DIGITAL ENGINEERING FACULTY
Computer Graphics Systems Group

VisCom: Visualization of Communication Networks

Visualization Approach and Evaluation Framework

Master's Thesis
in partial fulfillment for the academic degree
"Master of Science"
(M.Sc.)
in IT-Systems Engineering

Hasso Plattner Institute, Digital Engineering Faculty, University of Potsdam

submitted by
Valentin Schröter

Supervision:
Prof. Dr. Jürgen Döllner
Willy Scheibel

Potsdam,
May 2, 2025

Contents

Abstract

This thesis proposes *VisCom*, an approach for the visualization of communication networks. Communication networks are directed, cyclic multigraphs that are often visualized using node-link diagrams. In communication networks, nodes – representing participants, entities, or processes – exchange messages via named communication links, called *topics*. Communication networks occur in the field of machine-to-machine communication, for example in robotics, Internet of Things, and microservice architectures. Other domains can be modeled as communication networks as well, such as social networks. With state-of-the-art approaches, the loose assumptions about the structure of such networks often lead to information overload, cluttered visualizations, and a lacking representation of the underlying system structure. Thus, users and developers of such systems can only work effectively with smaller graph sizes, which limits the applicability of these approaches to real-world systems.

To improve the visual scalability, this thesis presents a two-step approach: First, a heuristic processing strategy analyzes the network data to gain information about the structural significance of nodes and edges. Second, the network data is visualized in a radial overview layout. This layout is guided by a proposed sorting algorithm called Weighted Flow Sorting, which produces structurally optimized node orderings in cyclic graphs. To support visual scalability, the layout is extended by community detection, connection filtering, and virtual nodes. To evaluate the proposed approach, this thesis presents an extensible evaluation framework that allows the comparison of multiple graph layout techniques side-by-side and the evaluation of their performance with regard to various graph quality metrics. The evaluation framework is available open-source at GitHub at /vschroeter/VisCom-Thesis, a limited demo of its user interface is hosted at vschroeter.github.io/VisCom-Thesis. The approach is further demonstrated on real-world communication networks derived from the robotics framework *Robot Operating System 2* (ROS 2), as well as on synthetic datasets with up to 150 nodes. VisCom performs comparably to existing node-link techniques and shows improvements in selected quality metrics.

Zusammenfassung

In dieser Arbeit wird *VisCom* vorgestellt, ein Ansatz zur Visualisierung von Kommunikationsnetzwerken. Kommunikationsnetzwerke sind gerichtete, zyklische Multigraphen, die oft mit Hilfe von Node-Link-Diagrammen visualisiert werden. In Kommunikationsnetzwerken tauschen Knoten – die Teilnehmer, Entitäten oder Prozesse darstellen – Nachrichten über benannte Kommunikationskanäle aus, die als *Topics* bezeichnet werden. Kommunikationsnetzwerke kommen im Bereich der Maschine-zu-Maschine-Kommunikation vor, zum Beispiel in der Robotik, im Internet of Things und in Microservice-Architekturen. Auch andere Domänen können als Kommunikationsnetzwerke modelliert werden, wie z. B. soziale Netzwerke. Bei aktuellen Ansätzen führen die schwachen Annahmen über die Struktur solcher Netzwerke oft zu einer Informationsüberladung, unübersichtlichen Visualisierungen und einer mangelnden Darstellung der zugrunde liegenden Systemstruktur. Daher können Benutzer und Entwickler solcher Systeme nur mit kleineren Graphen effektiv arbeiten, was die Anwendbarkeit dieser Ansätze auf reale Systeme einschränkt.

Um die visuelle Skalierbarkeit zu verbessern, wird in dieser Arbeit ein zweistufiger Ansatz vorgestellt: Zunächst analysiert eine heuristische Verarbeitungsstrategie die Netzwerkdaten, um Informationen über die strukturelle Bedeutung von Knoten und Kanten zu gewinnen. Zweitens werden die Netzwerkdaten in einem radialen Übersichtslayout visualisiert. Dieses Layout wird von einem in dieser Arbeit vorgestellten Sortieralgorithmus namens Weighted Flow Sorting geleitet, welcher strukturell optimierte Knotenreihenfolgen in zyklischen Graphen erzeugt. Um die visuelle Skalierbarkeit weiter zu unterstützen, wird das Layout mit Community-Detection, Kantenfilterung und virtuellen Knoten erweitert. Um den präsentierten Ansatz zu evaluieren, wird in dieser Arbeit ein erweiterbares Evaluierungs-Framework vorgestellt, das den Vergleich mehrerer Graphlayouttechniken nebeneinander und die Bewertung im Hinblick auf verschiedene Graphenqualitätsmetriken ermöglicht. Das Evaluierungs-Framework ist open-source auf GitHub unter /vschroeter/VisCom-Thesis verfügbar, eine eingeschränkte Demo der Benutzeroberfläche wird unter vschroeter.github.io/VisCom-Thesis gehostet. Der Ansatz wird an realen Kommunikationsnetzwerken aus dem Robotik-Framework *Robot Operating System 2* (ROS 2) sowie an synthetischen Datensätzen mit bis zu 150 Knoten demonstriert. Im Vergleich zu bestehenden Node-Link-Diagrammansätzen erzielt VisCom ähnliche oder bessere Ergebnisse bei den berechneten Qualitätsmetriken. VisCom zeigt vergleichbare Layoutqualitäten zu bestehenden Node-Link-Ansätzen und Verbesserungen bei ausgewählten Qualitätsmetriken.

Chapter 1

Introduction

For decades, our society has been exposed to a rapidly growing amount of complex, interdependent and dynamic data [van05]. A wide range of visualization techniques have been established as an essential strategy to address this complexity and support tasks associated with such data [Moo+24]. The goal of information visualization is to be simple and supportive, both in an effective and efficient way [van05]. This applies to many tasks, as well-designed visual representations are significantly easier to use than text-only descriptions, faster to comprehend, and provide the ability to reveal patterns, clusters, outliers, and other features in the data [Shn03].

However, reality shows that state-of-the-art visualization techniques often tend to overload the user with information. This information overload has been identified as particularly problematic for node-link diagrams as a subset in the spectrum of visualization techniques. At scale, the high density of links and complexity of node relationships can quickly overwhelm traditional visualization techniques, rendering them unusable as the size of the data to be visualized increases [HMM00; CHS21]. Only a few methods are able to deal effectively with a larger number of nodes while respecting the variety of rules regarding graph readability and graph aesthetics [HMM00; Ben+07; DS09].

This issue with graph visualization techniques can be observed especially in the domain of technical communication networks. In communication networks, nodes – representing participants, entities, or processes – are connected by named communication channels, which are referred to as *topics* and through which messages are exchanged.

Compared to traditional graphs, communication networks extend the common definitions [ER60; WS98; FFF99; BA99; Dog+18] by allowing multiple distinct connections between the same pair of nodes, each identified by a unique name. Thus, they exhibit a multi-graph structure with named edges. In the field of machine-to-machine communication, these networks occur in areas such as robotics [Qui+09; Mac+22], the Internet of Things (IoT) [Li+10; TK19], and microservice architectures [Cer+22]. Generally speaking, many technical systems can be modeled as communication networks.

Despite their importance, visualizing communication networks remains a significant challenge, especially for systems with a large number of nodes and connections. Existing tools quickly reach their limits in terms of scalability and already exceed their capacities when applied to systems with as little as 10 nodes [TB18]: The generated visualizations quickly become cluttered and unreadable as systems grow in size or contain dense connection patterns. The highly multi-connective nature of communication graphs also

poses difficulties for established methods such as force-directed graphs [FR91] or layered visualizations [STT81; GKN15]. These methods either display an overwhelming amount of visual entities or, in the case of complexity reduction techniques, obscure critical details [CHS21]. As a result, users are limited in their ability to understand system structures or trace communication flows effectively.

1.1 Problem Statement

Current visualization tools for complex communication networks face two major challenges:

1. **Information Overload:** Visualizations often contain a large number of graphical elements, resulting in cluttered diagrams.
2. **Structural Representation:** The overall structure of the network is frequently obscured, making it challenging to understand communication flows and the architecture of the corresponding systems. Particularly in hierarchical visualization techniques, nodes may be positioned far apart, hindering the interpretation of relationships

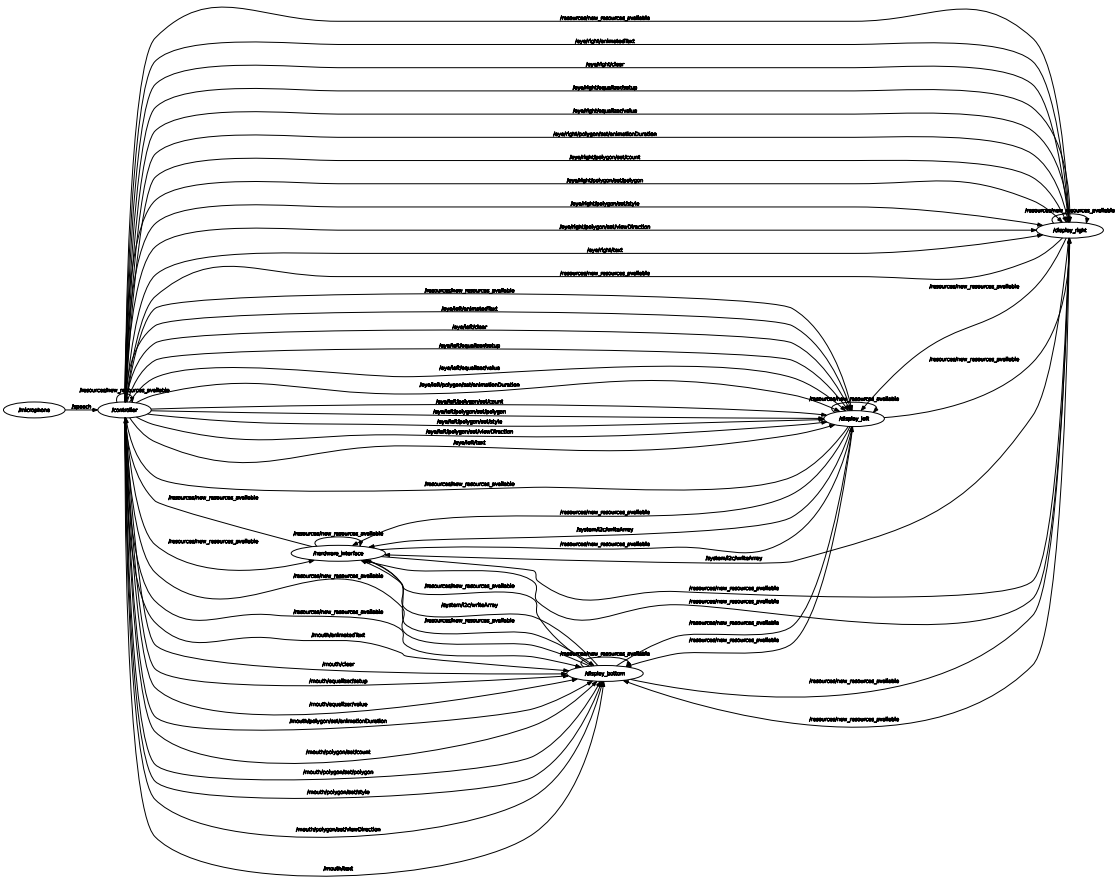
These problems are caused by the nature of communication networks. First, they typically contain multiple connections between the same node pairs. Second, they often include cyclic and broadcasting communication patterns, which are poorly supported by common visualization tools. Together, these aspects reduce clarity and readability of the networks' visual representations.

The stated problems even occur in small systems. Figure 1.1(a) illustrates a network of only six nodes, yet the visualization is already difficult to interpret due to excessive edge density and a lack of structural abstraction. This example is based on the default output of *rqt_graph* [TB18], a visualization tool for communication networks in robotics.

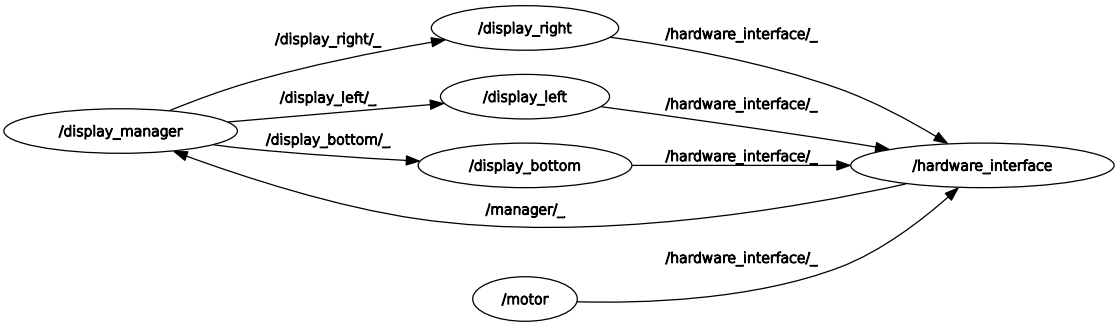
To demonstrate the effect of abstraction, a manually simplified version of the same network is shown in Figure 1.1(b). Here, semantically equivalent connections are aggregated, significantly improving the clarity of the system structure. This comparison highlights the need for automated techniques that can generate scalable and interpretable visualizations without requiring manual intervention.

1.2 Approach and Contributions

As a starting point for designing visualizations that effectively present data to users, Shneiderman introduced the Visual Information Seeking Mantra [Shn03]. It proposes the principle to first provide an overview, secondly allow the user to zoom and filter, and finally look at details if desired. This principle continues to be widely accepted and, along with other fundamental research in information visualization, motivates the use of interactive interfaces to overcome information overload and to support task-oriented human understanding [Ger+09].



(a) *Full Visualization of a Communication Network with Six Nodes and 53 Edges*



(b) *Manually Simplified Visualization of the Network in (a)*

Figure 1.1: *The visualization of a communication network in the tool rqt_graph [TB18], which is the default visualization tool in ROS 2, proposed for understanding the system principles [ROS24b]. The system consists of six nodes, each connected by multiple topics. Figure* ***(a)*** *shows the full visualization of the system, as it would be displayed in rqt_graph. Figure* ***(b)*** *shows the same system, but with a manually simplified connections. This is done by aggregating the connections between the nodes to only one edge.*

In order to address the challenges of visualizing complex communication networks with strong multi-connective characteristics, this thesis proposes a new visualization technique called *VisCom* with the aim of providing better visual scalability compared to existing techniques.
The proposed technique consists of two main steps:

1. A heuristic processing strategy analyzes the communication network and calculates scores for the nodes and the connections between them in order to gain information about their structural significance. This preprocessing is the foundation for subsequent algorithms.

2. The network data is visualized in a radial overview layout. This layout is guided by an algorithm called Weighted Flow Sorting, which produces structurally meaningful node orderings in cyclic graphs, emphasizing the dataflows between nodes. To support scalability, the layout is enriched with community detection and introduces concepts such as virtual nodes and connection filtering.

For practical demonstration, the visualization technique is applied to networks created in the widely used *Robot Operating System 2* (ROS 2) [Mac+22]. ROS 2 is an example of a framework that relies on communication networks to enable message exchange between distributed components.

To visualize and evaluate the approach described above, an evaluation framework has been implemented as part of the thesis. This framework provides a user interface to compare the proposed technique to existing techniques side-by-side for uploaded or generated graph data. It also includes the calculation of various graph visualization metrics to quantitatively assess the layouts.

