, packetNip = 100, threshold = 200, bufferSize = 40

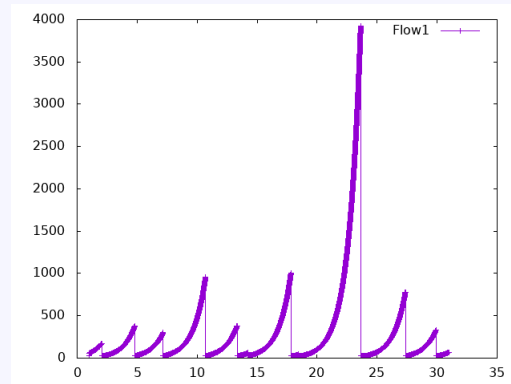
Flow1: FPS



Loss Rate = 0.1



Loss Rate = 0.3



Loss Rate = 0.5



Topology2

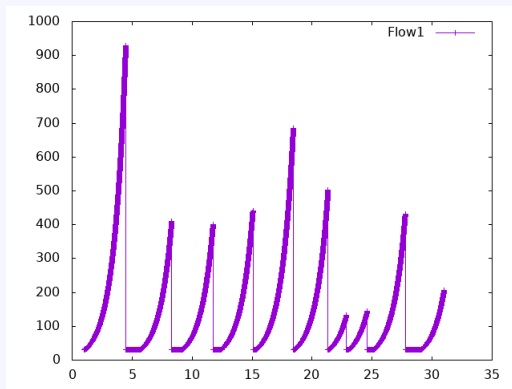


## Result (3-3)

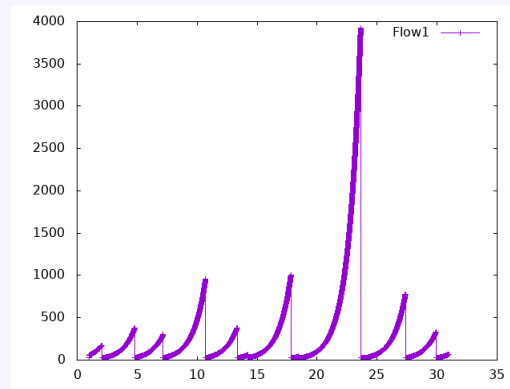
- Threshold에 따른 Server Streaming FPS 측정 (mode 1)

base parameter: packetSize = 100, packetNip = 100, threshold = 200, bufferSize = 40, loss 0.3

Flow1: FPS



Threshold = 100



Threshold = 200



## Topology2

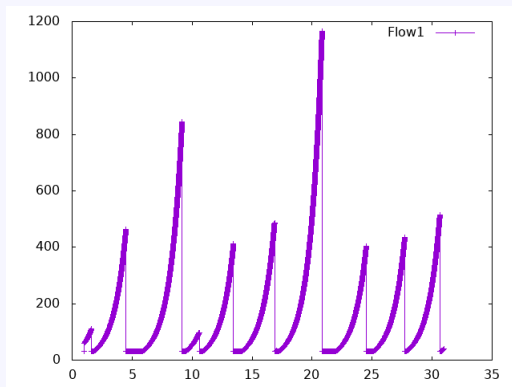


# Result (3-4)

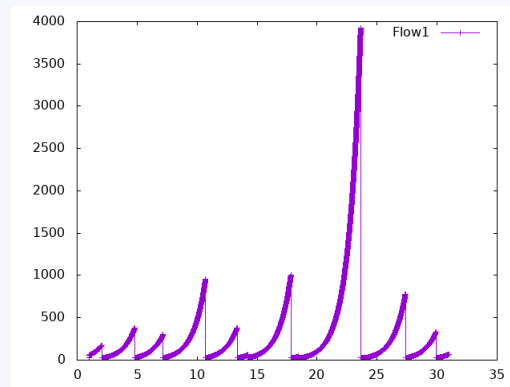
- Buffer Size에 따른 Server Streaming FPS 측정 (mode 1)

base parameter: packetSize = 100, packetNip = 100, threshold = 200, bufferSize = 40, loss 0.5

