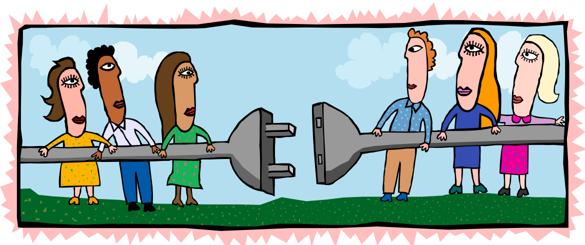
The PPSP Peer Protocol (PPSPP)

Arno Bakker, Victor Grishchenko, Riccardo Petrocco,
Johan Pouwelse

P2P-Next / Delft University of Technology

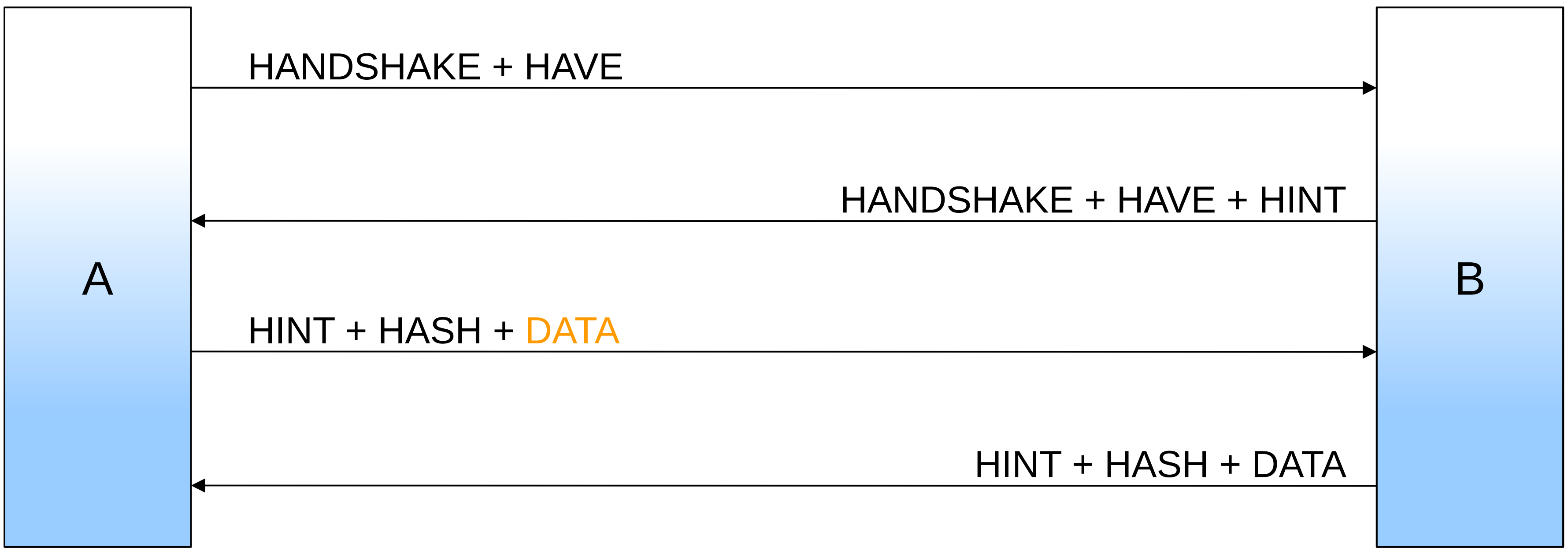
Refresh: PPSPP messages

- Basic unit of communication: Message
  - HANDSHAKE
  - HAVE: convey chunk availability
  - HINT: request chunks
  - DATA: actual chunk
  - HASH: MDCs to enable integrity verification
  - …
- Messages are multiplexed together when sent over the wire.



Example PPSPP on the wire

- Peer A and B both have some chunks



- Note: low latency, data transfer already in 3rd datagram.

PPSPP in detail

- Common set of messages across transports (UDP, RTP, TCP)
- Novel method of content integrity protection:
  - Merkle hash trees
- Novel method of chunk addressing:
  - Bins
  - = Address range of chunks with single integer

WG Item Status

- Identified 34 issues in post-Taipei discussion
- 10 simple textual ones resolved in -01
- Posted proposals for:
  - 10+13: Multiple content integrity and chunk addressing schemes
  - 26: Security of the handshake procedure
  - 17+20: Definition and security of Peer-Address Exchange (PEX)
- Identified new open issues from Requirements doc
- Posted security analysis for PPSPP messages
- 2 people in total responded on 1 proposal