In summary, this thesis presents the following contributions:

1. Novel node and edge scoring methods based on connection characteristics between nodes.

2. A novel flow-based sorting algorithm for cyclic and densely connected graphs.

3. A new radial graph layouting strategy with multiple layout and routing techniques.

4. An algorithm to synthesize random graphs with the characteristics of communication networks.

5. A quantitative evaluation framework for comparing graph layouts using multiple structural and readability metrics. The frontend of this evaluation tool is available at `vschroeter.github.io/VisCom-Thesis`.

6. The source code of the project as an open-source project on GitHub `/vschroeter/VisCom-Thesis`.

1.3 Structure of the Thesis

After the following Chapter 2 describes the general related work for this thesis, Chapter 3 introduces the fundamental concepts of communication networks, including nodes, channels, and mathematical representations. Chapter 4 presents the proposed approach with preprocessing steps and the layout algorithm as the main contributions of the thesis. The approach is quantitavely evaluated in Chapter 5 using graph quality metrics and real-world datasets. The implementation of the evaluation framework is described in Chapter 6, including the frontend and backend implementation, performance aspects, and deployment. Chapter 7 discusses limitations, outlines future work, and reflects on the design of the framework. Finally, Chapter 8 concludes the thesis by summarizing key findings and identifying potential applications. The Appendix provides additional visualizations and comparisons.

Chapter 2

Related Work

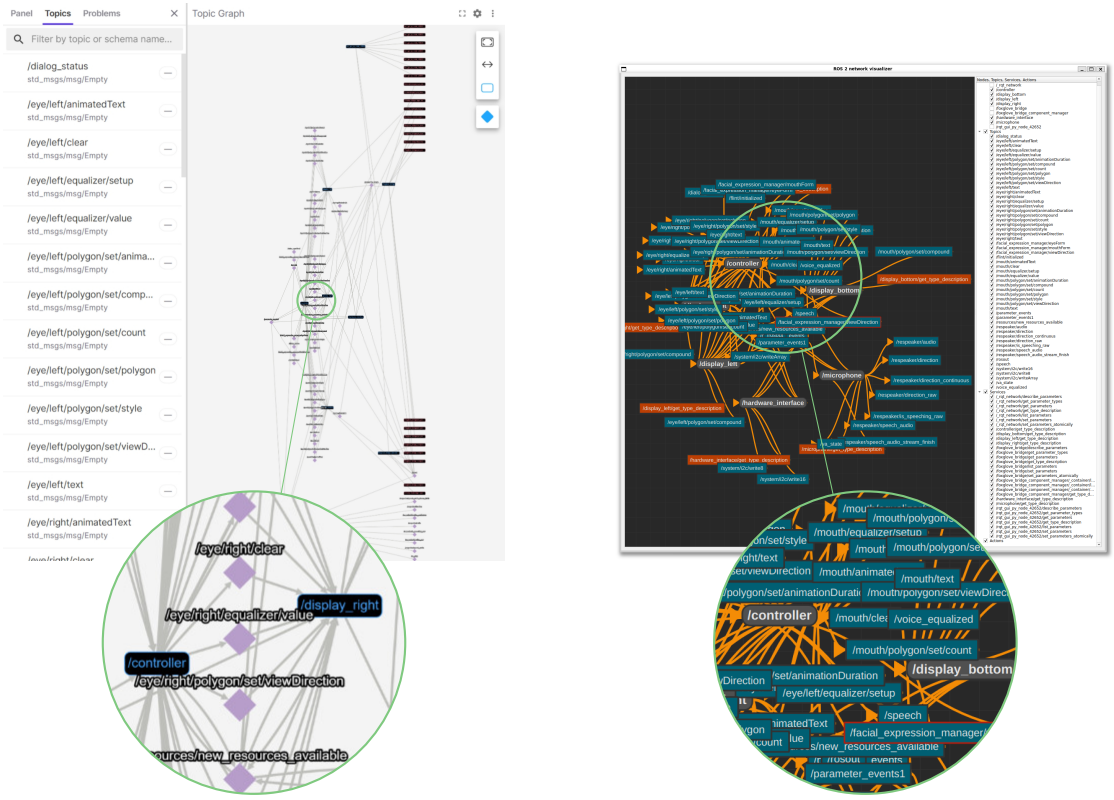
Areas of Communication Networks. Communication networks are part of many technical systems, including robotics [Qui+09; Mac+22], IoT-meshes [Li+10; TK19], microservice architectures [Nad+16], and other PC networks.

In robotics, frameworks such as ROS 1 [Qui+09] and the successor ROS 2 [Mac+22] serve as middleware for communication – modeled as directed message flows – between distributed software components. Typical robotic systems separate their software into multiple nodes, each responsible for a fine-grained task, such as sensor data processing or control of actuators. The communication between these nodes is organized around different connection types, such as asynchronous publish/subscribe or synchronous client/server mechanisms, and established through named channels called *topics*. Nodes can flexibly publish and subscribe to these topics, allowing for dynamic communication patterns.

Beyond robotics, communication network structures appear in other domains. In the IoT sector, protocols like MQTT form a similar topic-based communication between devices [YS16]. Communication technologies such as ZigBee or Bluetooth establish peer-to-peer or decentralized mesh networks between edge devices, which can also be considered as communication networks [Li+10; TK19]. Microservice-based architectures in cloud and web development result in similar networks comparable in structure and dynamics [Cer+22]. Those architectures often arise in large-scale, distributed systems where independent development, deployment, and scalability of components are critical, e.g. Netflix or Amazon [Nad+16; Par18].

Human communication, for instance via emails, can be modeled as a network structure of message exchange [Bir+06]. In an abstract sense, social networks in general could also be considered as communication networks [Sha64].

ROS Visualization. The Robot Operating System (ROS), as well as its successor Robot Operating System 2 (ROS 2), are widely used representatives for communication networks in the area of robotics [Qui+09; Mac+22]. ROS and ROS 2 include a set of libraries, drivers, and developer tools to build robot applications. It contains several tools to support the understanding of system-wide communication behavior, with the most prominent being *rqt_graph* [TB18]. This graphical tool visualizes the message flow between nodes using *GraphViz*'s layered layout engine [GKN15]. Nodes are shown as ellipses and individual topic connections are rendered as directed edges [TB18]. *rqt_graph* is recommended in the official tutorial for understanding the ROS communication model



(a) *Topic Graph in Foxglove Studio [Fox24]* (b) *ros_network_vis [Lal24]*

Figure 2.1: *Screenshots of currently existing alternatives for visualization communication graphs in ROS. Each tool visualizes the same system with six nodes as rqt_graph in Figure 1.1, connected by multiple topics.*

[ROS24b] and works well for small example systems. However, with increasing system size, the resulting visualization becomes difficult to interpret [ST15]. Nodes are often widely spaced, and edges follow complex paths, making it hard to see adjacent nodes or identify communication patterns, visible in Figure 1.1(a). Other tools like *Foxglove Studio* [Fox24] and *ros_network_vis* [Lal24] offer more interactive views but face similar issues in complex systems, suffering from information overload and cluttered diagrams as visible in Figure 2.1(a) and Figure 2.1(b).

Besides visualizing live ROS systems, other tools perform static system analysis. *HAROS* analyzes source code to detect issues early in development and can infer possible communication relations between packages [SCM19]. Its visualization module *HAROSviz* attempts to depict the expected topic connections and parameter flows. Its fully expanded view is similar to the Foxglove Topic Graph as in Figure 2.1(a) [San24].

Most other ROS tools focus on the transmitted live data rather than the abstract system structure. For example, *RViz* allows synchronized 3D visualization of the robot state using interactive markers [ROS24a; Gos+11]. Tools like *RosBoard* [Ven21] and *iviz* [ZH21] provide web or mobile interfaces for displaying sensor data from specific topics.

Graph Visualization Techniques. There are several visualization strategies for general network visualizations [Bur+21], as shown in Figure 2.3. One group of techniques uses linearized node placement. Examples for this approach are time-arc layouts [GBD09] or methods based on space-filling curves to optimize screen space usage [MK08].

Radial layouts are another popular category, used across disciplines. Chord diagrams offer a linearized circular layout to represent grouped node connections [Koo+17]. Hierarchical radial visualizations support layered systems and group structures, such as animated exploration of trees [Yee+01], hierarchical edge bundles [Hol06], or multi-circular layouts [BB08]. Gansner and Koren presented a method, to bundle edges in a circular layout also without explicit hierarchies [GK07]. The Circos visualization technique is widely used in bioinformatics for comparing genomes in a radial layout [Krz+09]. Focus+Context techniques like MoireGraphs [JK03] and Hive Plots [Krz+12] combine radial arrangements with structural grouping or aggregation to reduce complexity.

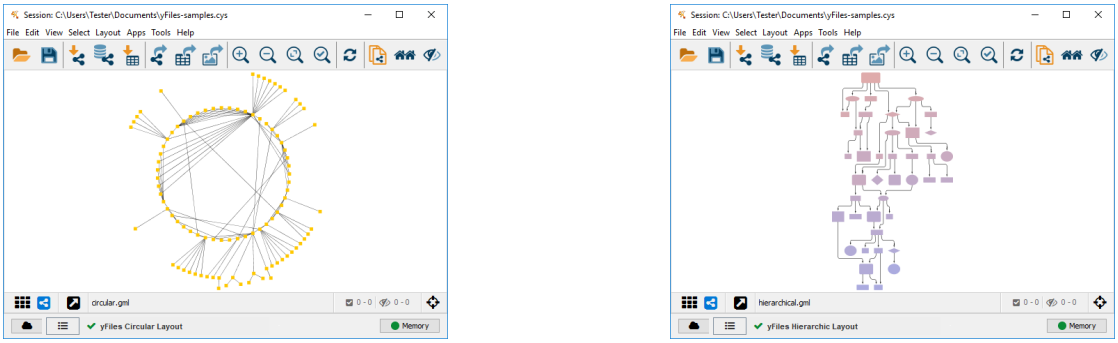
For general purpose as well as large-scale graph visualizations, force-directed layouts (FDG) are frequently used. Based on physics simulations of node repulsion and edge attraction [FR91; Kob12], FDGs create self-organized and often aesthetically pleasing layouts. They are used for visualizing social networks, infrastructure graphs, and internet structures [TDT16].

For hierarchical structures, treemaps are a popular method to visualize containment relationships and are widely applied in system overviews and software engineering [Sch+20]. Hybrid approaches, such as Elastic Hierarchies, combine treemaps with node-link diagrams to capture both structure and relationships [ZMC05].

Graph Visualization Tools. The popular open-source project *Gephi* [BHJ09; Bas25] provides an interactive framework for visualizing and exploring large graphs. In its main window, users can display and manipulate an uploaded graph, and choose from a selection of layout algorithms to apply to the graph. A separate data view lists all graph data in a tabular format. Furthermore, different analyses can be performed, including community detection and centrality measures.

The free tool *yEd* is another widely used and powerful graph visualization software [yWo25]. It focuses on the creation and the export of different types of diagrams, including for example flowcharts and UML diagrams. For the created or imported graphs, yEd provides a rich selection of automatic layout algorithms, including hierarchical, circular layouts, and organic layouts. The latter two are shown in Figure 2.2.

Graph Analysis. Structural analysis of graphs supports the understanding of communication patterns and the role of components within a network. Community detection methods aim to find densely connected subgroups of nodes. A widely used approach is modularity optimization [NG04], with the Louvain method being a fast and scalable implementation [Blo+08]. Extended community detection methods are capable of identifying overlapping communities, wherein nodes can belong to multiple groups [Shi+13; Din+16].



(a) *Circular Layout* (b) *Hierarchical Layout*

Figure 2.2: *Example of different layout types in the graph visualization tool yEd [yWo25]. The left image shows a circular layout, while the right image shows a hierarchical layout.*

Centrality measures aim to quantify the structural importance of nodes and edges. First introduced in the context of human communication networks [Bav50], the most common methods include degree, betweenness, and closeness centrality [Fre78]. Today, there is a wide range of centrality approaches based on eigenvector calculations, random walks, or variations of path-based scoring calculations [BJT23].

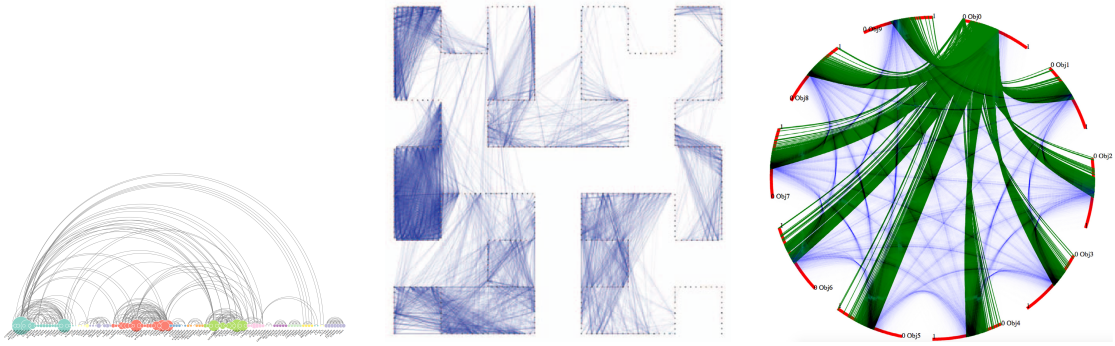
Graph Layout Evaluation. To assess the quality of graph layouts, several structural and readability metrics are commonly used. These include edge crossings, angular resolution, path continuity, and layout compactness [CP96; Ben+07]. Many of these criteria have quantitative formulations to enable objective comparison of different layouts [Pur02; DS09].

Evaluation frameworks aim to systematically compare layout techniques and visualize their performance. For example, *GraphVizdb* [Bik+16] is a web platform for scalable graph exploration using multi-level layouts and interactive filtering.

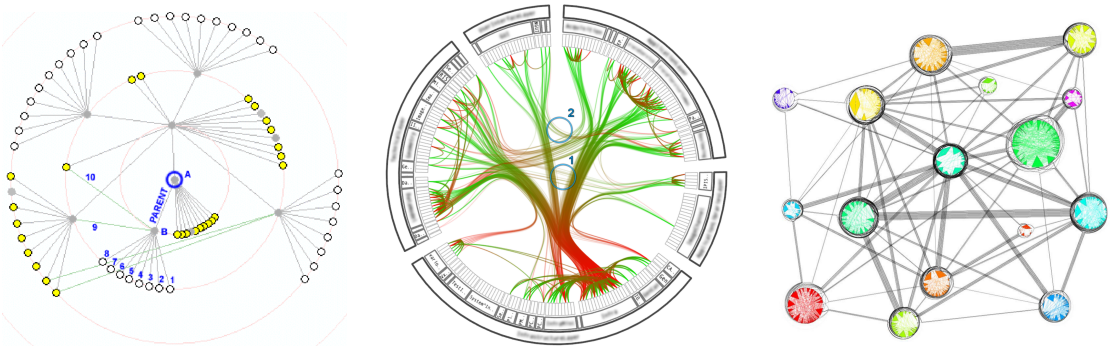
Kruchten *et al.* present a detailed comparison of layout types and their associated notations, using structural metrics to support design decisions in visualization development [KMM24]. Furthermore, several websites – like the *D3.js* gallery [Bos23] – present different layout techniques on different, predefined data without the possibility of a direct comparison. Mooney *et al.* evaluated a selection of popular layout techniques regarding calculated metrics based on synthetically generated graphs [Moo+24].