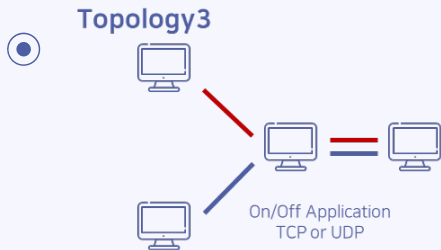
Flow1: FPS



Buffer Size = 20

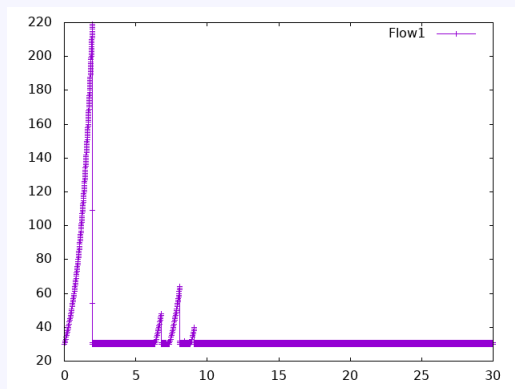


Buffer Size = 40

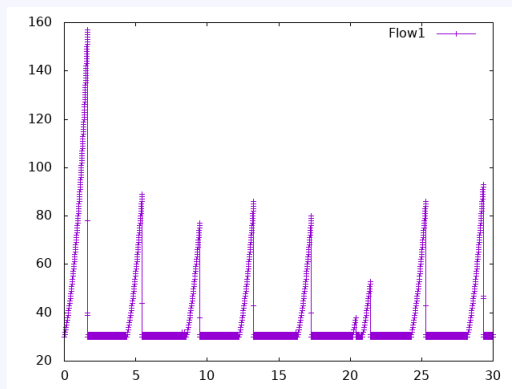


# Result (4-1)

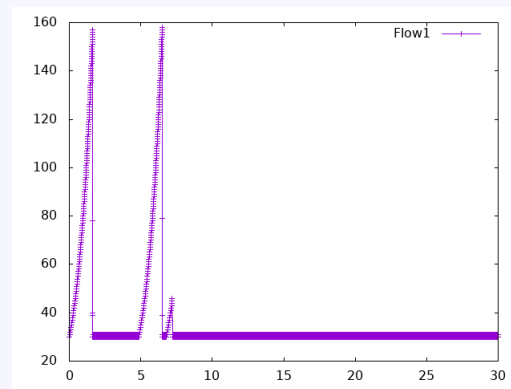
- Background Flow에 따른 FPS 측정 (loss = 0, mode 0)



UDP 1(on)-0(off)

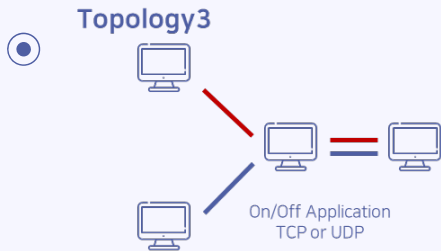


UDP 0(on)-1(off)



UDP 1(on)-1(off)

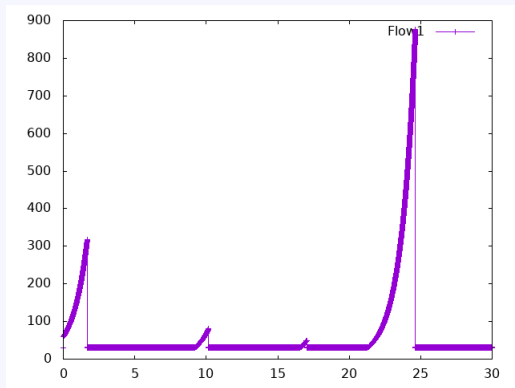




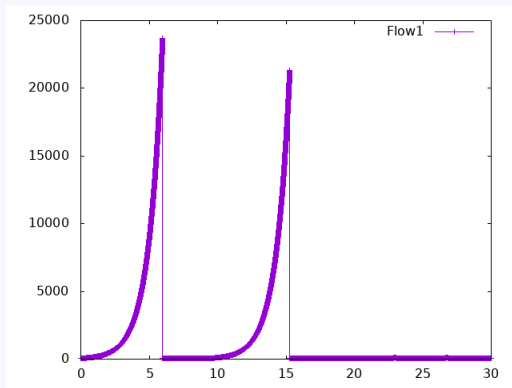
## Result (4-2)