Proposal 10+13

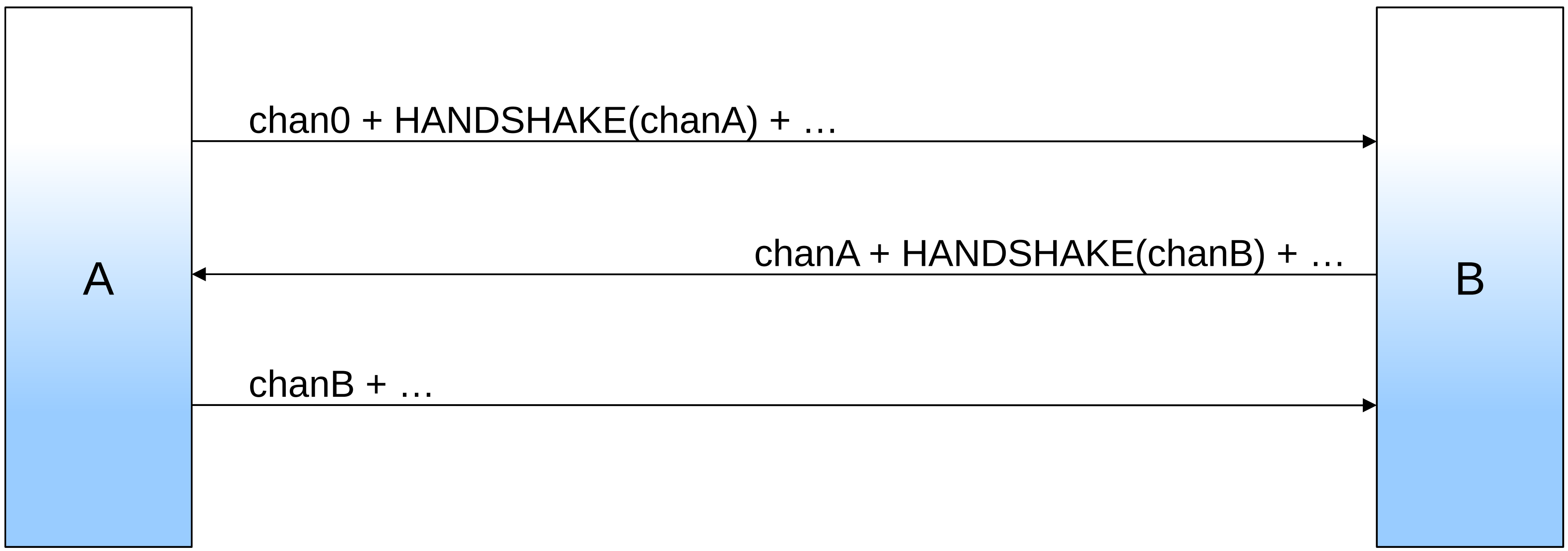
- “Multiple content integrity and chunk addressing schemes”
- Chunk addressing:
  - Scheme is extra metadata with swarm ID.
  - HINT+HAVE+… carry opaque “chunk spec”.
  - PPSPP SHOULD implement bin numbering.

- Integrity protection:
  - Scheme is extra metadata with swarm ID.
  - Or: Sender describes content integrity protection scheme in HANDSHAKE. Validity clear on first DATA message.
  - HASH message renamed to generic INTEGRITY.
  - PPSPP SHOULD implement Merkle Hash trees.

Proposal 26

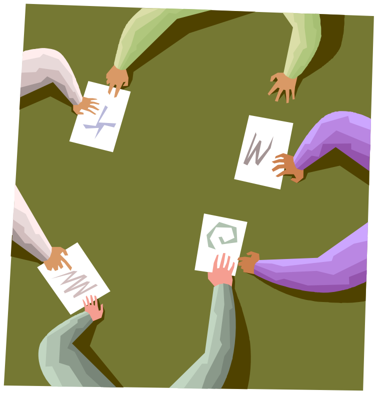
- “Security of the handshake procedure”
- Attacks:
  - DoS amplification: PPSPP peer amplifies traffic
  - DoS flood: state buildup at PPSPP peer
- Existing mechanism suffices
  - Clarify: no updates to unacknowledged peer.
  - Add: peer must reply immediately to HANDSHAKE, short timeout on state.
- Or: Copy RFC5971
  - No state till return routability check.
  - Adds latency.

PPSPP handshake procedure



Proposal 17+20

- “Definition and security of Peer-Address Exchange (PEX)”
- Rewrite definition:
  - PEX MUST contain addresses you exchanged messages with in the last 60 seconds.
- Security attacks:
  - Amplification: peer T causes peer A to connect to B1…n
  - Eclipse 1: Isolate single injector in live streaming
  - Eclipse 2: Isolate specific consumer peer



Protection against PEX Amplification attack

- Introduce membership certificates:
  - "peer A at address ipA+portA part of swarm S at time T"
  - Digitally signed
- Usage:
  - A sends cert to peer B during/after handshake.
  - B checks if sig OK, swarm OK and liveliness OK.
  - B puts cert in PEX reply to others.

- Different certification schemes:
  - Generic CA: hands out basic certificates, peer creates membership certs (CA -> basic -> membership trust chain)
  - Tracker as CA: creates membership cert on/after JOIN.

Protection against PEX Eclipse attacks

- Assumption: tracker returns a true random sample of the actual swarm membership.
- Live injector protected by:
  - Initiate percentage of connections itself
  - Disabling PEX
  - Or: PEX, but get percentage of peers from trusted tracker
- Protect consumer peer in same way:
  - Go to tracker if bad service

- Alternative PEX protection: PuppetCast
  - Set of peers in PEX reply externally controlled.

New Issues from PPSP Requirements

- REQ-8: QoS
  - More support needed? New issue #35
- PP.REQ-3: Get peers from peer
  - Satisfied by PEX
- PP.REQ-6: Peer status reporting
  - New issue #36
- SEC.REQ-1: Closed Swarms
  - New issue #37, propose P2P-Next solution
- SEC.REQ-2: Content confidentiality
  - Supported, add text (new issue#38)

New Issues from PPSP Requirements (cont'd)

- SEC.REQ-3: Encrypt peer links.
  - IPsec or DTLS, add text (new issue #39)
- SEC.REQ-4: Limit bad peer damage
  - Most attacks covered, will discuss (new issue #40)
- SEC.REQ-5: Exclude bad peers
  - Via content integrity protection, add text (new issue #41)
- SEC.REQ-6: Bad peers exhaust resources
  - Need PEX protection
  - Limit upload per peer
  - (Handshake procedure protects)
  - Add text (new issue #42)

New Issues from PPSP Requirements (cont'd)

- SEC.REQ-7: Decentralized tracking
  - Need PEX protection == issue #20
- SEC.REQ-9: Content integrity
  - Covered, add ref to Chung Kei Wong and Simon S. Lam for live (new issue #43)

Threat Analysis: HANDSHAKE

- Secured against DoS amplification attacks as proposed in mail dd. Jan 25th.
- Threat 1.1: Eclipse attack where peers T1..TN fill all connection slots of A by initiating the connection to A.
  - Solution: Don't accept all incoming connections, initiate e.g. 50% yourself (see also SEC.REQ-6 discussion).