With a focus on node-link diagrams, Guckes *et al.* presented a *GuidelineExplorer*, an interactive tool to recommend visualization guidelines based on different task perspectives [Guc+24]. Within this tool, different guidelines can be tested and directly applied to uploaded or randomly generated graphs.

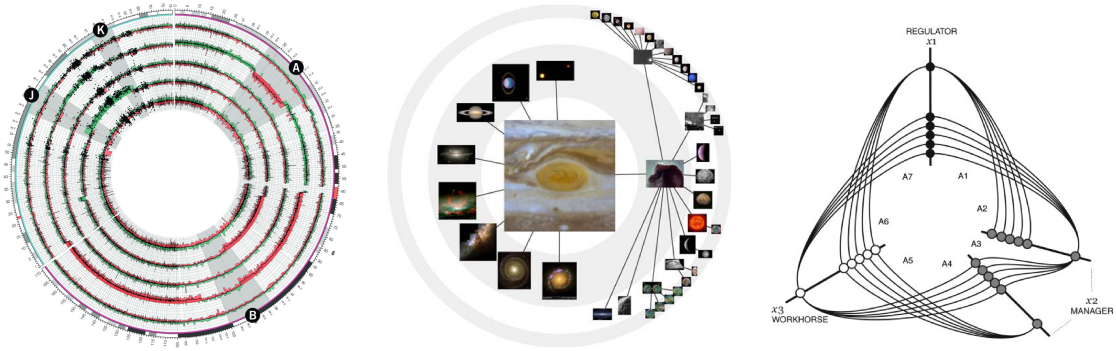
Graph manipulation and visualization tools like *yEd* [yWo25], offering a range of automatic layout algorithms, can also be used to evaluate different layouts.



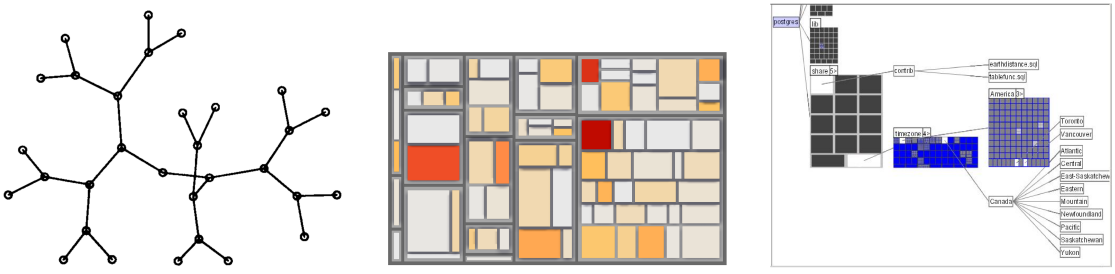
(a) *Arc Layout [GBD09; Hol24]* (b) *Space Filling [MK08]* (c) *Chord Diagram [Koo+17]*



(d) *Hierarchical Radial Layout [Yee+01]* (e) *Hierarchical Edge Bundling [Hol06]* (f) *Multi-Circular Layout [BB08]*



(g) *Circos [Krz+09]* (h) *MoireGraphs [JK03]* (i) *Hive Plots [Krz+12]*



(j) *Force-Directed Graph [FR91]* (k) *Treemap [Sch+20]* (l) *Elastic Hierarchies [ZMC05]*

Figure 2.3: *Selection of Existing Visualization Techniques for Graphs.*

Chapter 3

Fundamentals

Each graph with nodes and edges can be interpreted as communication networks, with nodes representing entities, and connections representing the relations between the nodes. In this thesis, the definition for communication networks is specified to contain connections between *communication nodes* based on *channels* and *topics*.

3.1 Communication Nodes

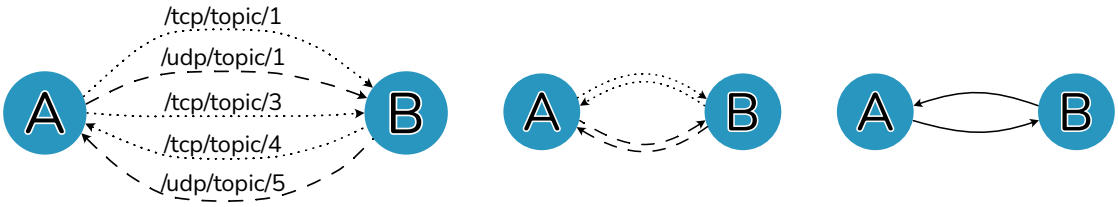
Communication nodes are the basic building blocks of communication networks, representing the entities that exchange information. The granularity and level of abstraction of these nodes can vary depending on the system and the context in which they are used. For instance, in social networks, nodes can represent individuals or organizations. In computer networks, nodes can represent devices (such as edge-devices in the Internet of Things (IoT) [Li+10; YS16]) or processes on the same or on different devices (like node processes in the Robot Operating System [Qui+09; Mac+22]). In either scenario, each communication node forms a semantically and syntactically coherent entity capable of sending and receiving messages from other nodes in the network.

3.2 Channels and Topics

The communication nodes in communication networks are connected via directed edges, each assigned to a communication **channel** c and a communication **topic** t.

Each **channel** represents a distinct type of communication. A network can contain one or multiple such channels, depending on the underlying system architecture and communication requirements. The concept of channels helps to refine the visualization, for example, by assigning different priority levels to different channels so that scoring and layout algorithms treat them differently. However, in the simplest case, a communication network can only contain a single channel for all connections.

In computer networks, channels can correspond to different transport protocols, such as TCP or UDP, or also communication endpoints like ports [KR12]. In robotics systems like in ROS, channels often distinguish between different connection mechanisms, such as asynchronous publisher/subscriber links and synchronous server/client interactions [Qui+09]. In the context of IoT, channels may reflect different physical or wireless



(a) *Topics between Nodes* (b) *Aggregated Topics per Channel* (c) *Aggregated Topics*

Figure 3.1: *Connections between nodes with different topics on different channels.* ***(a)*** *Nodes A and B connected by 5 distinct topics. In this example, these topics communicate on two different communication channels: Channel TCP visualized by a pointed line, channel UDP by a dashed line.* ***(b)*** *Variant to aggregate the topics per communication channel and direction.* ***(c)*** *Variant to aggregate the topics between nodes per direction, combining different communication channels.*

communication technologies, including Ethernet, WiFi, Bluetooth, ZigBee, or LoRa [Man+23].

Within each communication channel, a connection from a source node to a target node is associated with a specific **topic**. This topic defines the name of the data to be transmitted, allowing multiple connections to exist between the same pair of nodes, each using a different topic. In computer networks, such topics often semantically correspond to endpoints for data or services.

Derived from the semantics in ROS, topic-based connections can also be defined in a more implicit way by assigning outgoing and incoming topics to the nodes themselves [Qui+09]. Nodes having an outgoing topics t are called *publishers* and nodes having incoming topics t are called *subscribers*. This way, every publisher is connected to every subscriber on the same topic t, forming all possible types of connections: one-to-one, one-to-many, many-to-one, and many-to-many connections.

3.3 Characteristics of Communication Networks

Communication networks have several distinct structural characteristics and can vary greatly in complexity [Sha64]. Connections between nodes can be bidirectional, allowing participants to both send and receive information via different topics. While the most straightforward communications follow linear pipelines with unique topics, real-world graphs are typically more intricate with one-to-many or many-to-many connections [Qui+09]. Complex graphs often have a high degree of multiconnectivity between nodes, with multiple (sometimes up to 100 or more) topics established between two nodes [CHS21; Mac+22]. This leads to the notable characteristic that the number of connections with multiple interacting publisher and subscriber nodes can significantly exceed the number of nodes.

The density becomes especially evident in areas where **broadcast patterns** are implemented, creating fully connected subgraphs. These broadcast mechanisms are very

common in practical applications and frameworks like ROS, appearing in various forms such as system-wide logging, event handling, parameter updates, health monitoring, and status checks [Mac+22].

Commonly, communication networks contain **circular communication patterns**, where paths form loops through multiple nodes. This differentiates them from more constrained graphs like trees or directed acyclic graphs (DAGs), which are often inherent to data structures like build-systems or dependency graphs [STT81; GNV88].

Robotics frameworks frequently enable a hierarchical organizational structure. Similarly, in systems containing multiple devices, nodes belonging to the same device could be regarded as a group. For the purpose of this thesis, however, explicit **hierarchies** or **group memberships** within the communication network are not assumed in order to keep the approach as universal as possible. Thus, nodes can freely establish connections to other nodes without predetermined levels of organization. Similarly, there are no predefined group memberships that would cluster nodes into fixed sets.

To enable a more precise discussion of the nodes in this thesis, the following semantic categories are introduced. It is important to note that these classifications are used informally and loosely, as there is no strict or official categorization:

Source Nodes Nodes with mainly outgoing topics.

Sink Nodes Nodes with mainly incoming topics.

Processing Nodes Nodes with outgoing and incoming topics.

- **Splitter Nodes** Nodes with many outgoing and few incoming topics.
- **Merger Nodes** Nodes with few outgoing and many incoming topics.
- **Manager Nodes** Nodes with multiple outgoing and incoming topics.

Broadcast Nodes Nodes with connections to (almost) every other node on the same topic(s).

Figure 3.2 illustrates in (b) and (c) that both broadcast and manager nodes have a high degree of connectivity. To distinguish between the two types of nodes, broadcast nodes are defined as nodes with many connections on a smaller number different topics compared to manager nodes with many connections on a larger number of different topics.

3.4 Mathematical Representation

Let $G = (N, E)$ be a directed graph representing the structure of a communication network, where:

- N is the set of nodes in the network
- E is the set of edges between nodes in the network

The set of *topic edges* is defined as

$$E_T = \{(n_i, n_j, t) | (n_i, n_j) \in N^2, t \in T\},$$

where each topic edge has an associated topic t. We describe n_i as a *publishing node* and n_j as a *subscribing node* on t. A topic $t \in T$ is defined as a tuple

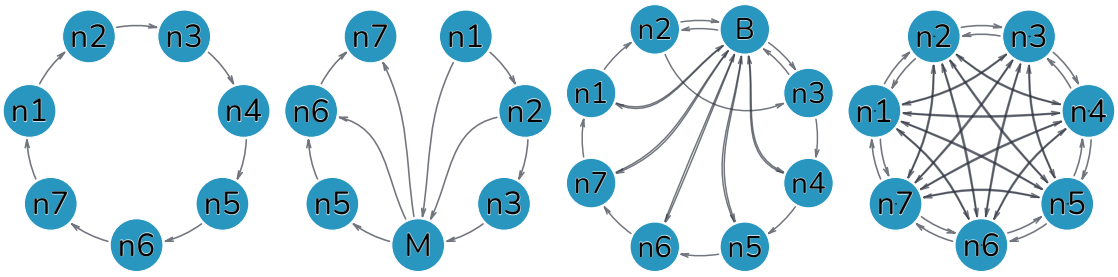
$$t = (id_t, c_t) \in \text{String} \times C,$$

where id_t is an assigned string identifier for t and $c_t \in C$ is the assigned communication channel for t.
Since there can be multiple topic edges between the same node on different topics, we can also define aggregated edges between nodes as

$$E_N = \left\{ \left(n_i, n_j, \{t_1, ..., t_k\} \right) | (n_i, n_j) \in N^2, t_k \in T \right\}.$$

Nodes can be grouped into communities, represented as a collection of subsets of N

$$\mathcal{P} = \{P_1, ..., P_k\} \text{ where } P_i \subseteq N \text{ and } P_i \cap P_j = \emptyset.$$



(a) *Cyclic Pipeline* (b) *Pipelines with Manager Node* (c) *Cyclic Pipeline with Broadcast Node* (d) *Cyclic Pipeline with Broadcast Connections*

Figure 3.2: *Selection of connection characteristics in communication networks.* ***(a)*** *Simple communication network containing 7 nodes in a cyclic pipeline.* ***(b)*** *A manager node M managing and combining two other pipelines $\{n1 \to n2 \to n3\}$ and $\{n5 \to n6 \to n7\}$. In this example $n1$ acts as source node, $n7$ as sink node, $n2, n3, n5$ and $n6$ are processing nodes.* ***(c)*** *The same pipeline as in (a) with an additional broadcast node B having connections to every other node in the network, e.g. for health monitoring. In contrast to the manager node, the connections on B are all on the same topic, e.g. health_status.* ***(d)*** *The same pipeline as in (a) but with broadcast connections between all nodes, e.g. for parameter update events.*

Chapter 4

Approach

The focus of this work is the visualization of an overview for communication networks, as this is the task that is currently insufficiently scalable in existing tools like the ones presented in Figure 2.1. According to the Visual Information Seeking Mantra [Shn03], the overview as first contact with data provides a broad view of the entire system. This overview should avoid information overload and visual clutter in order to be comprehensible. The tasks addressed by this overview visualization include, among others, the capturing of connected components or clusters, pattern or outlier identification, and the estimation of specific properties like the system size or density [Guc+24]. Following features based on this overview, such as *zooming in* and *filtering*, as well as the display of *details on demand*, can be implemented subsequently using user interaction and multi-view approaches [Ger+09].

In summary, the approach in this thesis for overview visualization of communication networks aims to provide the following information:

1. The overall **structure** of the network, giving insight into the size and density of the system, connected components, and clusters and outliers.

2. The **relationships** and **dataflow** between nodes, including the direction and structural importance of the connections, as well as the identification of major communication paths.

3. The **significance** of key nodes, emphasizing strongly connected and structurally important nodes.

4.1 Preprocessing

When looking at complex graphs, there is often a kind of intuition about which nodes seem to be more or less important for a network [Fre78]. By emphasizing those important nodes in a graph visualization – e.g. by size, color, or position – viewers can grasp the structure and flow of information within the network and recognize specific nodes preattentively [Hea92]. Especially node scores, as quantitative values, can be communicated well in terms of the size of their graphical representations [Pfe13].

In networks, the importance of nodes can be assessed based on their semantic meaning. Information about semantic meaning can be provided by developers of parts of the system

or by insiders familiar with the data. There are also frameworks, for instance for ROS, that observe the communication traffic in a live system [ROS24a] or statically analyze the source code to predict the communication style [SCM19; San24]. Both methods could be used to retrieve information about the communication frequency and data traffic. However, in many cases, this information might not be available or easily accessible. As a consequence, the goal of this work is to find a generically applicable method to measure the importance of nodes and edges based entirely on the structural, syntactic information of the graph. For communication networks, this structural information is mainly given by the topic connections between nodes, which is assumed to be available in the system's metadata.

In social networks, the importance of a node is often associated with its connectivity, i.e. the number of direct connections it has to other nodes [Fre78]. However, the presence of broadcast nodes in communication networks illustrates, that a high connectivity of a node does not necessarily indicate its structural importance, as shown in Figure 3.2(b) and Figure 3.2(d). The same applies to broadcast connections between nodes that also have other, structurally more important connections. This can be seen in Figure 3.2(d), where the actual important connections from Figure 3.2(a) (all on different topics) are extended by broadcast connections between every node (all on the same topic). In both cases – broadcast connections between existing nodes or to a dedicated broadcast node – the meaningful connections are obscured by less important broadcast connections.

