Peer-to-peer Media Storage and Casting – Draft 4

Jens Finkhäuser <jens@finkhaeuser.de>

2020

Abstract

This document outlines the problem domain by detailing current characteristics of video streams, internet connections and related technologies. It provides an analysis of related work, and proposes an alternative solution.

I. Version History

Publication Date	Document Version
2020-00-00	Draft 4
2020-01-08	Draft 3
2019-06-06	Draft 2
2019-01-11	Draft 1

II. Terminology

i. Streaming vs. Casting

The term *streaming* is nowadays synonymous with consuming media online without having previously downloaded them.

It's worth recalling that this is not the only meaning.

- In storage parlance, streaming just refers to sequential access of data as file systems or tape storage provide, as opposed to the *random access* mode granted by memory.
- In file format parlance, a streaming file format is one that structures data in records that can be sequentially consumed, in order to better support streaming storage access.

When used in the context of distributed storage, the term only disambiguates against types of file synchronization that use non-sequential patterns, such as employed by e.g. BitTorrent. The vast majority of file access methods are in fact streaming methods, rendering the common usage somewhat inaccurate.

A competing term is *casting*, which in networking parlance exists in different, well-defined variants:

Unicasting describes transferring a packet (or sequence of packets) from an originating node to a single recipient node.

Narrowcasting (*multicasting* in IP) describes transferring such a packet from a single originating node to a defined set of recipient nodes.

Broadcasting describes transferring such a packet to the world at large, for any interested recipient to pick up.

Swarmcasting is sometimes used as a form of casting that originates at more than one node.

On the packet level, most media streaming solutions today are in fact unicasting solutions. Many unicast streams might be served in parallel, but each packet typically has a single recipient.

This document uses the *casting* terminology for disambiguation when specific transport modes are implied, and *streaming* for the case where transport modes are unspecified.

ii. Record Streams

A useful definition for this document is that of *record streams*. As described in the previous section, in file format parlance, a stream is any sequence of records that can be consumed sequentially.

Video and audio files tend to come in streaming formats, defined as a sequence of (conceptual) frames.

But other kinds of record streams exist – conceptually, for example, a commit in a git repository is a record. Since all commits after the root commit refer to one or more preceding commits, a git repository can be laid out as sequential records. In fact, temporally speaking, this is almost always the case – except when two commits are created at *exactly* the same time, based on *exactly* the same preceding commit.

ii.1 SI vs IEC Units

One of the problems plagueing computer science is the mixed use of SI units and IEC units. In SI units, the prefix *mega-* denotes one million, whereas in CS we more often use it to mean 2^{20}.

Just like storage manufacturers, broadband providers tend to use SI units when specifying Mbit/s. Operating systems typically use IEC units when specifying file sizes in MiB.

This document keeps the use of SI unit Mbit/s for broadband uses, because it makes no explicit reference to file sizes. Implementations must be aware, however, that the *Mibit/s* requirements for streaming media files cannot be fulfilled by *Mbit/s* broadband speeds of the same numeric value.

ii.2 Bandwidth

The author(s) of this document understand that the *bandwidth* term is not correctly applied to what should be described as data transfer or transmission rates.

In common usage, though, *available bandwidth* is used synonymously with a maximum possible data transfer rate; other usages of the word tend to be similar, such as *required bandwidth* or *bandwidth usage*, etc.

For simplicity, this document conforms to the common usage.

III. Problem Domain

This section outlines the problem domain of peer-to-peer media streaming by analysing each domain component, and providing requirements derived from the each.

i. Internet Connections

We routinely build data centers these days with leaf-and-trunk architectures that see a Gbit/s of bandwidth provisioned for each CPU core, leading to Tbit/s availability on the trunk, so that it can be sobering to compare this to standard, affordable internet connection speeds around the world.

i.1 Average Bandwidths

Cable.co.uk [1] provides a good overview of world broadband speeds of 2018, which demonstrates the range to be between 5-60Mbit/s. The 2018 data highlights that the bottom end of the range improved by maybe half an Mbit/s, whereas the top improved by ca. 4Mbit/s – we consider that too little to update the calculations in further sections.

The average improvement of ca. 25 percent makes a big enough difference for updates since the last draft, but does not affect the general problem scenario enough to propose new solutions.

i.2 Minimum Bandwidths

Its worth noting that the data released only lists average speeds, no median or standard deviation. It therefore is unclear how this translates to geographic distribution; it is a well established pattern that metropolitan regions enjoy much faster bandwidths than the country side.

Theres a clue in a broadband map Germany publishes [2]. It maps where availability of broadband speeds exceeds particular percentages. In metropolitan regions, available broadband speeds in excess of 50Mbit/s are typical, while only speeds between 6-16Mbit/s seem to be available universally.

Of course, this map lists advertised broadband speeds while Cable's results list measured speeds. According to the FCC's 80/80 metric of measured speeds [3], median speeds vary greatly from advertised speeds from ISP to ISP. An average 80/80 performance across ISPs looks to be in the ballpark of 70% of the advertised speed.

If we apply this assumption to Germany's broadband map, it leaves the conclusion that universally available, actual broadband speeds in that country should be expected in the range of 4-11 Mbit/s, with an average of 7.5Mbit/s.

We can compare this average to the 24Mbit/s published for Germany by Cable; then it is lower by a factor of 3.2. This factor probably does not apply globally, but without more research, we may assume it does. This leads to a minimum available bandwidth assumption of 1.5-18.75Mbit/s globally. Applying the average increase of bandwidth over the last twelve months leads to a new bandwidth assumption of 1.9-23.5Mbit/s globally.

i.3 Latency

There is not much published data on internet latency around the world. Dotcom-Monitor publishes latencies between data centres [4], which is a useful starting point, but will bear little resemblance to end-user latency experiences.

The table does offer an insight, however. With inter-city latencies ranging from the 20s of milliseconds to the 400s, we do know that latency differences range by about a factor of 20 – more than an order magnitude. We can safely assume that latencies for end-users will generally be higher, but follow a similar order-of-magnitude distribution between various locations on the planet.

There is a second conclusion the table permits, and that is that – unsurprisingly – low latency is generally associated with geographic closeness.

The FCC report [3] also includes some data on latency, but only measures latency from the consumer to the nearest measurement server. The latencies measured here match up reasonably well with Doctom-Monitor's data, maybe adding up to 50ms on each customer's side, with a maximum of 100ms for customer-to-customer p2p connections.

i.4 Asymmetry

Broadband connections are rarely symmetrical in their up- and downstream capabilities. Most connections by far provide much higher downstream capabilities than upstream, limiting the available bandwidth for peer-to-peer streaming at the upstream point.

While we may expect changes to this [5], adoption of such new technology takes time to roll out. For the time being, it's reasonable to assume that upstream capabilities lie at 10-25% of downstream capabilities, as provided by current technologies.

i.5 Peering

Neither bandwidth estimate reflects issues of interconnectivity. In previous work at Joost, it was found that in China (serving as an example only), there were very limited peering arrangements between ISPs. Decent bandwidth and latency could routinely only be achieved between customers of the same ISP, while connections between customers of different ISPs were throttled so much as to be useless.

The principle of peering, i.e. offering a quid-pro-quo exchange of bandwidth between ISPs has been the founding stone of Internet architecture since its inception. China a decade ago demonstrates that this principle is not universally applied, leading to the expectation that there will be pockets of the internet effectively isolated from each other for peer-to-peer communication – even if bandwidth is available in principle.

i.6 IP Network Constraints

For the purposes of modern internet usage, it's safe to assume that traffic is transported over IP at the network layer (assuming an IP compatible adjustment to the OSI model).

Path MTU IPv4 and IPv6 place different constraints on packet sizes, but both are eventually bounded by the underlying data link

layer. In data centers, the data link layer might be based on any protocol, but for consumer connections, it's safe to assume that one of Ethernet or 802.11 are involved, and maybe a PPP protocol layered on top of either, such as PPPoE for broadband connections.

That means path MTU for peer-to-peer connections is almost certainly in the range of 1400-1500 Bytes [6]. We'll assume the lower of these to be on the safe side. The implication is that all peer-to-peer IP traffic will be segmented into transmission units of ca. 1400 Bytes.