Threat Analysis: HAVE

- Threat 2.1: Malicious peer T can claim to have content which it hasn't. Subsequently T won't correspond to requests.
  - Solution: peer A will consider T to be a slow peer and not ask it again.
- Threat 2.2: Malicious peer T can claim not to have content. Hence it won't contribute.
  - Solution: Peer+chunk selection policies external to the protocol will implement fairness and provide sharing incentives. Perhaps we should add CHOKE/UNCHOKE messages (Issue #4) as an extra mechanism for these policies to use.

Threat Analysis: ACK

- Threat 3.1: peer T acks wrong chunks.
  - Solution: peer A will detect inconsistencies with what it sent.
- Threat 3.2: peer T modifies timestamp in ACK to peer A used for time-based congestion control.
  - Solution: TODO. Could peer T use it to fake there is no congestion when in fact there is, causing A to send more data than it should?

Threat Analysis: DATA

- Threat 4.1: peer T sending bogus chunks.
  - Solution: The content integrity protection scheme defends against this.
- Threat 4.2: peer T sends peer A unrequested chunks.
  - To protect against this threat we would need network-level DoS prevention.

Threat Analysis: HASH

- Threat 5.1: Amplifcation attack: peer T sends HASHes, peer A checks hashes, spending CPU.
  - Solution: If the hashes don't check out A will stop asking T because of the atomic datagram principle and the content integrity protection.

Threat Analysis: HINT

- Threat 6.1: peer T could request lots from A, leaving A without resources for others.
  - Solution: Limit upload bandwidth per peer (see also SEC.REQ-6 discussion).

Threat Analysis: PEX_RES

- See above (mail dd. Feb 14th)

Threat Analysis: Unsollicited requests

- Threat: peer T could send a spoofed PEX_REQ or HINT from peer B to peer A, causing A to send a PEX_RES/DATA to B.
  - Solution: the message from peer T won't be accepted unless T does a handshake first (see mail dd. Jan 25th.), in which case the reply goes to T, not victim B.

Summary

- No show stoppers!
- Need more feedback!

PPSPP Implementation

Arno Bakker

Riccardo Petrocco

Richard Marsh

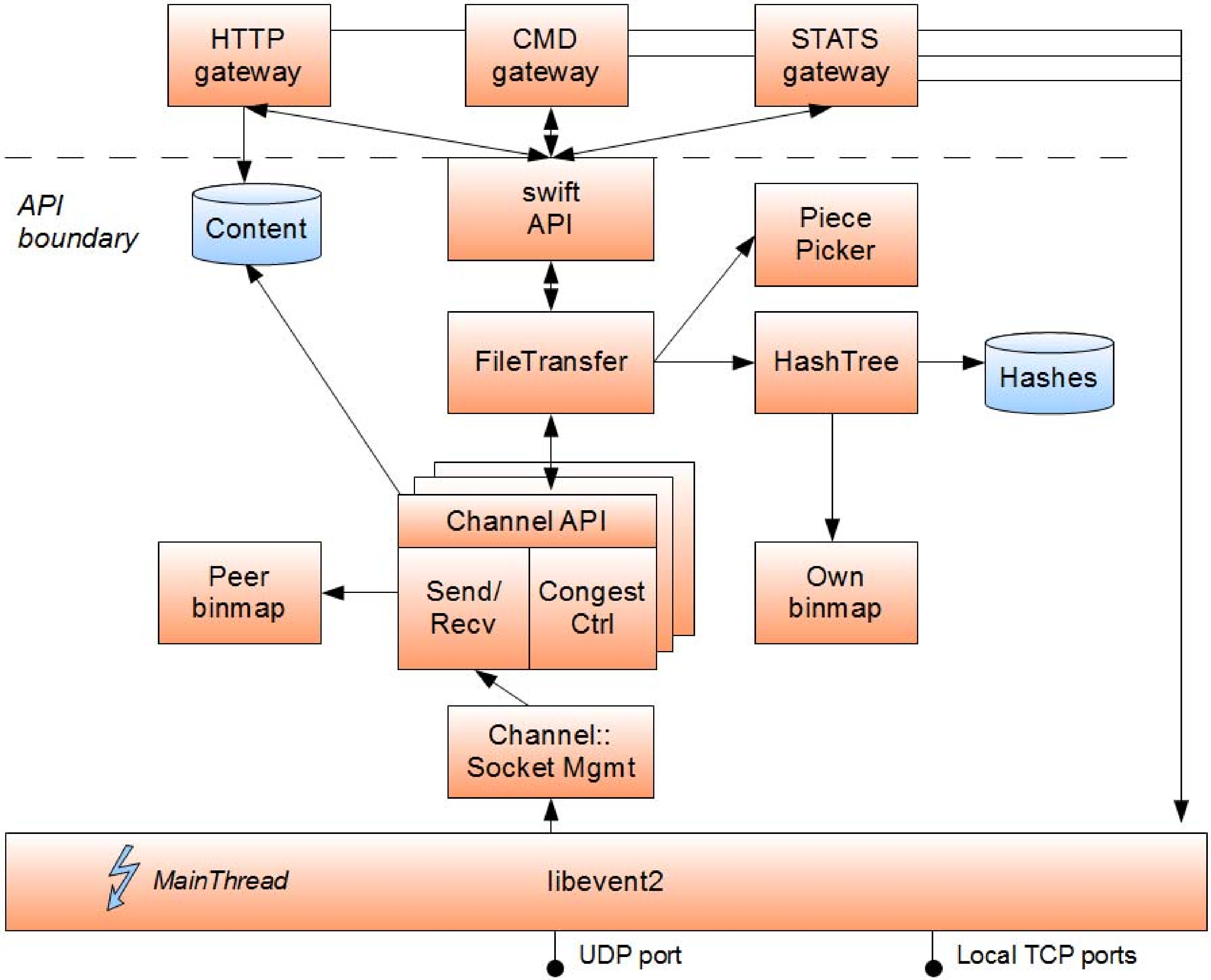
et al.

