Workgroup: The Interpeer Project
Published: 19 January 2026
Author: J. Finkhaeuser
Interpeer

Architecture for Human-Centric Networking

Abstract

This document follows on from [INTERPEER-REQUIREMENTS] to outline an architecture for a future Internet that has all of the desired properties laid out in that sibling document.

The requirements are derived from a problem statement contained in [INTERPEER-PROBLEM-STATEMENT].

About This Document

This document is a draft PIE and so adheres to the publishing process and naming described in [PIE.f92f09.00].

- This version is published at PIE.f92f09.architecture
- The latest version can be found at PIE.f92f09.architecture-00

Contributing

Responsibility for this document lies with The Interpeer Project.

Source code for it can be found at https://codeberg.org/interpeer/draft-jfinkhaeuser-interpeer.

Additional coordination and discussion occurs on a mailing list:

- Address: interpeer@lists.interpeer.org
- Archive
- Membership management

Table of Contents

1. Introduction

This document describes a novel architecture for a future Internet, designed to better serve human needs (human-centric) than the current Internet/Web architecture allows for.

The first part on desirable properties (Section 2.1) explores the properties from [INTERPEER-REQUIREMENTS] again, focusing on how they can interact. Then, a discusssion of architecture elements is undertaken in Section 2.2. The section on interactions (Section 2.3) focuses on how distinct interactions in the system are considered. Further notes are provided in Section 2.4.

A final Section 2.5 analyses the proposed architecture according to the criteria derived from the gap analysis.

2. Architecture

In [INTERPEER-REQUIREMENTS], desired properties of such an architecture are listed, and existing architectures examined against that list. For the purposes of designing a novel architecture, Section 2.1 reflects on how these properties can be grouped, or have overlapping effects.

2.1. Desirable Properties

We reproduce the properties summary table from [INTERPEER-REQUIREMENTS] here to ease referencing; refer to that document for more details and rationale.

Property	Description
User-Perceived Performance	Low time to completion of queries, or "time to first Byte".

Property	Description
Network Efficiency	The most efficient thing is to not use the network and use cached data.
Scalability	The architecture supports a large number of components.
Simplicity	From the systems theory definition, avoid complexity.
Evolvability	Be able to change a system component without affecting the rest.
Extensibility	Be able to add new functions safely.
Customizability	Allow clients to influence system behaviour to their needs.
Reusability	Be able to re-use components in the system.
Resilience	Be able to deal with (partial) failure of components.
High Intermittency Tolerance	Be able to support applications in dealing with high intermittency.
Low Intermittency Utiliziation	Be able to utilize low intermittency underlays.
High Latency Tolerance	Be able to support applications in dealing with high latency.
Low Latency Support	Be able to support applications that require low latency.
Low Reliability Tolerance	Be able to support applications in dealing with low reliability.
High Reliability Support	Be able to support applications that require high reliability.
High Throughput Support	Applications that need to move a lot of data should be able to do so.
Inconsequential Throughput Tolerance	It should be possible to prioritize the needs of high throughput applications against the needs of applications for which throughput is less of a concern.
Integrity	Verify and assure integirty of the payload.
Authenticity	Ensure that data comes from the source it claims.
Confidentiality	Prevent data from being shared without consent.
Security	Follow security considerations of [BCP72].

Property	Description
Privacy	Protect privacy (see also [RFC6973], Section 7).
Anonymity	Allow anonymous use.
Pseudonymity	Where anonymous use is impossible, allow pseudonymous use.
Unlinkability	Prevent linkability of multiple pseudonymous activities.
Consorship Resistance	Avoid choke points at which censorship can be enacted.
Accessibility	Provide an enabling environment for all.
Mobility	Tolerate changes in connectivity (network attachment).
Multi-Homing	Tolerate and be able to utilize multiple points of connectivity/ network attachment.
Self-Determination	Permit components to make local decisions.
Interest Grouping	Support components in communicating with other components with related interests.

Table 1: Properties of a Human-Centric Architecture

Quite a few of these properties either are related, or can be treated as a group in certain contexts. In this section, we'll discuss some such grouping, in order to ease some more specific discussion later in this document.

2.1.1. Performance

In the properties section of [INTERPEER-REQUIREMENTS], a combined "Performance" property was further split into individual properties listed separately in the table above.

It should be clear that *User-Perceived Performance* can be significantly enhanced by ensuring low latency and high throughput, for example.

For the design of a protocol more than an architecture, it is worth considering how much meaningful data can be sent in the initial network packet(s). This consideration includes reducing the amount of optional data in packets, which can also help achieve lower latency when fragmented packets are reassembled -- simply because fewer fragments may exist.

From an architecture perspective, it may be able to synthesize a requirement for low packet overhead from this, as well as a requirement for prioritizing data within a data stream.

2.1.2. Avoidance of Single Points of Failure

Several properties relate directly or indirectly to the avoidance of single points of failure. The most explicit one is *Resilience*, of course -- but it bears highlighting that *Interest Grouping* can also be viewed as such a property.

If communication flows from an interested node to multiple nodes purporting to share this interest, any of these nodes could potentially reply to queries, and responses from any of them can be treated as equivalent. This directly supports *Censorship Resistance* as well.

This may also support *Evolvability*, in that individual nodes can perhaps be upgraded others provide service stability.

2.1.3. Unlinkability

If multiple interactions are *unlinkable*, the implication is that each interaction is either *anonymous*, or that distinct interactions are *pseudonymous* with unlinkable identifiers.

It is worth stressing that HRPC considerations from which these properties are derived are mostly concerned with the anonymity of humans. This implies that *Unlinkability* is predominantly concerned with the inability to link multiple interactions to the same person or persons.

Nonetheless, any identifier used throughout the system should follow the same considerations, as to avoid the discovery of links in elaborate metadata analysis.

Identifiers should ideally not reveal anything about what is being identified, and ephemeral identifiers should be used whenever such a use is possible without undue impact on other properties.