Packet Throughput Rate One of the characteristics of internet traffic typically disregarded in transmission rates is that IP is a packet switching network. Specifically this means that

1. Each packet is transmitted in full over the underlying media before the next packet is processed, and
2. routers or other processing units along the path will only process and therefore forward complete packets. Their capacity for processing packets is bounded by their hardware design.

The implication is that the underlying hardware reason for bandwidth and latency characteristics is the maximum packet throughput rate of the networking equipment and links.

For applications streaming at high bandwidth, it is then imperative to keep the packet throughput rate below the path capacity in order to achieve the minimum possible latency – higher throughput rates will inevitably fill network interface buffers and may cause packet loss.

TCP vs. UDP TCP is the default stream oriented protocol on top of IP, and UDP the default datagram protocol. Both add a concept of ports to IP connections, in order to multiplex different senders and recipients at the same IP endpoint.

Their difference is that UDP does little more than that, whereas TCP adds a delivery guarantee for each packet. The method chosen to do so is to number bytes transmitted via a TCP connection, and for each received packet send an acknowledgement of the last received bytes.

In terms of packet throughput rate, TCP therefore halves it for the same payload bandwidth compared to UDP in order to guarantee delivery.

It's worth noting that the TCP SACK mechanism does not reduce the number of acknowledgements sent – it merely allows for more selective acknowledgements and retransmissions in case a packet was dropped, but subsequent packets were received. Without the SACK mechanism, only data up to the dropped packet could be acknowledged.

i.7 Casting

The terminology section refers to *casting* as opposed to *streaming*. It should be pointed out that several casting strategies are available at the IP layer, but support for them is limited.

Unicasting is the default mode of transferring IP packets.

Narrowcasting exists in the form of IP multicasting. Unfortunately, IP multicasting is in practice largely limited to a single IP network, or a group of IP networks under the management of the same organization. IP multicasting rarely works on the wider Internet.

Broadcasting exists in the form of the broadcast address of each IP network, and is therefore limited to the network, and not routed outside.

Swarmcasting does not exist on the IP layer.

The upshot is that any forms of casting other than *unicasting* must be emulated at the application protocol level. IP-based multicasting or broadcasting could be employed as optimizations for local networks.

Note that this is a viable strategy for e.g. IPTV services within hotels, etc. An implementation of a media streaming system should at least not obstruct such uses.

ii. Video and Audio

Video and audio streams have particular qualities over other types of files that require recognition when designing streaming protocols.

ii.1 Bitrate

Video streams encode more data than audio streams, and therefore require a higher bitrate, so we can neglect audio stream bitrate requirements for the moment.

Video stream bitrates depend highly on how noisy the image is, what quality expectations you have, and which video codec you use. If we assume 1080p/30fps as the standard of today, Blu-Ray requirements would be up to 40Mbit/s using the H.264 codec.

You can always compromise on quality, and that is the typical strategy for broadcasting over various media. Even so, typical bitrates are often no lower than 12-13Mbit/s with H.264 for *"unnoticeable"* quality degradation.

Googles VP9 has been shown to achieve up to 30% better compression rates, H.265 up to 50%; and AV1 claims to outdo them both.

This leaves a fairly wide range of 12-40Mbit/s for H.264, with hopes that H.265 or AV1 may lower them to 6-20Mbit/s. Given both ranges, its safe to assume that in the 12-20Mbit/s range, good quality can be achieved. Video streaming services often lower this to 5-10Mbit/s with reduced quality streams.

Of course, 1080p is strictly speaking yesterdays standard – were now living in a world of increasing 4k videos, and 8k is starting to become the new high end reference. Both formats quadruple bitrate requirements on resolution alone, and routinely offer 60fps framerates instead of 1080ps 30. With no further reduction in quality or improvements in video codecs, we can assume the following table provides reasonable data:

	Good	**Reduced**
1080p/30fps	12-20	5-10
4k/60fps	96-160	40-80
8k/60fps	384-640	160-320

Table 1: *Bitrates in Mbit/s for different quality requirements and video formats.*

It becomes very apparent that at while at 1080p it may be possible to stream video with fairly good quality, that becomes impossible for most internet connections for 4k. Reduced quality may work for 4k, but streaming 8k is basically reserved for LAN environments at the moment.

Video streaming providers spend a lot of effort on reducing these bitrate requirements further; there surely is some leeway. But 8k streaming is outside of most internet connection speeds by an order of magnitude, so compression improvements will only do so much.

ii.2 Subjective Streaming Quality

Research into subjective video quality during degraded conditions [7, 8] suggests that when other quality factors such as the above cannot be reduced, frame loss conditions may have less impact on perceived quality than could be expected.

Jitter, that is frame drops spread out over a period of time, is generally perceived to be preferrable over jerkiness or buffering.

Within limits, then, a streaming service could introduce or tolerate jitter better than congested paths between producer and consumer.

ii.3 Multiplexing

Each video stream is effectively a multiplexed stream of video and audio data, and may contain additional data streams. While downloadable videos do, in fact, multiplex this data into single streams, storage media such as DVDs or Blu-Ray do not.

What in storage media is offered as a convenience for bundling multiple audio languages with the same video content, also becomes a strategy for keeping transmission bitrate low in streaming.

A media streaming system should consider sourcing separate streams, and multiplexing them for transmission and decoding. In order to do so, individual bitrates in constant bitrate encodings, and additional

bitrate changes in variable bitrate encodings must be known by the streaming servers.

iii. Privacy

In peer-to-peer networks, potentially privacy sensitive data gets sent through peer nodes to their destination. Such routing nodes may keep data only in transient memory, but may also need to store data in on-disk caches. In the case of storage nodes, this is their main purpose.

Privacy measures need to be put in place to make this process safe, let alone GDPR compliant.

iii.1 Transport Encryption

The web's standard for ensuring privacy is transport encryption via TLS (formerly SSL), which is a form of endpoint-to-endpoint encryption. The underlying assumption is that transport endpoints represent the sender and recipient persons or organizations.

Since this is *conceptually* the case for the client-server architecture of the WWW, the only concern for TLS is to ensure it is also *actually* correct. Techniques such as domain certificates, OCSP stapling, etc. are employed for this reason.

In peer-to-peer networks, transport endpoints do not represend sender or recipients.

iii.2 End-to-end Encryption

In order to ensure privacy, some form of end-to-end encryption has to be employed. Using public key based cryptography, this is achievable – if and only if each of the sender and recipient are uniquely identifiable via a public key.

Senders can encrypt data with the recipient's key before sending it. The remainder of the peer-to-peer infrastructure must then treat all data as opaque.

This opaqueness presents particular challenges of optimization:

1. Naive end-to-end encryption effectively forces unicasting of data. The constraints of public key cryptography come to the rescue here; typically public key cryptography is only used for encrypting a symmetric cipher key. While transmitting this key remains a unicast operation, payload data can then be narrowcast to all recipients of the symmetric key.
2. Opaque media streams mean no intermediate node can perform optimizations, whether it is using different video compression methods, or selectively dropping frames, etc.
3. Adding caching capabilities of nodes means that the block cipher mode needs to be chosen differently from typical transport encryption in order to remain secure in the face of *random access updates* to data.

iv. Authentication, Authorization & Accounting

AAA is often performed by a central authority in networks, which when applied to a peer-to-peer system would turn the network hybrid.

iv.1 Authentication

Authentication in a peer-to-peer network means ensuring that a peer represents a particular participant on the network.

This is not the same as ensuring the peer represents a person or organization. Such authentication can additionally be performed, but is not required for the secure operation of the network layer.

If public key cryptography is employed, then public keys can effectively be used as participant identities – any data sent encrypted to the participant can only be used by the holder of the private key, thus ensuring that participants act on behalf of the key pair holder.

This solves the immediate need of the peer-to-peer layer. Some adjacent problems can trivially be solved by introducing a minimal PKI:

Individual/Organisation Authentication Authenticating individuals or organizations

as owners of the key pair requires a verification process that is outside the scope of this document. It must involve a trusted verification authority, and a verification process.

Such a process may involve sending a public key together with a photo ID for individuals to the verification authority. The result of the process should likely be a certificate signed by the authority, which peers can check for validity using standard cryptographic means.

