Open Risk White Paper

Connecting the Dots: Tensor Representations of ActivityPub Networks

Authors: Philippos Papadopoulos

February 2, 2024

www.openriskmanagement.com

The open future of risk management

SUMMARY

In this third Open Risk White Paper on *Connecting the Dots* we explore representations of online communication networks that are organized according to the ActivityPub protocol. We discuss the main relevant features of the protocol and the broader application ecosystem around it that shapes emerging online network topologies. We develop a stylized description of ActivityPub compliant networks as a mathematical multilayer network. Tensor representations of such complex graphs generalize the more familiar matrix algebra tools and can be useful in various ways: On the one hand they help empirical work in analyzing the characteristics of such networks, on the other they enable simulating and exploring network behavior.

Further Resources

- The Open Risk Manual is an open online repository of information for risk management developed and maintained by Open Risk. Various concentration risk concepts mentioned in this White Paper are further documented and explained using dedicated Open Risk Manual entries.
- The Open Risk Academy offers a range of online courses around risk and sustainable portfolio management, which utilize the latest in interactive eLearning tools.
- The Open Source Repository contains a variety of open source tools and resources.
- More content: Open Risk White Papers and Open Risk Blog

About Open Risk

Open Risk is an independent provider of training and risk analysis tools to the broader financial services community. Our mission is captured by the motto: The open future of risk management. Learn more about our mission and projects at: `www.openriskmanagement.com`

---

1 Introduction

In recent times, supported by developments in information technology, there is increasing focus among both academics and practitioners that aims to *connect the dots* by analyzing economic phenomena using *graph theory* and *network analysis*. Such developments create more detailed understanding of the structures and interactions between economic agents, ultimately guiding towards better policies and risk management. Examples of economic networks already studied include the system of interconnected financial institutions, where potentially adverse linkages are an issue. This topic rose to prominence after the financial crisis of 2008 (e.g.,[1, 2, 3] and more recent studies in [4, 5]). The explicit modeling of financial contracts as networks is now a fruitful line of inquiry ([6, 7, 8]). In broader economic context, networks show up e.g., in the analysis of core/periphery networks ([9]), the structure of multinational affiliate networks ([10]), shadow banking ([11]), collecting relevant economic statistics ([12]) and the underpinning of modern flow-of-funds analysis ([13]). Empirical analysis of economic networks is used also to understand how interlinked households and non-financial firms affect borrower delinquencies and defaults which ultimately affects the health of the entire system. ([14, 15]).

An important aspect of the above-mentioned economic networks is that they are, by-and-large, *implicit*. As information structures they do not exist in one integrated data universe. Rather, they materialize as the cumulative effect of disjoint activities such as: payments (electronic or in cash), as updates to corporate accounting systems and as many other contractual bookkeeping operations. In general only a fraction of that activity will be fully digitized and even smaller subsets will be available as a single data set.

In the meantime, network analysis is of growing importance also in many other disciplines outside finance and economics. The list includes chemistry, biology, epidemiology, sociology and communication networks. In all of these fields there are growing bodies of work that make use of generalized graph structures as representations of complex decentralized networks. Notably, the explosion for internet enabled communications and Web technologies has drastically increased the need to understand such systems. This has spurred the development of relevant tools from mathematics and data science. The emergence of large-scale social media networks, in particular, has spurred data intensive Social Network Analysis where the network structure is fairly explicit (available) and large data sets represent it fairly accurately [16].

Our focus in this White Paper is motivated by important recent developments in online networks[1] which use both existing and new Web technologies to establish novel *decentralized* communication patterns. The primary current example of such novel Web 3.0/Fediverse applications concerns new architectures for *social media*. The general pattern is an ensemble of distinct network elements which communicate with each other even while remaining independent platforms. A key enabler of this emergent web networking pattern is the *ActivityPub* protocol. This is a specification towards decentralized (more precisely, federated) social networking [17], based upon the exchange of ActivityStreams [18] messages that follow the Activity Vocabulary [19]. The ActivityPub proposal has been standardized and published by the W3C and has motivated the design of several federated social networking systems.

There are presently several concrete ActivityPub compliant implementations and the protocol sees meaningful adoption, primarily in the domain of federated social networks. Current ActivityPub networks consist of several million individual users, using several thousands of distinct servers, and a few dozens

[1] The domain is sometimes captured under the banner of *Web 3.0* or *Fediverse*, though these labels are not rigorous and are evolving.

of distinct server *types* [20]. Different ActivityPub server types might target a variety of use cases, or simply reflect different underlying technologies. Use cases include exchanges of smaller or larger pieces of text (termed Microblogging/Blogging), cataloging and reviewing of Books, Podcasting, the creation and sharing of Images or Videos, shared Calendar and Event Planning applications, Music Hosting, Bookmark or Link Aggregators, Discussion Forums and more. Federation of collaborative software development activities (based on the git paradigm and tools) is another important work-in-progress.

ActivityPub is built on top of the widely used HTTP (web) protocol which removes the need to install new software or to develop new networking libraries. It uses JSON as the data exchange format which is now a de-facto standard for online data exchange. As a W3C (World Wide Web Consortium) standard, it benefits from a degree of stability, assurance and potentially forward interoperability with other relevant web protocols.[2]

At the same time, ActivityPub does not specify all ingredients of fully deployed and interoperable information exchange networks. The precise form and nature of decentralized data storage and digital identity are not part of the specification, and it lacks a fine-grained *object-capability* model. As more concrete applications are pursued they raise requirements for more refined specifications, moderation, integration of payment methods, and collaboration mechanisms. As a result the framework is still actively developed [21, 22]. In particular, there is potential integration of social network infrastructure into a broader range of online activities with a possibly more explicit economic footprints. Thus, the two distinct strands of network phenomena we discussed here, the ubiquitous but still implicit economic networks and the emerging, decentralized, social networks come together in what one might term a new *decentralized, digital, socio-economic network* category.

> A decentralized, digital, socio-economic network would be defined as loosely coupled online infrastructure, with distributed control and ownership by diverse agents, which supports online activities and behavior such as the publishing and discovery of information, the establishment of interest-based relations and groups, and the exchange of digital services where those activities and exchanges may have an explicit economic dimension.

1.1 Motivation and Objective

The feasibility, likelihood, pace of adoption and precise shape of digital socio-economic networks remains open. An example of further specifications that aim to shape more concretely this emerging area is available here [23]. Security and privacy issues, other design trade-offs and choices that determine which information is distributed (and how) are still unresolved. More fully-fledged object capabilities[24] and access control models would provide technical foundations from which various unanticipated use cases can be met. Nevertheless, given the central enabling role and rapid maturing of digital network technologies, not least as seen at work during the recent pandemic, there is sufficient motivation to explore this emerging domain further.