Introduction

- Swift implemented in C++
- Libevent2 library for socket communication
- UDP

  + Multiplexing: Many swarms on same socket

  + IETF LEDBAT congestion control
- Video-on-demand + live prototype
- Source code:
  - www.libswift.org (GitHub)
  - LGPL License


HTTP gateway
CMD gateway
STATS gateway
API boundary
Content
swift API
Piece Picker
FileTransfer
HashTree
Hashes
Channel API
Peer binmap
Send/ Recv
Congest Ctrl
Own binmap
Channel:: Socket Mgmt
MainThread
libevent2
UDP port
Local TCP ports

Summary

- More info, sources, binaries:
  - www.libswift.org
- Acknowledgements
  - European Community's Seventh Framework Programme in the P2P-Next project under grant agreement no 216217.

Questions?

Arno Bakker (arno@cs.vu.nl)

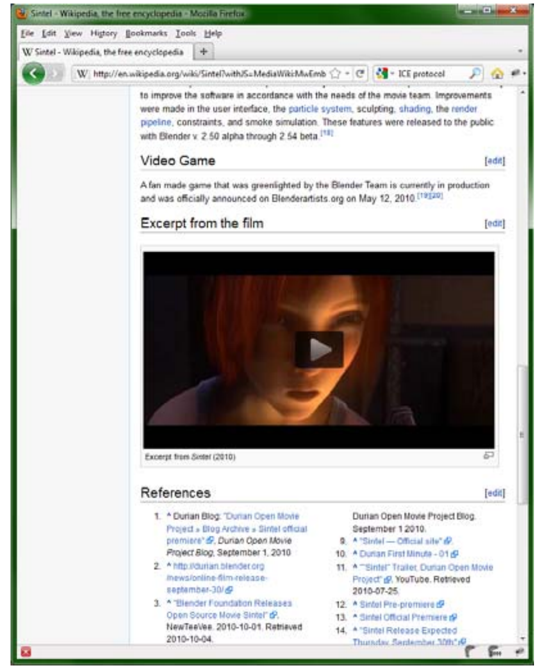
Riccardo Petrocco <r.petrocco@gmail.com>

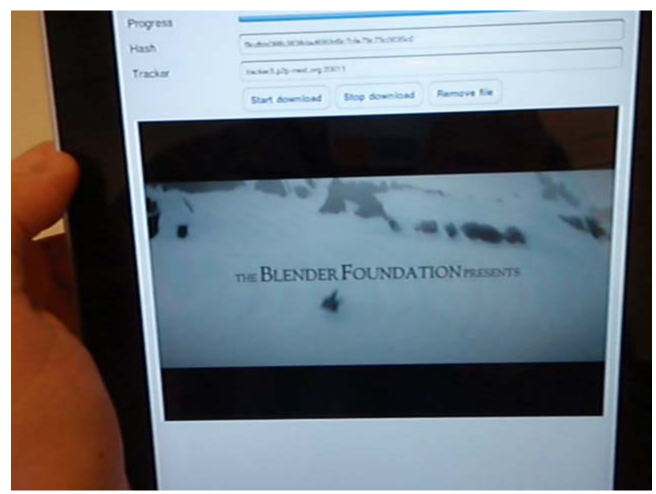
Johan Pouwelse (peer2peer@gmail.com)

Extra slides

Status

- Implemented in C++
  - Video-on-demand over UDP
- Running in Firefox:
  - <video src="swift://…
  - Via 100 KB plugin
  - Hooks on en.wikipedia.org
- Running on:
  - iPad
  - Android
  - set-top box
- Works with P2P caches





The Internet today

- Dominant traffic is content dissemination:
  - One-to-many
    - Download (ftp)
    - Video-on-demand (YouTube)
    - Live (Akamai, Octoshape, PPLive)
- Dominant protocol was designed for one-to-one:
  - TCP

What's wrong with TCP?

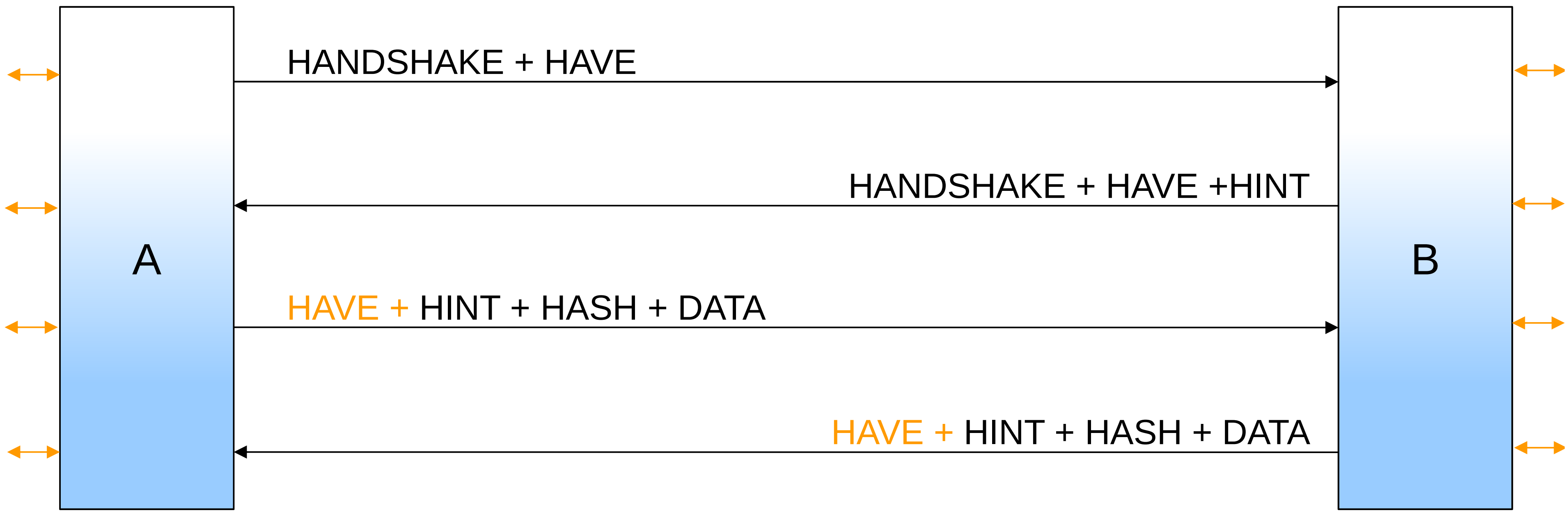
- TCP's functionality not crucial for content dissemination:
  - Don't need Reliable delivery
  - Don't need In-order delivery
- High per-connection memory footprint
  - Aim for many connections to find quick peers
- Complex NAT traversal
- Fixed congestion control algorithms
- I.e. not designed for "The Cloud"