Organizational Group Authentication Larger organizations will typically be split into smaller groups, who the organization may wish to keep separate for data access purposes.

Public key based authentication should then allow participants to generate a master key pair with which to sign group key pairs.

Device Authentication Nowadays, most individuals and probably all organizations are represented on a network by multiple devices. In peer-to-peer networks, it may be sensible to represent each such device by an individual key pair.

The PKI should work analogous to group authentication above.

iv.2 Authorization

After a participant has been authenticated, the next step is to determine what the participant is permitted to do. A peer-to-peer network should aim to achieve a decentralized authorization scheme for this.

Some inspiration can be drawn from OAuth [9] – here, permission tokens scoped to data objects, signed by a trusted authority over the data object serve as secure markers for authorization. A serialisation scheme for similar tokens exist in the JSON Web Token (JWT) [10] standard.

iv.3 Accounting

When a participant has performed an action, and if the action is deemed sufficiently critical, there should be a verifiable and private log of it.

IV. Related Technologies

This section outlines related technologies at a high level of abstraction, and provides a short analysis of how the technology meets or fails to meet issues of the problem domain.

i. Filesystems

An implementation of a distributed media storage system shares some similarities with local file system implementations. The main difference is that in a distributed system, a consensus algorithm needs to be employed to keep the system state valid.

A secondary difference is that storage and retrieval latencies are much higher in a distributed system – or potentially so. But the mitigation techniques of more localized caching are identical (in the abstract).

Therefore, a simplified overview of file system architecture can inform the design of a distributed storage system.

Some of the terminology used in this section is specific to UNIX operating systems, but the concepts are transferrable.

At its most basic, a file system manages which data *blocks* of a block device belong to a file. Block devices provide access to fixed-sized blocks of bytes, so files may use zero to many blocks, and leave blocks partially filled. All this is managed in what we'll conceptually call the *block allocation layer*.

A typical optimization of block allocation is to store files as much as feasible in sequential blocks. Such a block range is typically called an *extent*.

In file sharing parlance, the term *chunk* is more often encountered. A chunk is the more abstract term, encompassing single blocks as well as extents; it is the basic unit of file subdivisions.

Block allocation information is stored in *inodes* (possibly derived from "index node"), which also contain limited metadata about the file. The most common form of metadata is the file name, a permission bitmap, etc. We'll call this the conceptual *metadata layer* (even if in implementations it is indistinguishable from the block allocation layer). The metadata layer at minimum identifies a file.

On top of this, inodes also contain indexing information. Directory inodes are file system entries just like file inodes, but instead of referencing allocated blocks, they reference files contained in the directory. We'll call this the *indexing layer*, and it's used for content discovery.

Another component of many file systems is a *journal*, in which the intent of modifications is recorded before the modification is made, so that crash recovery can be optimized. Journals effectively form a consensus algorithm between the state before and after a crash.

An alternative to journals are full *copy-on-write* file systems which write new data to newly allocated blocks, then update metadata to account for the change, iterating all the way to updating the the file system root. If journals can be compared to a consensus algorithm, copy-on-write can be compared to eventual consistency.

ii. Content Addressable Networks, etc.

Content Addressable Networks exist in multiple forms these days. It's worth noting that the name derives from a specific research paper [11], which introduces the concept of Distributed Hash Tables as an optimization technique for 1990s hybrid peer-to-peer systems.

The key concept is to find values (such as file contents) based on a key (such as a name, a content hash, etc.). Any DHT used in this fashion forms a content addressable network.

In this, a CAN is a form of Information-Centric Networking [12]. A PARC research project, Content-Centric Networking [13] served as the basis for the current Named Data Networking project [14].

The main difference between CANs and NDN are that CANs are typically created as overlay networks over host addressed networks such as IP. By contrast, NDN aims to replace IP networks by using data names as routing information – routing decisions are made by matching the longest name prefix to prefixes advertised by networks or machines.

This leads to two interesting observations about the state of current CANs:

- The concept of CANs in no way requires the name of a data object to be a hash over the data. That is, data objects can be mutable under a stable name. In fact, NDN's longest prefix routing would prefer stable and hierarchical names as opposed to the pseudo-random data produced by cryptographic hash algorithms.

- A single DHT represents a flat routing space, while NDN is hierarchical. Nested DHTs containing ever longer name prefixes could be employed to structure peers in such a way that a) at each hierarchy level there are fewer unique name prefixes to manage, and b) peers serving content with related names can be looked up faster.

iii. LEDBAT

LEDBAT [15] is a congestion control algorithm specifically designed for background file synchronization, and employed as such by BitTorrent and derived technology.

LEDBAT works by timing acknowledgements sent by recipients of data, and therefore works only with underlying transport protocols that send such acknowledgements – or requires application protocols to send them.

The congestion control is opportunistic: it backs off from sending data very quickly when congestion is detected, but tries to use more bandwidth when the path is congestion free.

One of the great features of LEDBAT is that it effectively performs path congestion prediction by analysing increases in transmission latency, while other algorithms depend on congestion events having occurred already.

The downside of the protocol is that it is explicitly designed for background traffic, and not very aggressive in the face of other traffic on the same link. That makes it fairly unsuitable for video streaming, where reasonable effort should be expended to maintain a minimum bitrate.

It's worth mentioning TCP Cubic [16] as TCP's current state-of-the art congestion pro-

tocol. Where LEDBAT uses a linear function for determining window sizes, Cubic uses a cubic function (hence the name). The advantage of a cubic function is that it ramps up traffic very fast to the level before a congestion event occurred, and then probes for more bandwidth fairly aggressively after the network had time to recover.

iv. HTTP/3 aka QUIC

In October 2018, the working group for the QUIC [18] protocol formally requested the standard to be adopted as the next version of HTTP, HTTP/3.

For our purposes, QUICs secondary design goals are of particular interest: it introduces explicit congestion control as opposed to LEDBAT, and reduces the number of acknowledgement packets sent for received data. Where TCP can send at most four byte ranges in a SACK message, QUIC can send up to 256.

As QUIC applies these mechanisms to multiplexed streams over a single connection, this allows for very fine tuned retransmissions.

Of particular note is that QUICs congestion control mechanism is not fixed, and only in the current version employs TCP's Cubic algorithm [16].

One more interesting feature is the use of Forward Error-Correction Coding [17] to reduce the need for resending data. In particular, FEC-encoded larger data chunks spread over multiple data packets can tolerate the loss of some individual data packets, at the expense of slightly larger overall transmission size.

v. Kademlia

Kademlia [19] is a Distributed Hash Table implementation with many desirable properties, used in peer-to-peer systems today.

It's most notable feature is that peer node IDs and hash table keys occupy the same namespace. Using an XOR metric to compute notional distance between entries in the namespace allows Kademlia to select "close" peers for hash table entries.

The paper mentions peer node IDs to be randomly chosen, but refers to other methods possible. One could imagine using a public key, or hash of one. Since data keys are in the same namespace, it is likely that the same hash function is used with arbitrarily large input to derive the lookup key.

One point here is that in likely Kademlia use cases, node IDs and keys are highly randomized, leading to a very wide spread of keys across nodes. Another conclusion is that "distance" between nodes is notional only, and bears no relation to geographical or network distances.

These features combined likely lead to great robustness in the face of partial IP routing failures, but unfortunately allow for few optimizations related to network topology. As we have seen in the latency section above, such optimizations are highly desirable for streaming applications.

vi. SimHash

Cryptographically secure hashing algorithms share one property to be resistant to attacks, namely that slightly different inputs lead to vastly outputs.

By constrast, SimHash [20] tries to yield similar outputs for similar values. It is supposedly used by the Google crawler for finding duplicate documents.

For two strings s1 and s2 with a common prefix, the SimHash distance between them may be comparatively large when the postfix differs greatly. The interesting property of SimHash is that nevertheless the distances of each of these strings to a third randomly generated reference k are going to be very similar, if not downright identical.

vii. Double Ratchet Algorithm

In the Signal messaging app, the Double Ratchet Algorithm [21] is employed to provide forward security and break-in recovery. That is, if a symmetric encryption key were to be learned by an attacker for a specific message, future and past messages will not be compromised.

A Double Ratchet based scheme for groups of recipients is described in an older

paper [23] (referring to the previous *Axolotl* name of DR), and serves to turn the inherent Unicast operation of DR into an encrypted Multicast capable system.