More specifically, in this third Open Risk White Paper on *Connecting the Dots* [25, 26], we develop mathematical representations of federated online networks that help encode succinctly certain important

[2] While ActivityPub is a popular option it is only one of many federation network protocols under development (SoLiD, Diaspora, OStatus, DFRN, Zot/6, etc. being alternatives) and related Peer-to-Peer or distributed protocols that receive attention. There are also protocols, for example the Matrix and XMPP protocols that are federated but are not part of the ActivityPub network.

elements of the structure of such networks. In turn this enables using well-known mathematical operations from graph theory and network analysis. We focus on federated networks adhering to the ActivityPub protocol, which we will discuss in the relevant detail. Graph theoretic representations of ActivityPub networks have already been used recently to enable the consistent capture (data modeling) and statistical analysis of public fediverse data [27, 28, 29]. Supporting network data analyses is one key motivation for this work. Another motivation is enabling the theoretical modeling and *simulation* of ActivityPub network properties and behaviors (see e.g., [30]). Mathematical representations of ActivityPub networks can thus help the development of federated network patterns as they scale both the number of users, the variety of uses and the intensity of data exchange. In upcoming years the volume and distribution of data over such networks is likely to grow exponentially. The integration of, for example, *algorithmic agents* - which is explicitly envisaged in the ActivityPub specification - is likely to create significant traffic of its own. While the presence of automated, algorithmic actors in social networks and their potentially harmful impact is currently debated actively in the context of centralized digital platforms, tools that assist human users with managing ever-increasing information flows will require calculation agents (e.g., for discovery purposes [31]) of some sort. Faithful graph representations of data availability and network topologies can assist with the design of relevant and adapted algorithms in this direction.

With the above motivating factors, our objective here is to develop a concrete (if somewhat stylized) description of decentralized networks based on the ActivityPub protocol. The broader toolkit we will tap into is going under a variety of names, such as *multilayer networks*, *temporal graphs*, *multiplex networks* or *multidimensional graphs*. Good overviews of the mathematics involved and links to open source software that enables working with such structures are given in ([32, 33]).

1.2 Structure of the White Paper

- In this Introduction section we place the white paper in the context of contemporary developments around online networks and motivate the need for mathematical representations to help better understand growing information flows.
- Section (2) summarizes the technical specification of ActivityPub, focusing on the elements that are important for establishing network representations and mathematical graphs.
- Section (3) illustrates concretely how ActivityPub networks can be modeled using classic (simple) graph structures. This is an important preliminary step before we discuss more elaborate network models.
- Finally, Section (4) covers the concept of multi-layer networks and the associated tensor representations in the context of ActivityPub network patterns.

2 ActivityPub Network Structure

We review here some parts of the broader ActivityPub related specifications [17, 18, 19] with the objective to identify the aspects that determine most directly the *connectivity patterns* of compliant online networks.

2.1 A Bird's Eye View of ActivityPub

The list of core definitions, design choices and architectural constraints that are relevant for our purposes includes the following:

- **Client-Server Architecture**: The ActivityPub pattern suggests a Client-Server type network architecture. The specification provides both a Client-to-Server API for creating, updating and deleting online (web) content, and a federated, Server-to-Server, API for delivering notifications and content between Servers.
- **HTTP POST/GET verbs**: An ActivityPub network consists of digital devices (both Clients and Servers) connected over the internet and interacting via well-defined API's using exclusively HTTP POST/GET verbs.
- **Implementation Agnostic**: Client and Server implementations are not specified in any detail. They simply need to comply with a set of federation requirements when communicating within the network.
- **Actor Objects**: Most individual users participating in the network are represented by the *Actor* data model.[3] Actor objects represent participating users (might be persons or other digital agents) that exchange information in a voluntary (user initiated) and intermittent manner.
- **Actors are Server Specific**: The same real user may have accounts on multiple Servers, but each of these Actors is a unique object (e.g., with a unique network address).
- **Server based Authentication/Authorization**: Access authority to networked resources is regulated by Servers through the ability to login and access / manipulate objects.
- **Activity Objects**: Activity objects are messages that codify the range of possible user (Actor) *actions*. Different activity types codify the nature of the message exchanges (Posts, Follow requests etc.).[4]
- **Wrapped Data Objects**: The specification requires that the transmission of any actual data objects (Text, Images, Audio etc.) is being wrapped inside *Activities* objects.
- **JSON-LD Format**: Servers exchanging data objects must use the ActivityStreams object representation that is based on the JSON-LD format (media type: application/ld+json)[34].
- **Global URL's**: All data objects being *distributed* over the ActivityPub protocol must have unique global identifiers (with some exceptions for transient objects). These identifiers must be publicly de-referenceable endpoints (for example they must be secure HTTPS URL's).
- **Actor Inbox and Outbox**: Actor objects must have, in addition to various object identifiers, two mandatory properties: An *Inbox*, which is a reference to an ActivityStreams collection comprising all the messages *received* by the Actor and an *Outbox*, which is an ActivityStreams collection comprising all the messages *produced* by the Actor. These properties are the Actor *ports* through which they connect with other Actors.

[3] Administrators or other privileged users are not explicitly modeled.

[4] Activities are the recognizable verbs of the ecosystem as far as network users are concerned. As mentioned, the only underlying technically implemented verbs are the two HTTP GET and POST actions.

- **Client-Centric**: In ActivityPub it is Clients that initiate communication sessions with Servers (A Pull not Push Model). Specifically, Client to Server interaction takes place through Clients posting Activities to an Actor's Outbox.
- **Atomic Activities**: POST requests from Clients or Servers contain each a single Activity (but an Activity may contain multiple embedded objects).
- **Message Delivery**: Servers await incoming requests from other Servers or Clients. They must perform delivery on all Activities posted to the Actors outboxes. Depending on the type of Activity, Servers may be required to carry out further side effects (e.g., modify the state of a database).

The above short list of important ActivityPub elements and choices provides a quick sketch of what the ActivityPub network looks like. It is primarily the Server-to-Server API that most directly determines the federated network structure, hence it will be our primary focus here[5] For concreteness, an illustrative realization of an ActivityPub network would be a collection of running Web Servers (websites), connected to each other over standard internet infrastructure; The network will feature a population of human users, accessing those Servers via, e.g., their web browsers, which are running on home computers or mobile phones; Users would connect to Servers and produce / share information via their Clients, and finally federated Servers would distribute that information to other users who consume it through their own Clients. We will now discuss in more detail some of the key components more relevant for the network graph model.

2.2 ActivityPub Server Implementations

ActivityPub Servers (also called Instances, or Pods, or Services) are the fundamental building block of ActivityPub networks. Every active Server running on a host computer will have a valid, https-enabled domain or subdomain. A host machine will typically be a networked computer that runs one or more ActivityPub Server programs and listens to certain network ports.

In order to federate (reliably exchange information) with an existing ActivityPub network, a Server has to implement the Server-to-Server protocol. Servers are set up and controlled by administrators who have the authority to e.g., start or discontinue any service running on a host computer. They also have, in principle, access to any network user data that is routed through the Server (anything that is not encrypted).

The ActivityPub specification does not constrain Server designs beyond the basic required functionality that enables federation. Concrete Server implementations have to make choices on a range of important issues (indicatively):

- The number of (regular) users that have access to a Server (single-user versus multi-user instances). At least one Actor is required for a normal Server to be operational in the network, but there is no upper bound. Servers will in general have privileged users (moderators) and many related policies and tools. The internal, within-server, flow of information and detailed side-effects is not described in the protocol.

[5] Hence elements out-of-scope are i) the Client-to-Server API, ii) a significant part of the Server-to-Server specification that concerns how information propagates over the network (delivery) and iii) (m)any technical implementation details.