As an analogy, one can imagine a computer network with several servers that continuously process data via chained API calls. If another server is introduced whose only task is to ping every other server once an hour, this ping server would likely be the one with the most connections to other servers. Admittedly, it would also be the least important server in terms of the overall structure of the system. The same analogy applies to social networks, for example in the structure of a group of friends. If a postman is added into the system who has contact with every group member by delivering letters, this postman is the most connected, but might not be the most important person in the group of friends. The examples above show connections of lesser importance based on two criteria: firstly, a **lower frequency** of communication and secondly, a **similar type of connection** to all communication partners.

In research regarding social and communication networks, *centrality* was introduced as measurement to indicate, which nodes take up critical positions in a network [ZL17]. However, there is no standardized method for all graphs and cases to clearly evaluate the centrality and thus the significance of a node. For instance, the well-known Python library *NetworkX* provides over 40 distinct centrality methods [Net25].

There are three basic approaches of calculating the centrality for nodes in a graph [BJT23]:

Degree Centrality The degree centrality C_D focuses on the number of direct connections a node $u \in N$ has. This measurement provides information about the potential of communication activity [Fre77]:

$$C_D(u) = \frac{\deg(u)}{|N| - 1} \tag{4.1}$$

Betweenness Centrality Based on the count of shortest paths passing through a node. Nodes which are part of many shortest paths have a high betweenness centrality, emphasizing nodes, that undertake *mediation* role of the communication in a network [WB94]:

$$c_B(u) = \sum_{s,t \in N} \frac{\sigma(s, t \mid u)}{\sigma(s, t)}, \tag{4.2}$$

where $\sigma(s, t)$ is the number of shortest (s, t)-paths, and $\sigma(s, t \mid u)$ is the number of those paths passing through node u (with $u \neq s, t$).

Closeness Centrality Measures how near or reachable a node is to other nodes in the network. In its original form, closeness centrality can also be interpreted as inverse dependency of other nodes for communication. A node depending on a lot of intermediate participants to reach the communication target has a low centrality, and vice versa [Bea65]:

$$C_C(u) = \frac{n - 1}{\sum_{v=1}^{n-1} d(v, u)}, \tag{4.3}$$

where $d(v, u)$ is the shortest-path distance from v to u, and $n - 1$ is the number of reachable nodes from u.

While these basic centrality measurements may be viable for simple graphs like in Figure 4.3, they all fail in the presence of broadcast patterns in the network: A broadcast node with bidirectional connections to every other node will always have the highest degree, the highest betweenness as well as the highest closeness centrality. Other centrality calculations lead to the same result as long as broadcast connections are treated as edges with the same weight as other connections.

4.1.1 Edge Significance

To address this challenge, this thesis proposes a heuristic approach for calculating the topic significance based on its usage patterns. The fundamental assumption of the approach is that the same topics used by many different nodes likely represent general-purpose or broadcast communications, and thus carry less structural significance. On the other hand, topics used by only a few nodes for specific interactions likely represent more specialized and structurally significant communications. With this assumption, the weight $w_T(t)$ for a given topic t is defined as:

$$w_T(t) = \sqrt{\frac{1}{\max(u(t), 1)}}, \tag{4.4}$$

where $u(t)$ represents the topic usage count, calculated as

$$u(t) = |N_{out}(t)| \cdot |N_{in}(t)| - |N_{outIn}(t)|, \qquad \text{with} \tag{4.5}$$

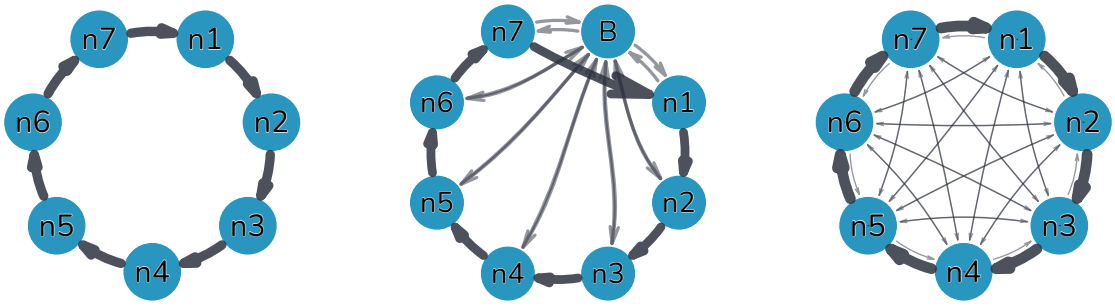
- $|N_{out}(t')|$ being the number of publishing nodes for topic t',
- $|N_{in}(t')|$ being the number of subscribing nodes for topic t', and
- $|N_{outIn}(t')|$ being the number of nodes that both publish and subscribe to topic t' (self-loops).

The total weight w_N of an aggregated edge e_{ji} between the nodes n_i and n_j is the sum of all topic edge weights between these nodes:

$$w_N(e_{ij}) = \sum_{t \in e_{ij}} w_T(t) \tag{4.6}$$

This approach distinguishes precisely between edges of manager nodes and broadcast nodes. Taking the examples of Figure 3.2(b) and assuming, that each edge from and to the manager node M is unique, they all have a weight of 1. On the other hand in Figure 3.2(c), assuming that all incoming broadcast edges to B have the same topic t_b, they would have a weight of $w_T(t_b) = \sqrt{1/7} \approx 0,378$. Adjusting the graphs according to the results of this edge weight calculation leads to better distinguishability between important and less important edges, as shown in Figure 4.1.

Based on the assumption to visualize communication networks on the basis of their structural information, semantic meaning or importance cannot be taken into account as it only focuses on the topics' usage count. Their might be *broadcast*-like nodes that



(a) *Cyclic Pipeline*

(b) *Cyclic Pipeline with Broadcast Node*

(c) *Cyclic Pipeline with Broadcast Connections*

Figure 4.1: *The pipeline examples as in Figure 3.2 but with the heuristic edge weight from Equation* (4.6) *applied. After applying the edge weighting, the structure of the pipeline from (a) is still visible in (b) and (c) despite the broadcast connections.*

provide indispensable and frequent information for all other nodes. Even in this case this approach is still valuable to evaluate the *influence* of edges regarding the network structure.

4.1.2 Node Significance

This thesis proposes, that a centrality score for each node in the communication network should fulfill the following properties to ensure an intuitive and meaningful comprehension:

- Nodes with a larger number of different connected topics should have a higher node score.
- Node scores should not depend on the direction of connected topics.
- Node scores of a system should not be influenced too much when add or remove a broadcast node.
- A system's node scores should not be affected too much by unimportant connections (i.e., broadcast connections that occur frequently in the system).

The calculated edge weights allow us to apply centrality measures based on shortest path calculations. To do so, the weight of an edge e_{ij} is converted to the following reciprocal distance:

$$d_N(e_{ij}) = \Big(\frac{1}{w_N(e_{ij})}\Big)^2 \tag{4.7}$$

The *Harmonic Centrality* [BV14] as extension of the closeness centrality calculates the centrality of a node u as $C(u) = \sum_{v \neq u} \frac{1}{d(v,u)}$, more precisely the sum of the reciprocal of the shortest path distances $d(v, u)$ from all other nodes $v \in G$ to u. Connections with a low weight – and thus a large distance – contribute less to the resulting centrality and vice versa.

In the shown default equation, the harmonic centrality for directed graphs calculates only the incoming centrality for nodes. This can be resolved by not only taking incoming shortest paths into account but also add outgoing paths. To further reduce the influence of low weighted edges, a root sum of squares approach can be added. Thus, in this thesis the following calculation of the *Harmonic Communication Centrality HCC* of a node $u \in N$ is proposed as

$$HCC(u) = \sqrt{\sum_{v \in N | v \neq u} \Big(\frac{1}{d(u,v)}\Big)^2 + \sum_{v \in N | v \neq u} \Big(\frac{1}{d(v,u)}\Big)^2}. \tag{4.8}$$

As variant of Equation (4.8), a second equation for calculating the communication centrality is proposed, that also rewards the nodes that are part of a shortest path, inspired by the concept of betweenness centrality [Fre78]. This *Communication Path Centrality CPC* of a node $u \in N$ is defined as

$$CPC(u) = \sqrt{\sum_{v \in N | v \neq u} \Big(\frac{1}{d(u,v)}\Big)^2 + \sum_{s,t \in N | u \in P_{st}} \Big(\frac{1}{d(s,u)}\Big)^2}, \tag{4.9}$$

with $d(u, v)$ being the shortest-path distance between u and v, P_{st} being any shortest path $\{s, n_1, ..., t\}$, and $d(s, u)$ being the distance from s to u on a shortest path between s and t. With this method, the centrality of u is not only influenced by shortest paths starting from and ending in u, but also by every other shortest path containing u. This approach is less penalizing on nodes that mainly offer low scored broadcast connections but are still part of a lot of paths between other node pairs. However, including intermediate nodes of the shortest paths introduces a potential problem of instability of the centrality computation in networks with very similarly weighted edges, as described by Segarra and Ribeiro [SR14].

The node scores $S_{CPC} = \{(u, CPC(u)) | u \in G\}$ (or the same for HCC) can be normalized to a maximum value of 1 to improve the comparability. The resulting normalized node scores S'_{CPC} is defined as

$$S'_{CPC} = \left\{ \left(u, \frac{CPC(u)}{\max(S_{CPC})} \right) | u \in G \right\} \tag{4.10}$$

Both Communication Centrality equations fulfill the stated requirements of handling communication connections in general and deal properly with less important broadcast connections. Including intermediate nodes in $CPC(u)$ leads to greater differences in centrality values and thus improved distinguishability.

As visualized in Figure 4.3, the introduction of a broadcast node can distort the centrality scores of established equations, obscuring the *true* relevance of nodes. Notably, both proposed centrality computations for communication networks, provide stable centrality scores, independent of the presence of broadcast nodes in the system.

4.1.3 Community Detection

Communication networks do not have predefined group memberships for their nodes. However, visualizations can benefit from automatic node aggregation to better reveal structural patterns within complex networks. The Louvain algorithm, known for its computational efficiency and accuracy, provides a robust method for identifying such communities by optimizing modularity through an iterative approach [Blo+08]. This algorithm can be used to divide complex networks into distinct, interconnected groups without any prior knowledge of their structure, as visible in Figure 4.2 and Figure 4.10. The modularity optimization of the Louvain algorithm is also capable of handling broadcast connections if edges are weighted using Equation (4.4).

4.2 Node Layout

The node layout process in the presented visualization approach consists of the following steps: **(1)** style computation of the nodes, **(2)** linear sorting of the nodes, and **(3)** radial positioning of the nodes.

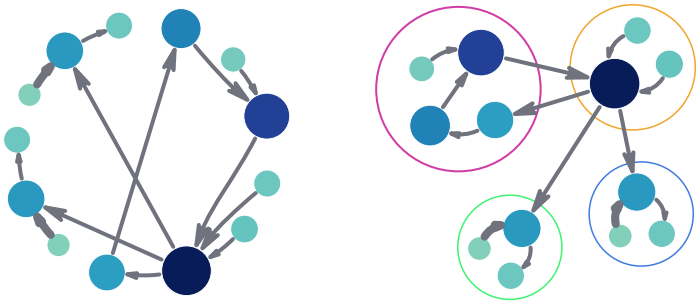


Figure 4.2: *The ungrouped nodes on the left layout are grouped into distinct communities on the right after applying the Louvain algorithm.*

4.2.1 Node Style

In larger graph visualizations it is common to adapt the visual mapping of nodes based on different criteria in order to support the recognizability of nodes and the preservation of a mental map [Bec+17]. The calculated node score (Equation (4.10)) is mapped to the node radii the following way:

$$r(s_n) = \max\Big(0\ ,\ \frac{ln(s_n) - ln(s_{min})}{ln(s_{max}) - ln(s_{min})}\Big) \cdot (1 - r_{min}) + r_{min}, \tag{4.11}$$

while

- s_n is the calculated score of a node n, e.g., $s_n \in S'_{CPC}$
- s_{min} is the score at which we want to have the smallest resulting node size in the visualization, e.g., $s_{min} = 0.2$. Nodes below this score will not get a smaller node radius.
- s_{max} is the score at which we want to have the highest resulting node size in the visualization, e.g. $s_{min} = 1$ (which is the highest value of normalized scores). Nodes above this score will not get a larger node radius.
- r_{min} is the smallest radius getting assigned at a node having s_{min} as score.

Thus, the scores of the nodes are mapped from $[s_{min}, s_{max}]$ to $[r_{min}, 1]$ using a logarithmic scale. s_{min}, s_{max} and r_{min} can be selected as required to keep the differences in node size within the desired range.

Furthermore, the node score is mapped to a sequential color scheme to further improve the preattentive distinctiveness between the nodes [Ban+13]. For example, in Figure 4.3 the *YlGnBu* scheme is used [HB03].

4.2.2 Radial Node Placement

Radial node placements are an approach found in various visualization applications, e.g., in bioinformatics [Krz+09] and other focus+context visualizations [JK03]. In this approach, nodes are linearly placed on a circle, inherently fulfilling requirements regarding graph aesthetics like an even node distribution, and a minimization of node-node and node-edge

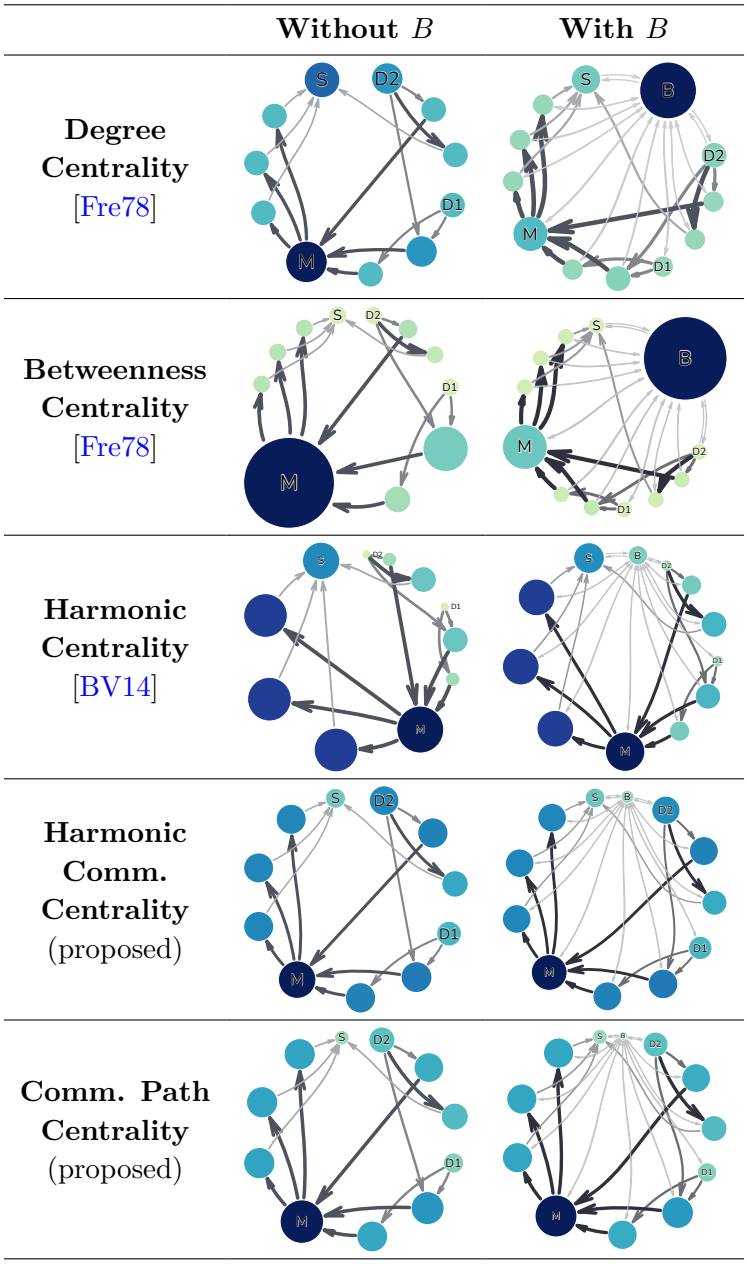


Figure 4.3: *Different node centrality calculation methods for the same 11-node communication network with edge weights calculated with Equation* (4.6)*. The score is visualized by the size and color of a node. In this network,* $D1$ *and* $D2$ *are data source nodes,* S *is a data sink node, and* M *is a manager node combining source nodes with sink nodes. Unlabeled nodes are intermediate processing nodes. The left column shows the scores of nodes in the network without the broadcast node* B*. The right column shows the network extended by the broadcast node* B*, which is bidirectionally connected to every other node in the network.*

overlaps. [Ben+07]. Unlike other linear placements like time-arc trees [GBD09], radial node placements efficiently fit data into square spaces, making them more space-efficient.

