HTLC-DASH:

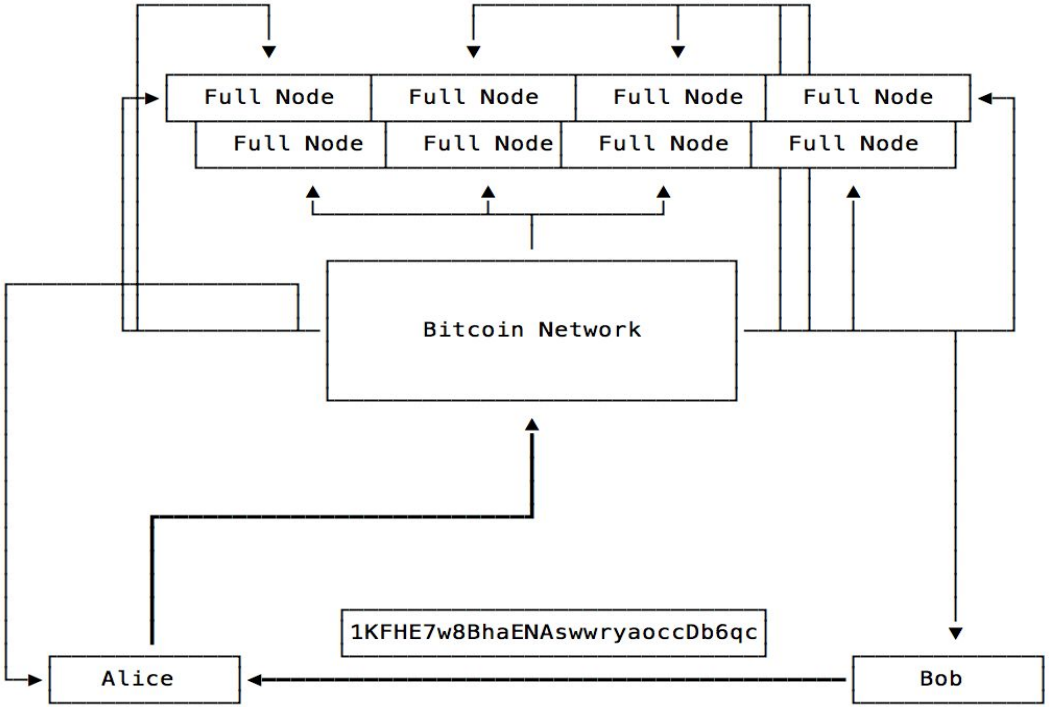
Micropayments for Decentralized Media Streaming