2.1.4. Quality of Service

A fair few properties essentially describe what can be grouped together as Quality of Service (QoS)-parameters, which include the distinct properties related to *intermittency*, *latency*, *throughput* and *reliability*.

The *Customizability* property essentially suggests -- at least when limited to these parameters -- that clients be able to specify their requirements in appropriate QoS flags. At the same time, *Self-Determination* implies that other nodes may desire to make decisions that do not respect such QoS flags.

An architecture can never control how remote nodes act. Explicitly mentioning *Self-Determination* is perhaps not necessary under this consideration. That being said, QoS can never be achieved if nodes behave arbitrarily.

The meaningful way out of this dilemma can be summarized by the Russian proverb "trust, but verify". Clients SHOULD trust that their QoS parameters are being honoured, but are best advised to also verify that this trust is not misplaced. Similarly, nodes acting as servers SHOULD trust the QoS parameters are not chosen arbitrarily and strive to meet them, but are advised to verify whether this has undesirable local effects (such as e.g. assigning all resources to one client).

All of the above is not particularly surprising, and follows engineering best practices in distributed systems.

However, the inverse is perhaps more interesting to state clearly: it does not make a whole lot of sense to assume QoS negotations are anything but informational.

Here, *Interest Grouping* can help again. Assume for the moment that a request to a group, an expression of an interest, is paired with QoS requirements. Further assume that all other group members respond with their QoS promises (it would be inadvisable to call them "guarantees").

This would permit a client to fall back on less stringent requirements if no peer can match the strict ones, if the application can support it. This fulfils *Self-Determination* both of the client, as well as of each individual responder.

Evolvability prohibits an architecture prescribing this behaviour, but implementations SHOULD describe the rationale behind the decision of any such strategies they implement here.

2.1.5. Functional Layering

There are several functional layers into which properties can be loosely grouped, with some properties making sense at multiple layers.

1. *Extensibility*, *Simplicity*, and *Confidentiality*, amongst others, are all eased if a distinction is made between payload and protocol. That is, if a protocol provides primitives for confidentially transporting extensible payloads, it is immediately simpler than if it itself contains extensibility mechanisms.

   This, at least, is true with regards to choices made by applications -- at the same time, it may not hurt to have an extensibility mechanism in the protocol. It is generally better to design an extensible protocol and make the predefined functions work within that extensibility mechanism, than try to add extensibility at a later stage.
2. Properties relating to QoS may depend on the properties of underlay networks. As such, a distinction between a "protocol logic" layer and transports that utilize underlays may increase *Simplicity*, but also *Evolvability*, as well as help fulfil those QoS requests.
3. A "protocol logic" layer should then contain the management of underlays according to QoS parameters, the ability to deal with *Multi-Homing* and *Mobility*, and provide primitives for an upper layer to ease dealing with *Integrity*, *Authenticity*, and other properties of a nature more related to human needs.

We will re-visit this grouping in more detail later.

2.2. Elements

In order for a novel architecture that shows all of the properties described in [INTERPEER-REQUIREMENTS], it must be designed in a human centric fashion. This encompasses human-to-human interactions, yes -- but in a computer network, each human actor is represented by one or more entities.

We will term these entities "resources"; a "resource" by this definition is very broadly defined as anything humans may be interested in that the computer network can represent, whether other humans or digital objects of some kind.

We can thus model a new architecture around the expected human-to-resource interactions. REST, ICN and DTN all contain the notion of resources, albeit not necessarily in the same form. IP's abstraction is that of an interface to which an IP address is bound -- but looking a little higher in the stack, we can find that both TCP and UDP define services as the main abstraction via the assignment of services to ports.

The recursive architectures described in the gap analysis do not offer any new forms of interaction, but rather employ a different view on layering and therefore do not require special consideration here.

Architecture	Name	Access method
Internet	Service/Port	Bi-directional packet exchange
REST	Resource	Request/Response
ICN	Content, Data	Interest/Data (similar to REST)
DTN	Endpoint	Bi-directional message exchange
Robotics	Topic	Publish/Subscribe, Request/Response, Action/Status/Result

Table 2: Resource Equivalents in the examined architectures

We note that these methods fall into two categories: the bi-directional exchange category is very general purpose; it does not define in which direction there is more data flow between endpoints. It is similarly not too well defined which endpoint initiates contact.

By contrast, the more request/response oriented methods clearly, if perhaps implicitly, define the role of a resource consumer who initiates a transmission, an a resource holder, who responds. In fact, resource creation is left largely undefined in ICN, and left incomplete in REST. The main advantage of this method is that it permits, at least in principle, removing the resource location from the equation, which induces increased network efficiency due to caching, etc.

Noteworthy is also that the request/response method maps fairly neatly to the document web use case, while the bi-directional exchange maps well to the real-time use case. The API use case can be fulfilled by either mode; many APIs follow a request/response pattern, but callback or notification patterns require bi-directionality.

The clear implication is that a new abstraction must provide both categories of access method. This abstraction must also be the central pivot point for the architecture, tightly coupled to the user interactions it affords.

Finally, robotic's Action/Status/Result interaction is novel in this architecture. But it is conceptually similar to a combination of Request/Response, with an implicit Publish/Subscribe mechanic with a well-defined end.

2.2.1. Users/Humans

Users must be represented in the system somehow, which means they need to be identified. Identification can rub up against HRPC properties, so it should be highlighted that identifiers under this architecture must be pseudonymous.

It must additionally be possible to create many ephemeral identifiers, in order to provide for unlinkability. If ephemeral identifiers are indistinguishable from other identifiers, this enables a quasi anonymous mode of interaction, as a random identifier reveals as little identifiable information as an absence of an identifier, provided that they remain unlinkable (see also Section 2.1.3).

The HRPC guidelines acknowledge that some rights are difficult to meet when others are fully embraced. For example, the right to remedy can be difficult to enforce when full anonymity is given to users in a system.