Given the assumption that circular characteristics are frequently present in communication networks, a radial placement approach is chosen for the visualization in this thesis. To support the comprehension of dataflows within networks and reduce visual complexity, an effective sorting of the nodes as basis for the placement is required.

Child Node Sorting. An effective sorting of nodes for a radial placement targets the optimization of multiple criteria. In particular, it should minimize distances between adjacent nodes to ensure that the edge distances in the layout are also minimized. Additionally, it should maximize the consistency of the data flow direction, which implies minimizing backward edges in the sorting [Ben+07].

Topological sorting is a widely recognized technique for sorting Directed Acyclic Graphs (DAGs) [Kah62; PK07] and thus for a linearized arrangement of nodes. However, when dealing with cyclic graphs, these conventional techniques are either not applicable or lead to weak sorts with an arbitrary node order within a cyclic node group. Especially when dealing with dense graph structures due to broadcast mechanisms, the common approach of the condensation of strongly connected components completely ignores substructures in the graph [Cor09].

To calculate a pseudo-topological sorting of nodes, this thesis proposes the *Weighted Flow Sorting* algorithm combining the following steps:

1. Edge-weight based, greedy **removal of cycles** in the graph.
2. Calculation and optimization of **topological generations**.
3. Visit nodes in an **modified depth-first search** to derive optimized sorting based on the topological generations.

In general, finding a minimum set of edges to remove for a graph to be acyclic – known as *minimum feedback arc set problem* – is NP-hard [MTB72]. Inspired by the simple heuristic approach from Eades *et al.* [ELS93], cycles are handled in a similar way in the first step (*getCycleFreeParents()*) of Algorithm 1. This is done by iterating the edges in descending order according to their weight and ignoring edges that would lead to circles. Based on the cycle-free graph the topological generations can be trivially calculated (*assignTopologicalGenerations()*) [Net24b].

As final step of Algorithm 1, the node assignment to topological generations can be adapted so that each node is in its latest possible generation (*adaptGenerations()*). This naturally leads to reduced distances between nodes in a sorting derived from the generations.

The *Weighted Flow Sorting* is then calculated in Algorithm 2 by traversing the nodes of the cycle-free graph in an adapted depth-first search, based on the topological generations. Starting with nodes of the first generation, a node n is added to the sort result, if all parents of n are already sorted.

Algorithm 1: Weighted Topological Generations.

```
   input  : nodes as list of nodes to get the topological generationAssignments for
   output : nodes with a generation number assigned
 1 components ← getConnectedComponents(nodes)
 2 foreach component ∈ components do
 3     assignTopologicalGenerations(component)
 4     adaptGenerations(component)
 5 return nodes
 6
   // Derive a cycle free graph as 1st step of the algorithm
 7 function getCycleFreeParents(nodes) : Map<Node, Node[]>:
 8     parents ← {n : ∅ for n ∈ nodes}
 9     visitedAncestors ← {n : ∅ for n ∈ nodes}
10     sortedEdges ← sortDescending(allEdgesOf(nodes) by edge => edge.weight)
11     foreach edge ∈ sortedEdges do
           // Remove cycles by checking if edge.target is ancestor of edge.source
           // If so, skip the parent assignment
12         if edge.target ∈ visitedAncestors.get(edge.source) then
13             continue
           // Store parent relationship with edge weights
           // Store parent relationship with edge weights
14         parents [edge.target][edge.source] ← edge.weight
           // Update ancestor relationships based on the current parent relationships
15         visitedAncestors.update(parents)
16     return parents
   // Assign topological generations to nodes based on the cycle free graph as 2nd step of the
      algorithm
17 function assignTopologicalGenerations(nodes):
18     parents ← getCycleFreeParents(nodes)
       // Nodes start with an undefined assigned generation.
       // Assign generation 0 to all nodes without parents.
19     foreach node ∈ nodes do
20         if parents [node] = ∅ then
21             node.gen ← 0
       // Assign generations to remaining nodes
22     while (nodesToAssign := filter(nodes, n ⇒ n.gen == undefined)) ≠ ∅ do
23         foreach node ∈ nodesToAssign do
               // Here every node will have at least one parent, otherwise they would have get gen
                  == 0 assigned before.
24             if every(parents[node], p ⇒ p.gen ≠ undefined) then
25                 node.gen ← max(parents[node], p ⇒ p.gen) + 1
   // Adapt the generations to minimize generation gaps as 3rd step of the algorithm
26 function adaptGenerations(nodes):
27     reversedGenerations ← sort({nodes.map(n => n.gen)}, descending)
28     parents ← getCycleFreeParents(nodes)
29     children ← reversed(getCycleFreeParents(nodes))
30     foreach gen ∈ reversedGenerations ( do
31         foreach node ∈ filter(nodes, n ⇒ n.gen == gen) do
32             foreach parent ∈ parents[node] do
33                 parent.gen ← min(children[parent], c ⇒ c.gen)
```

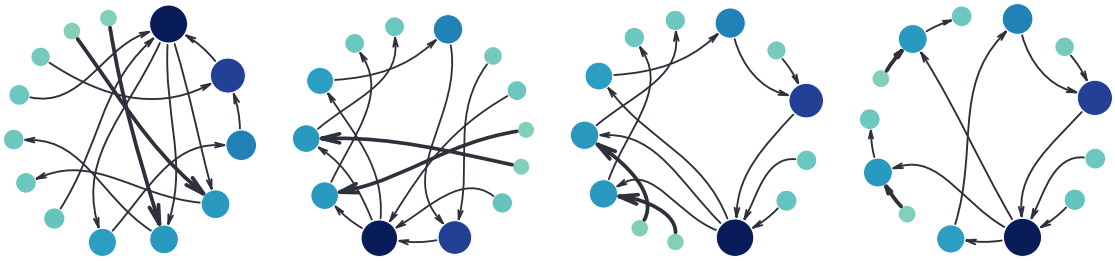
Algorithm 2: Weighted Flow Sorting.

```
   Input  : A list of nodes
   Output : A sorted list of nodes
 1 parentMap ← getCycleFreeParentMap (nodes)
 2 childrenMap ← reversed(parentMap)
 3 sortedNodes ← []

 4 function successors(node):
 5     return sort(childrenMap[node], descending by weight)
 6 function predecessors(node):
 7     return sort(parentMap[node], descending by weight)
   // Recursive helper function to visit nodes
 8 function visitNode(node):
 9     if node ∈ sortedNodes then
10         return
11     parents ← predecessors(node)
12     if parents.length == 0 or parents.every(p ⇒ p ∈ sortedNodes) then
           // Append node to sortedNodes
13         sortedNodes ← sortedNodes ∪ {node}
14     sortedChildren ← sort(successors(node), desc. by total successor weight)
15     foreach child ∈ sortedChildren do
16         visitNode(child)
   // Process each connected component of the nodes separately
17 foreach component ∈ getConnectedComponents(nodes) do
       // Visit the nodes based on the topological sorting until each node is sorted.
18     topoSorting ← getWeightedTopologicalSorting (component)
19     while component \ sortedNodes ≠ ∅ do
20         foreach node ∈ topoSorting do
21             visitNode (node)

22 return sortedNodes
```

In combination with the edge weight calculation of Equation (4.4), Flow Sorting effectively handles cyclic communications and broadcast patterns in complex graphs. Compared to other sorting algorithms, Weighted Flow Sorting has consistently better results in terms of total number of edge crossings and total path length, see Table 4.1. The result of *Weighted Flow Sorting* is exemplary visualized in Figure 4.4.

Table 4.1: *Comparison of different sorting algorithms on the same graph (26 nodes and 452 edges) using a radial layout approach. Edge Crossings is the metric calculated with Equation* (5.1)*. Path Length Ratio is the total path length compared to the best value. Algorithms marked with * are the proposed algorithms (in case of Flow an unweighted version of the algorithm).*

Sorting Algorithm	Edge Crossing Metric	Total Edge Crossings	Path Length Ratio
Weighted Flow*	**0.09**	**79**	**100.0%**
Depth First	0.11	96	118.3%
Flow*	0.11	96	121.5%
Weighted Topological*	0.21	187	140.1%
Weight. Topo. Unadapted*	0.21	191	141.1%
Topological	0.22	194	139.2%
Breadth First	0.25	221	131.7%
By Id	0.26	234	152.5%
Node Score	0.28	248	161.2%
Degree	0.37	333	172.0%



(a) *Node Score Sorting* (b) *Straight-forward Topological Sorting* (c) *Adapted Topological Sorting* (d) *Flow Sorting*

Figure 4.4: *Different linear sorting methods visualized in a radial placement. For each sorting the first node is placed shortly after the 12 o'clock position, subsequent nodes are placed clockwise.* ***(a)*** *shows one of multiple trivial sortings using the node score as criteria. Other trivial sortings could be based on a node's degree or its names.* ***(b)*** *and* ***(c)*** *are based on the Weighted Topological Generations calculated with Algorithm 1. In both sortings, nodes of the same generation are placed directly next to each other. In (c) the generations are adapted to minimize gaps by assigning every node the largest possible generation.* ***(d)*** *is the resulting Weighted Flow Sorting, based on Algorithm 2.*

Circular Node Placement. Nodes sorted by Algorithm 2 can be placed on a circle. The circular node distribution process shown in Algorithm 3 follows a three-step approach:

1. **Position Calculation**: First, the algorithm computes continuous positions for all nodes based on their sizes. It creates a linear arrangement where nodes are placed sequentially with configurable margins (m_p) between them. The total length of this arrangement determines the radius of the circle on which the nodes will be placed.

2. **Angular Distribution**: The linear positions are then mapped to angular positions of a circle in the range $[0, 2\pi]$. By adding a start angle of π, the first node is placed at the left of the circle, with subsequent nodes distributed clockwise.

3. **Circle Optimization**: Optionally, the algorithm can optimize the enclosing circle of the placed child nodes using an adapted version of Welzl's algorithm [Wel91]. The adaption of the enclosing circle is especially effective when the child nodes have large differences in their radii.

The placement approach ensures that nodes are evenly distributed while maintaining their relative spacing and preventing overlaps. The configurable margin factors m_p and m_r allow fine-tuning of the layout's appearance and density. In the basic layout without communities, Algorithm 3 is called on the root node of the graph, which contains all other nodes as direct children.

Grouped Positioning. In case the graph contains communities – either present in the data or computed during preprocessing – nodes can be arranged in a bottom-up manner across multiple layers. Nodes belonging to the same community are grouped

Algorithm 3: Radial Node Placement.

input : parent as the parent node for which the children are placed
m_p as margin factor for the node placement
m_r as margin factor for the parent node radius
adaptEnclosingCircle as option to optimize the enclosing circle of the parent node
output : Placed child nodes and
calculated radii of the parent

```
1  nodes ← parent.children ; positionMap ← ∅ ; lastPosition ← 0
   // Calculate continuous positions based on nodes' radii.
   // A margin of m_p = 0 would leed to touching nodes.
   // A margin of m_p = 1 adds the radius of the node as space between the next node.
2  foreach n ∈ nodes do
3      lastPosition ← lastPosition + n.outerRadius · (1 + m_p)
4      positionMap[n] ← lastPosition
5      lastPosition ← lastPosition + n.outerRadius · (1 + m_p)
   // Normalize positions to [0, 2π] to be used as angles.
6  foreach n ∈ nodes do
7      positionMap[n] ← 2π · positionMap[n]/lastPosition
   // Place the nodes on the circle with the calculated radius.
8  radius ← lastPosition/(2π)
9  foreach n ∈ nodes do
10     angle ← positionMap[n]
11     n.center ← getPositionOnCircle(angle, radius)
   // The resulting radius is based on the placed nodes and their sizes.
12 parent.radius ← (radius + max({n.radius : n ∈ nodes})) · m_r
13
   // Optional: Adapt to minimum enclosing circle.
14 if adaptEnclosingCircle then
15     enclosingCircle ← minEnclosingCircleWelzl(nodes)
16     parent.center ← enclosingCircle.center
17     parent.radius ← enclosingCircle.radius · m_r
```

into a *hypernode* [HMM00]. The size of a hypernode depends on its child nodes and the parameter m_r. Once the positions of the child nodes are set inside their hypernode, Algorithm 3 is used to position the hypernodes on the next layer. In this way, a multi-circular view is achieved [BB08]. In the simplest case, this grouping into hypernodes can be achieved based on the connected components of the graph, with each component having its own circular arrangement.

To improve the orientation of a hypernode in relation to its neighboring hypernodes, the child nodes can be slightly rotated in a subsequent step after Algorithm 3. This rotation is calculated based on the net direction of external connections coming from or going to nodes outside the hypernode, which is effectively a simplified form of a spring embedder [Kob12]. The resulting angle is used to locally rotate the children, aligning the hypernode more intuitively with the overall structure, as shown in the difference between Figure 4.10(b) and Figure 4.10(c).

4.3 Connection Layout

Complex node-link visualizations often suffer from visual clutter caused by edge crossings and poor connection routing. Existing visualization techniques for communication graphs – e.g., *rqt_graph* for ROS [TB18] – often render every single topic connection. In particular, given the multi-graph and broadcast characteristics of communication networks with a high potential number of connections between node pairs, it is not reasonable to render each connection in detail.

Following the *Visual Information Seeking Mantra*, connections should first be used to get an overview of the overall network structure [Shn03; Ger+09]. While the bundling of edges connecting different nodes is a common practice for hierarchical data or dense graphs – e.g., based on hierarchy [Hol06], on geometry [Wei+08] or on force simulations [HW09] – the presented approach only combines the edges of a node pair. In this way, the visualization inherently supports the task of identifying which nodes are connected to each other at the overview level.

Based on the radial placement of nodes, different connection layout approaches are proposed as part of this thesis, which completely avoid edge-node crossings and encourage clutter reduction [ED07].