Laolu Osuntokun
roasbeef@lightning.engineering
@roasbeef
Lightning Labs

Building Decentralized Apps 09/20/2017

Bitcoin Payments Today

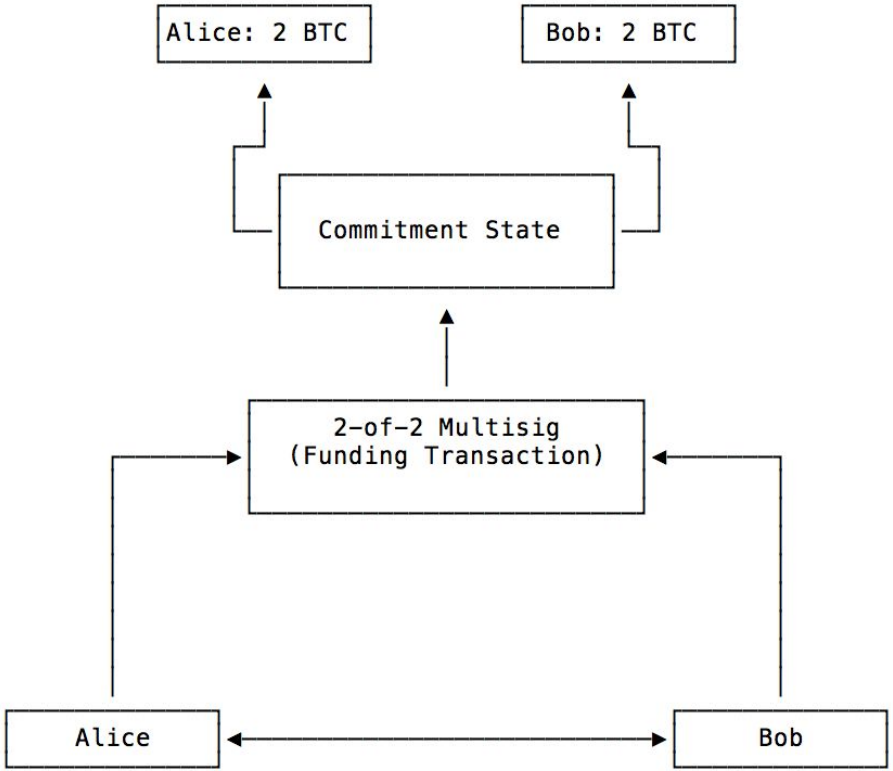
- All participants connected to **global** network
- **All** payments broadcast to **all other** participants
- **Each** payment **must** be fully verified
- Drawbacks:
  - Scalability limitations of **global broadcast** network
  - Each node does work even if not involved in payment
  - **Public** record of **each** payment kept for **ALL TIME**



Payments With Lightning

- Enter Lightning: **Off-Chain** Bitcoin payments

- Alice and Bob enter into a **contract**
- Contract creation:
  - Funds put into 2-of-2 multi-sig
  - **Before broadcast** transaction to *deliver* funds is signed
    - Requires malleability fix
  - Funding transaction broadcast
- Off-chain payments (sub-contract):
  - **HTLC**: Hash-Time-Lock-Contract
- Contract completion:
  - Closing transaction broadcast, final balance delivered
- On-chain footprint:
  - **2** transactions
  - All updates **point-to-point**
  - **Predictable** fees



The Holy Hash-Time-Locked Contract

- **Specific** implementation of generic: **“claim-or-refund”** functionality
  - **Conditional payment** upon reveal of **witness**
- The Hash-Time-Locked-Contract (HTLC)
  - Set up: **receiver** gives **sender** `H = Hash(R), where R <-$ {0, 1}^n`
  - Conditions: “I will pay you N BTC, iff you present R s.t `Hash(R)== H`”
  - Escape hatches: “If you don’t within T days, I get my money back”
- Enables **end-to-end** secure **multi-hop** payments through **untrusted intermediaries!**
  - Able to **connect** payment channels (tubes of money)
- Payment from **Alice** to **Dave** via **existing** channels
  - **A -> B -> C -> D** (Clear: balances get **decremented**, funds in **limbo**)
  - **A <- B <- C <- D** (Settle: balances get **incremented**, forwarders **credited**)

lnd- The Lightning Network Daemon

- One of many in-progress Lightning implementation:
  - Code (for `lnd`): https://github.com/lightningnetwork/lnd/
  - Spec: https://github.com/lightningnetwork/lightning-rfc/
  - Uses the **btcsuite (**a.k.a btcd**)** set of Bitcoin libraries
- Developed by **Lightning Labs**
  - Lead developer: **roasbeef** (the speaker!)
- Latest release: v0.3-alpha
  - **Feature** complete LN implementation
    - Able to: manage all channel states, passively forward, validate graph, onion payments, etc.
  - Release features:
    - **Macaroon** based authentication
    - **--autopilot** mode (!!)
    - Light client mode (**neutrino)**
    - **Spec compliance**

