Network Applications: P2P, Streaming, and CDN

September 24, 2024

Min Suk Kang
Associate Professor
School of Computing/Graduate School of Information Security

NetS&P Research Lab @ KAIST

KAIST

Client-server paradigm

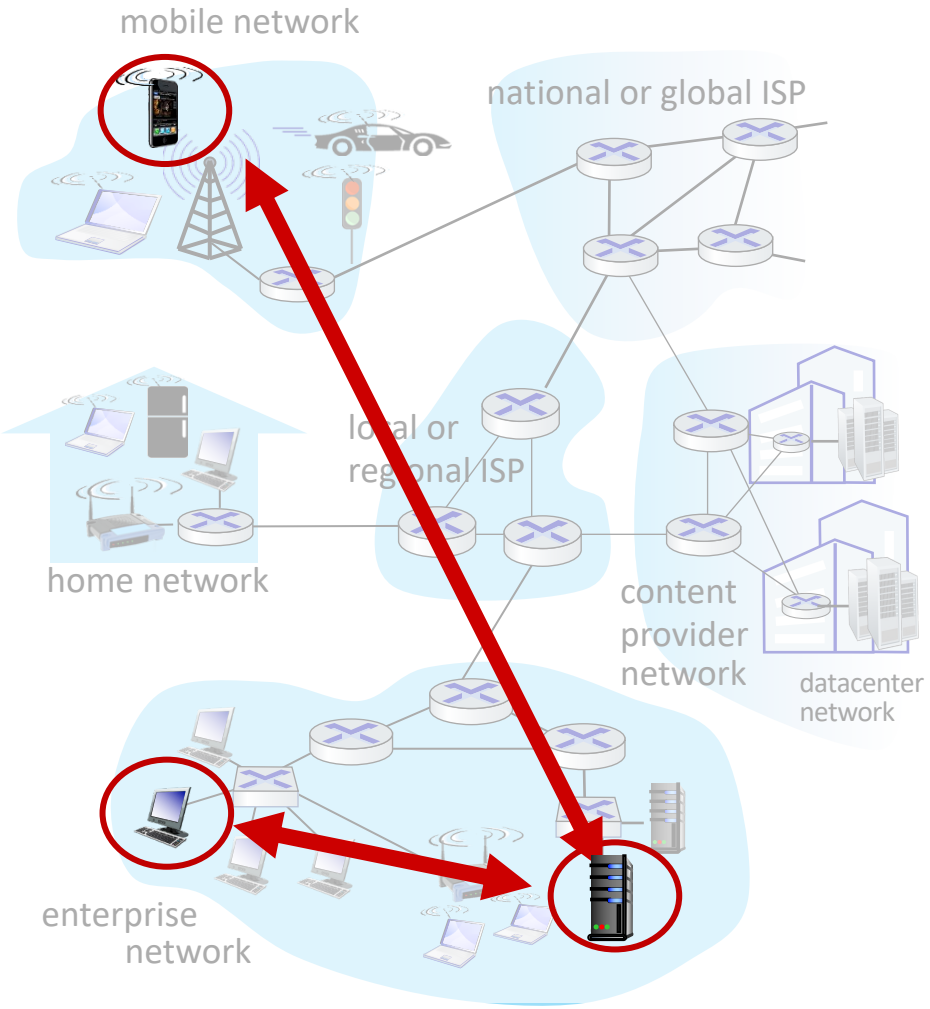
server:

- always-on host
- permanent IP address
- often in data centers, for scaling

clients:

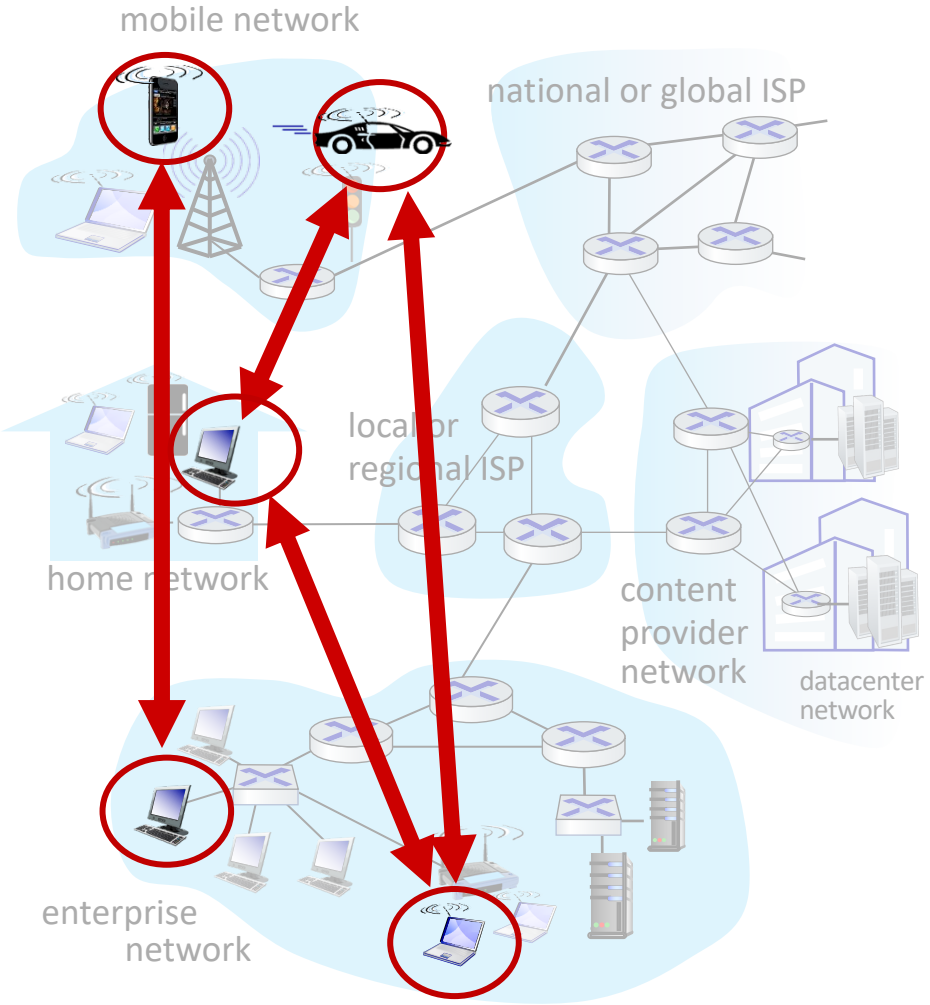
- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other

- examples: HTTP, IMAP, FTP



Peer-peer architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers may be intermittently connected and change IP addresses
  - complex management
- example: P2P file sharing



Real-world P2P applications?

- BitTorrent
- Decentralized webs
- Blockchain
- …

Defining blockchain ***without*** any jargons

- "**Agreeing** on **something** among a **group of parties** who **care about it**"

Defining blockchain ***with*** some jargons

- "**Agreeing** on **something** among a **group of parties** who **care about it**"

consensus algorithms
- safety
- liveness
- scalability
- finality

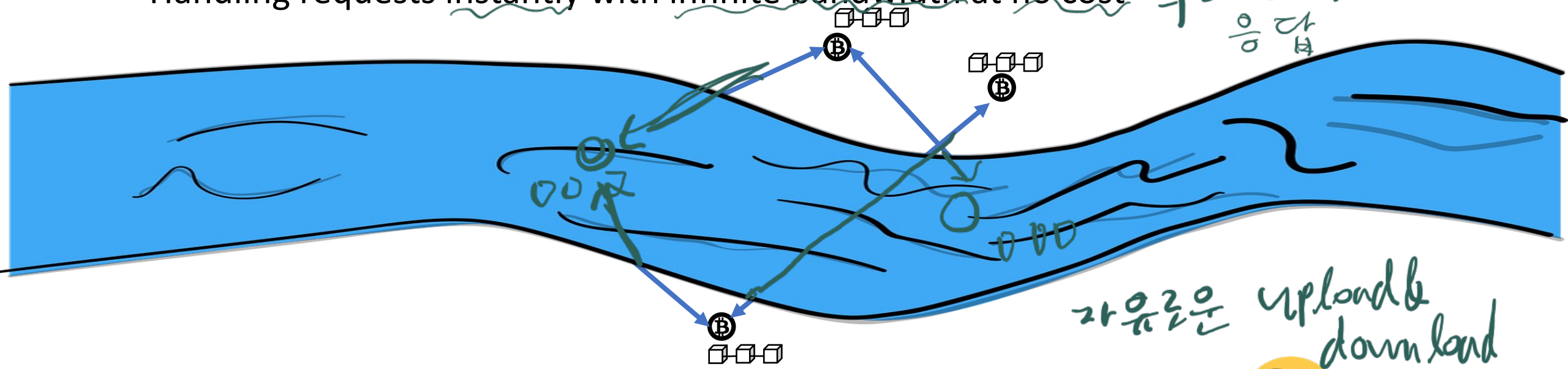
distributed ledger
- immutable
- total order
- fair ordering