Notably, the algorithm relies on a different protocol for the initial exchange of a shared secret. The same author provides X3DH [22] as a solution, which is explicitly designed for asynchronous use.

viii. Merkle Trees

A Merkle Tree [24] is a tree data structure used in cryptographic applications. Data is placed into leaf nodes, and each leaf node is labelled with a hash over the data it contains.

Non-leaf nodes are labelled with a hash over all hashes of their direct child nodes, all the way to the tree root. In this way, the root hash effectively becomes a hash over all content.

Content becomes addressable by these hashes: the entire data is identified by the root hash, subtrees by the hash of the node that forms the subtree root (in some uses referred to as a *munro hash*), and data blocks are identified by leaf node hashes.

viii.1 Copy on Write

When modifying data in a block, the data block hash changes. The implication of the Merkle tree structure is that then also the parent hash of the data block changes because the data block hash changed. This proceeds all the way up to the root hash.

A new root hash implies that semantically, a copy of the entire Merkle tree is created whenever a single data block is written – though data blocks in leaf nodes remain largely unaltered.

viii.2 Signed Hashes

Signing a hash in a Merkle tree cryptographically effectively signs the entire subtree worth of data. This is useful for e.g. file signing, in which the Merkle tree root would be signed.

But also when subtrees are updated because a data block changed, it is possible to just sign the subtree root to indicate that it's a valid part of the data as a whole.

Combined with an appropriate communications scheme, it thus becomes possible to synchronize Merkle trees without sending hashes from the modified data node all the way up to the root.

ix. Dat

The Dat Project [25] is a protocol specification and implementation of a peer-to-peer file synchronization system.

Dat's main characteristics are:

- Content discovery is based on DNS entries or Kademlia. As we've seen above, Kademlia has no notion of network vicinity between peers, and therefore cannot be used in its unmodified form for selecting optimal peers.
- Transports are either HTTP or uTP, a BitTorrent related protocol that uses LEDBAT for congestion control (even if the spec is strictly speaking transport agnostic). As we have seen, LEDBAT may not be a desirable property for video streaming. Conversely, HTTP which is currently based on TCP suffers from TCPs lower packet throughput than UDP.
- File chunk lookups are based on a Merkle tree. When casting, the file contents constantly change, leading to ever-changing root hashes in the Merkle tree having to be propagated to consuming nodes, adding undesirable overhead.

While Dat seems robust and well thought out for file synchronization, and uses efficient building blocks such as Kademlia and Merkle trees, these building blocks do not make it optimal for casting applications.

x. Interplanetary File System

Another peer-to-peer file system is the Interplanetary File System [26].

One has to be a bit careful with the naming here. The IPFS project as a whole encompasses several different protocols built on top of each other. The bottom layer is also named IPFS, but is not actually a file system – file system functionality is only offered at

higher layers. A the bottem layer, IPFS provides a content block addressing scheme. Unless each content block is chosen to be large enough to encompass an entire file, it does not even implement a full block allocation layer of a file system.

With IPFS' insistence on using content hashes as content names, it furthermore makes blocks effectively immutable – which isn't necessarily a problem, but demands that any higher level file system semantics built on top of it must follow a copy-on-write pattern.

Content discovery in IPFS is either based on DNS pointers or on their own hashtable based NDNS system, which is currently highlighted as being very slow.

xi. PPSPP

The Peer-to-peer Streaming Peer Protocol [27], as the name suggests, is a networking protocol designed for streaming media content between peers.

It's design centers around Merkle trees, much like in the Dat project – in fact, the Dat paper refers to PPSPP for inspiration in this regard. In contrast to Dat, PPSPP however provides a mechanism for partially updating Merkle trees, verifying updates with cryptographic signatures, such that casting becomes possible.

Furthermore, PPSPP supports two distinct access modes, recognizing the difference between file synchronization and casting requirements. The mode we'll term *on-demand mode* is based on a request-response cycle, much as in Dat. The *casting mode*, on the other hand, is initiated by a subscription message, whereupon the source pushes additional data to the subscriber without further data requests, until an unsubscribe message is received.

Where it lacks in comparison to Dat is any concept of file system hierarchy or metadata, or content discovery. It simply disregards discovery as a problem outside of its scope.

Much like QUIC, PPSPP accounts for many multiplexed channels across the same connection. Unlike QUIC, however, it does not provide congestion control or any form of voluntary rate limiting, either per connection or per channel. This makes timing dependent multiplexing of separate audio and video streams just a little harder over PPSPP than over QUIC (where it is also not considered to be a user visible feature).

Unlike QUIC, PPSPP uses LEDBAT for congestion control, but like QUIC it can acknowledge larger numbers of content ranges than would be typical for TCP.

V. Proposed Architecture

This section outlines a proposed architecture at a high level of abstraction.

i. Introduction

In the previous section, we have explored the problem domain, and analysed various related technologies. The conclusion from the analsysis must, unfortunately, be that *no single existing technology will satisfy the problem domain's requirements*.

This isn't to say that no solution can be found. The Dat Project, for example, successfully combines Kademlia distributed hash tables with a representation of file chunks in Merkle trees, and what is effectively the on-demand portion of the PPSPP protocol.

It lacks the casting portion of the protocol, and adds an indexing layer which isn't required. It also contains fairly simple notions of authentication and authorization, and lacks all accounting.

A simple starting point for an architecture that fixes the issues would then be:

1. Kademlia for content discovery.
2. PPSPP for content streaming/casting, with its Merkle tree representation of files.
3. A PKI for authentication.

As the more detailed analysis has shown, however, these components are still not ideally suited for low-latency, high-bandwidth media streaming. The following sections therefore describe casting-optimized versions of these components.

i.1 Overview of Roles

The architecture takes into account many of the roles likely necessary in a video distribution system.

Source in general describes the originating node of a video stream. In the more specific use case, it describes the originating node at which a video is "uploaded" to the network.

In an on-demand scenario, it is easily feasible for a video to exist at multiple nodes. The *source* then refers to the node that is contacted for transmitting the video.

Sink describes the recipient node for a video.

Multi-Sourcing describes an attempt to retrieve the same video from multiple nodes simultaneously, discarding duplicate data chunks. The best *source* for each chunk then "wins".

In a multi-sourcing scenario, a single *sink* is connected to multiple *sources*.

Narrowcasting refers to the opposite scenario, where a single *source* serves multiple *sinks* simultaneously. This is, as explained the terms section i, is not easy to achieve on IP networks except in special circumstances (that may be optimized for).

Unicasting refers to a single *source* serving a single *sink*. It is very linkely that the same *source* node serves many *sink* nodes simultaneously, but over separate connections, making this distinct from *narrowcasting* above.

In realistic scenarios, each connection may also be part of a *multi-sourcing* setup.

Amplifiers refer to nodes whose specific purpuse is to consume a video stream from a *source* and replay it to multiple *sinks*. They thus "amplify" the bandwidth from the original *source*.

Transcoders consume a stream, and become a new *source* for a different encoding of the same video. This might be at a different bitrate, resolution, or in a different format.

The exact way these roles interact in order to construct a full media streaming network is outside the scope of this document.

ii. File/Stream Abstraction

PPSPP's organization of files into chunks, which are addressable via Merkle trees is sound[1], and it's no wonder that Dat has adopted it. What is required on top of Dat's implementation, though, are the partial updates to Merkle trees via munro hashes that PPSPP provides.

Some minor modifications of this scheme will allow for better privacy in the face of forwarding peer-to-peer nodes that may not be fully trusted [2].

ii.1 Encryption

The obvious first modification is to encrypt the data chunks that make up the file. In contrast to transport layer encryption method, blocks of data should be randomly accessible, so the XTS-AES cipher [28] is recommended.

The cipher does not contain any protection against modifications, such that any random sequence of bytes will be "decrypted" to a different random sequence. XTS-AES leaves it up to the application layer to protect against such issues.

The proposal is therefore to concatenate raw data chunks with a checksum [3] before encryption. The result of both operations forms the entry into the Merkle tree. Such a checksum could conceivably be a cryptographically secure hash, as no error recovery is necessarily intended. Note that such

[1] It should be noted that sending hashes of data *ahead* of data is susceptible to extension attacks, and the Merkle tree usage in Dat and PPSPP does not mitigate against them. Double hashing is generally considered a sufficient solution

[2] In peer-to-peer networks, it's easier to arrive at a sound design when your default assumption is that no peer may be trusted.