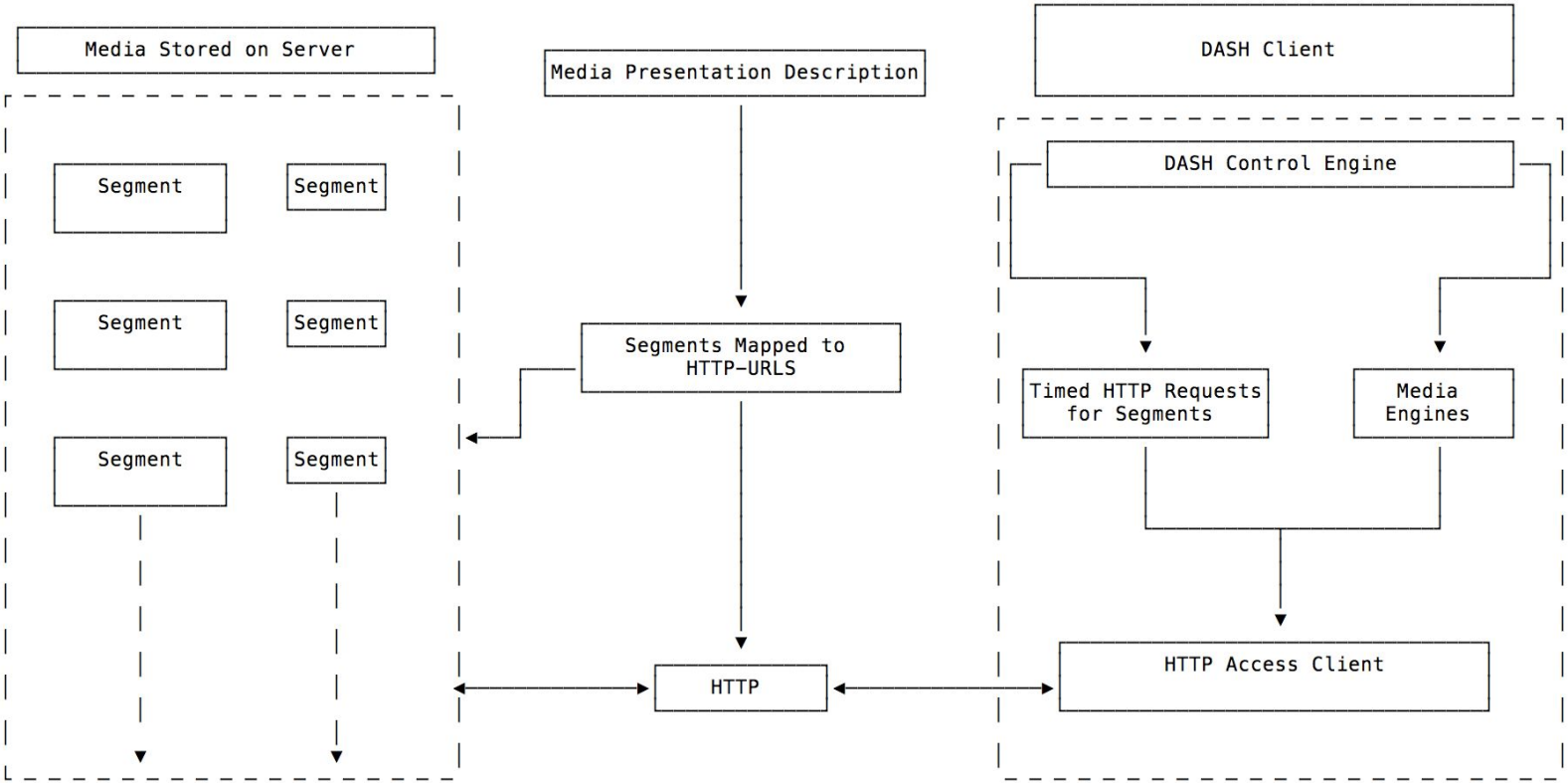
Lightning as an Application Platform

- API of **Layer-2** much **simpler** the raw base layer:
  - **OpenChannel(**nodeKey, amt, pushAmt**) -> Some(**chanPoint**)**
  - **CloseChannel(**chanPoint, *force=false*) **-> bool**
    where **chanPoint** = **txid:index**
  - **Pay(**payReq, *dest=nodeKey, hash=payHash, amount=n*) **-> Some(**route)
    Where **route** details path through network, total fees, etc
  - **Req(**amt, memo, fallBackAddr**) -> payReq**
- Developer Resources:
  - Overview, tutorials, example applications: http://dev.lightning.community/
  - Comprehensive documentation of `lnd`'s gRPC interface: http://api.lightning.community/
- Application platform for **intelligent agents**
  - **Machine-to-machine** payments
  - Micropayment **preference agents**
  - **Channel liquidity** optimizers
  - **Forwarding** profit **optimizers**