The solution we choose to this dilemma is to leave it to the specific application how much a user identifier can be traced to a person's real world identity. Using a lot of different ephemeral identifiers will provide more in the way of anonymity, though at the expense of remedy -- and conversely, permanent identifiers would provide zero anonymity and unlinkability, but help with full remedy.

In order to permit both, however, user identifiers must NOT contain, and so leak any information to an observer that would indicate the level of permanence the identifier enjoys. Instead the system must treat all identifiers equally, and must presume they are fully ephemeral, and exist solely for the duration of (a part of) some session.

Furthermore, identifiers must be related to some asymmetric cryptographic key pair. A typical such relationship would be for the identifier to either be a public key (such as e.g. in elliptic curves of [RFC8410], [RFC8032]), or be a key fingerprint, i.e. a hash over some public key for RSA or DSA keys ([NIST.FIPS.186-4]). Establishing this relationship means that it can be verified that e.g. a cryptographic signature is issued by the entity identified via a matching identifier, which in turn induces or helps incude the remainder of the HRPC properties.

2.2.1.1. Creators

Creators of a resource (Section 2.2.2) are those identities that generate a resource identifier.

Creators also own the resource. While many other participants can contribute (Section 2.2.1.3), the creator determines whether such contributions are authorized by adding other contributors to the resource.

If designed appropriately, this induces part of the *Confidentiality* property, in that resource creators ultimately control resource usage.

2.2.1.2. Consumers

Consumers of a resource request and process a resource. This requires read-only access to resources.

2.2.1.3. Contributors

Contributors to a resource are verifiable authors of a part of a resource, i.e. they have provided some signature or other means of verification that this resource part is authored by them.

The creator of a resource is a contributor, at least of the initial resource part. Other contributors are not creators.

Conceptually, contributors to a resource require write access to it. It is very likely that contributres also have read access in order to contribute meaningfully, but it should be noted that this is not strictly speaking necessary.

2.2.2. Resources

As alluded to above, we define a resource as an element in the network architecture that provides some utility to humans interacting with the system. Just as in REST, "any information that can be named can be a resource: a document or image, a temporal service (e.g. today's weather in Los Angeles), a collection of other resources, a non-virtual object (e.g. a person), and so on."

A resource is distinct from an ICN content chunk, because it is not bounded to any particular size (as e.g. chunks may be). If the resource is data, this implies that the data can grow and mutate over the lifetime of the resource. This induces REST's modifiability properties.

A resource is distinct from an IP service and from a REST resource, because it is location independent. A resource is distinct from an IP service also because it may persist beyond the lifetime of a service port. This induces the mobility and multi-homing properties, and both of the reliability properties. It also helps with some performance properties.

A resource is distinct from a DTN endpoint because it does not only name a destination for messaging, but may have other meanings (see above). This induces other modifiability properties, but may also include also portability.

A resource, in other words, is a data stream that may be transmitted efficiently, and has a purpose -- and this purpose must be embedded into the resource, or not all of the above properties can be fulfilled.

In order to retain these properties, a resource MUST also be self-contained. The moment additional metadata needs to be managed for a resource, such as a manifest, or ownership information, etc. the architecture introduces a dependency on a different element which can fail or be unreachable -- and so undermine the reliability and intermittency related properties in the worst case.

This also implies that a resource needs to be verifiable in itself, and so requires a method such as cryptographic signatures to be added.

For public resources, this is sufficient. But for modelling user-to-user interactions, not all resources can reasonably be public. Therefore, end-to-end content encryption also must be supported.

2.2.2.1. Resource Permance

ICN assumes that resources are data. Much the same applies to REST, at least insofar as various REST methods are used to create, update or retrieve resources.

REST also treats data sent in this manner as representational, which is the foundation for modelling resources that do not, themselves, consists of data, but that provide some kind of service.

The primary distinction, however, is not between whether a resource represents data or a service, but whether it can be considered somewhat *permanent*.

A resource with *permanence* can be cached, perhaps archived. If archived, it can be retrieved again at a later date. An *impermanent* resource is ephemeral, and exists only for a single interaction.

Interactions can be arbitrarily long, however. Consider a chat room as an example. The room persists, even if individual conversation topics within are of little interest some time later.

Resource permanence is a topic worth further consideration. From an architectural point of view, however, this is the only distinction between various resource "types" that should have an impact on other architectural considerations.

2.2.3. Nodes & Convergence

In order to provide mobility, multi-homing, improve reliability and deal with intermittency to varying degrees, we consider communications to occur between nodes, not between any particular network interfaces. We borrow a page from the DTN approach and indeed treat the architecture as able to function with arbitrary convergence layer protocols that transport the protocol messages we will discuss below.

In fact, it is likely that each node in the network should be addressable via multiple convergence protocols simultaneously. If the convergence protocol is e.g. IP based, this directly enables mobility, as node addressing can remain static even as attachment points to the IP network change.

One consequence of this approach is that nodes require addresses independent of the convergence layer. Additionally, there exists a many-to-many relationship between nodes and users -- a system may cater to multiple concurrent users, but at the same time, users may simultaneously use multiple devices, each with their own network attachment points.

For this to work, each convergence layer must provide more than a transport means, but rather also provide a means for managing the mapping of a node address to the convergence layer's own addressing scheme. Unlike in DTN, this architecture assumes that such mapping is highly dynamic (DTN is relatively agnostic here). The specification of such mechanisms is outside of this document's scope; however, in principle any mechanism such as the Domain Name System ([RFC1035] and its many extensions and updates) or more novel approaches such as Routing On Service Addresses (ROSA, [I-D.draft-trossen-rtgwg-rosa-arch]) can be used.

Note that as in the recursive architectures, the same protocol may be used for mapping and resolving node addresses to convergence layer addresses, and for mapping user idenitifiers to node addresses. Such re-use is not mandatory, but should be considered.

This architecture, however, defines more requirements on the convergence layer below.

2.2.3.1. Custodians vs. Caches

Any node that has custody of (parts of) a resource is a custodian of that part.