4.3.1 Circular Arc Connections

Straight lines between nodes that are close together in the radial arrangement can result in node-edge crossings [Ben+07]. Inspired by Lombardi drawings [Dun+11; Che+12], connections between the radially placed nodes can be laid out using circular or elliptic arcs improve graph aesthetics and also avoid node-edge crossings [Pur02].

The circular arc connections from a source node S to target nodes T_i, with all nodes lying on the circle C, can be constructed using the following steps:

1. Select a helping point P on the circle C. The segment $l = \overline{SP}$ defines the line on which the curvature of circular arcs changes from convex to concave.

2. The orthogonal $o = \perp \overline{SP}$ defines the line on which the center points of all circular arcs lie.

3. Each circle for the arcs is constructed to be coincident to the center points of S and T_i with its center lying on o.

4. The circular arc for the connection is obtained by trimming the constructed circle to the edges of the source and target node.

By choosing an angle $\angle SCP < 180°$ it is guaranteed by definition, that an outgoing connection to a node has a distinct path from a incoming connection from the same node. In addition, this way connections to nodes being closer in the linear sorting are favoured by being shorter. Figure 4.5 shows an example of the construction for $\angle SCP = 120°$.

The same construction principle could be applied to route connections outside the parent circle. With this approach inside arcs above a certain threshold could be replaced to better distinguish between *forward edges* and *backward edges* as visible in Figure 4.6.

A characteristic of circular arcs constructed this way is that the outgoing edges start at almost the same point, just as the incoming edges arrive at almost the same point, thus the connections neighborhood-agnostic. Although this behavior is intuitive with regard to the radial placement of the nodes, it leads to a low angular resolution between the edges and thus to poorer readability [BST00].

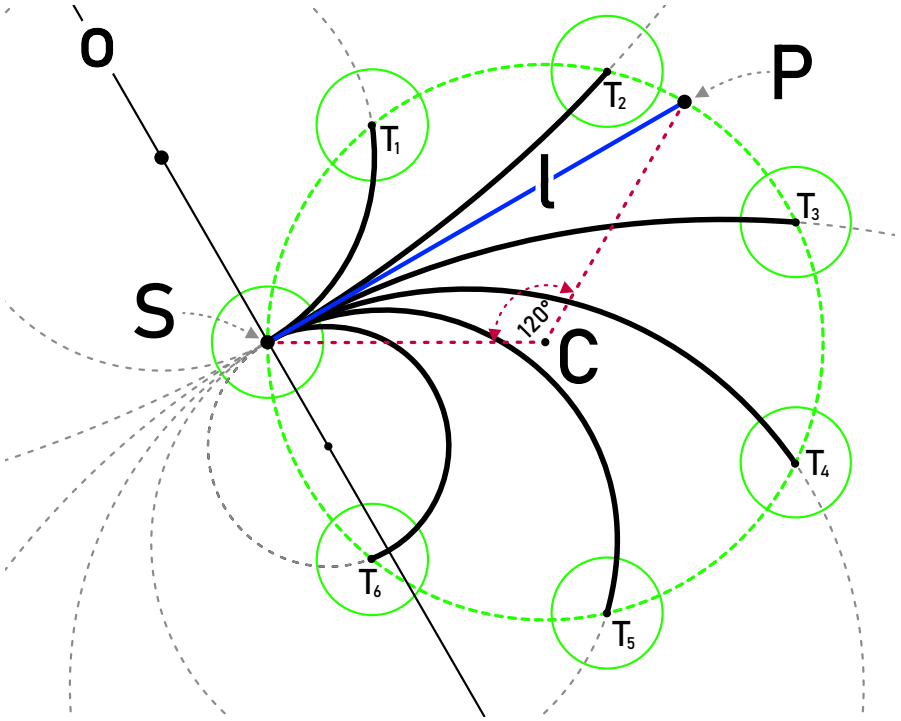
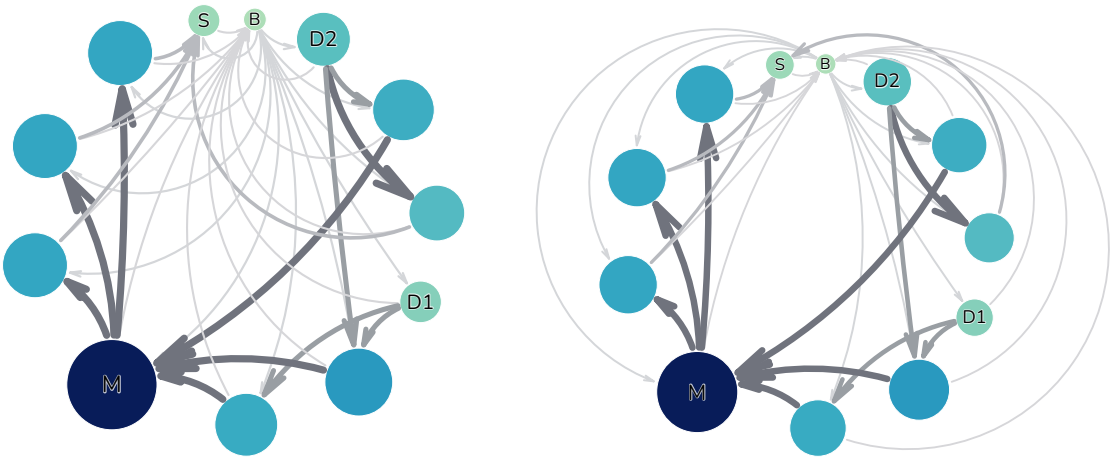


Figure 4.5: *The construction of circles for the circular arc connections from a source point S to all other nodes T_i in a radial layout with center C. In this case the point P is chosen such that $\angle SCP = 120°$. At this point the circular arcs switch from convex to concave.*



(a) $\angle SCP = 120°$ (b) $\angle SCP = 120°$ *with Outside Arcs*

Figure 4.6: *Circular arc connections with* $\angle SCP = 120°$*. Arcs in (b) are routed outside the parent circle for connections having a angular distance larger than* 180°*.*

4.3.2 Smooth Spline Connections

As a more flexible approach to using only circular arcs, this thesis proposes combining the following connection types:

Forward Arcs Circular forward arcs for clockwise connections between directly adjacent nodes in the radial positioning, with the same radius $r_{forward} = r_C$ as the circle C the nodes are placed on.

Backward Arcs Circular backward arcs for counterclockwise connections between directly adjacent nodes in the radial positioning, with a smaller radius for the circle segment, e.g $r_{backward} = r_C - f \cdot \min(r_u, r_v)$ with r_u and r_v being the radii of the connected nodes and f a scaling factor.

Smooth Splines Splines as more flexible curves for connections between other non-adjacent nodes.

To explain the layout of smooth splines, this work introduces the concepts of *connection ranges* and *connection anchors*.

Connection Ranges. In order to achieve maximum angular resolution, edges should ideally be distributed across the entire circumference of a node. However, in radial layouts, such distribution would introduce unnecessary edge crossings and edge-node intersections. To avoid these issues, the specific connection ranges for each node n are defined as follows:

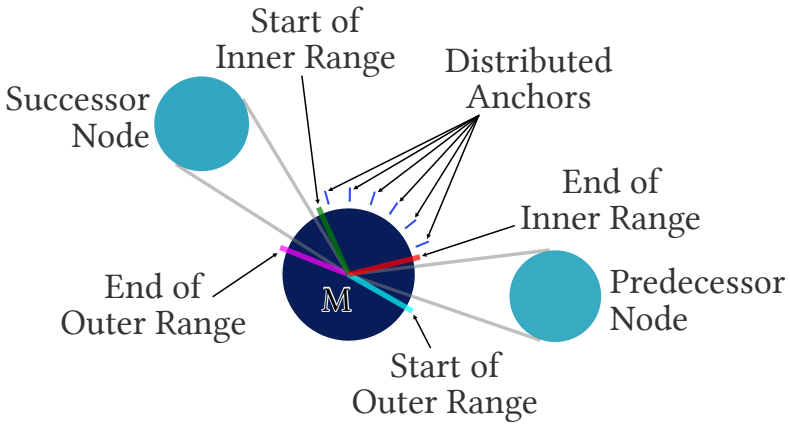


Figure 4.7: *Connection range construction of node M. The gray lines are the inner and outer tangents of node M's center to its directly adjacent nodes. The resulting inner range and the outer range both have a padding factor of $f_p = 0.1$ applied. Across the inner range six connection anchors are distributed, which are also used in Figure 4.8(a).*

Inside Connection Range

$$cr_{in} = \left[\theta_{in}^{start} + f_p \cdot \theta_{in}^{\Delta}/2 \ , \ \theta_{in}^{end} - f_p \cdot \theta_{in}^{\Delta}/2\right]$$

where θ_{in}^{start} is the angle for the inner tangent from n's center to the direct successor node's perimeter. Vice versa, θ_{in}^{end} is the inner tangent to the predecessor node. $f_p \in [0, 1]$ is a padding factor for the inner range.

Outside Connection Range

$$cr_{out} = \left[\theta_{out}^{start} + f_p \cdot \theta_{out}^{\Delta}/2 \ , \ \theta_{out}^{end} - f_p \cdot \theta_{out}^{\Delta}/2\right]$$

where θ_{out}^{start} is the angle for the outer tangent from n's center to the direct predecessor node's perimeter. Vice versa, θ_{out}^{end} is the outer tangent to the successor node. $f_p \in [0, 1]$ is a padding factor for the outer range.

The inner connection range handles connections between non-adjacent nodes on the same hierarchical level as in simple radial layouts. By using tangent lines between a node and its neighbors to define these ranges, natural pathways are created where connections can flow without intersecting node bodies. This geometric approach naturally responds to the layout context – when nodes are larger or positioned closer together, the valid connection ranges automatically become narrower. To further ensure visual clarity and prevent edges from coming too close to nodes, a padding factor f_p can be applied to these ranges (e.g., $f_p = 0.1$ to utilize only 90% of the calculated range), as shown in Figure 4.7.

Connection Anchors. A connection anchor $A = (P, V)$ represents the combination of a connection point P with a vector V that defines the direction of the connected edge. For a circular node n, the anchor A^α at a given angle α can be defined as:

$$\begin{aligned} A_n^\alpha &= (P_n^\alpha, V_n^\alpha) \quad \text{with} \\ P_n^\alpha &= \Big(PC_x^n + r_n \cdot \cos(\alpha),\ PC_y^n + r_n \cdot \sin(\alpha)\Big) \\ V_n^\alpha &= (PC^n, P_n^\alpha) \end{aligned} \tag{4.12}$$

where PC^n represents the center point coordinates and r_n the radius of node n. This formulation ensures that the anchor is orthogonal to the circumference of node n at point P_n^α. An anchor defined this way serves as the connection point for edges.

In the radial node layout, connections between non-adjacent nodes are assigned both start and end anchors within the inner connection ranges of their source and target nodes. To maximize angular resolution, anchors for multiple connections are distributed equally across the available range within each node's inner connection range. By sorting the anchors based on the target node's positions, unnecessary edge-crossings are avoided [Ben+07]. This context-aware approach differs from purely geometric layouts – such as pure circular arcs – by considering the full set of connections for each node when determining anchor positions. By distributing anchors within the previously calculated connection ranges, it is ensured that edges follow natural visual channels between nodes while maintaining even spacing.

Smooth Spline Connection. After establishing connection ranges and anchors, the construction of smooth spline connections is a visual coherent and flexible approach for connecting non-adjacent nodes. Each smooth spline connection is constructed as a cubic Bézier curve defined by four points: the start anchor point, two control points, and the end anchor point. The control points are placed along the direction vectors of the respective anchors to ensure smooth transitions at both ends of the connection.

To support a smooth appearance of the curves, the distance of each control point $d_{control}$ to its corresponding anchor can be chosen proportional to the total distance between the anchors $d_{anchors}$. For the proposed layout, the smoothness factor $f_{smooth} = 0.4$ is chosen, such that $d_{control} = d_{anchors} \cdot f_{smooth}$. This proportional scaling ensures that longer connections have more gradual curves, while shorter connections remain compact.

Given the source anchor $A_s = (P_s, V_s)$ and the target anchor $A_t = (P_t, V_t)$ for a connection, the resulting cubic Bézier curve B is defined as:

$$\begin{aligned} B &= (P_s, P_{c_s}, P_{c_t}, P_t) \quad \text{with} \\ P_{c_s} &= P_s + \text{normalize}(V_s) \cdot d_{control} \\ P_{c_t} &= P_t + \text{normalize}(V_t) \cdot d_{control} \end{aligned}$$

4.3.3 Connection Anchor Refinement

Equally distributed connection anchors across a node's valid inner range provide a basic solution for connection placement, optimizing the angular resolution. This approach however lacks consideration for the actual geometric relationship between connected nodes potentially leading to unnecessarily long paths with additional bends and avoidable edge crossings. This is particularly problematic in dense radial layouts where many connections compete for limited space at node boundaries. A multi-step refinement process is proposed to improve a simple equal distribution of anchors: (1) Treat the distributed anchors not as single points, but instead as angular ranges, (2) combine leading and trailing edges between identical node pairs, (3) optimize anchor range distributions based on desired connection directions.

Valid Anchor Ranges. Instead of restricting each connection to a single fixed anchor point, this is extended to a more flexible approach: For each anchor A the valid angular range $ar_A = [\theta_A^{start}, \theta_A^{end}]$ is defined within the node's available connection range.

Within this range, the anchor point is assigned dynamically based on the *desired anchor point*, which is determined by analyzing the geometric relationship between the connected nodes. In the simplest case, this corresponds to the point where a straight line between the source and destination nodes would originate.

If this desired anchor point falls within the valid range, it is directly used. Otherwise, the anchor point is positioned as close as possible to the desired location, aligning with the nearest boundary of the valid range.

To keep a minimum angular distance between adjacent connection, a padding can be added to the limits of the anchor ranges. This can be done by applying a scaling factor $f_{padding} \in [0, 1]$, such that

$$\begin{aligned} ar_A^{\text{Padded}} &= \left[\theta_A^{start} + \frac{\text{Padding}_A}{2}\ ,\ \theta_A^{end} - \frac{\text{Padding}_A}{2}\right] \\ &\textit{with} \\ Padding_A &= f_{padding} \cdot (\theta_A^{end} - \theta_A^{start}) \end{aligned} \tag{4.13}$$

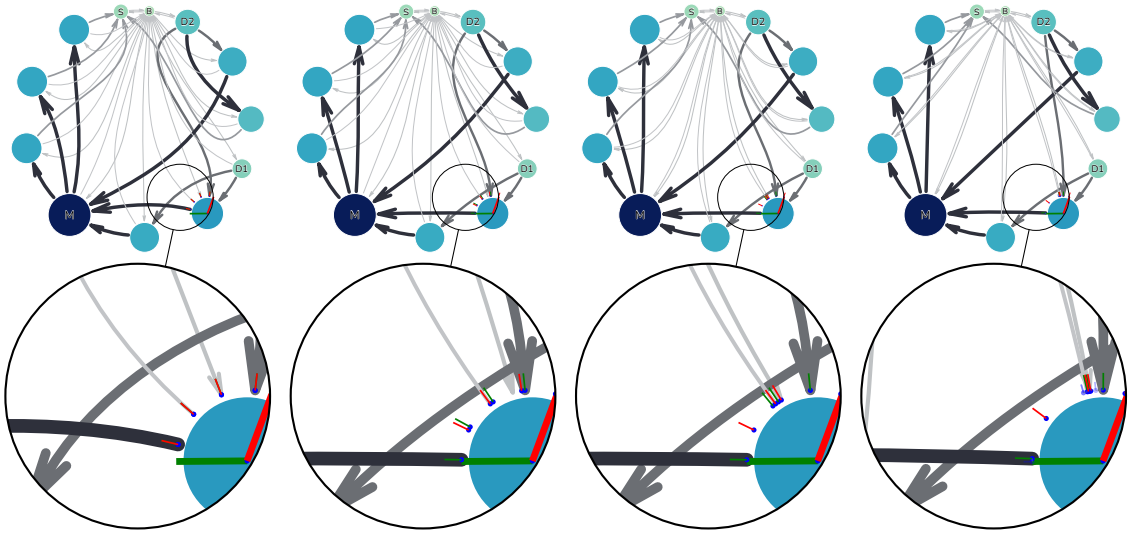
By combining the concept of valid ranges with desired anchors, connections can better adapt to the overall layout geometry, producing paths that follow more intuitive and direct routes.