- On the type of media and formats that get exchanged, processed and stored through Activities (Text, Image, Audio, Video, etc.). In other words, while ActivityPub specifies the data envelope shape and the addressing and delivery mechanisms, it does not specify what to do with the envelope *content*.
- How to persist the state of the federated communication network using e.g., Server host filesystems and/or different types of databases.
- Which privacy controls to implement and the demarcation of public versus private boundaries more generally.
- What kind of network federation policies to apply (blacklists and/or whitelists of other Servers).

These choices are fairly important in concrete applications and have implications on the network structure and information flow patterns. In this work we aim to keep an agnostic stance versus such Server specific details.

2.3 ActivityPub Object Hierarchy

Within and between each running Server, somewhere in their computer memory, or in transit between Servers there will be the data objects originated, distributed and stored in the network. The specification provides a high level data model for those objects. It sketches certain logical object relations and hierarchies (one can think of it as an Ontology (in the Web Ontology Language sense) or alternatively as an UML sketch of an Object-Oriented class structure). This part of the specification is useful as a starting point for Server *implementations*. With a few important exceptions, it does not play a direct role in determining the *topology* of the federated Server network but is nevertheless useful to have as background knowledge.

- **Object**: Describes an ActivityPub data object of any kind. The Object type serves as the base type for most of the other kinds of objects defined in the *Activity Vocabulary*, including other Core types such as Activity, IntransitiveActivity, Collection and OrderedCollection.
- **Link**: A Link is a reference to a resource identified by a URL. Many of the properties defined by the Activity Vocabulary allow values that are either Objects or Links. When a Link is used, it establishes a relation connecting the subject (the containing object) to the resource identified by the link.
- **Activity**: An Activity is a subtype of Object that describes some form of user action that may happen, is currently happening, or has already happened. The Activity type itself serves as an abstract base type for all types of activities and does not carry any specific semantics about the kind of action being taken.
- **IntransitiveActivity**: Instances of IntransitiveActivity are a subtype of Activity representing intransitive actions. They do not involve an object, hence none is provided.
- **Actor**: Formally, participants of ActivityPub networks are conceptualized as Actor objects.
- **Collection**: A Collection is a subtype of Object that represents ordered or unordered sets of Objects or Links. A collection is simply a logical container of objects (similar to arrays or lists in programming languages).

- **OrderedCollection**: Is another subtype of Collection, whose members are assumed to be strictly ordered (e.g., in chronological order).
- **CollectionPage**: An object used to represent distinct subsets of items from a Collection (useful for presenting to users manageable chunks of information).
- **OrderedCollectionPage**: Same as the CollectionPage, but used to represent ordered subsets of items from an OrderedCollection.

2.4 Actors

As mentioned already, individual users are represented by an Actor data object. An Actor is the entity that is modeled as performing an Activity (e.g., posting a message to the network). This object must somehow get stored in the Server. Actor objects will be central elements in our graph models, typically represented as the nodes of various networks. Normal network users are represented by Actors, but as already alluded, administrators, moderators and potentially other special users may have interactions with Servers that are not captured in the specification. Importantly, user accounts on different Servers correspond to different Actors.

The Actor ID is a valid URL that describes the entity. An Actor is documented via a publicly accessible JSON-LD document (effectively an online user profile). The Actor object can be queried with a GET request to obtain the profile. The Actor data object gives identification information (e.g, the username). An ActivityPub Actor has a mandatory Inbox and Outbox properties. This is an essential and important attribute (discussed more below). Servers can POST to an Actor's Inbox to send them a message (as part of an Activity) and Servers can GET from an Actor's Outbox to see what messages they have posted (subject to authorization).

2.4.1 Actor Types

An Actor might represent real persons (whether a person acting as individual, a representative of organization or other grouping of people). But an Actor can also represent a software application (for example a bot with various levels of ascribed intelligence). The diversity of Actor types is captured by the following range of possibilities indicated in the specification:

- **Person Actor**: Represents an individual person. E.g., social media accounts that represent humans using their personal computing devices to access the network.
- **Group Actor**: Represents a formal or informal collective of Actors (e.g., members of an online forum).
- **Organization Actor**: Represents an organizational entity. These are types of online accounts that represent a real-world or digital organization. This will typically be operated by one or more designated humans (e.g. a social media manager) that have delegated authority to engage in the network on behalf of the organization.
- **Application Actor**: Describes any software that gets authorized to operate as a programmatic participant in the network. These might be, e.g., algorithmically driven agents or other pieces of software. The simplest examples would act as automated publishers or repeaters of content, while more advanced generations may include bots with more advanced capabilities.

- **Service Actor**: Represents a service of any kind. These may include for example IoT devices providing sensor measurements. Seen as network Actors, these sensors may be considered as passive nodes, focusing on data and context provision, as part of a heterogeneous digital network.

Actors may have additional properties depending on their type. E.g., People Actors may have detailed names, other identifiers, gender, age or ethnicity information, location or device data, etc. Organizational Actors may alternatively have identifiers, category of business, size, foundation date, mission statements, location, etc. Application Actors may have identifiers, functionality description, source code documentation, version information etc. Finally, automated IoT services may have identifiers, manufacturer info, indicators of available connectivity protocols etc.

2.5 Activities

Along with the all-important Actor object we just discussed, Activities are another central object in ActivityPub (it is in the name after all!). The Activity object acts as a conduit for delivering the network's objective: enabling the exchange of standardized information. An Activity is realized via a message that may or may not include a more detailed data object. It captures some form of action (past, present, or future intention). Every message sent on behalf of an Actor has a Link to the Actor's JSON-LD file. Any Actor receiving the message can look up relevant information and start interacting as appropriate.

2.5.1 Activity Format

The schema used for expressing activities in ActivityPub makes use of the ActivityStreams specification. All exchanged Activities must respect this format, otherwise there is no guarantee they will be understood by federating Servers. In turn ActivityStreams is expressed as JSON-LD (JSON for Linked Data) format. JSON-LD is an extension to plain JSON (Javascript Object Notation) which extends JSON objects with hypermedia capabilities. The *context* mechanism (a special field) provides a window to the Linked Data and Semantic Web universes. This enables, for example, Activity payloads to contain metadata links in standardized and machine-readable ways. These links, in turn, enable potentially a broader and more flexible *Semantic Network*, where federated Servers and Clients make use of a wider cloud of information services (e.g., Open Data) and are less constrained by fixed data formats and vocabularies. This design decision may potentially help the exchange of semantically more complex, not a-priory explicitly specified Activities.

2.5.2 Activity Types

The Activity object type serves as an abstract base type for all types of Activities. There is a vast range of Activities that can be represented (for example, Create, Update, Follow, Like, Block, etc). In addition, implementations are free to introduce new desired types of Activities beyond those defined by the Activity Vocabulary. Activity objects enable both passive and imperative operations. In the passive sense, the Activity is used to record that an Activity has occurred or is occurring. In the imperative sense, the Activity can be used as a form of command. It instructs the Server to modify the network state in a manner consistent with the action being described.