[3] A more complete proposal is to use a signing scheme here that, combined with encryption, does not fall prey to the problems outlined in Davis' analysis of common schemes [30].

a hash would differ from the hash entered into the Merkle tree.

Such a cipher requires the symmetric encryption key(s) to be communicated out-of band, which naively implemented adds protocol complexity one would like to avoid.

ii.2 Record Stream

When splitting file data into a series of chunks, each chunk is appended to the previous chunk to form the final file again. In casting situations, where there is no concept of a full file until the stream ends, this is more apparent, but applies to all chunking mechanisms.

The proposal is to make this explicit. Instead of adding file chunks to the Merkle tree, add *append records* with file chunk data.

The overhead of this is very low – naively a single byte header is sufficient. But the benefit is that we can now multiplex other types of records into the stream without need for out-of-band communications, such as symmetric encryption keys.

Adding a header does carry security implications. For the most part, it needs to be ensured that the header actually matches the following chunk data. This is partially ensured by the chunk hashes in the Merkle tree.

ii.3 Key Exchange

Key exchange can occur via X3DH [22]. Since that protocol is designed for offline use, and we can now embed other records into our record stream, the proposal is to embed some parts of the X3DH messages into the file stream.

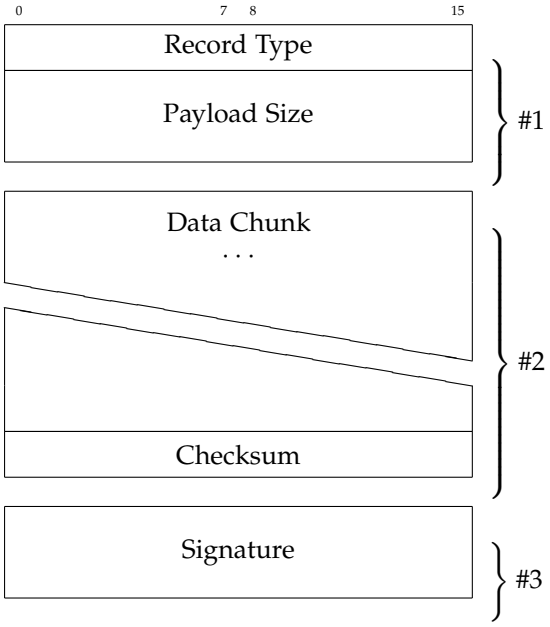
In particular, X3DH requires Bob to publish prekeys to a server, which Alice can use to establish a secure connection. Alice, then, wands to *send* data securely, and Bob wants to *receive* data securely. The burden then falls on file consumers to publish prekeys.

Prekey bundles contain public keys and signatures, nothing that cannot be shared with the public. It is therefore conceivable to store them in a distributed hash table used by the network (see following sections).

Once prekey bundles are successfully distributed on the network, the sender's initial message to each accepted recipient can be embedded directly into the Merkle tree. This initial message contains a shared key for further communication with the recipient. In this proposal, the X3DH initial message should immediately be followed by an encrypted message containing the file encryption key.

ii.4 Extension Attack Mitigation

As mentioned previously, the Merkle tree usage in Dat and PPSPP is susceptible to extension attacks. The proposed full specification of each entry into the tree that mitigates against this issue follows.



1. We begin with a 16 bit field for the record type. Probably an 8 bit field would be sufficient, but let's add the extra extensibility.

2. We follow with a payload size of 32 bit. This is the size of the encrypted data that follows. 32 bit would allow rather larger chunks than useful, but 16 bits would leave chunks artificially short[4].

   This size contains the raw data chunk size, the checksum size, the size of any

[4] If extra overhead is a concern, an 8 bit record type and a 24 bit payload size would also align nicely into 32 bits total.

padding bytes added during encryption *and* the size of the final signature.

3. The next part is the encrypted data chunk. It contains:
   (a) The raw data chunk.
   (b) A checksum.
4. Finally, a signature over the entire preceeding byte data is appended.

The choice of checksum and signature algorithm is left up to the implementation. It is conceivable for the signature to be a simple hash – but in practice, the best choice would be a public key signature as PPSPP proposes [27].

Note that the entirety of #2 above is encrypted, meaning the data chunk is effectively verified with a sign-then-encrypt-then-sign scheme, where the initial signature may only validate the data itself, i.e. with a checksum. This scheme fixes common signing and encryption weaknesses, as well as extension attacks – though double hashing for any hash-based checksums is still recommended.

Mismatching signatures or payload sizes must lead recipients to discard the entire record. In fact, the use of signatures allow recipients to selectively accept chunks from multiple signatories, as long as they have all been deemed authoritative.

We define three types or records in this scheme so far:

1. Append records that append file data,
2. X3DH initial message records for specific recipients, and
3. An encrypted file encryption key addressed to the same recipient.

There is likely a good use-case for a fourth type of record, a metadata record. The metadata payload is somewhat outside the scope of this section; however, following sections may refer to such a record.

One metadata field that immediately springs to mind is a flag indicating whether the file is encrypted at all. Public files do not need encryption and the overhead associated with it. Nothing much changes in the rest of this section, except that the payload size does not need to account for padding bytes with unencrypted files, and only append records are necessary.

iii. Authentication, Authorization and Accounting

The previous section refers to the use of X3DH for exchange of file encryption keys. This already assumes that some authentication is to be performed via a public key infrastructure.

iii.1 Authentication

The two major concerns for a PKI in the context of this document are to provide a chain of authentication, and allow for key revocations. The exact workings of a PKI are beyond the scope of this document.

The proposed hierarchy of keys as follows:

1. Each participant in the network must create a root or master key. This key is to be validated or revoked by the PKI, and forms the basis of all other operations. We'll call this the *identity key*.
2. Each participant must create one, and may create many device keys. These keys allow a participant with multiple devices to join the network. Device keys or hashes of them act as peer IDs (discussed later).

   This distinction between identities and devices allows for addressing each with distinct data.

   Device keys are signed by identity keys.
3. Finally, X3DH mentiones ephemeral session keys. Unlike X3DH, such keys can be issued by identities *or* devices, so the issuer must always be referred to in their usage.

   This allows for key rotation across all communication channels.

   Ephemeral keys are signed by identity or device keys, depending on their purpose.

From the point of view of the peer-to-peer network, this distinction between different key types is sufficient for all proposed use-cases.

iii.2 Authorization

Authorization on the network occurs on several levels, but can be separated into distinct categories.

Network authorization refers to peers verifying that identities or their devices may connect to the network. If an identity is part of the PKI and not revoked, network authorization is granted.

Service authorization refers to peers verifying that a device may perform the requested action – this is the information that allows peers to deny service.

It is beyond the scope of this document to define all possible types of service authorization, but later sections of the document will refer to some of them.

At its most basic, service authorization can be granted if a tuple of an identity and a permission specification is signed by an authority, and no revocation for this tuple exists.

Any identity can be an authority in this scheme; however, it can only be an authority over its own permission specifications. That is, permission specifications are always scoped to the signing authority. Participants in the network can decide whether or not they accept an authority.

The first step in service authorization is for a participant claiming authority over some permission specification by publishing this claim, providing the permission specification and their signature. It may be sufficient to publish this claim, or the claim needs to be validated.

The exact validation process is outside of the scope of this document. However it is implemented, the permission specification will include in some form *what action* is permitted on *which object*[5].

[5]Objects could be groups of other objects; the scoping is very flexible.

If a claim is valid, then a signature by the claimant over the permission specification and an identity becomes a valid permission grant that any node can independently verify.

Permission revocations are similar signed tuples of the permission specification, the identity, and a marker indicating a revocation. Timestamping the tuples for grants and revocations allows for retracing their history.

Subsequent sections of this document assume that *service authorization* is implemented in this fashion.

File authorization refers to file access permissions. If the X3DH scheme is used for embedding encryption keys into the file stream as described previously, then being able to decrypt such a key record translates to read permissions on the file.