Note that this definition excludes nodes that merely cache a data resource. In contrast to a cache, a custodian is charged with continued storage of the (parts of the) resource they manage. Caches merely provide best effort storage.

By this definition, it is deliberately and explicitly possible for multiple nodes to have custodianship of a resource at any time.

2.3. Interactions

Defining the elements of this architecture is the easier task. There is a strong distinction made between the human as represented by a user identity, and the network node. Another distinction is made between the node and its convergence layer protocols. Finally, a relationship -- in the abstract -- between users and resources in the form of cryptographic signatures is established.

We can now elaborate the interactions users have with resources, and examine how this may related to network nodes.

We have already explored how ICN and REST vs. the Internet and DTN model different access patterns. In particular, one group is more heavily focused on resource consumption, while the other on bi-directional messaging.

We have to observe that neither fully describes the user-to-user interactions we see either on the Internet, or in fact in the physical world. While it is common enough for people to have personal conversations with other folk, a lot of our time we spend speaking to and collaborating in groups.

Rather than treat groups as a special case of interaction, one should consider that a pair of people are still a group, albeit a very small one. In fact, one-to-one communications should be considered the special case, while group communcations must be the general case, if we are to model human needs well in the digital realm.

Within group communications, we can identify the speaker role and the listener role. It is by no means given that there is only one speaker, and only one listener at any given time. In fact, the roles constantly shift back and forth (some people claim they can speak and listen simultaneously, but the data on that is apocryphal at best).

Neither are groups static; members constantly get added or leave. Finally, individuals can be in multiple groups simultaneously.

With so much in flux, this makes it hard to pinpoint exactly how to define a group. The response, in most any digital system, is that groups are "things" that can be created, and that provide affordance for the management and self- management of its members, and relay messages between members.

Now that we have this group "thing" described, should we add a new element to the architecture?

It turns out, that is not necessary. We can simply define a group as the set of users currently concerned with a particular resource.

This has two major implications.

The first, and simpler one is that the convergence layer protocols really must provide group communications, and their respective method for mapping node identifiers to convergence layer addresses is really an exercise in group membership management.

One obvious way in which this can be provided is via IP multicast, e.g. by mapping a resource identifier to a multicast address in some way. But as we will see later, this approach could be a little too naive.

The second implication is because resources must be self-contained: this means also that group membership -- at the level of user identifiers, not at the level of node identifiers -- must be contained within the resource, and by extension such related information as permissions.

It must be, because if it were not, then we would again introduce a dependency of the resource on other elements (nodes, extra metadata) which provides challenges for fulfilling the desired architectural properties.

Note also that this treatment of a group as a set of users concerned with a resource has a clear correspondence with the Publish/Subscribe mechanism described in the Robotics application. Notably, the "resource" here can be viewed as roughly equivalent as the topic to which nodes subscribe or publish to.

2.3.1. Resource Creation

Resources, like communication groups, must be created. In the process of creation, a creator user provides an identifier for the resource, as well as information on the intended usage of the resource.

If we wish to maintain the property of customizability, then the end-to-end principle must be redefined. We no longer treat this as relating to network endpoints (nodes), but rather to the principal elements of users and resources.

Specifically, we establish two separate processes: the first is a user-to-resource process, in which the creator of a resource documents their intent. This is not fundamentally different from e.g. choosing a TCP address and port. The choice of TCP as a protocol documents a streaming intent, and the port typically correlates to a service protocol which defines how higher layer interactions are to occur.

The key difference is that in this architecture, this intent is not an ephemeral state of a single machine, but rather a permanent feature of the resource itself.

This creation cannot be advertised to the group, because prior to the existence of the resource, no resource specific group can exist. This appears to introduce a chicken-and-egg problem, which we will resolve later.

2.3.2. Resource Consumption

Resource consumption works quite similar to how it does in ICN: a consumer user posts an interest in a resource to its neighbouring nodes.

The interest specifies not only the resource identifier, but also the user identifier that expresses an interest. It may furthermore provide one or more node identifiers as routing information, and could even contain current convergence layer addresses for these nodes.

Recipients of an interest have have no obligation to store this interest as in ICN. They can respond in one of several ways.

1. If they have custodianship of the resource, they can decide whether to add the interested user to the resource group or not.
2. If they know or suspect nodes that may have custodianship, they can forward the interest to that node. If so, they should add the originating node identifier to the interest (if none is present), as well as the convergence layer address by which the interest was received.
3. If they are neither custodians nor can locate a custodian of the resource, they should return an error response to the interested consumer.

Adding a user to the resource group is a multi-step process.

1. First, custodians needs to check whether the user can join the group as a consumer.
2. If that is permitted, the resource data itself is updated to record that the user is now part of the resource group (this step may be omitted for public resources). This may mean distributing additional resource data within the resource group.
3. The custodian now signs the interest, and repeats it to the resource group. Depending on the convergence layer, this can result in several communication packets being sent along different channels.

Whether or not the custodian does permit this joining of a user to a resource or resource group can depend on many different factors. The custodian may be the node that currently hosts the creator of the resource; in this case, the creator can be explicitly asked to consent to this request.

Or the creator can have already added the user identifier to the resource; in that case, only the convergence layer operations for joining the resource group must be performed.

It's also worth highlighting that in principle it is possible to deterministically map a resource identifier to an IP multicast address, e.g. via an ORCHID v2 ([RFC7343]) or similar process. In that case, the interest can be posted directly to the resource group, and no routing of the interest to a more likely destination has to occur.

The key point here is not that all of the above steps have to be performed in precisely this order -- but rather that at the end of a successful initiation of consumption, the user identifier is recorded in the resource, and a matching node identifier has joined the convergence layer(s) groups.

In order to satisfy all of the use cases outlined above, an additional thing needs to happen: just like the resource creator needs to record intended usage into the resource, a consumption interest needs to specify desired usage. This can provide the second process, the resource-to-user negotation whether this desired usage matches the creator's intent.