2.6 Actor Inbox and Outbox

On an ActivityPub compliant Server, individual user accounts (Actors) have both an Inbox and an Outbox endpoint. These API endpoints accept HTTP GET and POST requests (they are URLs that are accessible on the Server and listed in the ActivityPub Actor's profile description). Actor-to-Actor message exchanges in ActivityPub networks are thus somewhat analogous to email communication and the choice of language aims to suggest that. The Inbox and Outbox properties are live parts of the system and a cornerstone aspect of the implied ActivityPub Social Graph. They are essentially the *endpoints of the graph edges*.

More specifically, the Inbox is the mechanism that enables Actors to *receive* Activity messages from the ActivityPub world. The Inbox URL Link is discovered through the Inbox property of an Actor's profile. Inboxes of Actors on federated Servers accept HTTP POST requests that send such messages. The Inbox stream will contain all Activities received by the Actor. The user/owner of an Inbox is generally able to access all of their Inbox contents. On the other hand, the Outbox is how Actors *send* Activity messages to other Actors in the network. The Outbox link is discovered through the Outbox property of an Actor's profile. A Client GET request from a Actor's Outbox URL will receive a JSON-LD list of Activities. The Outbox will be an OrderedCollection which represents a chronologically ordered list of Activities. In summary, we have the following actions:

- **GET from Inbox URL**: Actor receives new Activities sent to them by network Actors.
- **POST to Inbox URL**: Servers push Activities intended for the Actor.
- **GET from Outbox URL**: Other network Actors get Activities created by the Actor.
- **POST to Outbox URL**: Actor publishes new Activities.

2.6.1 Following and Followers

According to the protocol, Actors should have (but need not have) the *Following* and *Followers* properties. While not mandatory, these properties are the raison-d'être of a federated network. Without them there is no real network structure. An alternative wording of those properties that abstracts away from social media usage might be *Subscribing to* and *Subscribed by*. The Following property of an Actor is a Link to an ActivityStreams collection of Actors that this Actor is Following (subscribing to). It expresses the interest of an Actor to be informed (be passed messages by this list of other Actors). It also captures the fact that a list of other Actors have consented to be followed. The Followed property is a Link to an ActivityStreams collection of the Actors that follow (subscribe) to this Actor. It expresses the interest of other local or remote Actors to be informed by means of Activity messages from this Actor.

2.6.2 Moderation

This is a specialized topic in the context of the present discussion, but it is instructive to discuss the level of additional detail that might be required before one has a realistic representation of what happens in such as federated network in terms of actual information flow. Broadly speaking, in social network parlance *moderation* is a set of policies available to privileged users (administrators or moderators) that can modulate both the *graph evolution* (who can see and connect with whom) and the *propagation of information* (what messages are seen by whom). This topic is particularly relevant for the user experience.

It is discussed extensively in the context centralized social networks but applies as well to federated networks [35]. Growing user bases creates challenges for administrators [36].

In terms of moderation primitives, ActivityPub defines in this direction the *Block Activity* (within the user "Reaction" category of Activities). This action indicates that the Actor is blocking an object. The typical use of this functionality is to support systems that allow a user to block Activities or content of other users. The blocking of other Servers by Server administrators (termed *defederation*) is outside the scope of the ActivityPub specification but is important for defining the network structure. A Blocking activity can in principle be conceptualized as a negative edge [27]. Such a network can be captured, for example, as a weighted directed graph, with the weights being $w \in (+1, -1)$.

In practice, the degree of interoperability within ActivityPub networks depends on the *ability and willingness to federate*. Ability reflects foremost the implementation choices of respective software platforms. Willingness reflects rather administrative choices of individual Servers.

3 Fundamental Graph Correspondences

We now have enough of a sketch of how ActivityPub network work to formulate the core correspondence to mathematical graphs. The salient features of ActivityPub networks that we need to capture first are the following:

- Actors exist within Servers.
- A Server must have at least one Actor to be part of the network.
- Connections between Servers are established through Actor actions. Servers do not proactively discover and network with other Servers of their own accord. Modifications of the Social Graph are linked to Actor actions.
- Actors receive and send information messages through Server operations.

The ActivityPub social graph is thus constructed and expanded via the Following and Followers collections of each Actor. Practically this happens when a user on one Server finds and follows a user on another Server. Servers maintain a list of remote Actor accounts that their own users/Actors follow and subscribe to their posts. Servers communicate with other Servers and *propagate information* across the social graph by posting Activities to Actors' Inbox endpoints.[6]

[6] In reality the information diffusion patterns in popular current ActivityPub implementations are more complicated: To reduce resource requirements for typical social media use cases (e.g., supporting a public *timeline*) a number of concepts such as shared inboxes and public addressing have been added to the protocol.

The term Social Graph in the context of ActivityPub networks has the following two distinct meanings:

1. The *full graph* that is the union of all decentralized sets of nodes and edges across all federating Servers (encompassing both private and public relations). This graph captures the real user experience, but is in principle only available to a theoretical party (a panopticon) that can authenticate with all existing ActivityPub Servers and has access to all Actor's Following collections.

2. The public and in principle *partial graph* that is the union of the subset of public relations. This partial graph can be reconstructed by any interested party that will crawl existing Servers. It may deviate significantly from the full graph (and potentially be a biased representation of it).

The importance of that distinction is maybe more clear if we consider again the two main motivations for mathematical graph representations: the empirical analysis of network data and the simulation of network performance. The former will in general be performed on public data while the latter can simulate more faithful configurations.

Administrators or other third parties can crawl the public web and discover other Servers, but the actual persistent connectivity and, thus, eventual exchange of information can only be achieved via Actor actions. We have already seen that Actors are naturally mapped to being nodes of a graph. By the same token, Actors are connected by graph edges, which describe relations, interactions or associations. When relationships are symmetric (bidirectional) they are described most economically with an undirected graph (the edges have no directionality or arrows). ActivityPub networks will in general exhibit asymmetric relationships, so we will use directional graphs by default. We can already make an introductory visual presentation of Actors and Follow/Following relations in Fig. 1.

3.1 Simplified Graph Representation

The classic, simple (with no self-loops), directed graph is sufficient to describe the Following/Followed connectivity of Actors within a single Server. Consider an ordered pair $G = (V, E)$ comprising:

- V, a set of vertexes (also called *nodes*) with each $v_i \in V$ representing a Actor.
- $E \subseteq \{(v_i, v_j)|(v_i, v_j) \in V \times V \wedge i \neq j\}$, a set of edges (also called *links*), which are ordered pairs of vertexes representing relations between Actors.

We put that definition to work immediately in Fig. 2. We imagine a small collection of Actors all registered within the same Server. The graph depicts various possible Actor connectivity patterns: Actors that do not follow and are not followed by anyone and Actors that have a mix of symmetric / asymmetric connections. This model can be useful to analyze *sub-graphs* of a network, in particular if the Server instance is large.