decentralized peers
- permissionless
- permissioned
- Byzantine peers

Sybil protection
- proof of work
- proof of stake

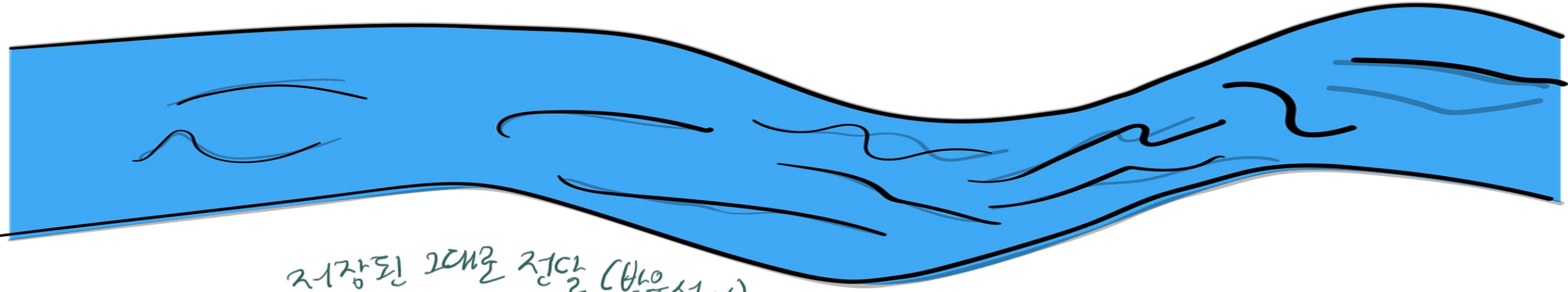
Let's Imagine "Ideal" Blockchain

- A publicly visible, verifiable, totally-ordered, fully-decentralized ledger
- Handling requests instantly with infinite bandwidth at no cost



- **Can we build a new decentralized web on this "ideal" blockchain?**

Building a New Web



- Static web
- Microblogging
- Facebook
  - Social graph, chats
- ...

- How would this new web change the ecosystem?

Back to Reality: Existing Blockchains

- Reality check:
  - 7 transactions per second, 1-hour for finizing a transaction, burning too much fossil fuel?
- Getting better (very quickly)
  - >100k transactions per second
  - <1sec finalization
  - near zero cost operation
- Technologies behind them
  - new distributed system techniques (e.g., DAG, proof of stake)
  - new cryptography (e.g., zero-knowledge proof)
  - …

Use Case: Lens Protocol

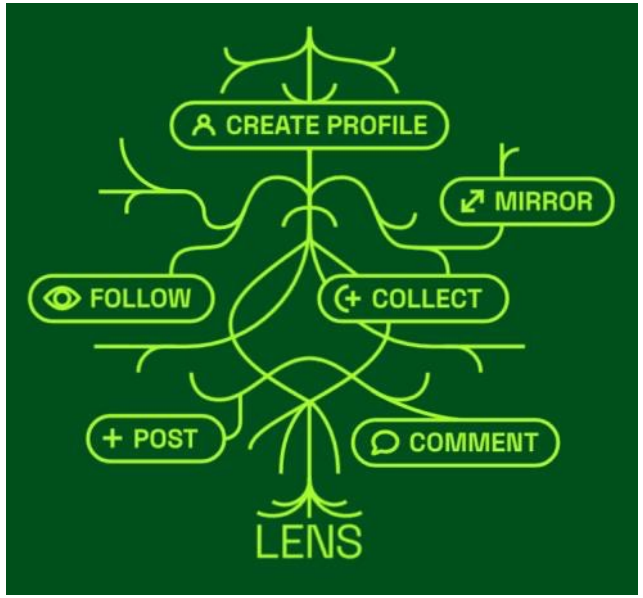
A user-owned, open social graph
that any application can plug into

나 보유하고 있는 소셜 그래프 프로토콜

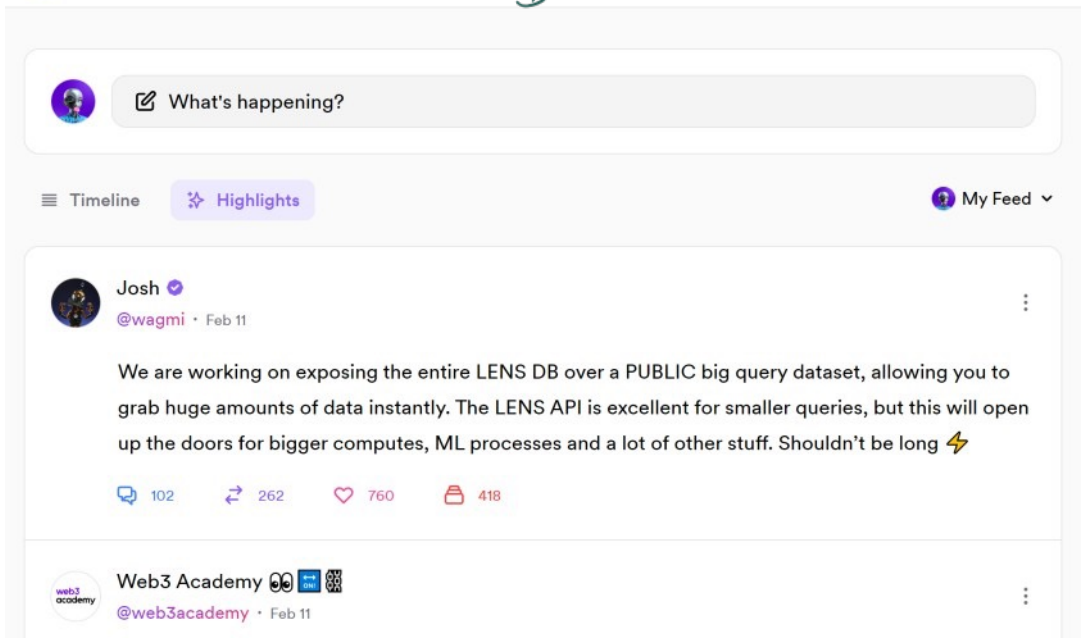
Lenster – Web3 twitter

그 소셜 데이터 통하기

social graph
data elements



https://www.lens.xyz/

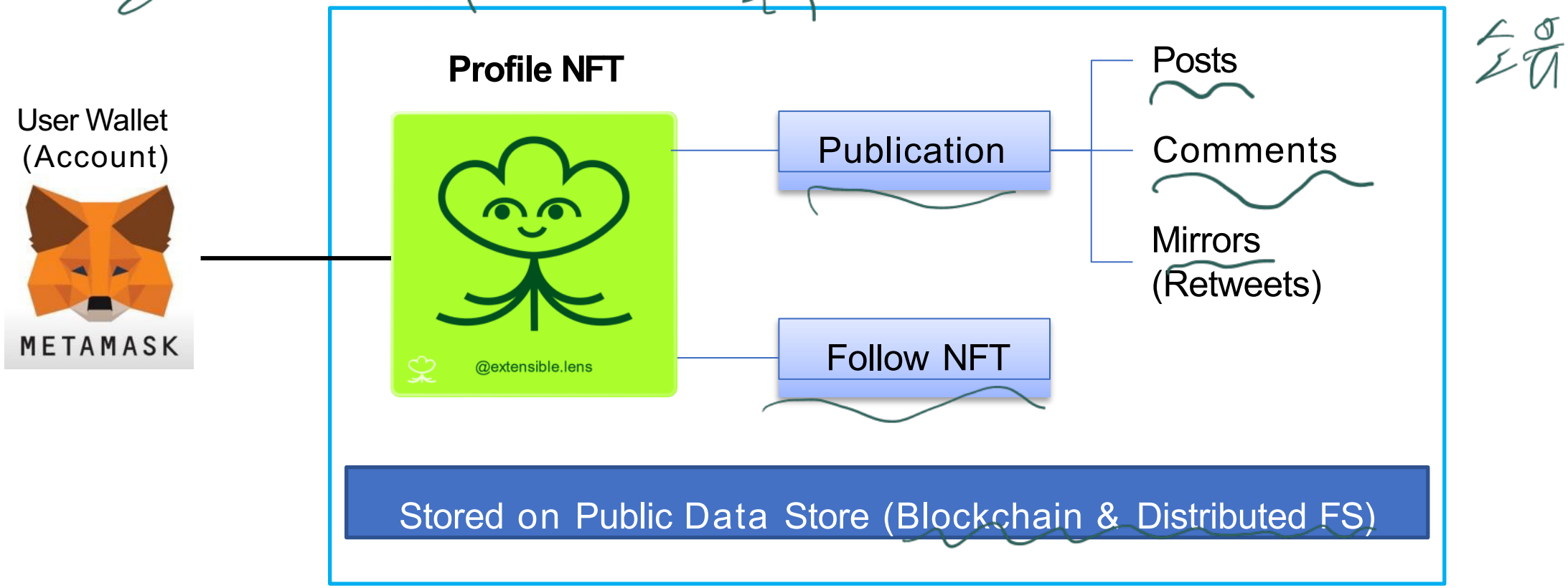


https://lenster.xyz/

1. Data Ownership to Users

Profile NFT is a key element to owns your data and gives you control of your contents