This provides more information to the custodian node to decide whether joining the resource group is feasible. For example, when the consumer desires some bi-directional communications, and the creator just names some static data, that effectively represents a failed user-to-user negotation of the communications parameters.

Interests must also contain one or more of the following pieces of information:

- On the one hand, it is likely that a resource *is* in some way chunked up for better transport, even if it must be self-contained. An interest could be expressed for a (set of) particular content chunk(s) for the resource.
- Alternatively, the interest could be in the entire resource, which implies a subscription to updates to the resource -- either from the origin chunk, or from some chunk position specified in the interest.

At any rate, and unlike ICN, either of the above means that an interest can yield more than a single response. For this reason, nodes sending an interest must choose an identifier -- a cookie -- for the interest, which responses must contain. In this way, a single interest can be responded to multiple times.

For this to be manageable, interests must also contain a time stamp until which the interest is valid. Custodians must not respond to a timed out interest. Note that timeout of an interest, does not automatically imply timing out of group membership.

2.3.3. Data

Caches of a resource must ignore unsigned interests. Interests signed by a custodian of the resource must be responded to by sending data according to the interest, if the cache contains such data. Data responses must contain the interest cookie.

Care must be taken how to send data responses. While it is safe to assume that all members of a resource group share some interest in a resource, it is not a given that all the interests are equivalent.

Consider a video broadcast -- it is quite likely that when a user intends to join a broadcast, they wish to do so at the current time point. But perhaps they also wish to catch up with what has already been sent. Both are subscription interests, but they specify different starting content chunks (or perhaps none, in the case of the current time point).

To stay with the IP multicast example, it would not make sense to flood the multicast group with data some of the members would discard. For this convergence layer, it may be best to maintain multiple resource related groups -- one for sending interests to caches, perhaps, and one for each group of nodes that wish to consume the resource from (roughly) the same offset.

In other words, convergence layers require significant knowledge of the interest that data is sent in response to. Conversely, it is infeasible to suggest that data is sent to the entire resource group all the time.

Instead, data is sent to the convergence layer group(s) that this layer determines is best suited for the interest(s) at hand.

A few comments should be made on the distinction of custodians, caches, creators and contributors at this point.

Any node that hosts a creator or contributor acts as a cache, at least of the resource chunks that this user has created. Such nodes may also be full custodians. The key characteristic for the purposes of this section, however, is that they store some data, and can therefore send it in response to an interest, i.e. act as a cache.

2.3.4. Custodianship Provision

Where the previous sections have somewhat glibly assumed that the consumer and creator of a resource share some means to find each other, be they part of an IP multicast convergence layer group that can be derived from a resource identifier, or by some other means.

In order to provide the high intermittency tolerance property first and foremost, custodianship cannot merely lie in the node that happens to host the resource creator. At the same time, the mobility property requires that the network is not statically designed, but that nodes can be flexible in providing custodianship -- a static network design of custodians may not suffice here.

In order to find custodians, creators must send a custodianship request to neighbouring nodes. Again, what this notion of "neighbouring" entails is dependent on the convergence layer.

Nodes can respond in one or all of the following ways:

1. Generating a custodianship offer response to indicate that they wish to become custodians of the resource.
2. Forward a previously received offer of possible custodianship from another node.
3. Forward the custodianship request to other nodes it knows.

The specifics of the custodianship request and response are outside of the scope of this architecture.

A physically highly reglemented network may provide custodianship only from nodes operating on a specific converge layer protocol address. A logically highly reglemented network may provide custodianship only from nodes who can prove they are designated custodians by providing a signature of that fact from some mutually recognized authority. Other networks may provide more flexible custodianship.

A custodian node has two tasks:

1. First, it must permanently store any parts of the resource it receives. It may decide that it can best serve its purpose by also becoming a consumer of the resource in general, so that it accumulates all of the resource eventually. Note that a request for custodianship may request this behaviour explicitly.
2. Second, it must act on behalf of the resource owner to the best of its ability. As such, it may decide to permit consumers or contributors to join the resource.

In order to perform this job, authorization information that can ultimately be traced back to the creator must be embedded into the resource. For example, a resource may contain a section that explicitly marks a user identifier as a contributor or consumer. Or it may record other custodian nodes in the resource. It may delegate the ability to name other custodians to a particular (set of) custodians.

When a creator or authorized custodian accepts an offer, a custodianship acceptance response is sent to the newly inaugurated custodian. The same or equivalent may be sent to the resource group, and/or stored in the resource itself.

Custodianship, unlike group membership, is not technically a property of the resource itself. However, as the resource is shared amongst all group members in some way, recording primary custodians in the resource may be a convenient choice.

2.3.5. Custodianship Offers

Custodianship offers are generally the response to a custodianship request. Since custodianship requests pertain to an identified resource, the offer should typically also contain the same resource identifier.

It is also possible for nodes to spontaneously send custodianship offers for unspecified resources, to indicate capacity. Receiving nodes must store these offers, and respond with them as described in Section 2.3.4.

Offers are not permanent; they must be equipped with a lifetime. After an offer expires, nodes should discard them.

Note that there is no particular requirement for nodes to keep offers for a specific duration, or to keep all offers it receives; nodes can apply local policies here, including flood and DDoS protection policies, etc.

The main purpose of spontaneous offers is to pre-populate offer tables in nodes so that finding a suitable custodian can be accelerated. An implementation which forwards custodianship requests is equally viable.

Note that the above formulation of custodianship requests and responses makes it compatible with a large variety of convergence layer mechanisms.

In a local area network, for example, nodes may periodically broadcast spontaneous custodianship offers on the data link layer. Conversely, the criteria for custodianship selection are just wide enough to also e.g. permit mapping this mechanism onto a distributed hash table such as described in [KADEMLIA].

2.3.6. Custodianship Removal

Removal of custodianship effectively demotes a custodian to a mere cache. It is primarily information that needs to be communicated to the resource group, and so could be embedded into the resource.