Swift design goals

1. Generic protocol that covers 3 use cases (vod, live, dl)
2. Have short prebuffering times
3. Be extensible:
   - Different congestion control algorithms (LEDBAT)
   - Different reciprocity algorithms (tit4tat, Give-to-Get)
   - Different peer-discovery schemes (tracker, DHT)
4. Can be carried over different transport protocols (UDP,TCP,RTP profile)
5. Traverse NATs transparently
6. Low footprint

Swift on the wire: Example 2

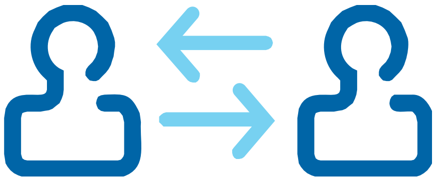
- Peer A and B both have some chunks
- Are receiving chunks from others in parallel



- Note: Chunk availability always up-to-date by pushing

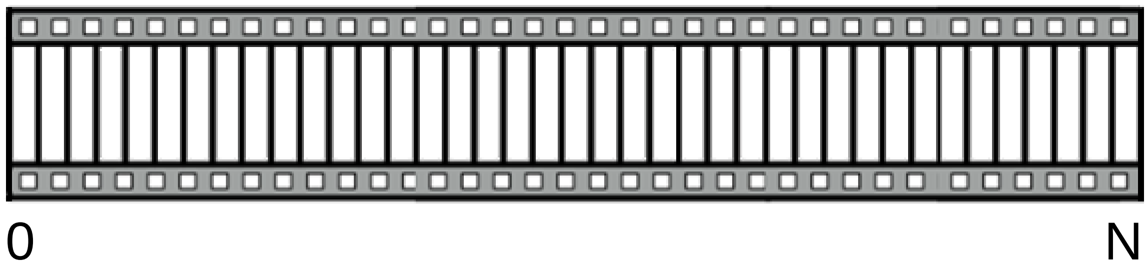
Chunk availability and Rarest first

- Rarest-first is common element in chunk selection policies:
  - Peers download chunk that least peers have
    - Low supply
  - Peers can upload that to many peers
    - High demand
- Result: Upload capacity of peers exploited !
- Requires:
  - Peers have good view of neighbours' chunk availability
  - Hence: Swift pushes HAVE messages



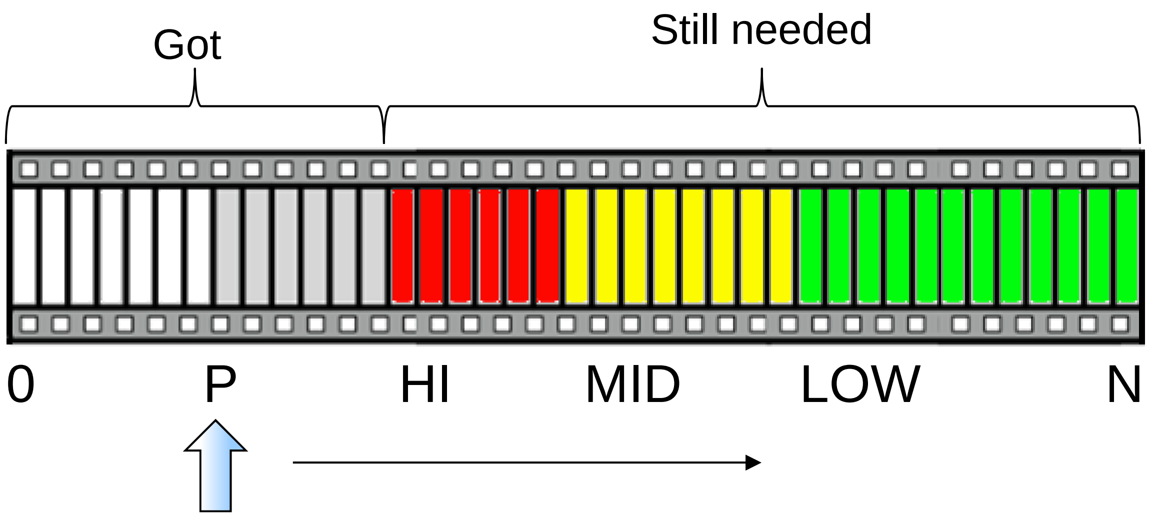
BitTorrent basics

- Content divided into fixed-sized pieces: 0..N



- Computers exchange pieces following economic model
  - Rarest-first (Low Supply -> High demand)
  - Not in order!
- Bootstrap and security data in .torrent file:
  - Address of peer tracker
  - Cryptographic hash of every piece (integrity checking)