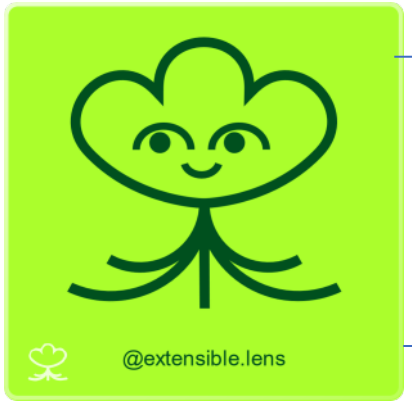
NFT로 프로필, 게시물, 댓글, 소셜 상호작용을 소유



2. Data as Digital Assets

Follow NFT, Collect, Profile NFT as a digital asset

Profile NFT



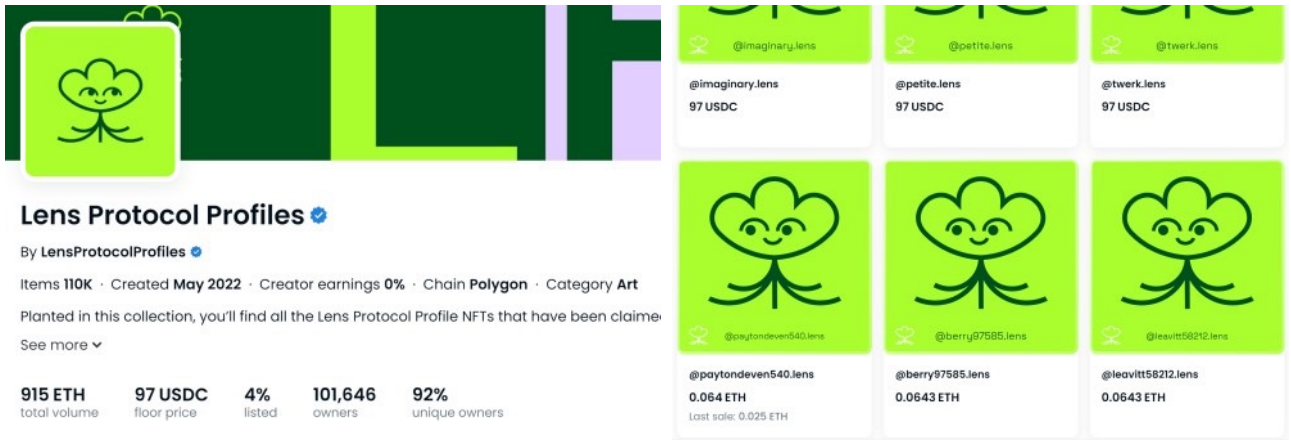
- Publication
  - Posts
  - Comments
  - Mirrors (Retweets)
- Follow NFT

Collect
Paid content

Buy & Sell
a Profile NFT
with all content

Follow
with a fee

Profile NFT market on OpenSea



https://opensea.io/collection/lens-protocol-profiles

3. Communities as data governance actors

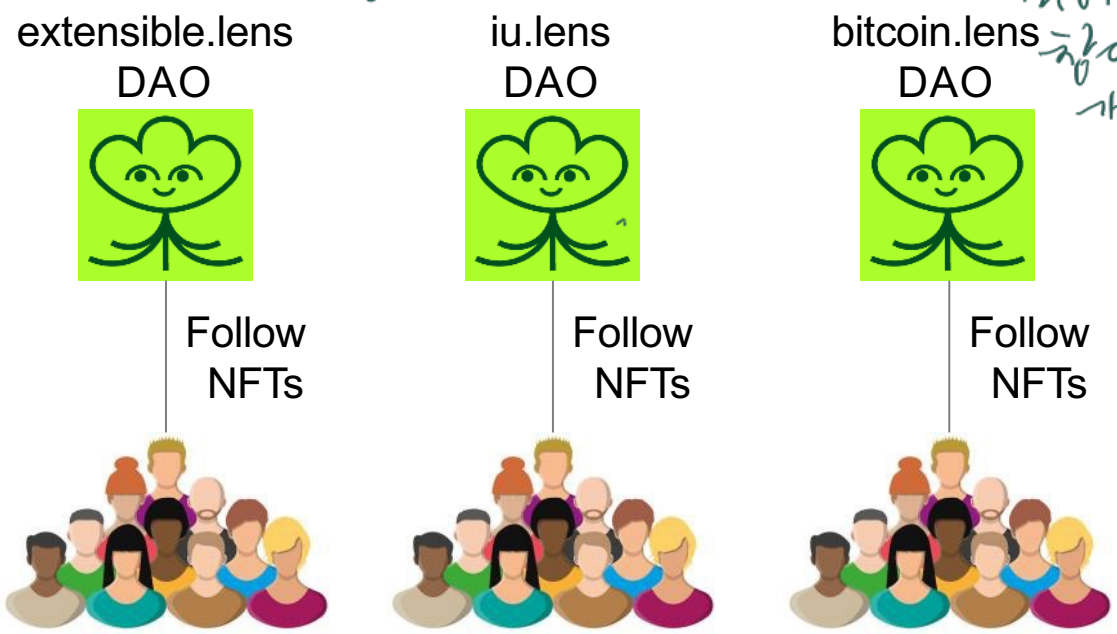
Governance for Lens Protocol

Community Multisig
by the Lens community

- Setting up Governance and Emergency Admin Addresses
- Setting Treasury Addresses and Fees
- Whitelisting Assets
- Moving the Lens Protocol System into a Publishing Paused or fully Paused state
- Whitelisting addresses to create profiles
- Whitelisting Follow, Collect, and Reference Modules
- Upgrading the Lens Protocol Hub Contract

Governance for social communities

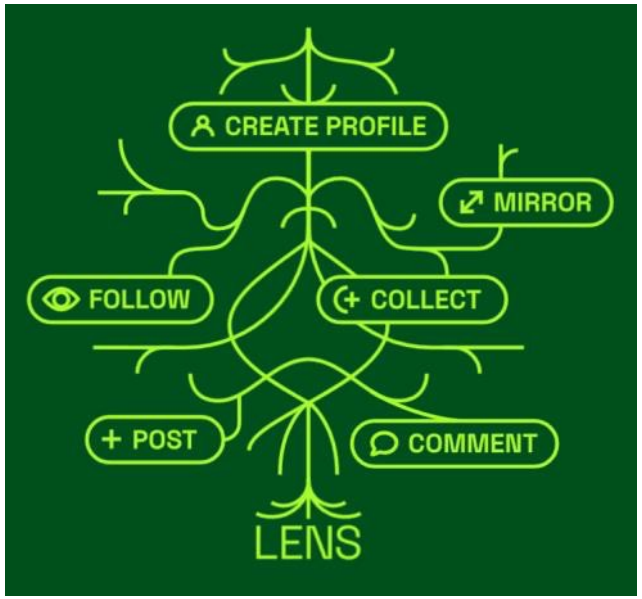
Built-in governance of Follow NFTs
to create a social DAO
(Decentralized Autonomous Organization)



4. Protocols as data utilization tools

Lens Protocol makes user's social graph data **a social graph protocol**
Applications utilize Lens Protocol to access user's social graph

b protocol을 기반으로 app 생성용이

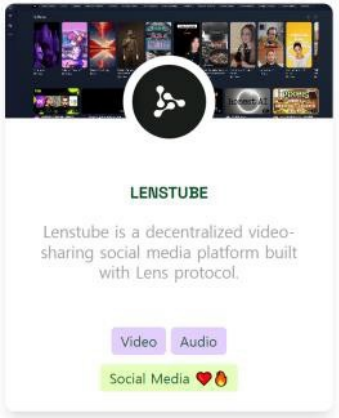


Lensverse

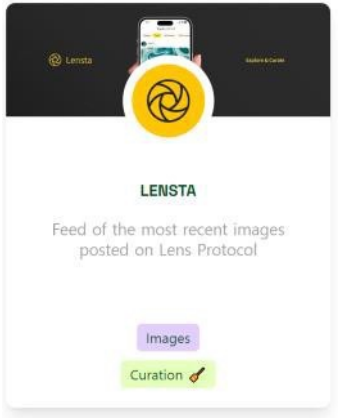
Hundreds of applications built on top of Lens Protocol