If there are both a leading and a trailing edge between a node pair, the padding can be further refined. Rather than treating these as separate connections with independent anchors, counter-path pairs are identified and their anchors are positioned closer together. This creates a visual cue that these connections form a logical pair while still maintaining enough separation to avoid visual confusion.

Dynamic Anchor Range Distribution. The shifting of anchors towards desired connection points inside a valid range can be further extended. By softening the strict equal distribution of valid anchor ranges with the definition of a minimum range size, the available space can be better redistributed between all ranges. The minimum size $ar_{minSize}$ can be defined as $ar_{delta} \cdot f_{min}$, with, e.g., a scale factor $f_{min} = 0.5$.

For nodes with desired anchors, it is prioritized to place them near their geometrically optimal position, while the minimum range size is maintained and sufficient space is reserved for the distribution of the remaining connections. If the placement of anchors close to their desired position is not possible due to limited space, the maximum suitable limit is selected for the range.

This optimization brings the connections as close as possible to a straight connection while at the same time maintaining a suitable angular resolution between the connections, depending on the choice of f_{min}.



(a) *Equally Distributed Anchors* (b) *Equally Distributed Anchor Ranges* (c) *Combined Leading and Trailing Edges* (d) *Dynamic Anchor Range*

Figure 4.8: *Different refinement steps for the advanced connection layout using circular arcs for adjacent nodes and cubic Bézier curves for other nodes. The magnified area shows details for a single node with its valid inner range between the larger green and the larger red line segment.* ***(a)*** *Equally distributed anchor points across the range. Each small red line stands for a placed anchor.* ***(b)*** *Equally distributed anchor ranges for all connection, each extends between a small red line and the clockwise next small green line. A padding factor of $f_{padding} = 0.1$ is applied. Within its assigned range the final connection anchor can be placed freely based on the desired anchor point.* ***(c)*** *Outgoing and incoming edges to and from the same node are combined by decreasing their valid ranges, in this case scaled by a factor of* 0.1. ***(d)*** *Dynamically sized anchor ranges. The ranges are shifted towards desired anchor points while preserving a minimum size.*

4.3.4 Inter-Group Connections

The grouped placement of nodes – given by the data or calculated by community detection algorithms – emphasizes local neighborhoods by placing closely related nodes close together. However, this grouping brings with it the challenge of how to layout the connections between nodes of different groups.

The most straightforward approach is to draw all inter-group connections as straight lines. However, this can result in a high number of edge crossings and even node-edge overlaps, especially in multi-circular layouts [BB08]. Instead, appropriately curved edges can help reduce such issues [Ben+07].

But using curved edges introduces new problems. When poorly designed, strongly curved edges can increase the number of edge crossings [Xu+12]. Additionally, excessive edge bending harms traceability and violates basic readability metrics for graph layouts [Pur02; Ben+07].

To address these challenges, this thesis proposes two alternative approaches for laying out inter-group connections: (A) a hierarchical connection strategy based on aggregated hyperedges, and (B) a direct spline layout analog to the in-circle connections, based on the valid outer connection ranges of nodes.

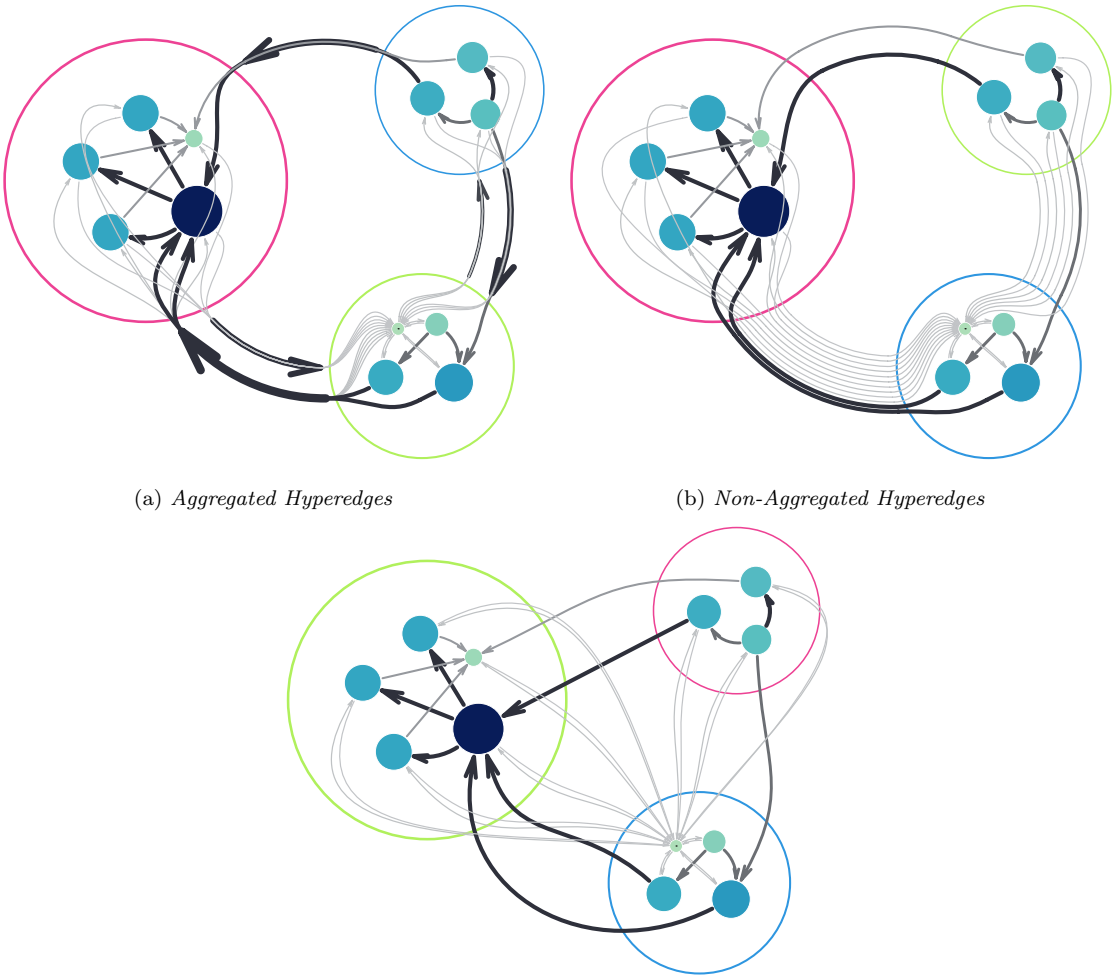
Hierarchical Connections. Inspired by the multi-circular layout from Baur and Brandes [BB08], inter-group connections are split into two distinct path segments: (1) path segments between both hypernodes containing the actual target and source nodes, referred to as *hyperedges*, and (2) path segments from the hyperedge to the actual source and target nodes.

In the first step, one or more hyperedges are routed between the hypernodes in the same way as intra-circle connections on a higher level – thus, circular arcs for directly adjacent nodes and smooth splines for the other nodes. In the second step, each connected node is linked to its respective hyperedge. Depending on its position relative to the hyperedge, different visual styles can be used. For example, nodes that directly face the hyperedge can also use smooth spline curves. For nodes positioned behind other nodes, circular segments – as used in the original multi-circular view – can provide a more appropriate route without crossing nodes.

Hyperedges themselves can be used in two ways: (1) as aggregation channels, so that each hypernode pair is connected by at most a single outgoing and incoming edge, which bundle all contained connections like in Figure 4.9(a). (2) Alternatively, every inter-group connection can be assigned its own hyperedge. This approach could reduce ambiguity while increasing visual density, as visible in Figure 4.9(b).

Direct Spline Connections. In contrast to the hierarchical approach, direct spline connections lay out inter-group edges analogously to the intra-group splines, using the outer connection range cr_{out} of each node. Similarly, the connections can be either distributed equally across the outer range or using dynamic anchor optimization to preserve angular resolution.

Analogous to inner splines, anchors are sorted according to the target group positions to reduce edge crossings and maintain visual coherence. Compared to the hyperedge approach, direct splines can provide shorter and more direct connections with fewer bends, as visible in Figure 4.9(c). This reduces layout complexity, although it may lead to higher visual density in cases of many inter-group connections.

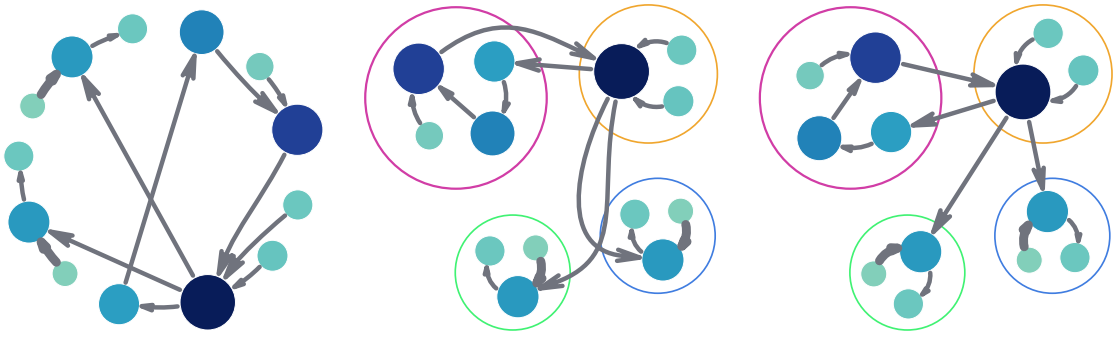


(a) *Aggregated Hyperedges*

(b) *Non-Aggregated Hyperedges*

(c) *Direct Spline Connections*

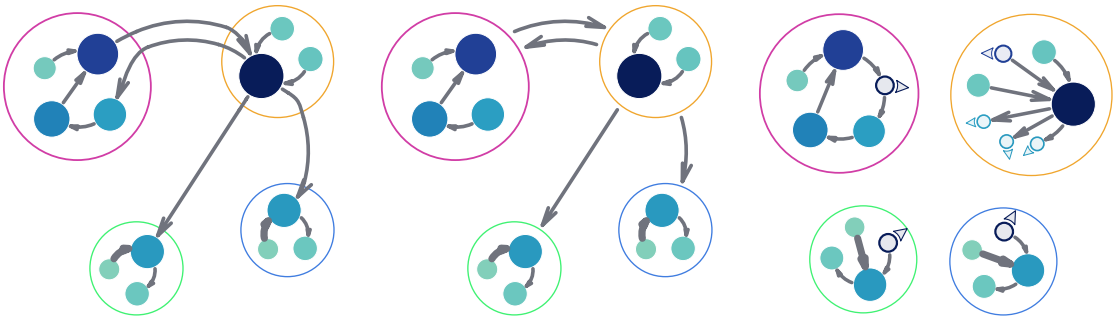
Figure 4.9: *Different approaches for inter-group connections of nodes. The edges in (a) and (b) are routed in a hierarchical way, thus they follow the hyperedges between the hypernodes. The hyperedges are shown in (a) and are aggregated, so that between two hypernodes there is only one leading and one trailing hyperedge. In (b), the hyperedges are not aggregated, so that each connection has its own hyperedge. In (c), the connections are routed directly between the valid outer ranges of the nodes using smooth splines.*



(a) *Default Layout without Node Grouping*

(b) *Direct Connections*

(c) *Direct Connections (Rotated Hypernodes)*



(d) *Hierarchical Connections*

(e) *Hierarchical Connections with only Hyperedges being visible.*

(f) *Added Virtual Nodes*

Figure 4.10: *Different visualization configurations for the same communication network with 13 nodes. The ungrouped nodes in* ***(a)*** *are grouped in* ***(b)*** *to* ***(f)*** *into distinct communities after applying the Louvain algorithm. The different groups containing the assigned child nodes are called hypernodes and are indicated by different colored circles. The hypernodes are placed in the same way as the child nodes using Algorithm 3.* ***(b)*** *and* ***(c)*** *use the direct connections approach inter-group connections, where (c) has the hypernodes rotated based on their connections. In* ***(d)****, the connections are routed in a hierarchical way by first layouting hyperedges (edges between hypernodes) and afterwards the connections between these hyperedges and the actual nodes. For illustration,* ***(e)*** *shows only the hyperedges between the hypernodes used for the layout in (d). In* ***(f)****, the groups are extended by virtual nodes. This way, inter-group connections are avoided.*

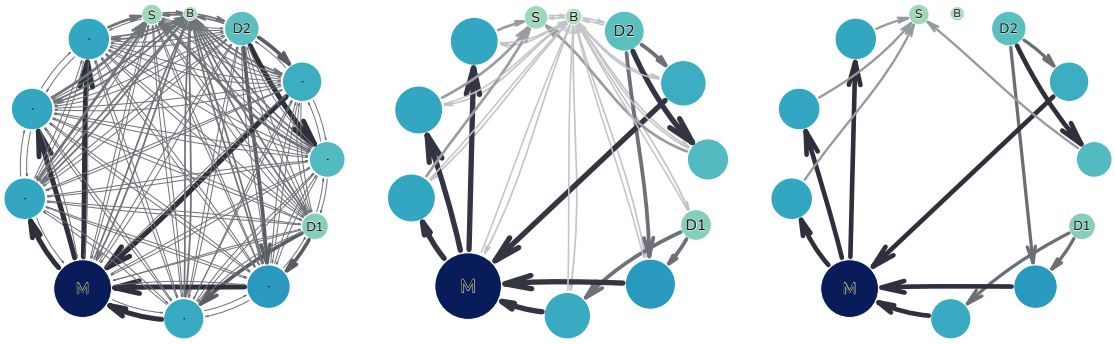
4.4 Visual Scalability

A fundamental problem with node-link diagrams is that they generally scale poorly. As the complexity of graphs increases, the edges will become longer and more difficult to trace, especially for connections between different groups. Even the best layout algorithms will result in hairball-like visualizations at a certain graph size [Yog+21]. This phenomenon particularly arises for dense graph structures relevant in communication networks with a lot of broadcast characteristics. This complexity therefore requires forms of simplification in order to remain comprehensible for the user.

Filtering of Visible Connections. The overview of nodes should also act as such and focus on the tasks of identifying important connections that define the dataflow between nodes. Therefore, *less important* connections can be filtered out at this level of overview [Ger+09].