3.1.1 Adjacency Matrix Representation

The duality between network graphs and matrix representations is an important aspect of graph theory. Matrix algebra is a useful tool in graph theory as it allows the expression of important algorithms in

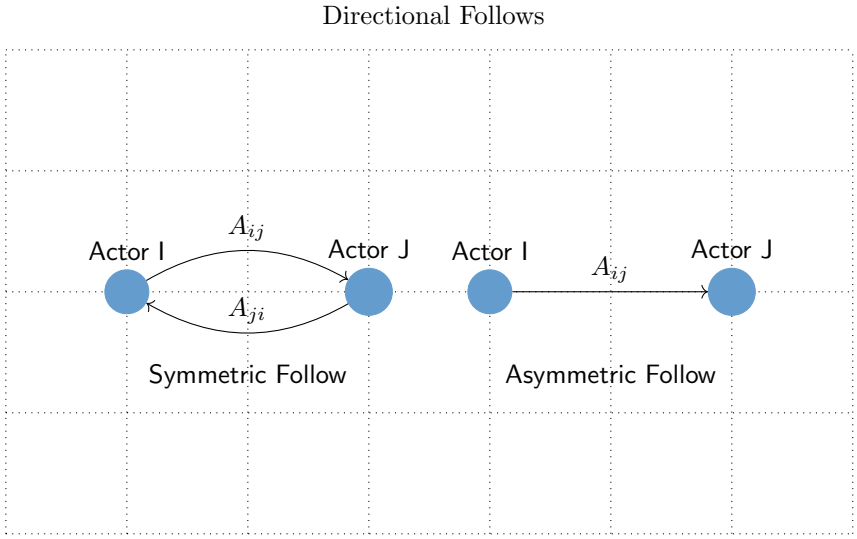

Figure 1: Representation of ActivityPub Actors as the indexed nodes and the Follow / Following relations as edges of the Graph. The direction of the edge arrow points from the Actor who follows to the Actor that is being followed. A_{ij} denotes the adjacency matrix values (more on that below).

concise mathematical notation and the utilization of a very extended body of linear algebra knowledge. The core correspondence is established via the notion of an *Adjacency Matrix*. In the simplest case this is a boolean matrix (composed exclusively of zeros and ones) representing the connectivity between nodes in a graph.[7]. More precisely, for a graph with N nodes, the adjacency matrix is an $N \times N$ matrix whose (i, j)-th entry A_{ij} is 1, if the vertex v_i is connected with an edge to vertex v_j and 0 otherwise.

Let us write down the adjacency matrix for our sample graph. We need to first map the Actor nodes to an index (lets say using alphabetic order). Our example will be a 7×7 matrix: the 7 rows are laid out horizontally (ranging over the From nodes) and the 7 columns vertically (ranging over the To nodes). To obtain the matrix we simply examine all Actors in turn and establish whether they have edges pointing to other actors (Following). The result of this exercise is:

$$A_{ij} = \begin{bmatrix} \mathbf{F}/T & A & B & C & D & E & F & G \\ A & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ B & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ C & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ D & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ E & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ F & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ G & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

How can we use this result? There is a long list of useful tools, here we only sketch how this works at high level. A common operation involving graphs is breadth first search (BFS). A breadth-first-search algorithm returns the set of all nodes reachable from node i, following the direction of the links (edges) of the graph. The corresponding matrix operation is to multiply the adjacency matrix A of the graph

[7] One can introduce also weighted graphs by attaching a real number to each edge

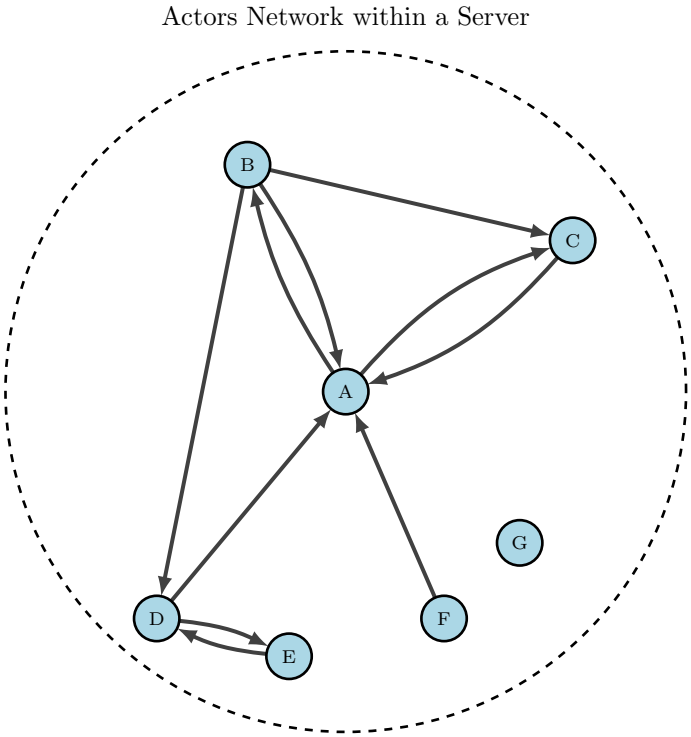

Figure 2: A simple directed graph is useful when all Actor connectivity is realized within a single multi-Actor Server. This is the configuration of a centralized, non-federating platform. This model can be useful to analyze *sub-graphs* of a network, in particular if the Server instance is large. It ignores connectivity outside the Server. The graph depicts various possible Actor connectivity patterns: Actor G does not follow and is not followed by anyone (They may still receive content from shared inboxes and public information flows). Actor F only Follows. Other Actors have a mix of symmetric / asymmetric connections whereas Actor A might be what is termed an *influencer*, a node with a large number of Follower relations. The dashed line surrounding all Actors is the periphery of the Server that is hosting them.

by a vector u of size N that has a single entry (unity) corresponding to the node i. Using powers, e.g. $A^2 = AA$ of the adjacency matrix one can progressively probe ever wider neighborhoods of this starting node. Computing the eigenvalues of the adjacency matrix is another important tool in spectral graph theory that examines fundamental properties that distinguish and characterize graph structure.

3.1.2 Adjacency List Representation

While an adjacency matrix is a mathematical object when used as a data structure it may be inefficient[8] when the network is sparse (that is, it has many nodes with few edges). An alternative representation is the *Adjacency List*. More precisely, for a graph with N nodes, the adjacency list is a hashtable of size N, of the form $v_i : S_i$, where S_i is the set of nodes to which v_i has an edge to ($v_j \in S_i$ if $(v_i, v_j) \in E$).

$$\begin{aligned} A &: (B), (C) \\ B &: (A), (C), (D) \\ C &: (A) \\ D &: (A), (E) \\ E &: (D) \\ F &: (A) \\ G &: (\emptyset) \end{aligned} \tag{2}$$

3.2 Single-Actor Servers

Another use case where the simple direct graph pattern has sufficient realism to usefully describe an ActivityPub network is when each Server only involves one Actor. This is not a theoretical possibility, several existing projects implementing ActivityPub are premised on being single-user, self-hosted instances. In this context the natural identification is to consider each *Server* as a graph node, and thus leave the Actor object implicitly associated with its hosting Server. This configuration is visually depicted in Fig. 3. Importantly, the adjacency matrix is in this case identical to the one we saw previously (Eq. 1).

3.3 More General ActivityPub Networks

The classic graphs discussed above show that are expressive and flexible mathematical tools. Yet as modeling tools for more general ActivityPub networks they have limitations that have already been alluded to. Some important open questions are the following:

1. How to represent multiple **distinct federating Servers** (hence distinguishing between Actors local to Server and remote Actors that live in different Servers).

2. How to represent **Actors of different types**. We re-emphasize that Actors are not necessarily Persons. In the presence of multiple types of Actors (e.g., Groups, Bots etc.) the topological properties of the network may be quite a bit more complicated.