MPEG-DASH: Overview

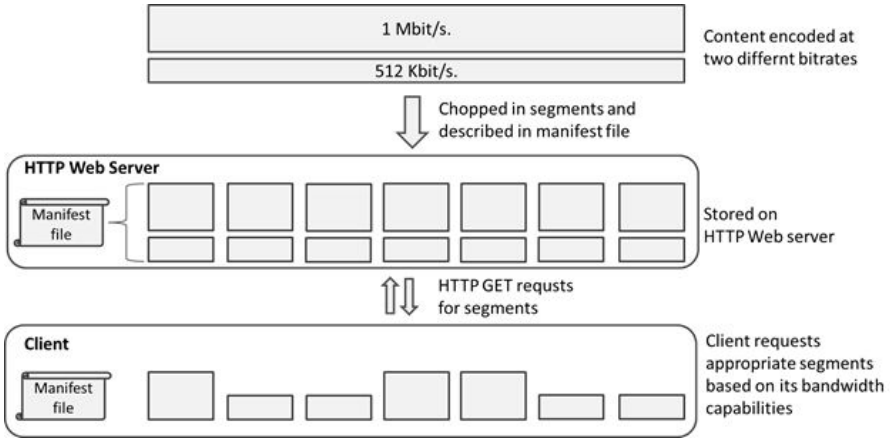
- Industry standard for **adaptive bitrate streaming:**
  - **DASH**: Dynamic Adaptive **Streaming** over **HTTP**
  - HTTP used as **transport layer** for meta-data + streaming chunks
  - Widely used for **pre-stored** and **live** streaming
- Core components:
  - Segment:
    - Encoded time-slice of media (**2s - 10s**)
  - Media Presentation Description (MPD)
    - Describes timeline of **segments** at various **bit-rates**
  - Dash Client
    - DASH **Control Engine**
    - **Samples bandwidth**, figures out **which** segments to play
  - Media HTTP Server
    - **Serves** the MPD
    - **Stores** pre-encoded **segments** or generates in real-time (live streaming)