Web3 Twitter

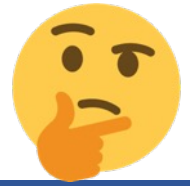
Web3 Youtube



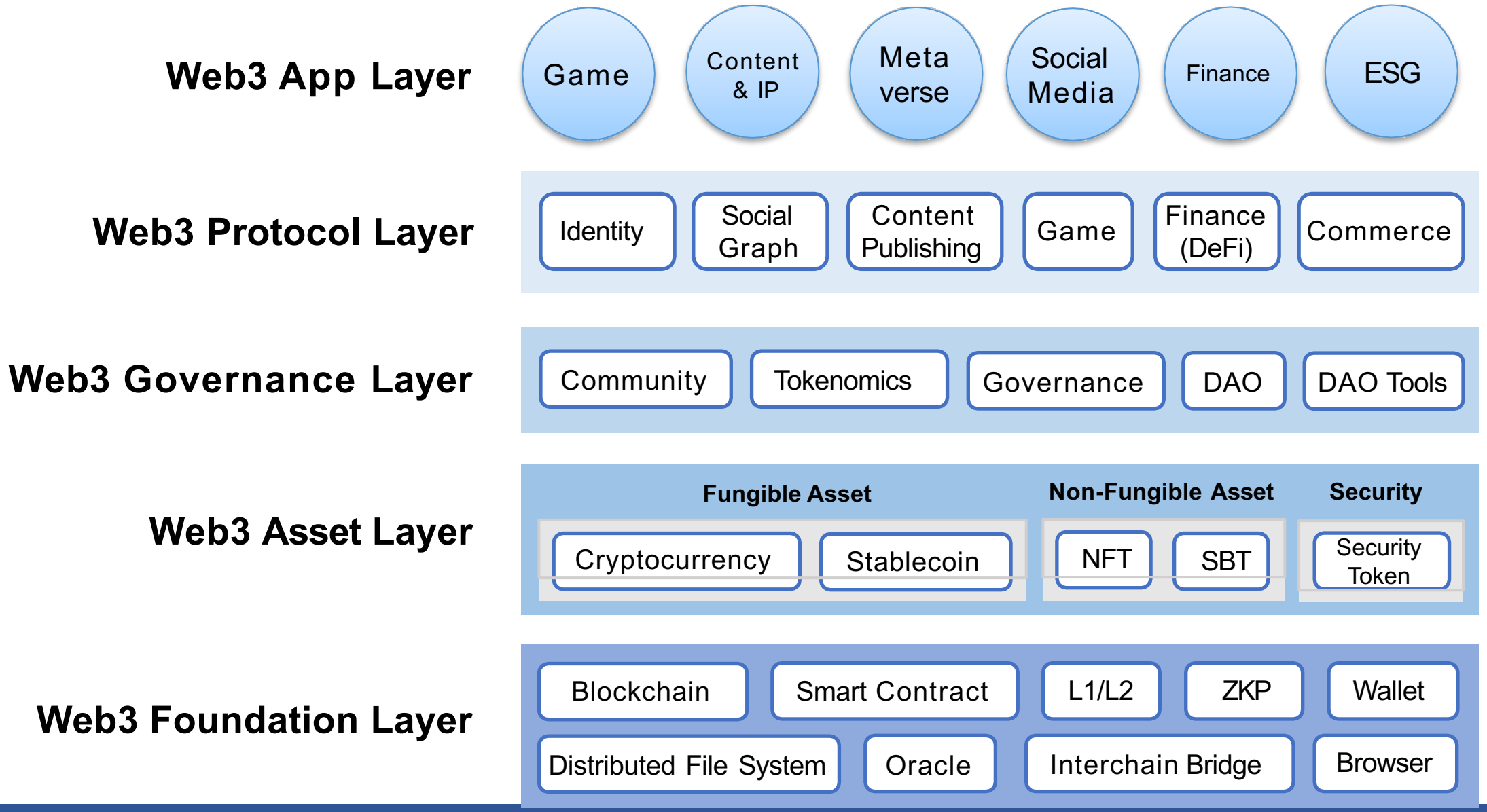
Web3 Instagram



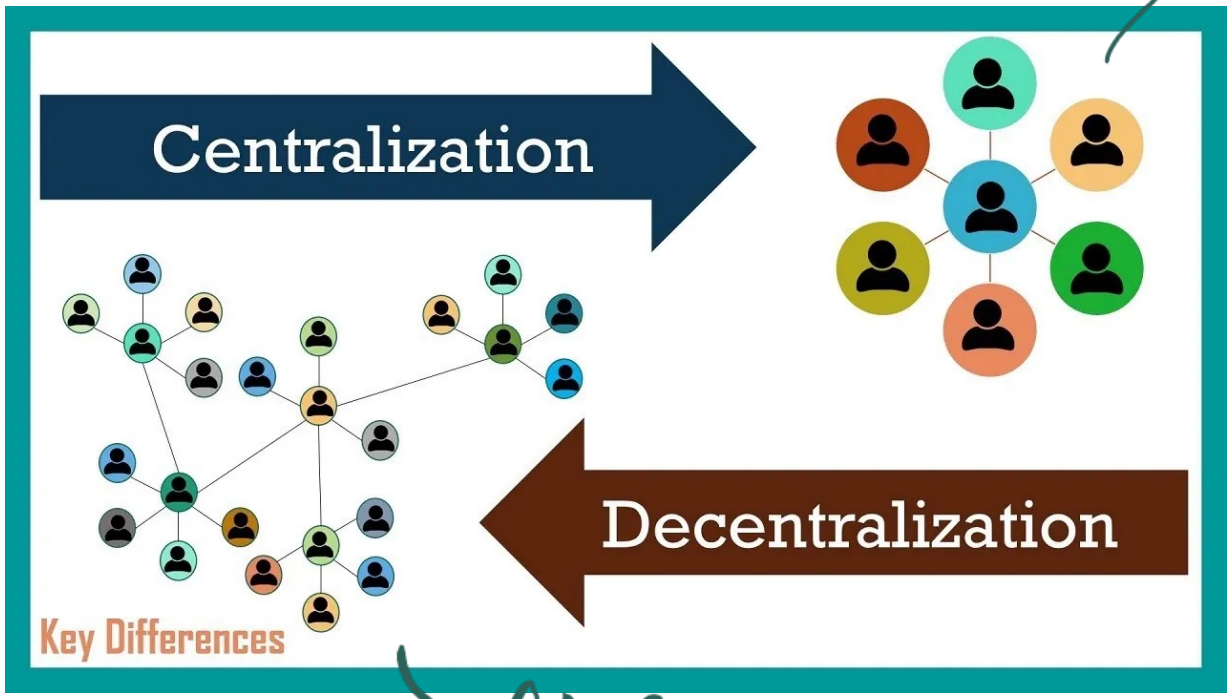
https://www.lens.xyz/apps



Web3 Stack (in the view of data)



Takeaways



router protocol
↓
속도↑,
중앙 통제

SNS → 탈중앙화,
검열 or 조작X
(데이터 게이트키퍼 귀속)

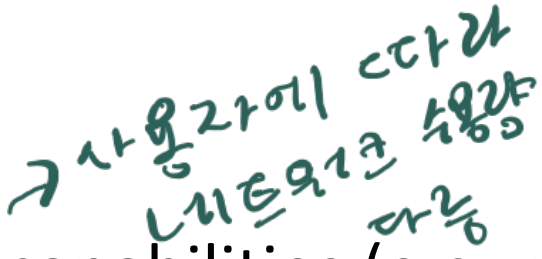
- Can you imagine the next Internet or Web?

Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- **video streaming and content distribution networks**
- socket programming with UDP and TCP

Video Streaming and CDNs: context

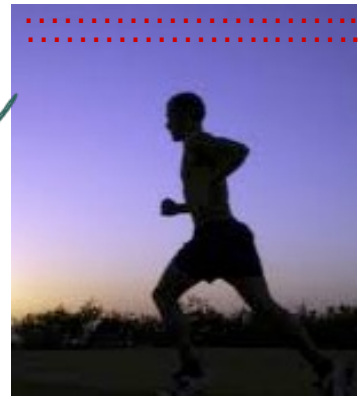
- stream video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic (2020)
- *challenge:* scale - how to reach ~1B users?
- *challenge:* heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure



Multimedia: video

- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) and *number of repeated values* (N)



frame *i*

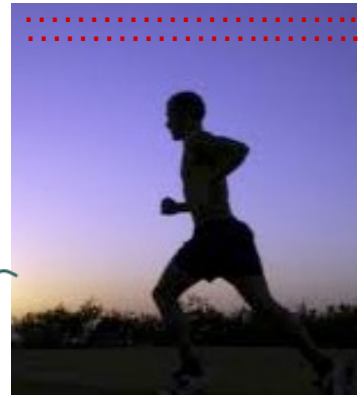
frame *i+1*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes → coding 변화량에 따라 encoding 속도 변화
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps → CBR
  - MPEG2 (DVD) 3-6 Mbps → VBR
  - MPEG4 (often used in Internet, 64Kbps – 12 Mbps) → VBR

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) and *number of repeated values* (N)



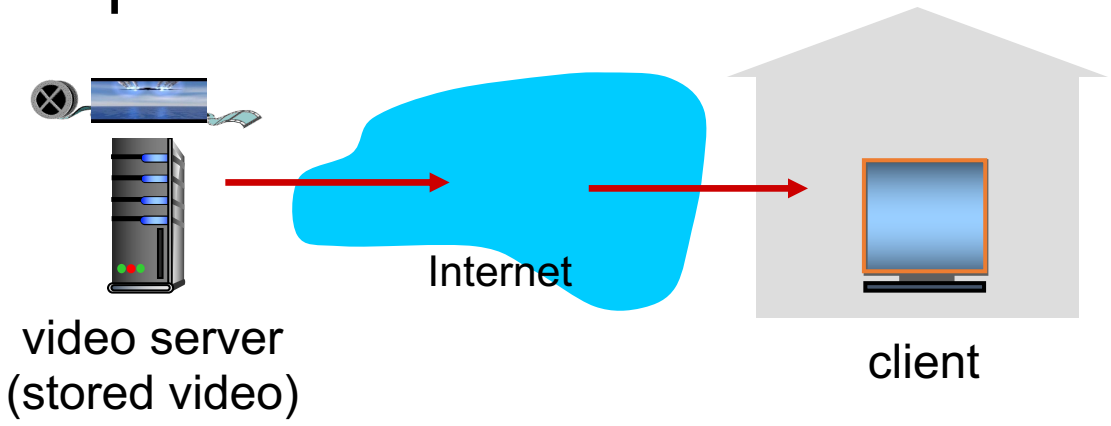
frame *i*

frame *i+1*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

Streaming stored video

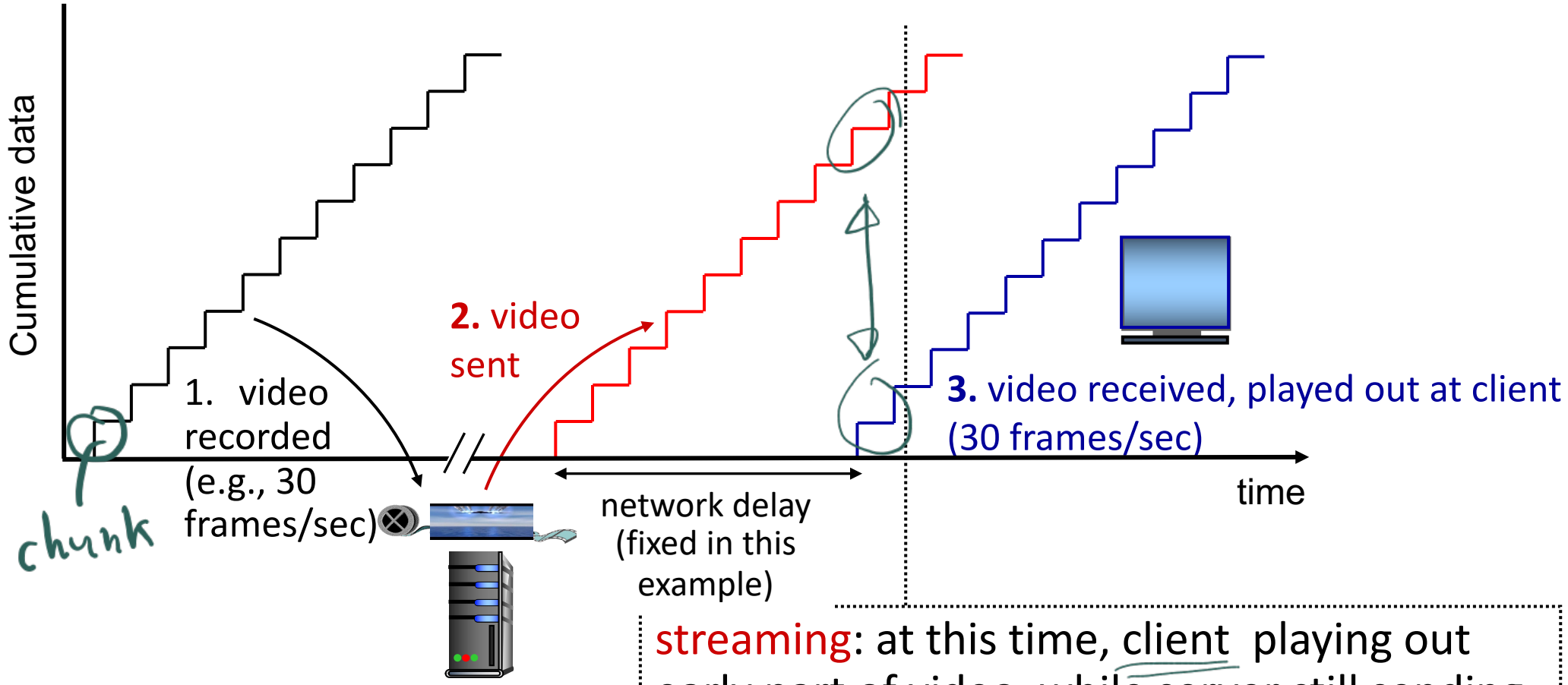
simple scenario:



Main challenges:

- server-to-client bandwidth will *vary* over time, with changing network congestion levels (in house, access network, network core, video server)
- packet loss, delay due to congestion will delay playout, or result in poor video quality

Streaming stored video



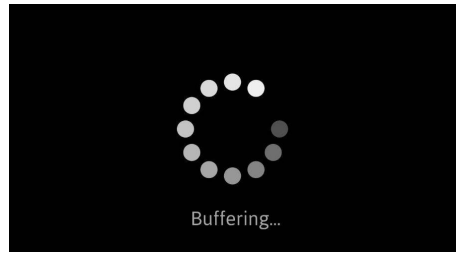
streaming: at this time, client playing out early part of video, while server still sending later part of video