3. How to represent Servers that enable *different types of Activities*. Servers that focus on different use-cases may only partially interoperate.

[8] Actual performance depends on how the matrix is represented in computer memory as it does not necessarily need to follow a list of arrays pattern (i.e. sparse matrix storage).

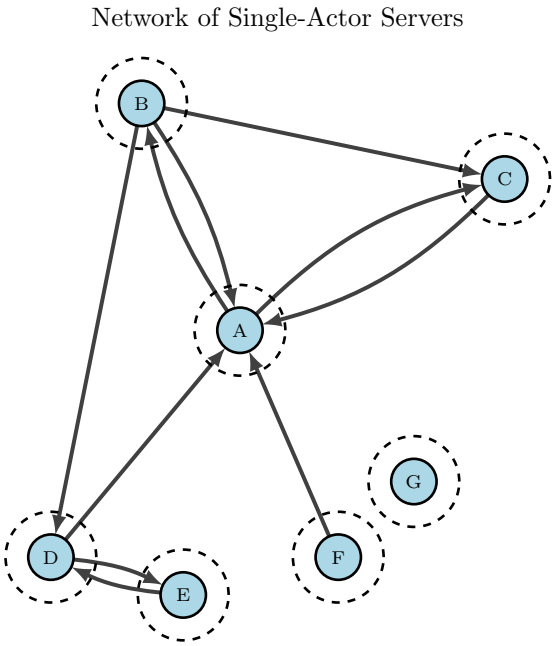

Figure 3: When each Server includes only one Actor, another representation of the ActivityPub network as a simple directed graph is possible. The graph is effectively identical to our previous example. The difference is that we have multiple servers (indicated by dashed lines), each hosting a single Actor. One might think of this pattern as one way the *decentralized* nature of the ActivityPub network approaches a *distributed* (or Peer-to-Peer) network. In this simplified architecture analyzing the network of Servers is identical to analyzing the network of Actors. While single-user Server implementations do exist, with current technical and usability constraints it is unlikely that such a pattern will be the only relevant network pattern. Hence more elaborate network models are required.

4. How to represent more nuanced relations between Actors. The Following/Followed relation is pivotal but we already saw how the Blocking Activity introduces more complicated connections. As another example, Likes/Dislike Activities establish other type of relations between Actors etc.

5. How to represent *evolving networks*. New or Removed Servers and/or Actors might be represented as Graph editing operations and require a temporal network approach.

6. Finally, to the degree that we want to track in more detail the Activities pursued by Actors (the sharing of data objects) we might have to expand the graph structure into more realistic but less mathematically tractable Property Graphs (which would include e.g., models for data Collections).

Pictorially, the general structure of a fully developed ActivityPub network might look like Fig. 4. Such a *heterogeneous* ActivityPub network aims to capture the likelihood of different types of Servers. Heterogeneity implies that the type and amount of Actor connectivity and Activity objects that diffuse over the network varies.

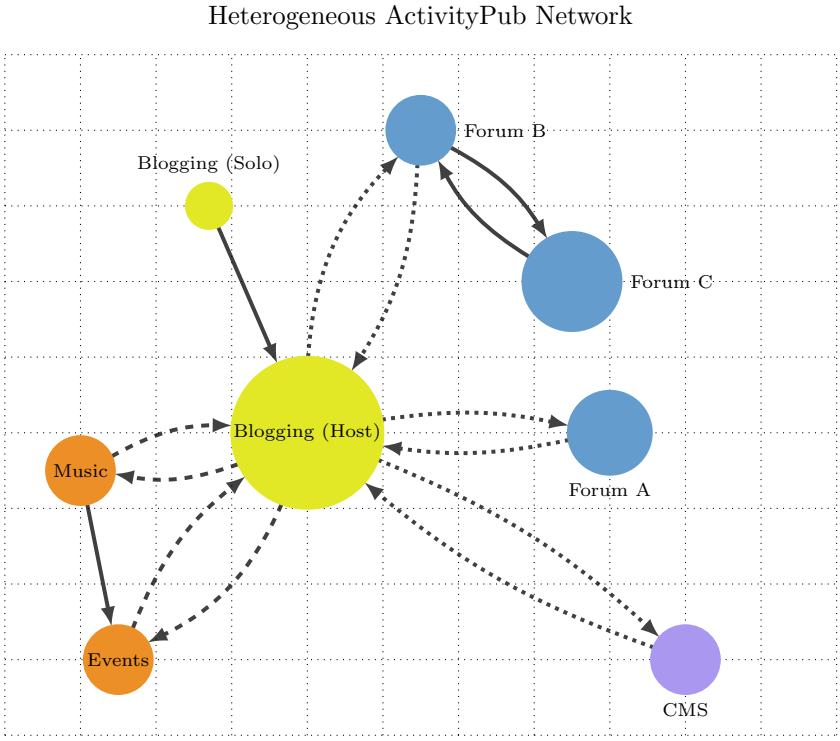

Figure 4: A heterogeneous ActivityPub network that illustrates the coexistence of different types of Servers. Individual Actors are not shown (in practice one or more Actors will be registered within each Server node). Heterogeneity implies that the type and amount of Actor connectivity and Activity objects that diffuse over the network varies. This figure frames the level of our ambition around the mathematical representation of the network in terms of adjacency tensors, slightly more than a simplest graph representation, but substantially far from a detail model of everything that is happening. The trade-off of generalizations is that there are increasingly fewer linear algebra algorithms available for analysis.

4 Multi-Layer Network Representations

4.1 Multilayer Network Notation

Mathematical frameworks for multilayer networks that use tensor algebra have been presented in [33, 37] and we will follow their general notation and definitions here. A multilayer network starts with the familiar set of elementary nodes V. We enumerate these nodes explicitly $V = \{v_i\}_{i=1}^{N}$. The order $|V| = N$ denotes the total number of nodes. In addition, we define layers (or aspects) that will encode qualities of nodes and edges. Sometimes these additional dimensions are denoted colors.

A multilayer network can have any number (d) of such aspects. We define a sequence $L = \{L_a\}_{a=1}^{d}$ of layers for each aspect a. This can be seen as a generalization of a single-layer network (where $d = 0$, i.e., where the elementary nodes are monochromatic). The total number of layers is $|L_1| \times \ldots \times |L_d|$, where $|L_a|$ is the number of layer instances for each aspect.

We can then define the multilayer nodes as the Cartesian product $V_M = V \times L_1 \times \ldots \times L_d$. The nodes are identified using a set of $d + 1$ indices:

$$V = \{v_i^{l_1,\ldots,l_d}\}$$

where $i \in [1, N]$ and $l_a \in [1, |L_a|]$. An edge set E_M of a multilayer network is a set of pairs of possible combinations of nodes: $E_M \subseteq V_M \times V_M$. Putting everything together we obtain the definition of a multilayer network as the quadruplet $M = (V_M, E_M, V, L)$.