In order to prevent situations in which custodianship is repeatedly accepted and removed by competing parties, we define that custodianship can only be removed by the party that accepted it, or by any party higher in the authority chain.

To illustrate this, assume that creator A delegated selection of custodianship to contributors B and C. B selects a custodian node CN.

Now contributer C cannot remove CN as a custodian. Contributor B could, or creator A could, because A initially delegated custodianship selection.

2.3.7. Data Removal

Data removal in a distributed system is difficult to guarantee. The preferred mechanism e.g. in ICN is to provide end-to-end encrypted data only, and then lose the encryption key, making data unrecoverable, which is similar in effect.

This mechanism is sound enough, but suffers from a data race. If a decryption key has leaked before legitimate nodes forgot it, the data remains accessible. Worse, it only remains accessible to illegitimate nodes.

Implementations are strongly encouraged to find complementary means to ensure data deletion. Some are discussed below.

1. In ICN, where resource chunks are addressed via a hash of their content, they are effectively immutable as any mutation creates a new content hash identifier, and so a distinct chunk.

   If resource chunks are mutable, on the other hand, zeroing out data is as valid a mutation as writing any other data, and so can help protecting plain text data (either in public resources, or the plain text prior to encryption).

2. Creators may send explicit deletion requests to caches and/or custodians.

A conforming implementation SHOULD provide as many of these complementary methods as feasibly to best provide data protection, but MUST provide at least the above three -- unless some future revision of this document obsoletes them.

2.4. Notes

A number of notes apply to this architecture which are not easily expressed either as elements or interactions.

2.4.1. Performance

Several of the desired properties can only be achieved by selecting an appropriate convergence layer protocol for the combination of the creator and consumer intents.

Assuming that the two parties wish to engange in a video call; the resource may then represent the call session. This is a high throughput, low latency scenario with bi-directional messaging. A choice of BP as a convergence layer protocol may not yield the desired results here, due to it's design of dealing primarily with high intermittency.

Conversely, a creator may create a live video stream, but a consumer may not care at all to watch it as it is being created. They may merely wish to record it for later consumption. Here, even though the creator's intent is similar to the above scenario, the consumer's intent relaxes the requirements and can make BP a more viable choice.

The key thing to stress is that it is the combination of the creator's intent (as embedded in the resource origin) and the consumer's intent that makes for the best choice of convergence layer protocol, and implementations MUST take this into consideration when choosing from available protocols.

It follows, then, that resources SHOULD encode such parameters, and nodes SHOULD compare these to their own capabilities and requirements.

Additionally, implementations MUST provide such convergence layer protocols as necessary to induce all of the desired properties in order to be considered a full implementation of this architecture.

2.4.2. Intermittency

The explicit custodianship mechanism described above is different from the one in DTN, in that it applies to storing and making available of a resource rather than to taking care of forwarding a message.

One implication is that the end-to-end principle is not violated by the contributor discarding a resource chunk after a custodian has received it.

The main distinction to DTN here is that in DTN, the sender of a message is still part of the communications flow. Moving the effective endpoint to a custodian which can then fail leaves little means for notifying the sender, and allowing it to find a contingency solution.

In this architecture, because resources are self-contained, once a contributor has transferred custody of a resource chunk, it is -- conceptually -- no longer involved in how the resource is being accessed; the end-to-end scenario is ended. When a consumer accesses a resource, a new end-to-end scenario is established.

That said, in order to achieve intermittency mitigation akin to DTN, custodians MUST explicitly acknowledge the receipt of all data. Such recepts should be made at the granularity of resource chunks, however, not at the data packet or message granularity of TCP vs. DTN.

2.4.3. Resources and Chunks

The above deliberately avoids being too detailed about how resources or their respective chunks may be identified. Interests can be in an entire resource, however, or in an individual chunk. It follows that these identifiers may occupy a shared namespace -- but no such requirement is imposed here.

It is worth emphasizing that the notion that a resource is subdivided into chunks is not necessarily given. A resource may consist soley of a single mutable chunk -- if so, then why distinguish between the chunk and resource?

Rather, the distinction is explicitly made so that implementations consider the ramifications of how resources should be represented.

One point to stress again is that resources MUST be self-contained. That is, information on which chunk(s) appear in the resource in which order must be embedded into the resource itself. Only then can we guarantee that no dependencies on additional architecture elements are introduced.

2.4.4. Multiple Convergence Layer Protocols

Each node may not only provide multiple convergence layer protocols, but may also use them simultaneously for a single resource. This implies the existence of a messaging abstraction in implementations whereby a node sends a message into a resource group. Each convergence layer can then forward the message according to its means.

If a receiving node receives the same message via multiple convergence layer protocols, it must discard duplicates of the message and process them only once.

If nodes A and B communicate via one convergence layer protocol, and nodes B and C via another, incompatible one, this does not pose a problem. What counts are that messages -- intents, data, etc. -- are forwarded, not that all nodes communicate in the same way.

2.4.5. Pivot Point

Due to the exchangable convergence layer protocols, we have a narrow waist in the architecture that is different from e.g. the Internet architecture, where the narrow waist is IP packets.

Downwards, towards the convergence layer, the narrow waist consists of the messages in a compatible protocol. Upwards, towards the user, the narrow waist consists of a self-contained, shared resource. The architecture does not place any constraints on the data embedded in the resource (with the exception of meta information discussed in this document).

It is likely necessary to acknowledge this dual layer narrow waist. That said, the leading abstraction is the self-contained resource. It is feasible that several competing messaging protocols exist that conform to this architecture.

2.4.6. Multiple Contributors

The notion that multiple parties can contribute to a single resource is unusual, and derives from group communications as the primary mode of communications. But it is also what makes this architecture effective at modelling real life user-to-user interactions.

This has some implications on how to model a self-contained resource, but this architecture should not be prescriptive of the means, only the effect.

In particular, it implies that updates to the resource should likely be structured in such a way that updates by multiple contributors do not conflict with each other. One set of methods for this are conflict-free replicated data types (an overview can e.g. be found in [CRDT]).