network

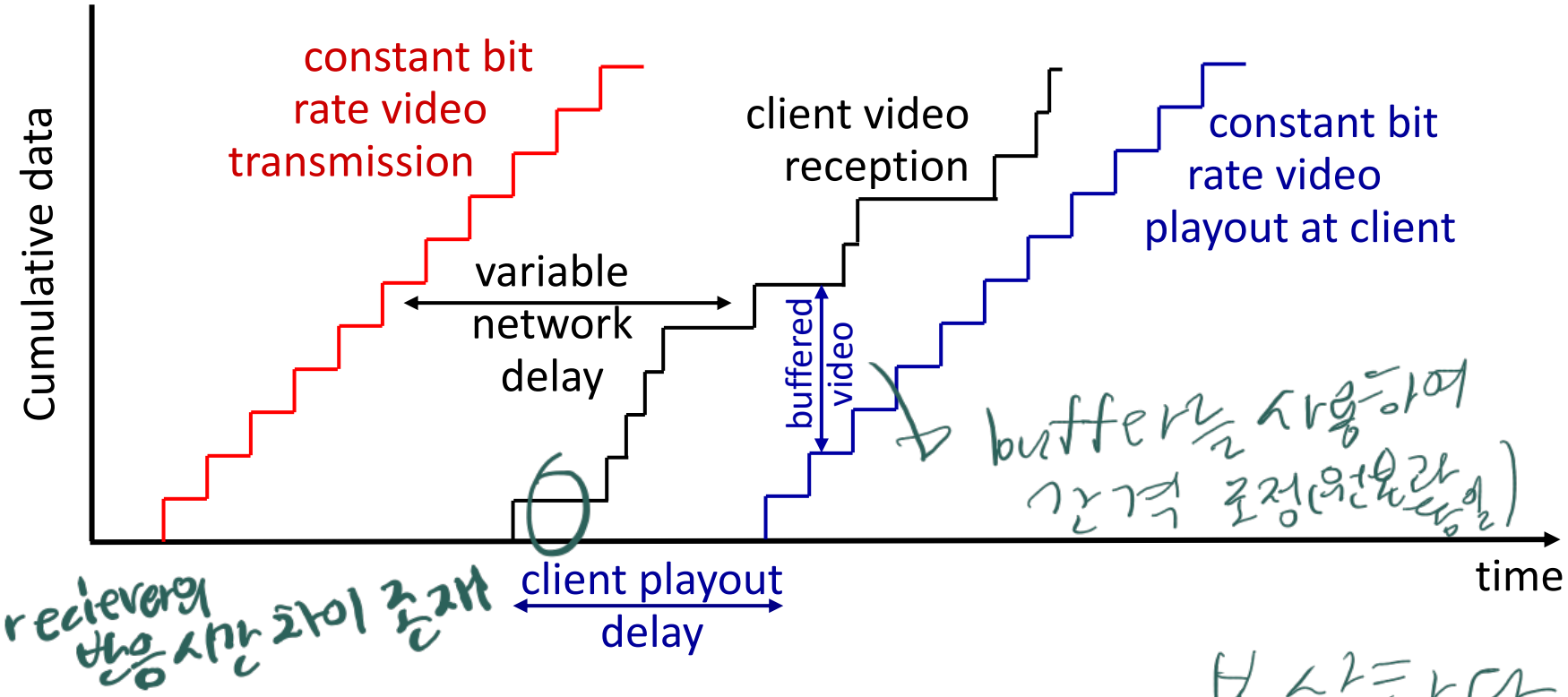
⇒ delay 로 인하여 실행 part 다름

Streaming stored video: challenges

- continuous playout constraint: during client video playout, playout timing must match original timing
  - … but network delays are variable (jitter), so will need client-side buffer to match continuous playout constraint
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video
  - video packets may be lost, retransmitted



Streaming stored video: playout buffering



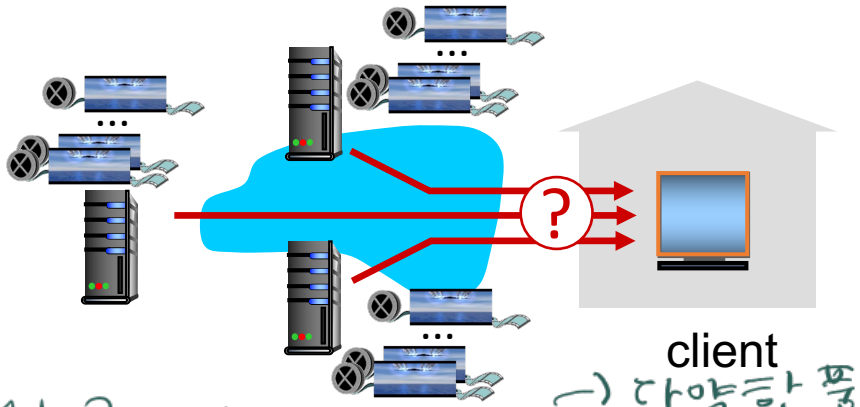
- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Streaming multimedia: DASH

*D*ynamic, *A*daptive
*S*treaming over *H*TTP

server:

- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files
- files replicated in various CDN nodes
- *manifest file:* provides URLs for different chunks



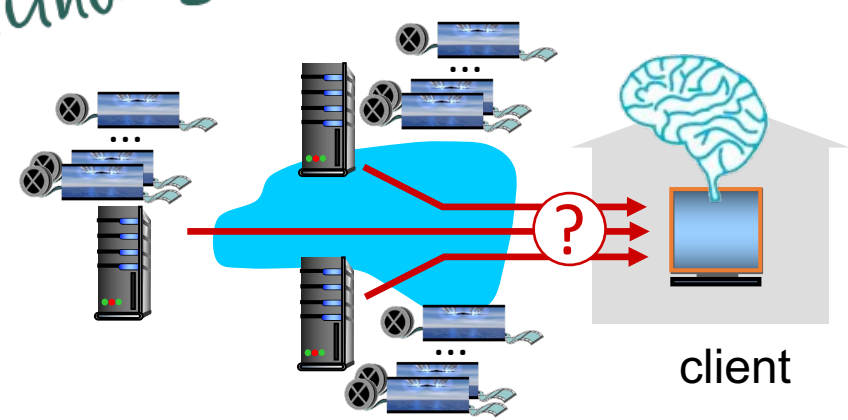
client:

- periodically estimates server-to-client bandwidth
- consulting manifest, requests one chunk at a time
  - chooses maximum coding rate sustainable given current bandwidth
  - can choose different coding rates at different points in time (depending on available bandwidth at time), and from different servers

Streaming multimedia: DASH

- *"intelligence"* at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)

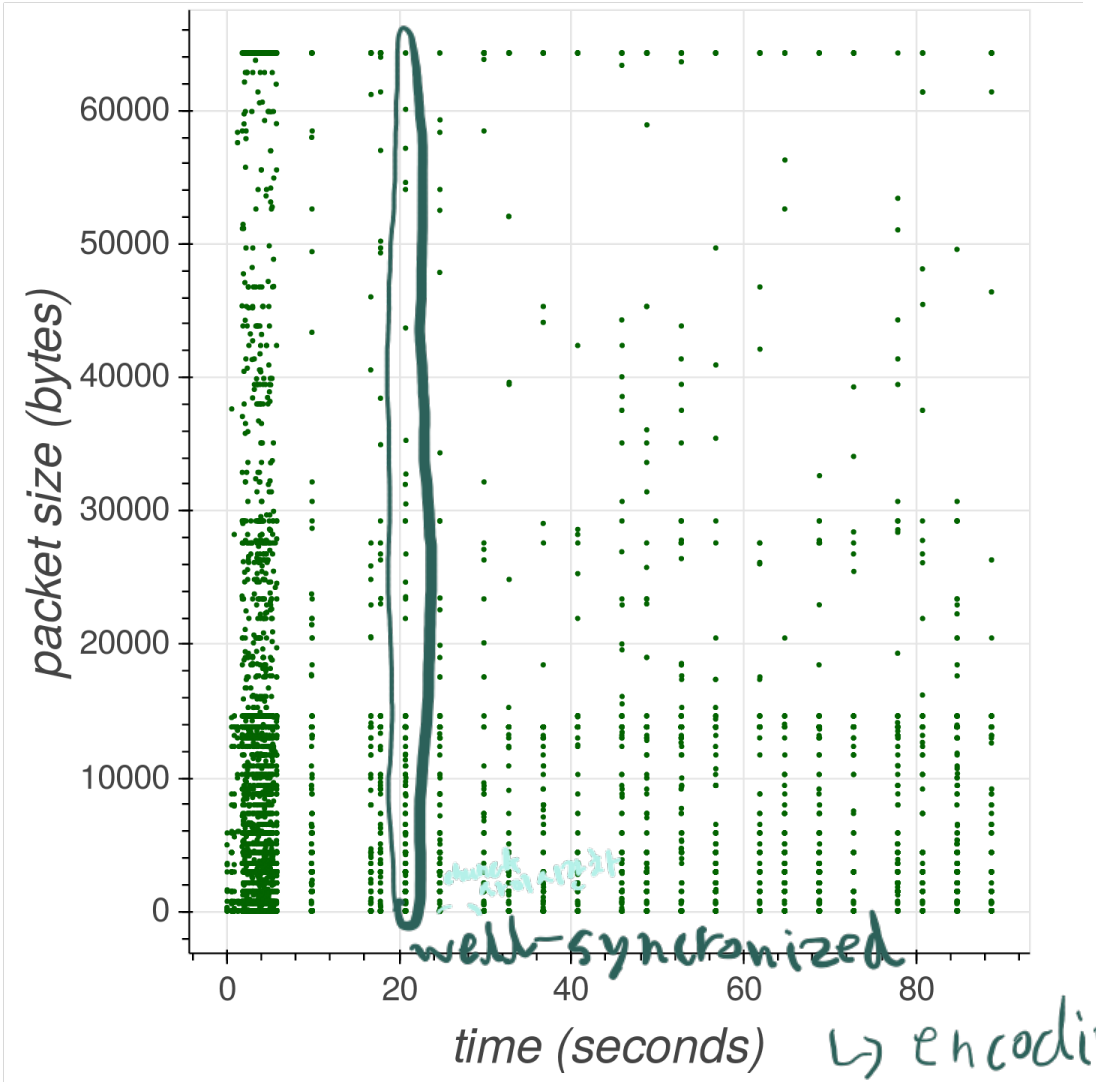
  Streaming video = encoding + DASH + playout buffering