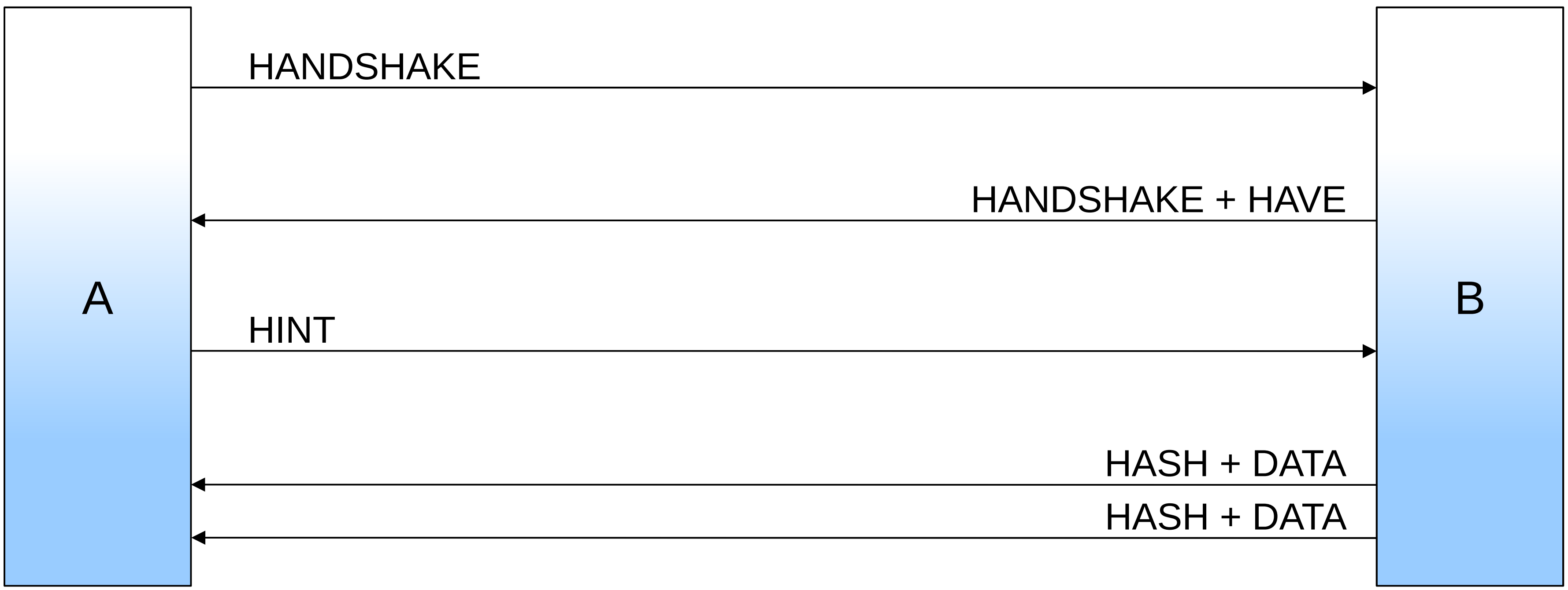
P2P-Next video-on-demand



- Divide set of needed pieces into:
  - High: always, in-order
  - Mid: if no high, rarest-first
  - Low: if no high or mid, rarest first
- Use new Give-to-Get algorithm for uploading
  - Upload to best forwarders

Swift on the wire: Example 3

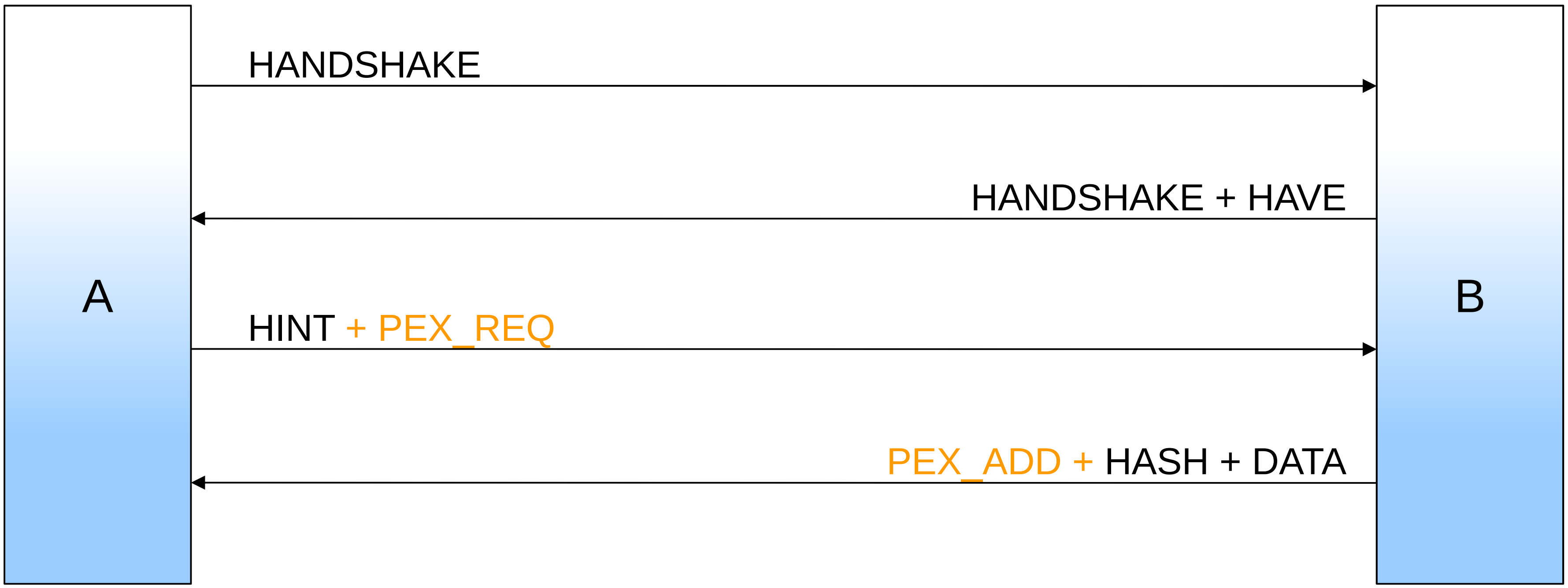
- Peer A is starting leecher, peer B is seeder