But just because the existence of multiple contributors is explicitly acknowledged and considered, this does not imply that every application of this architecture must in fact provide for multiple contributors. A single contributor/creator is equally supported. In such a scenario, a resource payload may also consist of a simple file of well-established type, etc.

2.4.7. Self-Contained Resources

This document stresses that resources must be self-contained, but it should hopefully be apparent at this point that this relates to not introducing dependencies on other users or nodes. It is perfectly fine for a resource to *refer* to other resources, e.g. in the same way that a hyperlink in an HTML document does.

On the other hand, it is equally possible for a resource to contain several multiplexed data streams, as is e.g. the case for most video file formats.

2.5. Analysis

With the elements, interactions and notes elaborated, we can now analyse whether this architecture induces all of the desired properties, and we must conclude that it does.

Note, however, that in the final sections of [INTERPEER-REQUIREMENTS], we conclude that some properties are not particularly desirable after all.

In particular, we must note that REST's *Visibility* property is somewhat incompatible with the HRPC properties. In particular, while the architecture does provide for some theoretical visibility into aspects of custodianship management, end-to-end encrypted resource payloads mean that the type of visibility that REST provides is not possible. But we must end-to-end encrypt in order to guarantee the HRPC properties.

We must, therefore, discard *Visibility* as a desired property. We similarly discard *Portability* in that it is a property best relegated to a higher layer. It may be feasible to design a portable application architecture which utilizes this current architecture as its transport, however.

Finally, we also discard the *Network Performance* property, largely because it is better described in other properties still included in the consideration.

User-Perceived Performance: Similar to the above, the choice of convergence layer may satisfy this property. That said, the architecture explicitly encourages caching of data, as well as custodians. In this manner, nodes are free to select the lowest latency data cache available to them, which may be local.

Network Efficiency: See the use of caches above.

Scalability: Scalability is induced in multiple ways: - In much the same way as REST provides scalability by separating interactions around specific services, this architecture separates interactions around specific resources. - The use of custodianship management is chosen to make it optional (though necessary for inducing other properties). Additionally, it is designed so that not every node needs to be aware of every custodian and vice versa.

Simplicity: The amount of elements and interactions is deliberately kept low. Complication is introduced by the interdepencence of the messaging layer with the convergence layers, but this is necessary complication in order to induce other properties. All things considered, the architecture is simple enough.

Evolvability: The architecture establishes interaction patterns and some requirements on individual elements, but does not perscribe how these requirements are fulfilled. It thus provides evolvability of the system.

Extensibility: The architecture defines a minimum set of different roles and interactions, but does not limit extensions. Once group messaging is implemented, additional messages can be added to permit extensions, different groups from the one(s) described can be created, etc.

Customizability: The client does not so much initiate server behaviour as it negotiates its intent with the intent recorded in the resource. The server should chose to respond such that both intents are satisified.

Reusability: Components are re-usable in the same sense as REST. The caches provide a uniform interface, and could so also be implemented as proxies. This implies the inclusion of a user agent-like component, which may then communiacte with downstream caches.

Resilience: As in REST, the architecture provides resilience against failure of components not involved in a particular resource's group. Unlike REST, the custodianship transfer mechanism provides additional resilience against failures of all but one custodian and all caches.

High Intermittency Tolerance: The custodianship transfer mechanism as well as the requirement that resources are self-contained provides for high intermittency tolerance.

Low Intermittency Utilization: The selection of convergence layer protocols based on consumer and creator intents permits for effective utilization of low intermittent connections.

High Latency Tolerance: Treating resources as self-contained and mutable effectively introduces a high tolerance for latency, in that a resource is always "complete". Whether the current node has all the parts that other nodes have does not affect this notion of completeness.

Low Latency Support: (((low latency support)) Where low latency convergence layer protocols are available, they can be utilized, supporting applications with low latency requirements.

Low Reliability Tolerance: Tolerance for low reliability is provided in - Not assuming a 1:1 relationship between a node and a resource, i.e. allowing for fallbacks, - allowing for multiple convergence layer protocols that may or may not provide a measure of reliability, and - encoding protocol logic independent of convergence layer, and authorization logic idependent of the protocol layer, such that failures in one do not need to affect the other.

High Reliability Support: Support for high reliability is provided via explicit custodianship management and the base group communications.

High Throughput Support: Much as with latency, high throughput convergence layers can be chosen according to the needs of the consumer and the intent of the creator.

Inconsequential Throughput Tolerance: (((inconsequential throughput)) Similar to reliability, high throughput is not a necessary choice. The architecture does not require high throughput in its design, and so permits for implementations to schedule low throughput resource streams.

Integrity: The requirement of resources to be signed by contributors provides integrity.

Authenticity: The signature above also provides authenticity.

Confidentiality: Cryptographic confidentiality is provided by end-to-end encryption of resources. Other meanings of confidentiality are supported, in that resource creation and contribution are, at the abstraction of the architecture and protocols implementing it, conscious user choices. Applications MUST NOT implicitly share resources for this property to be maintained.

Security: Where appropriate for an architecture document, [BCP72] is followed. Implementations MUST ensure further compliance themselves.

Privacy: The considerations of [RFC6973], Section 7 are reflected to the best an architecture document can.

Anonymity: Anonymity is provided by making user identifiers for consumption relatively irrelevant; contributors are identified. However, each identifier can be ephemeral and limited to (a subset of) a resource.

Pseudonymity: User identifiers are fully pseudonymous.

Unlinkability: Ephemeral user identifiers provide unlinkability.

Censorship resistance: The architecture utilizes caches and custodians in order to ensure that a resource can exist in multiple places, making censorship resistance difficult here. Furthermore, ephemeral user identifiers make it difficult to censor individual people.

Accessibility: Many accessibility concerns are outside of the scope of an architecture. However, as the architecture places no constraints on resources or identifiers that make them particularly inaccessible, we can consider this property fulfilled.