- Background Flow에 따른 FPS 측정 (loss = 0.3, mode 0)

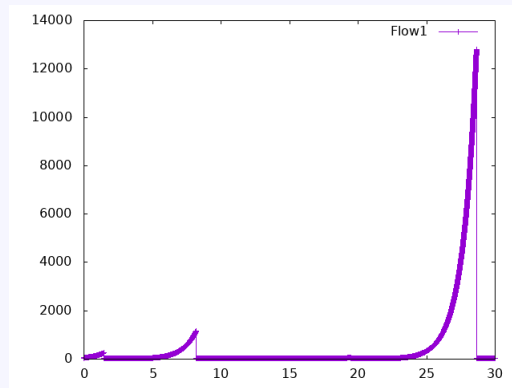
Flow1: FPS



UDP 1(on)-0(off)

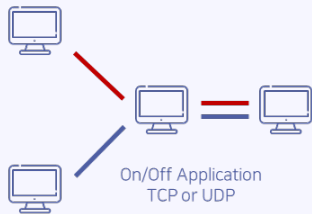


UDP 0(on)-1(off)



UDP 1(on)-1(off)

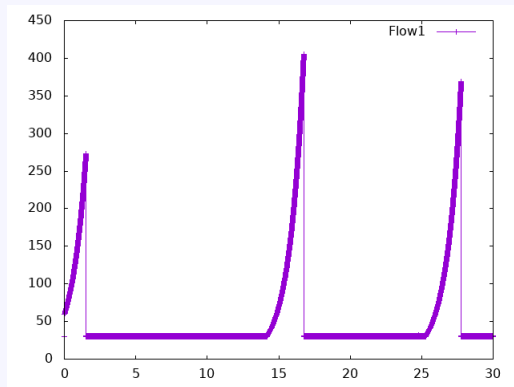
### Topology3



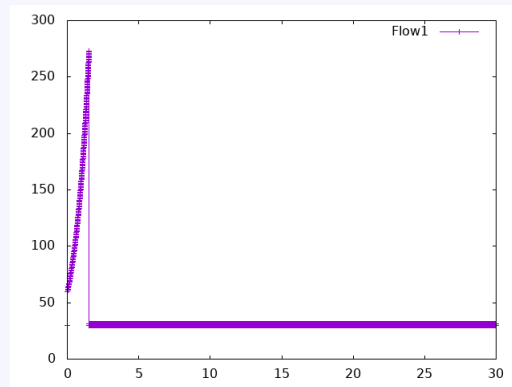
## Result (4-3)

- Background Flow가 TCP일 때, Loss Rate에 따른 FPS 측정 (TCP 1(on)-0(off))

Flow1: FPS



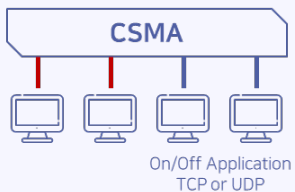
Loss Rate = 0



Loss Rate = 0.3



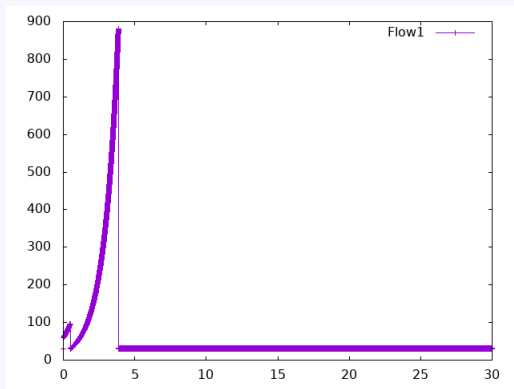
## Topology4



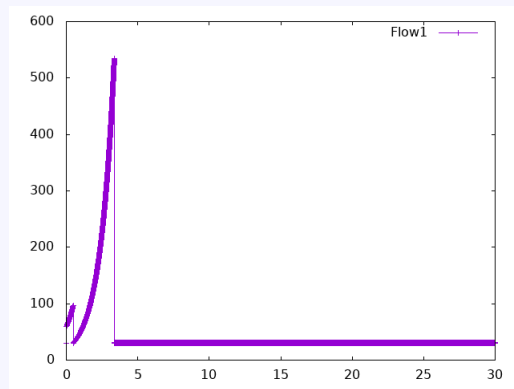
# Result (5-1)