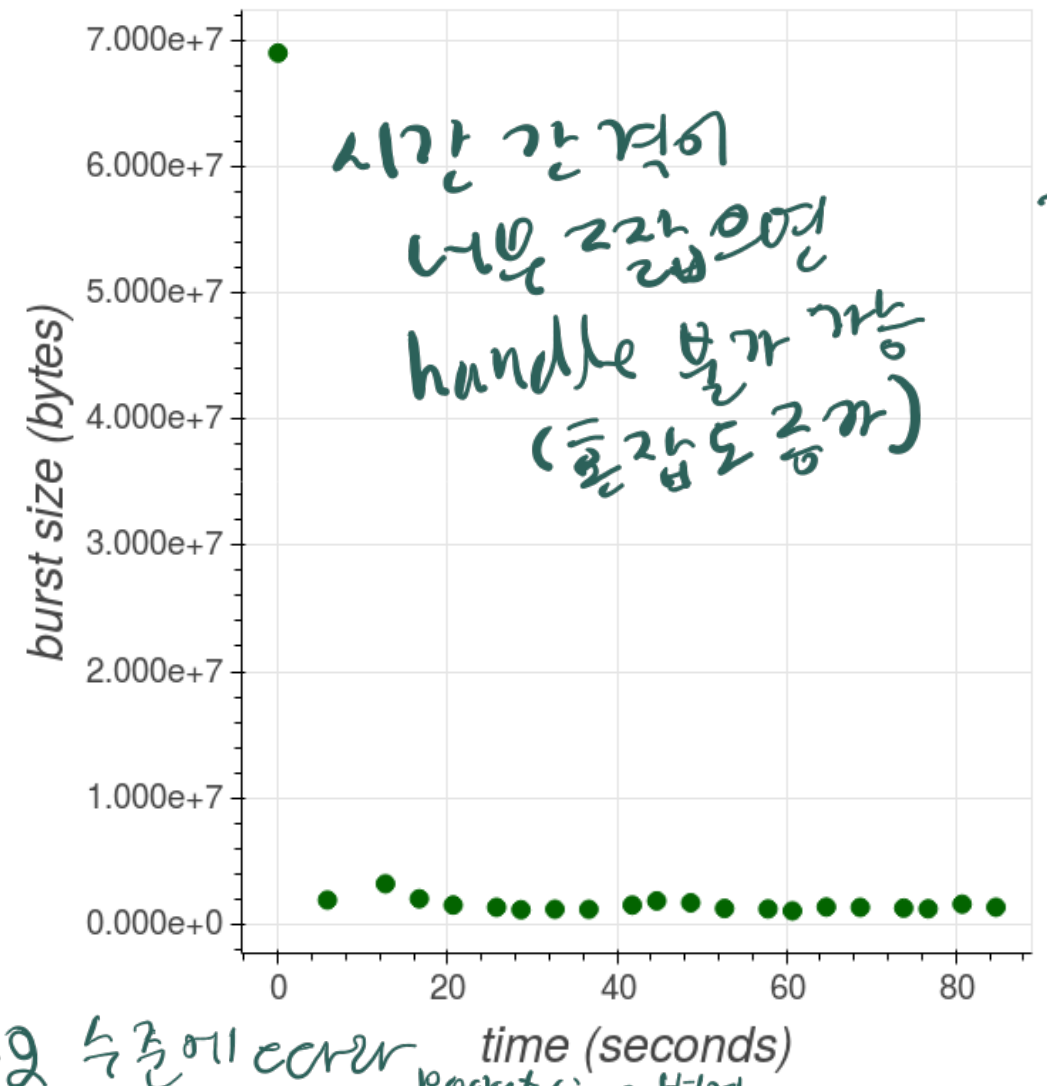
내용을 일련의 작은 크기의 HTTP 기반 파일 세그먼트들로 분리시킴으로써 동작하며, 각 세그먼트는 영화나 스포츠 이벤트 생방송 등 잠재적으로 수 시간에 걸친 내용물의 재생 시간의 짧은 간격(interval)을 포함하고 있다. 이 콘텐츠는 다양한 비트레이트로 이용이 가능하다.

콘텐츠가 MPEG-DASH 클라이언트에 의해 재생되면 클라이언트는 비트레이트 적응(ABR) 알고리즘을 사용하여 재생 시 멈춤이나 재버퍼링을 일으키지 않고 다운로드할 수 있도록 가능한 최고 비트레이트의 세그먼트를 자동으로 선별한다.