4.2 Adjacency Tensors

Loosely speaking, tensors can be seen as generalizations of vectors and matrices. Vectors could be called rank-1 tensors, they have one index. Matrices are rank-2 tensors, they have two indexes or dimensions. The more indices, the higher the rank of a tensor. Adjacency Matrices are rank-2 tensors, so they are inherently limited in the complexity of the relationships they can capture. *Adjacency Tensors* are higher-order objects that can potentially capture more complex network relations. The multilayer adjacency tensor A is a very general object that can be used to represent a wealth of complicated relationships among nodes. The simplest generalization of an Adjacency Tensor beyond an Adjacency Matrix would be the fourth-order, or rank-4, tensor A_{ij}^{ab} where the indices (a, b) range over a single additional layer dimension (e.g., a count of Server instances) and (i, j) ranges over the nodes (e.g. Actors).

$$A_{ij}^{ab} = \begin{cases} 1 & : \text{ if link from node } (i,a) \text{ to node } (j,b) \\ 0 & : \text{ otherwise} \end{cases} \tag{3}$$

The corresponding adjacency list is of the form $v_i^a : S_i^a$, where S_i^a is the set of nodes to which v_i^a has an edge to ($v_j^a \in S_i^a$ if $(v_i^a, v_j^b) \in E_M$).

4.3 ActivityPub as Multilayer Networks

The above machinery is what we need to get started with the mapping of concrete ActivityPub networks into mathematical multilayer networks. What are our options? A large variety of network types can be represented using the general multilayer-network framework. Here we review some relevant attributes identified in [33] and how they relate to ActivityPub networks.

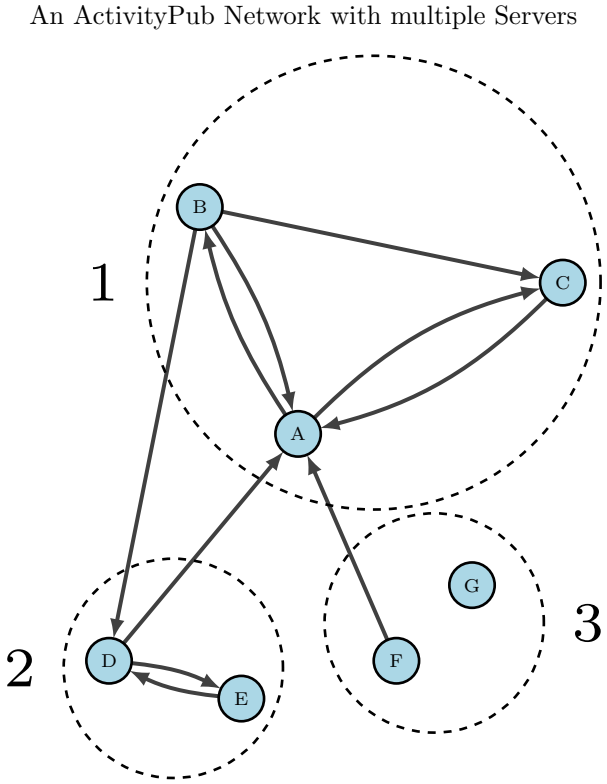

Figure 5: An ActivityPub network comprising three Servers with varying numbers of Actors. The underlying Actor-to-Actor connectivity is the same as our previous examples.

A first question is about node alignment between layers. A multilayer network is said to be *node-aligned* (or fully interconnected) if all the layers contain all nodes. This will in general not be the case for snapshots of an ActivityPub network. A potentially interesting exception is a node-aligned *temporal graph* where all Actors would be present in all temporal layers[9]. This approach may be a good representation to enable the analysis of information propagation on an existing network.

At the other extreme, a network may be *layer-disjoint*. A multilayer network is layer-disjoint if each Actor exists in at most one layer. This will be typically a relevant category. For example, under the assumption that ActivityPub Actors are unique to Servers a multilayer network that distinguishes Server instances will in general be layer-disjoint.[10] Another example where the layer-disjoint pattern is relevant is when we want to represent different Actor types. Each distinct Actor type becomes a layer instance.

From a network modeling perspective we have two degrees of freedom we can exercise: i) the number of dimensions or aspects we model, this determines the rank of the tensor representation and ii) the size of each aspect (the number of distinct qualities in each dimension), which determines the index range of each tensor dimension. As an illustration, a multilayer network representation that recognizes distinct Servers of the same type is given in Fig. 5.

Note that while the Adjacency Matrix representation provided already for our example is still valid, it does not encode the fact that Actors reside in distinct Servers. As we have seen in the description of the protocol, the role of Server instances is quite explicit in the specification (it is a Server-to-Server

[9] Focusing on an time interval where there are no new nodes
[10] That assumption will break down if Actors belonging to the same user being active in different Servers get connected.

protocol). This is even more so in practical implementations, as it shapes information discovery and distribution patterns. A second aspect that we can readily model is Server Type.

Multilayer Network: Server Count as Layer

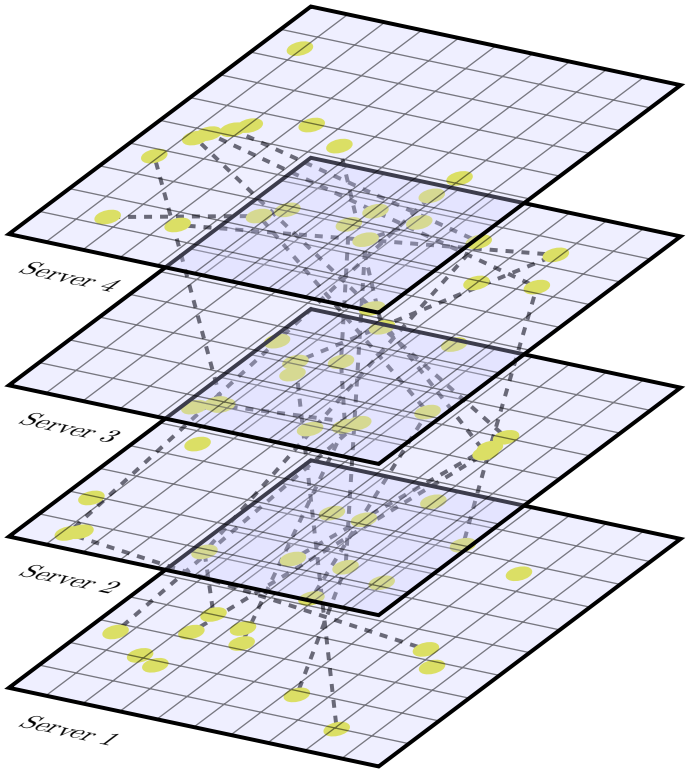

Figure 6: The architecture of Mastodon as a decentralized online social network has been discussed in [29]. The architecture distinguishes between two layers: the Server-to-Server layer, made up of all interconnected Servers (Server N); and the social network layer, formed by the follow relationships between Actors (User K @ Server N) hosted by the different Servers.