Write permissions are also fairly easily handled, allowing for a file to be appended to by multiple sources. It makes use of the above service authorization scheme as follows:

1. As a file owner, a participant publishes a claim that they can grant write permissions on the file in question. This claim is validated.
2. The file owner then publishes a permission grant for writing on the file. The identity of the grantee is, of course, included.
3. The permission grantee publishes a file append record (or some such).
4. Recipients of the record can verify that they can trust the contents, because the owner delegated write permissions to the publisher of the record.

We should similarly publish read permissions in order for peers to deny service early. This prevents peers who have lost their cached copies of the file from redownloading them.

The proposed encryption scheme using X3DH is still lacking in information on how prekey bundles are to be distributed.

In the X3DH specification, it is the server's responsibility to hand out one- time prekeys

one by one, until none are left. In a distributed environment, such accounting is fairly hard to do in a timely fashion, especially when nodes fail. The proposal, therefore, is to not use one-time prekeys [6].

Scoped File Authorization is also possible. In the above scheme, we assume that access to a file is either granted or not.

We can scope permission fairly easily by changing the encryption key before all file append chunks are added to the Merkle tree. This can be done fairly easily by employing the Double Ratchet [21] algorithm, and publishing newly generated keys in the same way as described in the previous section as part of the Merkle tree.

This is not the cheapest operation. All changes to the encryption key need to be sent to all permitted readers. That may mean encrypting a (small) key thousands of times, depending on how widely the file is used, and embedding records for each into the Merkle tree.

The change would require the publishing message of the key to be extended to refer to the last hash encrypted with the previous key, so that reasonably rapid random access can be granted.

In the unmodified Double Ratchet scheme, each message (aka chunk) should be encrypted with a different key. As an optimization, it is proposed that this key rotation is not performed for every chunk, but rather every time the reader set changes, and optionally at a configurable interval.

Similarly, write permission grants must be scoped not only to a file, but to a hash offset – this is entirely possible under the proposed scheme[7].

Delegation One of the desirable side effects of the proposed permission scheme is that it's possible for file owners to delegate jobs to other peers. When casting video, for example, it is unlikely that a peer has the bandwidth for all possible video resolutions (and therefore bitrates) that clients require.

A published transcode permission allows its grantee to authoritatively publish a downscaled video, without the need for the stream producer to sign each downscaled video chunk.

Similar reasoning applies to peers verifying casting sources other than the originating peer, if that is a desired feature.

iii.3 Accounting

In the previous sections, the publication of prekey bundles or of permission claims already require some form of accounting of some actions. Whatever method chosen for publication, it can serve for accounting for other actions.

iv. Transport Protocol

As we have seen in prior sections, the choice of transport protocol for media streams can significantly affect the viewing experience, especially when casting.

To summarize:

- Latency introducing jerkiness or buffering must be avoided, and
- congestion prediction via LEDBAT is desirable, but
- instead of LEDBAT's adjustment to congestion window sizes, possibly TCP Cubic leads to better stream performance.

The major factor affecting latency is the use of TCP's ACK/SACK and resend mechanism. In particular, as TCP has strict sequencing of packets, when multiple streams are multiplexed over the same TCP connection, the loss of one packet of one stream immediately stalls all other streams as well.

Much as QUIC proposes, it is therefore desirable to implement a new protocol capable of multiplexing several channels (or streams) over the same connection. Such a protocol must necessarily be based on UDP, because IP is the only viable lower layer on the Internet.

Both QUIC and PPSPP propose multiplexed channels, and in theory, either of

[6] An alternative might be to use short-lived keys, but that introduces the problem of time synchronization across the network

[7] Complications arise when write permissions are granted and revoked for conflicting offsets. This must be addressed in an implementation.

them is suitable enough for our purposes – except that QUIC contains special provisions we don't need (such as handshakes and channels relating to HTTP only), while not containing much casting support. PPSPP on the other hand relies on a transport that has ACK messages of some sort, which UDP does not.

The minimal transport implementation would be to use PPSPP, with the addition of an ACK messages, over UDP[8].

iv.1 Acknowledgements

TCP sends ACK ranges in TCP headers, and introduces SACK messages for sending larger ranges. By contrast, QUIC only sends the equivalent of SACK messages, but may contain many more ranges in a single message.

This all seems a little back to front.

On a reasonably stable connection, it should be expected that in excess of 50% of packets sent by a sender are received by the receiver. That value is not arbitrarily chosen, but rather represents the tipping point at which either sending acknowledgements of received packets, or requests for unreceived packets becomes cheaper (in terms of less data to send).

The proposal, then, is instead of sending ACK ranges, to send messages with MISSED ranges instead. If in excess of half the packets are received, that makes for fewer messages and smaller messages to send, and will lower the required packet throughput rate for a given stream compared to TCP or QUIC, thus reducing latency.

If fewer than half of packets are received, it is doubtful that streaming media is still possible – no amount of error correction or skipping frames will likely restore the signal to acceptable quality. This, or indeed potentially a higher threshold still, is a clear signal for a receiving peer to seek different sources for the same stream.

Missing the Future The purpose of sending ACK instead of MISSING is that a recipient can hardly know that it misses any packets after the last one received, that is, it cannot miss future packets.

Sending ACK ranges leaves the recipient passive, and makes the sender responsible for re-sending everything that might have been missed.

It makes sense to send the last received packet as a kind of ACK range end, in order for the sender to make any decisions. However, as an optimization over ACK behaviour, sending MISSED ranges is still sensible.

Missed Range Determination On the transport layer, MISSED ranges are any missing packets with sequence IDs lower than the latest received packet. However, the application layer may determine that some of these are not actually missed, even if they are missing.

The default behaviour should be to report all MISSED packets, but let the application layer override this as necessary.

Use with LEDBAT The only other consideration is that for LEDBAT-like congestion prediction, periodic MISSED packets need to be sent, even if they contain zero missed ranges.

A periodic packet containing the above last received packet and no MISSED ranges, together with a timestamp of the last received packet would enable LEDBAT.

The amount of packets sent is well below the amount generated by TCP's ACK mechanism, and corresponds more to the TCP Delayed ACK mechanism.

Real implementations of TCP Delayed ACK send acknowledgements within very few milliseconds of retrieving packets. A good period for LEDBAT usage would have to be a research topic.

iv.2 Congestion Management

LEDBAT requires periodic messages sent from receiver back to the source in order to measure round trip times. LEDBAT then shrinks the send window when RTTs increase, and grows it when RTTs decrease. In that way, it predicts, and if possible avoids congestion on the entire path from peer to peer.

[8] In practice, it may well be that a subset of PPSPP suffices, and other small changes to the protocol are desirable – but that can safely be considered an implementation detail at this stage.

By contrast, TCP Cubic reacts only to actual congestion by also reducing the send window, and then attempts to increase it while no further congestion is detected.

It's clear that LEDBAT is better for video streaming, as it helps avoid congestion in the first place. Where LEDBAT is not suitable to streaming is that there is no lower bound on the decrease of the send window. Unfortunately for us, a minimum bitrate is required for maintaining a video stream without stutter or buffering.

The problem is that LEDBAT is not cooperative – yes, receivers send back enough data to measure RTTs, but otherwise are passive participants in the scheme.

The proposal is to introduce a RTT threshold, based on the combined stream bitrates of all streams sent on the path, that informs the sender that it can no longer guarantee quality. This information is sent to the recipient as a congestion warning message.

The congestion warning message should, if possible, include addresses of other live nodes serving the same streams. This should allow the recipient to start recovering individual streams by receiving them from other sources, and selectively unsubscribing from them at the original sender.

TCP Cubic is such a popular congestion control mechanism for TCP, that it bears examining whether it's cubic curve might make for better recovery from congestion than the linear function of LEDBAT. This is necessarily speculative, and actual research in a live system should be conducted.

The Initial Send Window Size should likely be determined based on the target packet throughput rate, and modified appropriately by the congestion management algorithm afterwards.

iv.3 Forward Error-Correction Coding

Much like QUIC, our proposed transport would benefit from FEC [17] for avoiding resends from MISSED packet ranges.

The issue with FEC is that initially it increases bandwidth in order to reduce latency introduced by resends. However, since increased bandwidth can also increase packet loss, requiring more packets to be recovered, the exact redundancy factor used by FEC for best results is difficult to determine.