One note should be made about identifiers that are hashes of something, such as e.g. user identifiers. A hash is not a particularly accessible datum. However, this architecture does not require that such data are visible to the user at all -- a hash may be an identifier in the protocols, but may be represented by a human readable and screen reader friendly string in the user interface, for example.

Mobility: The distinction between node identifiers and convergence layer addresses provides for mobility.

Multi-Homing: The same distinction permits for multi-homed nodes.

Self-Determination: Nodes are free to select how they communicate; they can reject data or interests from other nodes as dictated by local policy.

Self-determination is also guaranteed at the user layer, with some caveats. In particular, the model of resource ownership by the creator implies that contributors cannot force their way into contributing to a resource; in that sense, their self-determination is limited. But they are not equally limited when it comes to resources they create themselves.

Interest Grouping: The definition of group communications along the shared interest in a resource directly supports interest grouping.

As the above list demonstrates, this architecture incudes all of the properties defined as desirable based on the problem section, with the exception of the visbility property from REST.

The key points that make this architecture distinct from previous attempts and contributes to these results are:

1. The architecture is focused on user-to-user interactions, which are group efforts. Endpoint-to-endpoint interactions are described, but in the service of the above.
2. The architecture pivots around the notion of a group resource, which members can contribute to and/or consume.
3. The inclusion of user-to-user interaction implies the existence of user identifiers, which provide the hooks for making such resources end-to-end encrypted by default.

3. IANA Considerations

This document has no IANA actions.

4. References

4.1. Normative References

[INTERPEER-ARCHITECTURE]

Finkhaeuser, J., "Architecture for Human-Centric Networking", PIE PIE.f92f09.architecture-00, 19 January 2026, <https://specs.interpeer.org/PIE.f92f09.architecture/PIE.f92f09.architecture-00>.

[INTERPEER-PROBLEM-STATEMENT] Finkhaeuser, J., "Problem Statement & Gap Analysis for Human-Centric Networking", PIE PIE.f92f09.problem-statement-00, 22 July 2025, <https://specs.interpeer.org/PIE.f92f09.problem-statement/PIE.f92f09.problem-statement-00>.

[INTERPEER-REQUIREMENTS] Finkhaeuser, J., "Gap Analysis & Requirements for Human-Centric Networking", PIE PIE.f92f09.gap-analysis-00, 19 January 2026, <https://specs.interpeer.org/PIE.f92f09.gap-analysis/PIE.f92f09.gap-analysis-00>.

[PIE.f92f09.00] Finkhaeuser, J., "PIEs - Proposals for Interpeer Enhancement", PIE PIE.f92f09.00-00, 14 March 2025, <https://specs.interpeer.org/PIE.f92f09.00/PIE.f92f09.00-00>.

4.2. Informative References

[BCP72] Best Current Practice 72, <https://www.rfc-editor.org/info/bcp72>.
At the time of writing, this BCP comprises the following:

Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

Gont, F. and I. Arce, "Security Considerations for Transient Numeric Identifiers Employed in Network Protocols", BCP 72, RFC 9416, DOI 10.17487/RFC9416, July 2023, <https://www.rfc-editor.org/info/rfc9416>.

[CRDT] Preguiça, N., "Conflict-free Replicated Data Types: An Overview", arXiv, DOI 10.48550/ARXIV.1806.10254, 2018, <https://doi.org/10.48550/ARXIV.1806.10254>.

[I-D.draft-trossen-rtgwg-rosa-arch] Trossen, D., Contreras, L. M., Finkhäuser, J., and P. Mendes, "Architecture for Routing on Service Addresses", Work in Progress, Internet-Draft, draft-trossen-rtgwg-rosa-arch-01, 9 July 2023, <https://datatracker.ietf.org/doc/html/draft-trossen-rtgwg-rosa-arch-01>.

[ISOC-FOUNDATION] Internet Society Foundation, "Internet Society Foundation", n.d., <https://www.isocfoundation.org/>.

[KADEMLIA] Maymounkov, P. and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric", Springer Berlin Heidelberg, Lecture Notes in Computer Science pp. 53-65, DOI 10.1007/3-540-45748-8_5, ISBN ["9783540441793", "9783540457480"], 2002, <https://doi.org/10.1007/3-540-45748-8_5>.

[NGI-Assure] PNO Digital Srl, "NGI Assure", DOI 10.3030/957073, Grant Agreement ID 957073, 31 August 2024, <https://doi.org/10.3030/957073>.

[NGI0-Discovery] Stichting NLNet, "NGI Zero Discovery", DOI 10.3030/825322, Grant Agreement ID 825322, 1 November 2018, <https://doi.org/10.3030/825322>.

[NIST.FIPS.186-4] "Digital signature standard (DSS)", National Institute of Standards and Technology (U.S.), DOI 10.6028/nist.fips.186-4, 2013, <https://doi.org/10.6028/nist.fips.186-4>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/rfc/rfc1035>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/rfc/rfc6973>.

[RFC7343] Laganier, J. and F. Dupont, "An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers Version 2 (ORCHIDv2)", RFC 7343, DOI 10.17487/RFC7343, September 2014, <https://www.rfc-editor.org/rfc/rfc7343>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/rfc/rfc8032>.

[RFC8410] Josefsson, S. and J. Schaad, "Algorithm Identifiers for Ed25519, Ed448, X25519, and X448 for Use in the Internet X.509 Public Key Infrastructure", RFC 8410, DOI 10.17487/RFC8410, August 2018, <https://www.rfc-editor.org/rfc/rfc8410>.

Acknowledgments

Development of this document started as work undertaken under a grant agreement with the Internet Society Foundation [ISOC-FOUNDATION], but has since seen a number of revisions. Some revisions are inspired by work undertaken under grant agreements from Horizon Europe, specifically [NGI0-Discovery] and [NGI-Assure].

Copyright Notice

Index

Author's Address

Jens Finkhäuser
Interpeer gUG (haftungsbeschraenkt)
Email: ietf@interpeer.org
URI: https://interpeer.org/