MPEG-DASH: Overview



MPEG-DASH: Segments

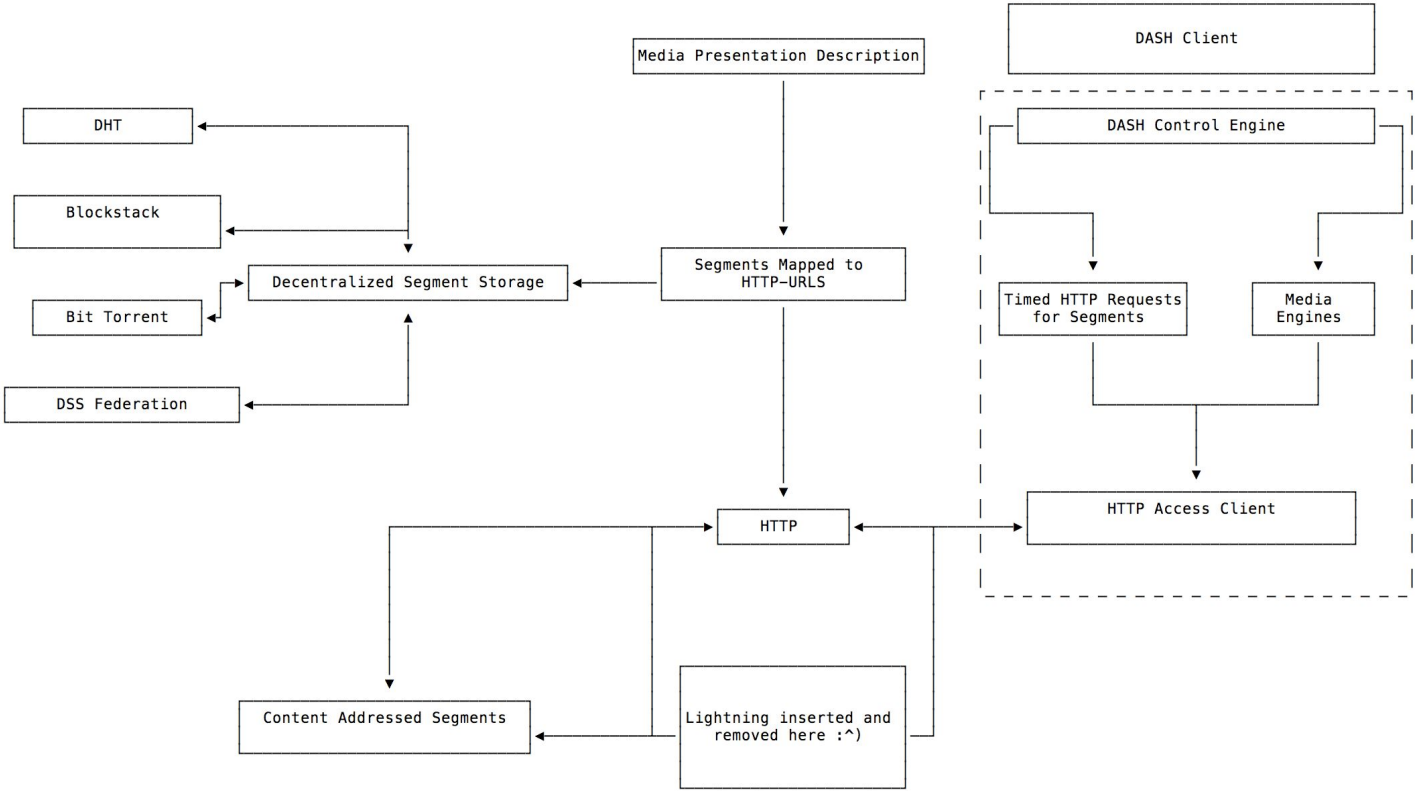
- Typically from **2s** to **10s**
  - **Lower** segment size:
    - +High switching granularity
    - +**Suited for live**
    - -Large number of files
  - **Higher** segment size:
    - +Small number of files
    - +**More cacheable**
    - -Not well suited for live
- Workflow:
  - Client fetches segments to ensure **adequate buffer**
  - If **bandwidth changes**: request higher/lower quality to **compensate**
  - How **YouTube** is able to scale quality based **connection quality**



HTLC-DASH: Overview

- Lightning + MPEG-DASH + Decentralized Segment Storage = 💗
  - Create a **decentralized marketplace** for **media streaming**!
  - **Replaces adverts** when watching videos online!!!
- Lightning + MPEG-DASH
  - Extend the **MPD** to include **pricing information**
    - **Larger** segment (higher quality) -> **higher** price
  - **HTTP** server responds w/ error code **402** (payment required) if segment not paid for
  - **DASH Client** able to factor in **user preferences** when fetching segments
- MPEG-DASH + DSS:
  - **MPD** Server need not have all **data locally**
  - Instead can fetch from independent sources in decentralized manner
  - Incentives to **cache popular content** locally, possible proxy-fetching and re-server
  - **Agent** in decentralized **marketplace** for media