- Note: Receiver controls flow

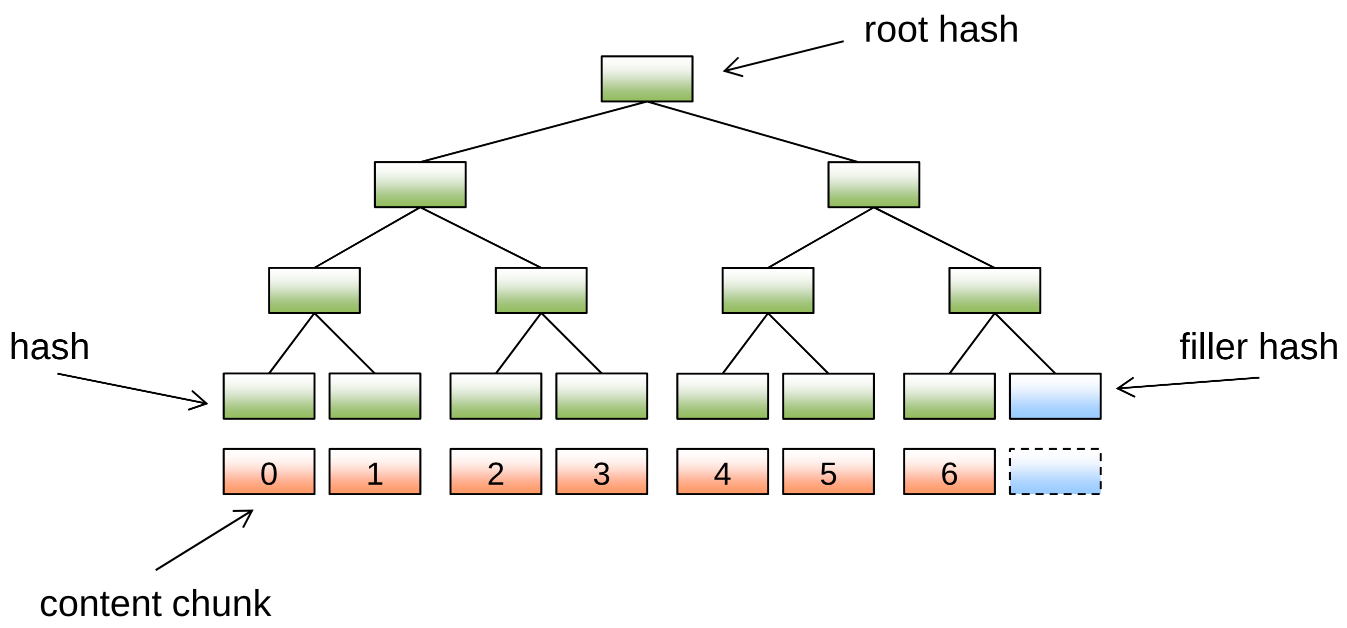
Swift on the wire: Example 4

- Peer A is leecher, peer B is seeder,
- Peer A requests peer list



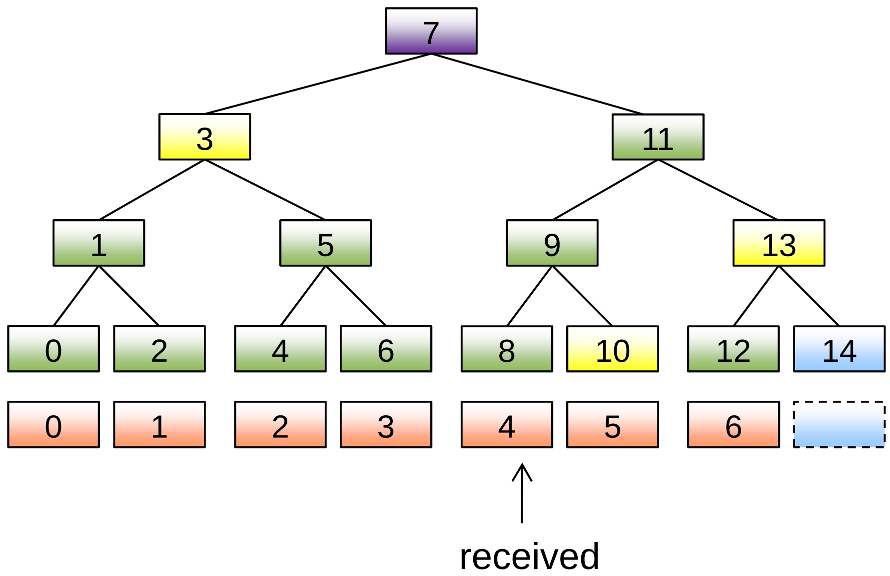
Swift integrity checking

- Content identified by single root hash
- Root hash is top hash in a Merkle hash tree



Swift integrity checking (cont'd)

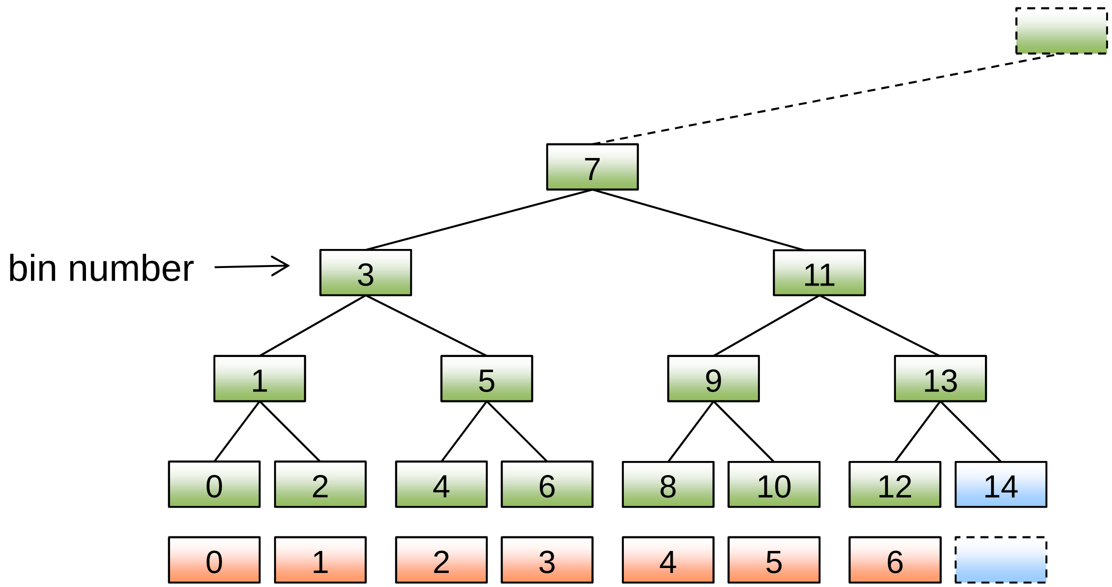
- Atomic datagram principle:
  - Transmit chunk with uncle hashes
  - Allows independent verification of each datagram



  - Root hash + some peer addresses enough to start download!