We propose an adaptive FEC mechanism: starting with a standard redundancy factor, the factor should be increased when MISSED ranges are reported – because when packets can be recovered due to the use of FEC, MISSED ranges need not be reported at all.

If this leads to an *increase* in MISSED ranges, the factor needs to be reduced to less than its value before the experiment started, until a FEC redundancy factor of 1 is reached and FEC is disabled.

Once FEC needs to be reduced, it is possible to send this as a warning to the application layer that the available packet throughput rate for the path is possibly not sufficient to meet the target rate.

A Splitting/Joining Algorithm is at the heart of the translation from the file representation layer to transport layer. Each chunk of data from the Merkle tree must pass the algorithm to be split or joined into appropriately sized network packets. It is at this point that FEC is used to help determine how many distinct packets a data chunk is encoded into.

The FEC redundancy factor to use is actually composed of two distinct factors: the application layer may define a stream factor that the transport layer should try to use. Additionally, each transport path may have its own FEC factor based on the congestion conditions. Each data chunk should be FEC-encoded with a factor that takes both into account. The exact method for determining this factor is a research topic, but most likely the transport path FEC factor must be authoritative.

iv.4 Wire Protocol

The wire representation is fairly similar in concept to the data chunk format described in Section ii.4.

The main difference for the wire protocol is that it might be useful to merge message IDs as outlined in Section vi.1.

Instead of detailing the entire wire protocol here, it suffices to note that for the splitting of data chunks into individual packets, what's necessary to encode is:

1. The message type, indicating that the message contains a data chunk part,
2. the part number of the data chunk, so the recipient can reassemble the chunk,
3. the FEC factor used, so that reassembly can commence once sufficient parts are received,
4. the data part, and
5. a checksum for integrity checking of the part.

Any signing of the message may happen for each message separately, or in the case of short messages concatenated into a packet, for the packet as a whole.

iv.5 Multi-Peer Sourcing

As referred to previously, nodes are free to choose multiple sources to stream the same video from, letting the first arrival of a data chunk "win", and discarding all others.

It is feasible for an overlay protocol to request only every Nth data chunk from any given source, thus interleaving partial streams. This reduces transfer rate requirements on each source, while maintaining the incoming transfer rate at the sink. The strategy is particularly useful considering that upstream capacity at sources is likely far more limited than downstream capacity at sinks.

Similarly, using FEC, it is possible for each source to provide only every Nth FEC-encoded chunk. The benefit here over the simpler approach above is that given a sufficient number of sources, the total loss of one of them can be recovered due to the error coding.

v. Content Discovery

When discussing content discovery, it bears recalling the conceptual distinction of a *metadata layer* and an *indexing layer* in file systems. Recall that the former stores file metadata, and the latter effectively stores file locations (within directories).

v.1 Metadata Layer

The metadata records alluded to in the file representation section cover most of the immediate requirements on a metadata layer. That is, metadata records should exist for and within each file that, at minimum, mark the file as encrypted or unencrypted.

It is largely unimportant to the file representation in what form metadata records are stored. All we can say for certain is that they should contain structured data. The choice of serialization format is outside the scope of this document.

> **Note:** The main consideration really is the generation of signatures for the records. Popular representations such as YAML or JSON all suffer from the problem that they need to be normalized in order to be compared – multiple different byte sequences can in fact refer to the same data. This can make validation of signatures difficult as well, if a signature of generated content is to be compared to previously serialized content, for example. See, for example, how JWT [10] solves this by signing a specific serialized form of the content.

Given some form of structured data, we define the following metadata entries with special meaning. An unencrypted metadata record with (some of) these fields MUST be the first record in a compliant record stream.

`name`
: Contains the name under which this file can be resolved. That is, resolving this name and retrieving the first record must result in a record identical to the one containing this field (but may come from a different peer). If the field occurs in subsequent metadata records, it MUST be ignored.

`encrypted`
: A boolean flag indicating whether encryption is enabled for the file. If the flag is true, it means any chunk append records cannot be processed by

higher application layers unless appropriate key sharing records have been received[9].

Other metadata fields might help the application layer decide how to process the file. The proposed optional fields are:

`type`
: An IANA Media Type [31]. This information helps the application layer decide early what sort of stream to process.

`bitrate`
: An expected bitrate for the stream. This bitrate cannot sensibly specify the *actual* stream bitrate, if non-file chunks are multiplexed into the stream. For variable bitrate encoded data, this bitrate should represent a reasonable expected average. The field largely serves to inform the transport layer's congestion management algorithm.

  The bitrate and path MTU result in an *target packet throughput rate*, which is the more useful metric for the transport mechanism.

  Publishing the bitrate is very useful for sender and recipient, and should be the default:

  1. Senders can better multiplex streams when they know the target packet throughput rate, prioritizing packets from each stream so that they meet the target.
  2. Recipients can detect when a sender fails to meet the target and find a new source, if available.

v.2 Indexing Layer

The indexing layer of the proposed system does not have to be particularly complex. In particular, we do not need to transport file system hierarchy information as e.g. Dat does.

[9] As record streams are ordered, it is theoretically possible for subsequent metadata records to switch encryption of following blocks on or off again. For the sake of simplicity, the proposal is to only consider the flag in the first occurring metadata record.

However, we should let clients choose which substreams of a de-multiplexed media stream to subscribe to, in order to maximize the efficient use of available bandwidth.

The proposal is to publish an index file tying these different substreams together. It is a regularly published file, and just as any other file must start with a metadata record. However, the file does not need to include anything other than metadata records, and in fact most likely shouldn't.

> **Note:** This is similar to how Dat handles indexing data. The difference is that Dat uses an indexing file format, whereas we can use metadata records.

Metadata records in this file must supply one additional field, but may contain any number of application-defined fields. The additional field is:

`contents`
: The contents field is a list of reference entries to other streams.

`live`
: The live field is a boolean flag indicating a live record stream, e.g. one containing a live video feed. It's an indication to the transport layer to switch to casting mode instead of on-demand mode, and applies to all referenced files alike. If absent, the default value is false.

Each reference entry defines the same mandatory and optional fields as metadata fields described in the metadata layer. This allows for early selection of substreams.

A requirement is, though, that the metadata entries listed here MUST match the metadata entries in the resolved file, otherwise the reference is considered invalid.

In addition to these fields, other fields are likely useful for selection by the application layer, but are file type dependent. A video stream entry, for example, should contain metadata specifying the stream resolution, for selection by the user. Metadata on audio streams should include information on the audio language. Specifying mandatory metadata for individual stream formats is outside the scope of this document.

> **Note:** It is a fairly trivial exercise to extend this indexing layer to support file system hierarchies. For example, merely referencing another index file is enough to create a hierarchy, though this mechanism is not very efficient.
>
> If and when need for such hierarchies arises, this system is extensible enough to support them.

v.3 Hash Table

While the previous sections cover how metadata is structured and (minimal) indices are managed, no mention is as yet made how file names are resolved.

First, it's important to note that in PPSPP, the so-called *swarm ID* is used to identify a data stream. For casting, PPSPP requires signed Merkle trees, which are considered optional for on-demand mode. If signed Merkle trees are used, partial tree updates can be pushed to receivers, enabling verification of new chunks without renegotating the Merkle tree root.

It is proposed to make this the default behaviour for all files. Under the PPSPP specs, the swarm ID for signed Merkle trees is the public key used for signing. In terms of our previously defined PKI, this is an ephemeral key – that is, a key used solely for the signing of this particular stream.

With PPSPP's requirements thus covered, the proposal is to use a modification of Kademlia for resolving file names.

Node IDs or peer IDs in a network are the peer addresses of nodes participating in the network. According to our previously defined PKI, those are equivalent to device IDs.

Kademlia defines its key namespace to be 160 bits in length; the proposal is to loosen the requirement somewhat and define the key namespace to be the same length as device IDs. It's not unlikely that a hash over public keys that's 160 bits in size is the best choice; however, for future extensibility it may be better to make that size based on other requirements of peer/node/device IDs.

Since Kademlia expects node IDs to be hashes of this length, it further requires value keys to also be hashes of this length, and then computes an XOR distance as part of its algorithm.