These two facets of a minimal multilayer network that resolves Server types and individual instances thereof is illustrated in Figures 6 and 7. Putting it all together we obtain the following rank-6 Tensor:

$$A_{ij}^{abkl} = \begin{cases} 1 & : \text{ if link from node } (i,a,k) \text{ to node } (j,b,l) \\ 0 & : \text{ otherwise} \end{cases} \tag{4}$$

We close by illustrating once again our small network example. Let us write down the adjacency tensor for our sample graph where we now introduce a Server aspect. The object is A_{ij}^{ab}. For brevity we will not list all elements (the adjacency tensor is very sparse in this instance) but only indicatively. First, the tensor elements A_{ij}^{11}, which captures the relations within Server 1:

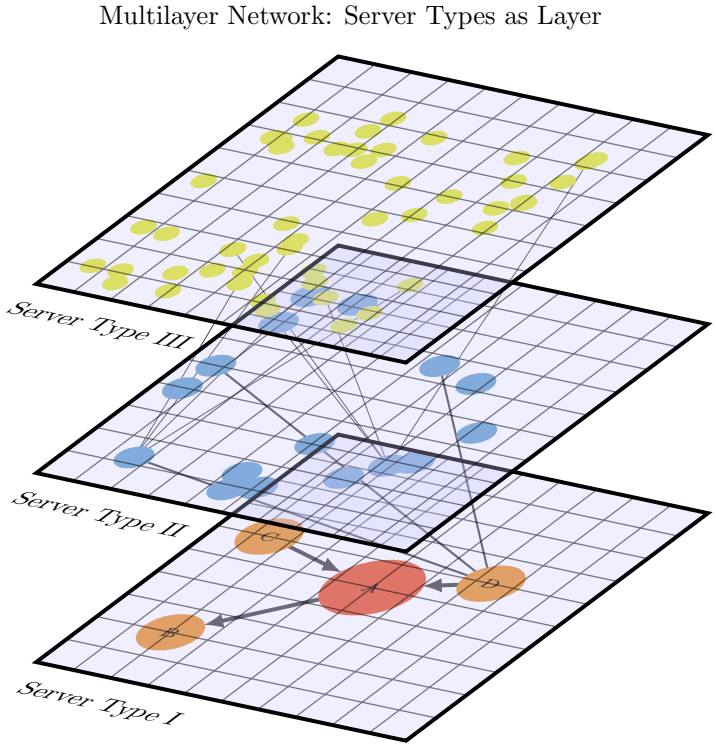

Figure 7: An ActivityPub network comprised of three Server Layers (I, II, III), each representing a different type of Server. While Actors can in principle Follow and exchange Activities that are jointly recognized, there will be some subset of Activities that can only occur between some Layers.

$$A_{ij}^{11} = \begin{bmatrix} & A & B & C & D & E & F & G \\ A & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ B & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ C & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ D & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ F & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ G & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

Next, the tensor elements A_{ij}^{12} which capture relations emanating from Server 1 towards Server 2:

$$A_{ij}^{12} = \begin{bmatrix} & A & B & C & D & E & F & G \\ A & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ F & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ G & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6}$$

We notice that the components of adjacency tensor are block sub-matrices. This is a general pattern: an

adjacency tensor can always be unrolled into what is called a *supra-adjacency matrix*. The corresponding adjacency list will in this case read like:

$$
\begin{aligned}
(A,1) &: (B,1),(C,1) \\
(B,1) &: (A,1),(C,1),(D,2) \\
(C,1) &: (A,1) \\
(D,2) &: (A,1),(E,2) \\
(E,2) &: (D,2) \\
(F,3) &: (A,1) \\
(G,3) &: (\emptyset)
\end{aligned}
\tag{7}
$$

References

[1] M.D. Koenig and S. Battiston. From Graph Theory to Models of Economic Networks. A Tutorial. *A.K. Naimzada et al. (eds.), Networks, Topology and Dynamics*, 2009.

[2] M.D. Flood. On Counterparty Networks. 2015.

[3] Andrea Aguiar, Dror Y. Kenett, Richard Bookstaber, Thomas Wipf. A Map of Collateral Uses and Flows. *Office of Financial Research (OFR) Working Paper Series*, 2016.

[4] G. Marti et al. A review of two decades of correlations, hierarchies, networks and clustering in financial markets. *Preprint*, 2017.

[5] Simonetta Rosati, Francesco Vacirca. Interdependencies in the euro area derivatives clearing network: a multi-layer network approach. *ECB Working Paper Series*, 2019.

[6] D.D. Gatti H. Dawid. Agent-Based Macroeconomics. *Universitat Bielefeld: Working Papers in Economics and Management*, 2018.

[7] J.P. Gleeson T.R. Hurd. A framework for analyzing contagion in banking networks. *Preprint*, 2011.

[8] E. Cerutti and H.Zhou. The Global Banking Network in the Aftermath of the Crisis. *IMF Working Paper*, 2017.

[9] P. Csermely et al. Structure and dynamics of core/periphery networks. *Journal of Complex Networks*, 2013.

[10] C. Joyez. Network Structure of French Multinational Firms. *Preprint*, 2017.

[11] Z. Poszar et al. Shadow Banking. *FRBNY Staff Reports*, 2012.

[12] J. Duesenberry. A Process Approach to Flow-of-Funds Analysis. *National Bureau of Economic Research*, 1962.

[13] G.L Breton L.B Duc. Flow-of-funds analysis at the ECB. Framework and Applications. 2009.

[14] Financial Stability Board. The Financial Crisis and Information Gaps. *Report to the G-20 Finance Ministers and Central Bank Governors*, 2009.

[15] A.G. Haldane. On microscopes and telescopes. *Bank of England Speech*, 2015.

[16] Wikipedia Authors. Social Network Analysis. Link.

[17] ActivityPub online documentation (as of jan 2024). Link.

[18] Activity Streams 2.0 online specification. Link.

[19] Activity Vocabulary online specification. Link.

[20] Pixelfed Developers. FediDB: Fediverse Network Statistics. Link.

[21] Fediverse enhancement proposals. Link.

[22] Activitypub issue tracker. Link.

[23] Valueflows vocabulary for distributed economic networks. Link.

[24] Wikipedia Authors. Object Capability Model. Link.

[25] P. Papadopoulos. WP8: Connecting the Dots: Economic Networks as Property Graphs. *Open Risk White Papers*, 2019. Online Link.

[26] P. Papadopoulos. WP10: Connecting the Dots: Concentration, diversity, inequality and sparsity in economic networks. *Open Risk White Papers*, 2021. Online Link.

[27] Lucio La Cava, Domenico Mandaglio, Andrea Tagarelli. Polarization in decentralized online social networks. 2017.

[28] Lucio La Cava, Andrea Tagarelli. Information consumption and boundary spanning in decentralized online social networks: the case of mastodon users. *Online Social Networks and Media*, 30, 2022.

[29] Matteo Zignani, Sabrina Gaito, Gian Paolo Rossi. Follow the mastodon: Structure and evolution of a decentralized online social network. In *Proceedings of the Twelfth International AAAI Conference on Web and Social Media (ICWSM 2018)*, 2018.

[30] Nickey Obreykov. Evaluating network generation algorithms for decentralized social media platforms. 2021.

[31] Jan Trienes, Andrés Torres Cano, Djoerd Hiemstra. Recommending users: Whom to follow on federated social networks. 2018.

[32] S. Boccaletti et al. The structure and dynamics of multilayer networks. *Physics Reports*, 2014.

[33] M. Kivela et al. Multilayer networks. *Journal of Complex Networks*, 2014.

[34] JSON-LD online documentation. Link.

[35] Alan Z. Rozenshtein. Moderating the fediverse: Content moderation on distributed social media. 2023.

[36] I. H. Anaobi et al. Will admins cope? decentralized moderation in the fediverse. 2023.

[37] Manlio De Domenico et al. Mathematical formulation of multilayer networks. *Physical Review X*, 2013.