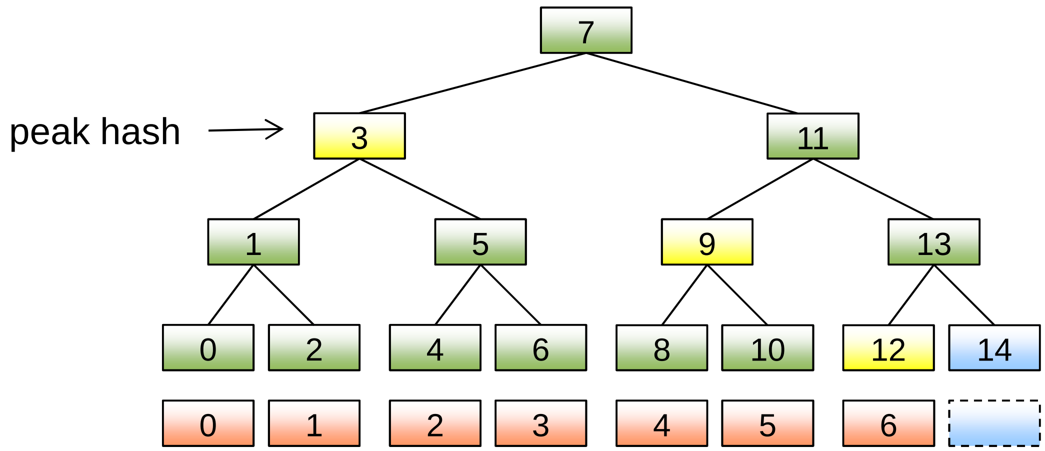
Swift chunk IDs and live trees

- Nodes in tree denote chunk ranges: bins
  - Used for scalable acknowledgements + low footprint
- Dynamically growing & pruned trees for live



Swift Peak Hashes

- Used to automatically, and securely calculate content size
- Don’t need size to start download (i.e., metadata is just root hash)

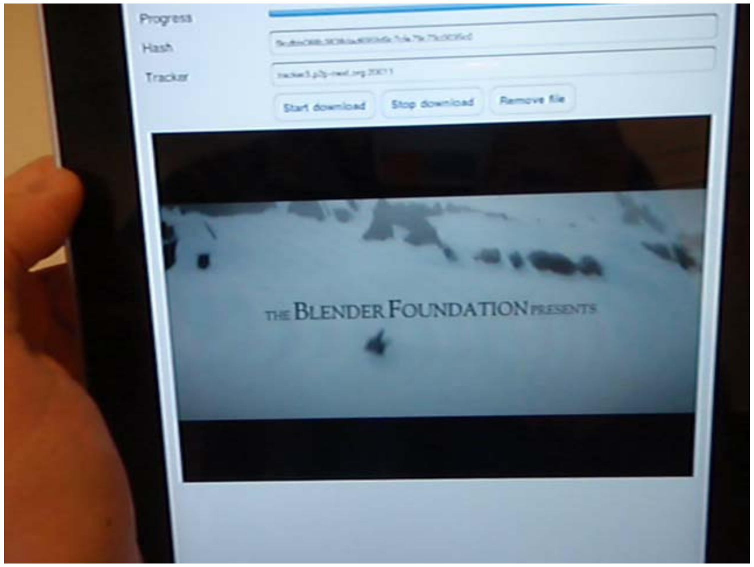


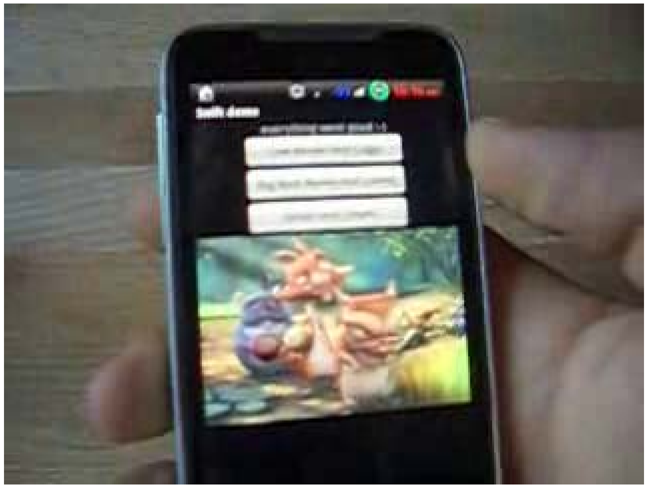
Transport protocols

- Swift over UDP
  - Implemented
- Swift as RTP profile (charter hint)

Swift over UDP

- Datagram consists of channel ID + multiple messages
  - Channels allow different swarms on single UDP port
- Message is fixed length, first byte message ID
- IETF LEDBAT congestion control
- Simple NAT traversal via protocol itself





Swift as RTP profile

- cf. Secure Real-time Transport Protocol (SRTP)
  - "layer residing between RTP app and transport layer"
- Chunk = RTP packet

V P X CC M PT	Sequence Number
Timestamp	
SSRC Identifier	
Extension ID	Extension header length
Data …	
HINT+HAVE+HASH	
Length of swift messages	

Swift as RTP profile (cont'd)

- RTP header protected against malicious modification
- Merkle tree can handle variable-sized chunks (if req)
- Advantages of UDP