HTLC-DASH: Overview



HTLC-DASH: Payment Required

- Enforce payment w/ **proxy** in-front of **DASH Client** + **Lightning Server**
  - Served **MPD** now has s**atoshi-granularity** pricing for segments
    - Proxy intercepts, generates mapping
  - Server grants client **anonymous credential** to be used as **salt**
  - HTTP **402 Payment Required** returned by default
  - **Lightning Proxy** will intercept requests and proxy responses
    - Uses **Lightning** to **pay** server for fragment!
- **Strawman** Approach
  - Payment for segment **exchange dilemma**
    - Who goes first? Pay then send, or send then pay?
- **Atomic Exchange** w/ HTLC's
  - Enforce **atomic exchange** via HTLCs!
  - **Payment** hash is **segment** content **hash**:
    - Claim of payment **reveals** segment

HTLC-DASH: Storage Backend

- DSS Server doesn't **need** all content **locally**
  - Able to dynamically fetch from **independent** sources
  - Possibly will **pay upstream** DSS, **resell to down stream clients**
- HTLC-DASH servers participate in **incentivized** media serving **marketplace**
  - **Availability, reliability, diversity** of content make providers more attractive
  - Servers become **aggregators** of **desirable content**
- Possible sources:
  - **Blockstack**
  - DHT
  - Bittorrent (**littorrent**)
    - Re-use merkle-tree of infohash content in HTLC's
    - New v**ersion of Bittorent** switching to **SHA-256** (due to SHA-1 break)
  - DDS **Federation**
    - Dynamically share content amongst each other

HTLC-DASH: Alternative Advanced Implementations

- 1:1 **segment** hash to **payment** hash mapping not feasible
  - Due to limitations in Bitcoin Script segment **size** may be too **large**
    - **5s** segment of **10Mbps** -> **5MB**
  - Solution?
    - Merkle Trees!
    - Add ability to **validate merkle tree branches** to Script
    - Commitment txn has 100's of HTLC's, settled in parallel
- Exchange **Non-Interactive Zero Knowledge Proof** of Segment Knowledge
  - NIZKPoSK (kek), server proves to client that has segment:
    - Enables client to not enter into contract unless server can deliver
    - Eliminates nuisance server attack (funds in limbo, but it's satoshis!)
  - **ZKBoo**
    - "MPC-in-the-head" based NIZKP
    - Proof generated in **ms** (for our case)
    - Amenable to **real-time** ZKCP's (zero knowledge contingent payments)

HTLC-DASH: Challenges

- **Latency**, latency, latency
  - DASH client needs to be able to **maintain sufficient video buffer**
    - Otherwise, video pauses, annoyed user
  - Decentralized fetching may introduce prohibitive **latencies**
    - Client can prefer providers with **lower latency**, providers use **IP Anycast**
  - Client can be more aggressive in pre-fetching further in stream
- **Update speed** on Lightning side
  - Lightning Commitment Protocol (**LCP**) optimized for **pipelined, batched** updates
  - **1.5 RTT’**s required for full update with current bi-di channels
    - **1000’s of HTLC’s** able to settled+cleared in a **single update**
  - Can transition to uni-directional channels, for lower latency updates
    - Only requires **0.5 RTT for a payment**, may require reset if imbalanced

Extending HTLC-DASH Clients w/ Intelligent Agents

- HTLC-DASH client can factor in user **preferences:**
  - Able to factor in **attributes** to make more intelligent **fetching decisions**
  - Can surveil market in real-time to locate best price
    - Also will need to factor in latency to media server, etc
- Examples:
  - Give agent **budget** of X BTC, able to **throttle segment quality** based on budget
  - **Dynamically scale up/down bit-rate** based on content (dark scenes don't need high bitrate!)
  - **Podcast** requires lower bitrate compared to **music** (hi-fi, FLAC, etc)
- Agent able to factor in relevant **attributes** and user **preferences**
  - Reduces mental txn costs for micropayment systems
  - Micropayment and Mental Transaction Costs by Nick Szabo (MUST READ!)

Lightning Labs is Hiring!

Think HTLC-DASH is cool?

- 👀 out for the code!

https://angel.co/lightning/

Looking for:

- Crypto Protocol Engineer
- Frontend Engineer

Contact: roasbeef@lightning.engineering