- Background Flow에 따른 FPS 측정 (Loss = 0)

Flow1: FPS



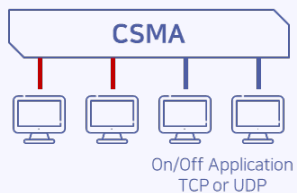
TCP 1(on)-1(off)



UDP 1(on)-1(off)



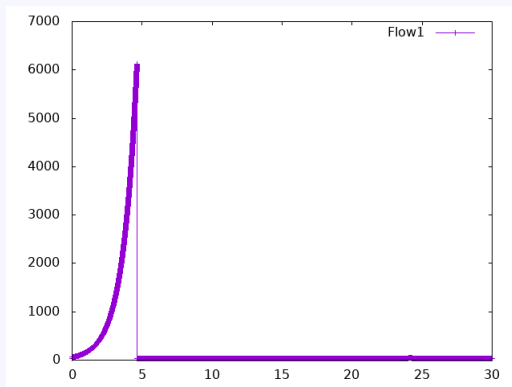
## Topology4



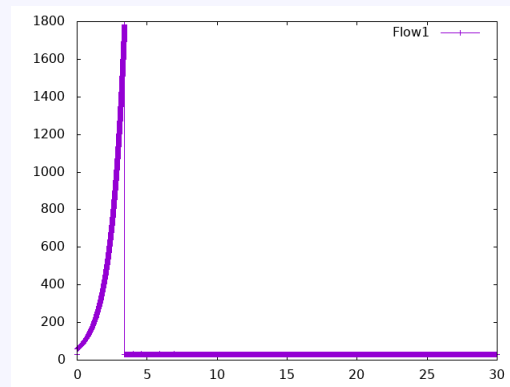
# Result (5-2)

- Background Flow에 따른 FPS 측정 (Loss = 0.3)

Flow1: FPS



TCP 1(on)-1(off)



UDP 1(on)-1(off)

The slide features decorative geometric patterns in the corners, consisting of thin blue lines, dots, and circles. In the top-left, there are several parallel lines and a small cluster of dots. The top-right has a circle with a dot inside and a line with a dot. The bottom-left shows a circle with a dot and a line with a dot. The bottom-right has a circle with a dot and a line with a dot.

# 05

---

## Conclusion

# Conclusion (1)

1. **Packetnip**: packetnip가 작을수록 resend check하는 주기가 작아짐
2. **Packet size**: packet size가 작을수록 streaming rate가 커지게 되어 resend check하는 주기가 작아짐
3. **Mode**
  - mode 0일 경우: drop 안 나면 fps+1씩 증가하여 그래프 지수적으로 증가, drop 날 경우 fps/2씩 되어 그래프 선형으로 증가
  - mode 1일 경우: 증가속도 빨라서 초반부터 drop → drop 안 날 때 fps 급격히 증가 → resend check할 때 loss난 packet들의 sequence number가 면적으로 그래프 나타남
4. **Threshold**: threshold 가 클수록 fps가 2배 되는 주기가 길어짐 → sending rate 급격히 증가 → drop 되는 packet 개수 증가 → resend check할 때 loss난 packet들의 sequence number가 면적으로 그래프 나타남

## Conclusion (2)

5. **Loss rate:** loss rate 가 클수록 drop 되는 packet 개수 증가 → resend check할 때 loss난 packet들의 sequence number가 면적으로 그래프 나타남
6. **FPS:** fps가 클수록 보내는 packet 개수 증가 → client 측에서 도착하는 packet 증가 → frame resend check할 때 loss난 packet 발견 횟수 잦음 → fps 작아짐
7. **Buffer size:** buffer size 클수록 → buffer full로 인해 packet loss될 확률 감소 → fps 증가
8. **UDP-TCP:** UDP가 TCP보다 더 많은 packet을 보냄 → UDP가 TCP보다 loss가 빠르게 일어남 → UDP가 TCP보다 빠른 시간에 줄어듦

The background features abstract geometric patterns in the corners, consisting of thin blue lines, dots, and concentric circles, creating a modern, minimalist aesthetic.

# Thank you & QnA