The choice of cryptographic hashes for node IDs is a good one – it allows for a wide spread of IDs across the key namespaces, with the result that Kademlia's hash table should be resilient to partial network failures.

Value Keys However, forcing a cryptographic hash function for value keys also creates a wide spread across the namespace, meaning that related keys will not be clustered together. In terms of resolving substreams faster, this is not a desirable property.

The proposal, then, is to use SimHash hashing for value keys, and modify Kademlia to use SimHash distances instead of XOR distances. Recall that SimHash has the property that similar keys produce SimHash values that have similar distances to (most) randomly chosen reference points, such as node IDs. This property means that there is a high likelihood that a node serving one key will also serve similar keys, leading to far fewer lookups.

The worry that this might lead to chokepoints on the network should be mitigated by the modified LEDBAT scheme with feedback – peers can easily choose further distant nodes for substreams.

Hierarchical Naming One side-effect of locating names with common prefixes at the same nodes is that this lends itself to hierarchical naming schemes.

We propose a mixture of URL and the reverse domain name scheme commonly used in software namespacing: `de.finkhaeuser/p2p/storage_casting.pdf`.

This scheme lends itself to defining namespace authorities just as in the case of the Domain Name System (DNS). Within each namespaces, authorities are free to organize related files into the same prefix.

Note that a system for managing ownership over a namespace is outside of the scope

of this document. However, claims such provided by JWT [10], signed by an appropriate authority would be verifiable – the claim would need to include a signing public key that can be used to verify all resources within the namespace. In that, a system very similar to DNSSEC [29] would provide for delegation and key update mechanisms.

k-bucket Sorting In unmodified Kademlia, nodes for each key distances are placed into a *k-bucket*, which is sorted by uptime of the node. This mechanism implies that stale nodes are rarely contacted, and drop out of the hash table entirely.

Unfortunately, uptime is not the only metric for choosing a desirable node. As we have seen, latency increases dramatically with network topology distance – it is clear that of two candidates, the one closer in network topology is far more desirable than the other.

The proposal is to sort k-buckets not just by uptime, but also by other metrics such as network distance.

The optimal procedure for this still needs to be determined. However, as a starting point, a weighted average of normalized network distance and uptime can be used. It should be noted that in the in Joost's streaming layer, using network topology distance as a selection metric made an enormous difference for reducing stutter, both due to reduced overall latency, but also because with shorter paths, packet drops become far less likely.

Network distance, of course, can only be determined at the node – that is, a participant serving values in the Kademlia table can only measure its own network distance (easily) in order to sort values in the k-bucket. However, each querying node also maintains its own k-buckets, thereby cascading the effect.

Hash Table Values In unmodified Kademlia, the protocol either returns values or it returns nodes to query that are closer to the value. The latter is exactly what we want, and should not be modified.

Naively in our case, the former should resolve SimHashes over file names to the actual file – but in the case of PPSPP, we instead need a swarm ID.

It should be the case that the node responding with the swarm ID can also serve the file via PPSPP.

These optimizations over Kademlia therefore come to mind:

1. in addition to key lookup, allow one-step subscription to the key. This is a useful property for subscribing to substreams containing actual content, and removes one round trip to the same node.

2. if a subscription was requested, a node can respond with the swarm ID, but immediately also send the first metadata chunk of the subscribed file – or whatever chunk is considered current in casting.

3. in addition to the swarm ID, a response should always include relevant alternate peers for the same file. This allows the client to update its own k-buckets for faster switching to alternative sources.

vi. Miscellaneous

There are a few miscellaneous considerations to the proposed architecture, listed in this section.

vi.1 Message Encapsulation

The previous sections blithely assume that packets for PPSPP, Kademlia and some proposed addtions are all sent across the same connection. This is of course likely to cause fairly catastrophic failure.

There are three approaches to solving this issue, with different advantages.

Merging Message IDs solves the issue by modifying all protocols to use the same message ID namespace, re-assigning message ID values from this namespace. The main advantage is reduced wire overhead, which should be highlighted is a real concern.

Protocol Encapsulation on the other hand offers a simple message envelope specifying a protocol ID in its header, and leaving the payload to be entirely defined by subprotocols. The main advantage lies in much clearer separation of concerns.

Protocol Separation finally expects different connections for each protocol. While this is very simple to implement, it suffers from multiple drawbacks. Stream multiplexing is much harder to manage if each stream is over a different connection – the network path is the same, but endpoint ports must be merged into a single virtual connection.

Which of these approaches is to be chosen, and their exact implications is left as an implementation detail – suffice to note that a choice must be made.

vi.2 NAT piercing

If moving to a proposed UDP-based protocol, peers must be identified by their public IP address, which may be the address of the router connected to the internet. Furthermore, the router must forward packets sent to a particular IP and port combination on to the peer. The general process is NAT piercing.

STUN is the default protocol for piercing NATs. It's widely deployed, but it should be noted that it is only efficient against some NAT types. In either case, STUN requires two participants in the network with publicly reachable IP addresses, not behind NATs.

The basic principle is that in order for assuming NAT to be pierced, a packet sent to one address, if returned from the other address, must be received by the originating peer.

Typically this is implemented as a single server listening on two distinct ports or IP addresses. With a forwarding mechanism, it would also be possible to utilize two peers, of which only the receiving end must have a public IP.

Universal Plug'n'Play is a networking protocol that allows peers to query their router's public IP. It is much simpler than STUN, but not supported by all devices.

Whichever mechanism is used, UDP-based protocols rely on NAT piercing for peer-to-peer networks. Neither QUIC nor PPSPP have this issue, as QUIC is designed for client-server use (as is IPFS), and PPSPP delegates this issue to outside of its scope (as does Dat).

References

[1] Cable.co.uk, Worldwide broadband speed league 2019 [p. 2]

[2] Bundesministerium für Verkehr und Digitale Infrastruktur, Der Breitbandatlas [p. 2]

[3] FCC, 8th Measuring Broadband America Report, Dec. 2018 [p. 3]

[4] Dotcom-Monitor, Network Latency [p. 3]

[5] Cable Labs, Full Duplex DOCSIS/DOCSIS 3.1 [p. 3]

[6] Wikipedia has a good summary table for MTU sizes of common media protocols, with references to the respective specs. [p. 4]

[7] H.-T. Quan et al., "Temporal Aspect of Perceived Quality of Mobile Video Broadcasting",
IEEE Trans. on Broadcasting , vol. 54, no. 3, pp. 641651, Sept. 2008. [p. 5]

[8] K.-C. Yang et al., "Perceptual Temporal Quality Metric for Compressed Video",
IEEE Trans. on Multimedia , vol. 9, pp. 15281535, Nov. 2007 [p. 5]

[9] OAuth 2.0 [p. 7]

[10] JSON Web Token [p. 7, 19, 22]

[11] S. Ratnasamy et al., "A Scalable Content-Addressable Network" [p. 8]

[12] Information-Centric Networking [p. 8]

[13] Content-Centric Networking [p. 8]

[14] Named Data Networking [p. 8]

[15] Low Extra Delay Background Transport, RFC6817 [p. 8]

[16] M. Ahmad et al., TCP CUBIC: A Transport Protocol for Improving the Performance of TCP in Long Distance High Bandwidth Cyber-Physical Systems [p. 8, 9]

[17] Forward Error-Correction Coding [p. 9, 18]

[18] QUIC Working Group [p. 9]

[19] P. Maymounkov, D. Mazières, "Kademlia: A Peer-to-peer Information System based on the XOR Metric" [p. 9]

[20] M. S. Charikar, " Similarity Estimation Techniques from Rounding Algorithms" [p. 9]

[21] M. Marlinspike, "The Double Ratchet Algorithm" [p. 9, 16]

[22] M. Marlinspike, "The X3DH Key Agreement Protocol" [p. 10, 13]

[23] N. Unger et al., "SoK: Secure Messaging" [p. 10]

[24] Merkle Tree [p. 10]

[25] Dat Project [p. 10]

[26] Interplanetary File System [p. 10]

[27] Peer-to-peer Streaming Peer Protocol, RFC7574 [p. 11, 14]

[28] IEEE P1619 [p. 12]

[29] DNSSEC [p. 22]

[30] D. Davis, "Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML" [p. 12]

[31] IANA-registered Media Types [p. 20]