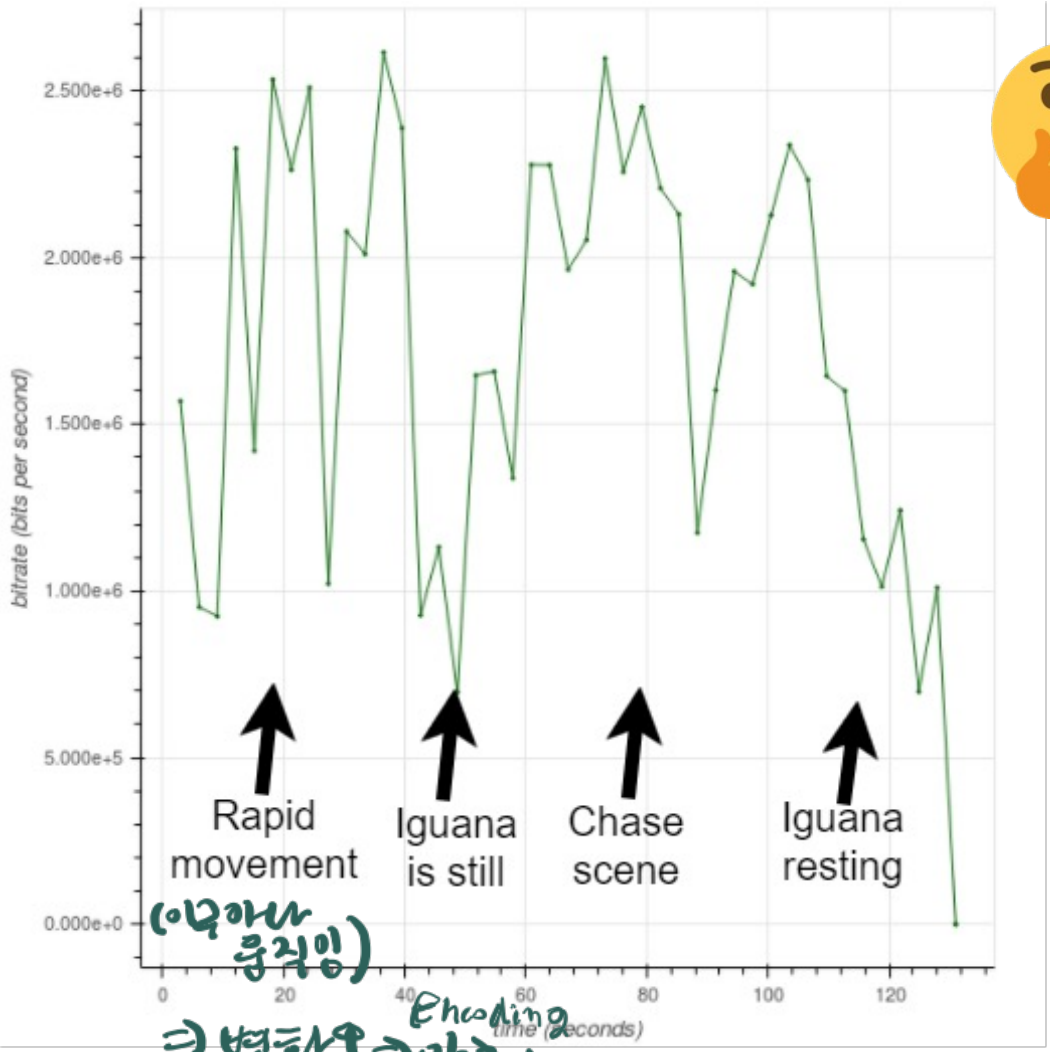
Privacy concerns in DASH



well-synchronized

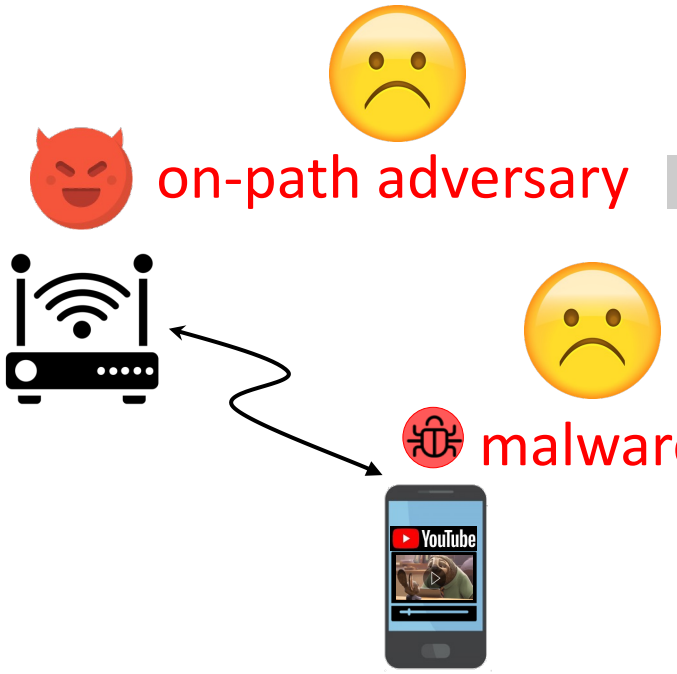


Privacy concerns in DASH

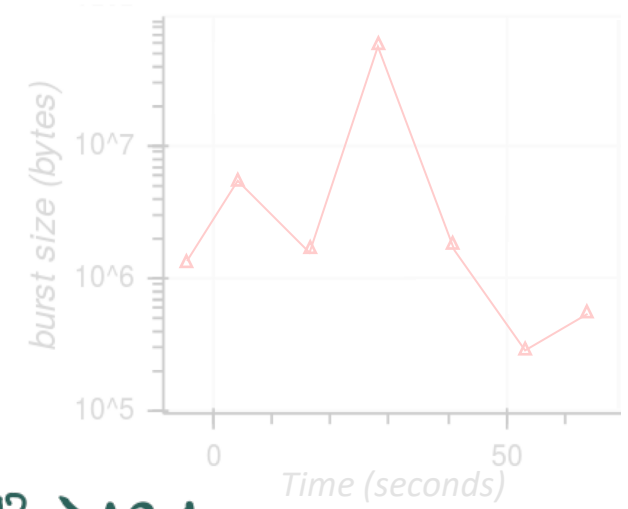


Infer Video Titles from ***Encrypted Traffic***

wifi-traffic 그래프를 통해 무엇을 보고 있는지 파악



traffic burst pattern
leaks video title
[USENIX Security 2017, CODASPY 2017]

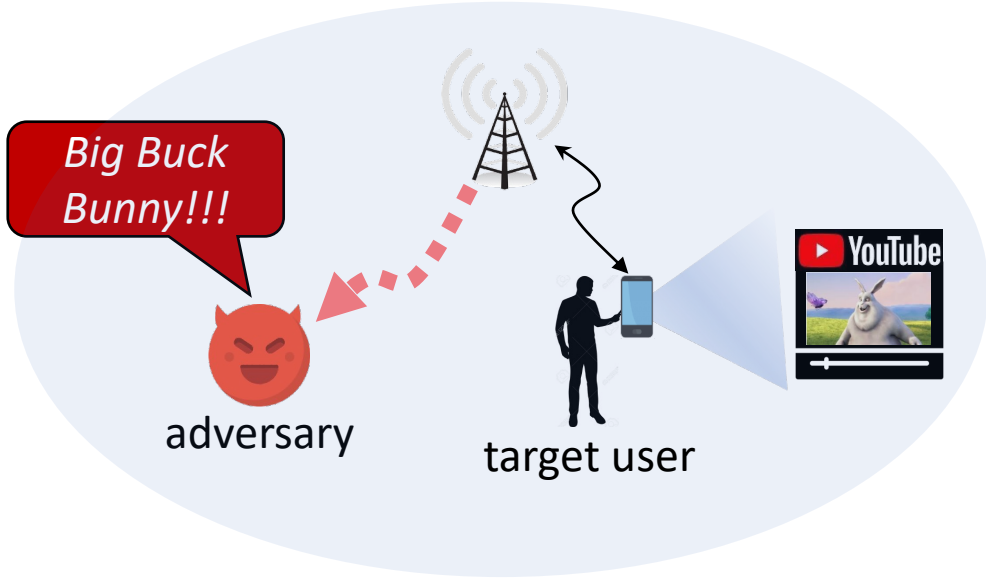


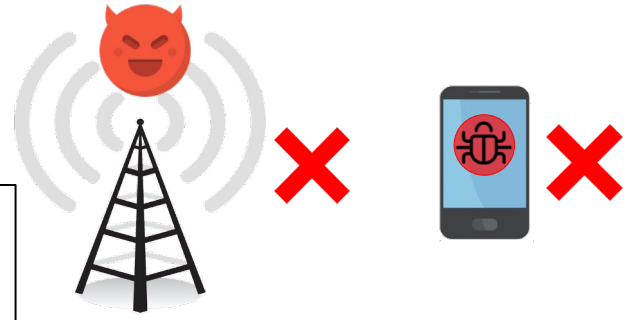
모든 비디오는 리소스 Unique pattern 존재

Moba: A Novel Video Inference Attack

('What are you watching?' in Korean)

- ***Identify YouTube videos*** watched by the target user
- ***Indirectly*** estimate the downlink traffic burst pattern
- No requirement for an ***on-path adversary*** or ***malware installation***





Nitya Lakshmanan, Abdelhak Bentaleb, Byoungjun Choi, Roger Zimmermann, Jun Han, and Min Suk Kang, "On Privacy Risks of Watching YouTube over Cellular Networks with Carrier Aggregation," In *Proceedings of ACM **IMWUT/UbiComp*** 2022.

Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

- *option 1:* single, large "mega-server"
  - single point of failure
  - point of network congestion
  - long (and possibly congested) path to distant clients

….quite simply: this solution *doesn't scale*

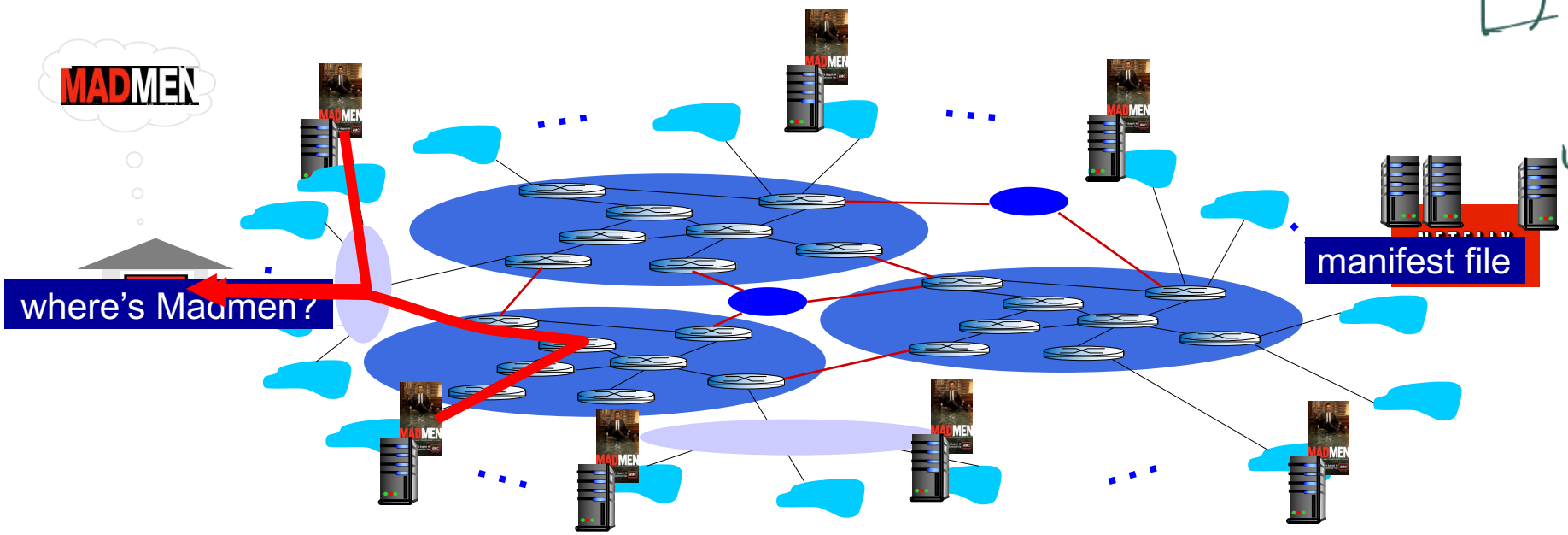
Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - Akamai: 240,000 servers deployed in > 120 countries (2015)
  - *bring home:* smaller number (10's) of larger clusters in POPs near access nets
    - used by Limelight

Content distribution networks (CDNs)

- CDN: stores copies of content (e.g. MADMEN) at CDN nodes
- subscriber requests content, service provider returns manifest
  - using manifest, client retrieves content at highest supportable rate
  - may choose different rate or copy if network path congested



Content distribution networks (CDNs)

Internet host-host communication as a service

OTT challenges: coping with a congested Internet from the "edge"

- what content to place in which CDN node?
- from which CDN node to retrieve content? At which rate?

경색되다.

Next…

- *Chapter 3.1 Transport-Layer Services*
- *Chapter 3.2 Multiplexing and Demultiplexing*
- *Chapter 3.3 Connectionless Transport: UDP*