Based on the calculated edge significance (Equation (4.6)), a minimum score can be assigned for connections to be rendered. A value for this threshold can be assigned by users with system knowledge. The chosen threshold can be calculated based on the count of similar topic connections: If, for example, topics with more than 8 usages should be filtered out, the score threshold τ_s can be set to $\tau_s = \sqrt{1/8} \approx 0.36$.

Filtering of edges below this score does not influence the node layout, as their placement is based on the Weighted Flow Sorting (Algorithm 2), which already handles connections with less weight. The effect of connection filtering is visible in Figure 4.11. The visibility of filtered edges can be restored through user interaction or the focus+context visualization of a single node.



(a) *Unfiltered Connections*

(b) *Connection Filtering with* $\tau_s = 0.15$

(c) *Connection Filtering with* $\tau_s = 0.36$

Figure 4.11: *Illustration of the effect of connection filtering on the same graph.* ***(a)*** *shows all connections.* ***(b)*** *has a threshold value of* $\tau_s = 0.15$. ***(c)*** *has a threshold value of* $\tau_s = 0.36$, *filtering out any broadcast connections. The choice of a suitable* τ_s *will preferably filter out more frequently used broadcast connections.*

Virtual Nodes. Existing graph visualization tools such as `rqt_graph` face several challenges when used on large communication networks. Connections tend to result in long edges, especially in layered or radial layouts, as shown in Figure 1.1(a). These edges are difficult to trace visually, particularly when they span multiple levels or connect distant node clusters. To follow such edges, users frequently need to zoom out significantly, which leads to a loss of detail and makes it harder to understand local structures and dataflows. The same is true for multi-circular views. While interactive features such as highlighting can offer some help, they often fail to fully compensate when dealing with long or intersecting edges.

To address these issues, this thesis proposes the concept of *Virtual Nodes*: For every inter-group connection, the involved nodes are mutually duplicated into the respective other group, connected correctly to all nodes in that opposite group. The virtual nodes are integrated into the layout algorithms such as the Weighted Flow Sorting, allowing them to reflect the dataflow of the original network.

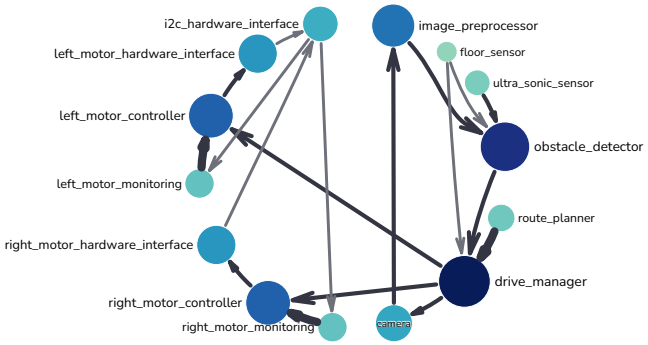
Introducing virtual nodes offers several advantages. It completely avoids long and visually disruptive edges between distant groups. All nodes involved in external communications are directly visible within the local group. As a result, users do not need to follow long outgoing edges to understand group-level dataflows, enabling more efficient subsystem-level analysis. This makes the method especially useful in large and complex communication networks with multiple interconnected subsystems.

The described introduction of virtual nodes is visible in Figure 4.10(f). It clearly shows the absence of inter-group connections, as they are implicitly covered by the virtual nodes. It furthermore visualizes the possibility to better comprehend the dataflow between all nodes of a group.

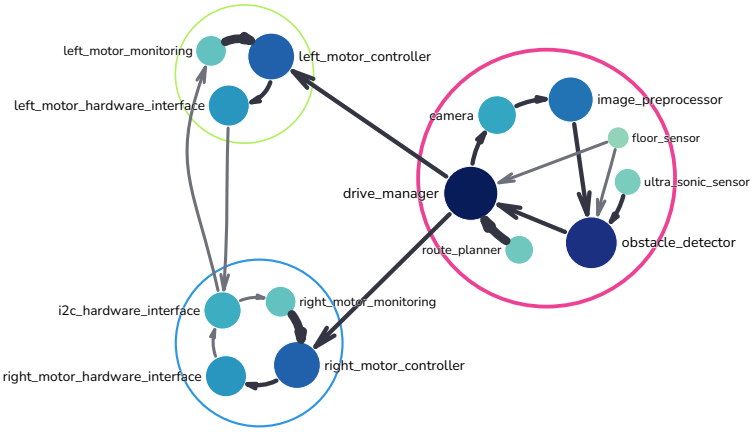
To distinguish virtual nodes from normal nodes, they are visually marked using adapted style properties and an arrow indicating the direction of the original node. On user interaction, an additional line is drawn between the virtual node and its origin to further improve traceability.

The approach also introduces trade-offs. In highly interconnected systems, many nodes could be duplicated across groups, which may result in the system being perceived as much larger. By removing long inter-group edges in favor of local duplicates, the global structure of the network becomes less apparent and requires interaction to be fully understood. Virtual nodes must be clearly distinguished from original nodes to avoid misinterpretation by forgetting actual existing connections.

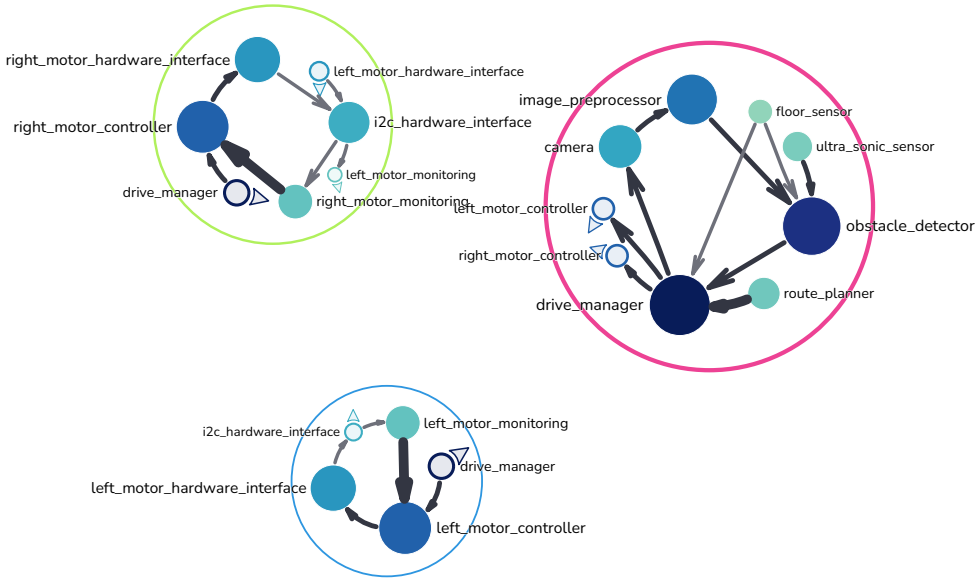
Figure 4.12 shows the effect of virtual nodes on a system with 14 nodes. In this system, seven virtual nodes are added to the communities to remove inter-group connections. These virtual nodes are implicitly connected to the original nodes in their respective groups.



(a) *Default Layout*



(b) *Layout with Communities*



(c) *Layout with Virtual Nodes*

Figure 4.12: *A system with 14 nodes.* ***(a)*** *shows the default layout without any grouping or virtual nodes.* ***(b)*** *shows the same system with community detection applied, resulting in a layout with hypernodes.* ***(c)*** *shows the same system with seven virtual nodes added, which are implicitely connected to the original nodes.*

Chapter 5

Quantitative Evaluation

5.1 Graph Quality Metrics

To evaluate graph layouts, several graph quality metrics are calculated, that measure different aspects of the layout's structure and readability.

Aspect Ratio. The *Aspect Ratio Metric* is the ratio between width and height of the layout's bounding box. A layout that is too wide or too tall is harder to display and understand [Moo+24]. The smaller of both possible ratios between the width w and the height h of the layout is used to get an AspectRatio between $[0, 1]$, with 1 being a square layout:

$$\text{AspectRatio} = \min\left(\frac{w}{h}, \frac{h}{w}\right)$$

Edge Crossings. The *Edge Crossing Metric* counts the number of intersections between edges in the graph layout. Fewer edge crossings improve the readability of the graph, as excessive crossings can obscure relationships between nodes and make the layout harder to interpret [Moo+24].

To obtain a comparable edge crossing metric $\aleph_c$ in the interval of $[0, 1]$, the number of edge crossings c is scaled against the maximum possible number of crossings c_{max} [Pur02]:

$$\begin{aligned} \aleph_c &= \frac{c}{c_{max}} \\ &\text{with} \\ c_{max} &= c_{all} - c_{impossible} \\ &= \frac{|E| \cdot (|E| - 1)}{2} - \frac{1}{2} \sum_{i=0}^{|N|} [degree(u_i) \cdot (degree(u_i) - 1)] \end{aligned} \tag{5.1}$$

Path Efficiency. The *Path Efficiency Metric* measures the ratio between the actual path length and the direct Euclidean distance between nodes. This can be seen as generalization of the metric for edge bends [Pur02]. The path efficiency as a value

between $[0, 1]$ is calculated as

$$\text{PathEfficiency} = \frac{\sum_{e \in E_N} \text{DirectDistance(e)}}{\sum_{e \in E_N} \text{PathLength(e)}}, \tag{5.2}$$

with the assumption, that a path will never be shorter than the direct distance between two nodes.

This equation can be normalized so that the efficiency ratio of each single path segment is observed:

$$\text{PathEfficiency}_{norm} = \frac{1}{|E_N|} \sum_{e \in E} \left(\frac{\text{DirectDistance}(e)}{\text{PathLength}(e)} \right) \tag{5.3}$$

The normalized version is also known as *Distortion* [Wal+22].

Higher values of PathEfficiency indicate that paths are closer to straight lines, with $\text{PathEfficiency}_{norm} = 1$ indicating only straight lines between nodes.

Path Continuity. The *Path Continuity Metric* evaluates the smoothness of paths by analyzing angular differences between consecutive segments of shortest paths in the graph [DS09].

The root mean square of angular differences is normalized to [0, 1] by dividing by π:

$$\text{PathContinuity} = \frac{1}{\pi} \sqrt{\frac{1}{|E_N| - 1} \sum_{i=1}^{E_N} [\Delta(\theta_{i-1}, \theta_i)]^2}, \tag{5.4}$$

with θ_i being the polar angle between source node and target node of the i-th connection of E_N, and $\Delta(\theta_{i-1}, \theta_i)$ the angular difference between two angles. Lower values indicate smoother path continuity, which improves the perception of visual flow and reduces abrupt changes in direction.

The *Weighted Path Continuity Metric* extends the Path Continuity Metric by weighting angular differences based on edge weights. This emphasizes the importance of high-weight connections in the layout.

$$\text{WeightedPathContinuity} = \frac{1}{\pi} \sqrt{\frac{1}{W} \sum_{i=1}^{E_N} [w_i \cdot \Delta(\theta_{i-1}, \theta_i)]^2}, \tag{5.5}$$

with $W = \sum_{i=2}^{n} w_i$ being the total weight of all edges included in the sum.

Path Angular Prediction. The *Path Angular Prediction Metric* measures how well paths follow a consistent angular trend. For each triplet of consecutive segments, the angle of the third segment is predicted based on the trend of the first two. Deviations from the predicted angle are squared, weighted, and aggregated. Lower values indicate more consistent angular trends, which contribute to a more intuitive layout.

$$\text{PathAngularPrediction} = \frac{1}{\pi} \sqrt{\frac{1}{W} \sum_{i=2}^{E_N} w_i \left(\theta_i - \theta_{\text{pred}}(\theta_{i-2}, \theta_{i-1})\right)^2}, \tag{5.6}$$

where $\theta_{\text{pred}}(\theta_{i-2}, \theta_{i-1})$ is the predicted angle of the next edge based on the last two angles, and $W = \sum_{i=2}^{n} w_i$ is the total weight of all edges included in the sum.

Stress. The *Stress Metric* is a typical metric in the area of multidimensional scaling, measuring how well distances in a higher-dimensional space are preserved in a lower-dimensional embedding, with a value of 0 indicating a perfect match, a value of 1 a inappropriate match [Kru64]. This is transferable to visualizations by comparing the graph-theoretical shortest path distances in the communication network with the Euclidean distances between nodes in the layout.

The original stress equation requires the original and the embedded distances to be in a comparable scale, therefore it is not scale-invariant. This can lead to issues when comparing abstract graph distances – in common networks, depending on the size, approximately in the interval of 1 to 100 – with distances of visual embeddings – usually pixel based coordinates in a typical value range up to multiple 1000.

While there are simple approaches to achieve scale invariance – like maximum value normalization – the stress S is defined as follows [Sam69; SMK24]:

$$S = \frac{\sum_{i,j}^{N} \left[\delta_{ij} - \alpha \cdot d_{ij}\right]^2}{\sum_{i,j}^{N} \delta_{ij}^2} \quad \text{with} \quad \alpha = \frac{\sum_{i,j}^{N} \left[\delta_{ij} \cdot d_{ij}\right]}{\sum_{i,j}^{N} d_{ij}^2}, \tag{5.7}$$

where

- d_{ij} is the euclidean distance between nodes i and j in the 2D layout,
- δ_{ij} is the observed dissimilarity, thus the graph-theoretical distance (shortest path) between nodes i and j in the graph structure, using Equation (4.7) for the distance score of nodes, and
- α is the calculated scaling factor to achieve scale invariance.

For disconnected nodes, the maximum shortest path distance found in the graph is used. A smaller value of S indicates that the embedded distances d_{ij} more accurately preserve the original dissimilarities δ_{ij}, providing a more accurate visualization of the relationships between nodes.

Node-Edge Overlaps. The *Node-Edge Overlap Metric* – also known as *edge tunnel* or *edge bridge* metric – counts the number of overlaps between edges and nodes in the graph layout. Fewer overlaps improve the traceability of edges in the layout [CP96]. To normalize the metric node-edge overlap metric $\aleph_{NE}$, the amount of node-edge overlaps o is scaled against an upper bound [DS09]:

$$\aleph_{NE} = 1 - \frac{o}{o_{max}} \quad \text{with} \quad o_{max} = |E| \cdot (|N| - 2) \tag{5.8}$$

Node-Node Overlaps. The *Node-Node Overlap Metric* – also known as *node occlusion* – counts the number of overlaps between nodes and nodes in the graph layout. Node occlusions contradict the recognized guidelines for the readability of graph visualizations [Sug02]. Fewer overlaps improve distinctiveness of nodes, making it easier for the user to capture the count of individual elements in the graph [DS09]. To normalize the metric node-node overlap metric $\aleph_{NN}$, the count of node overlaps o is scaled against an upper bound:

$$\aleph_{NN} = 1 - \frac{o}{o_{max}} \quad \text{with} \quad o_{max} = \frac{|N| \cdot (|N| - 1)}{2} \tag{5.9}$$

5.2 Evaluated Datasets

For evaluation, a combination of real-world and synthetic datasets is used. The real-world datasets are gathered from running communication network systems in the domain of robotics, based on ROS 2 [Mac+22]. The synthetic datasets are generated using a custom generator that simulates typical characteristics of communication networks.

ROS 2 Data with RosMetaSys. The data acquisition process for analyzing live ROS 2 systems was carried out using the tool developed for this purpose *ROS 2 Meta System Exporter* (rosmetasys) [Sch24a]. This command-line tool extracts meta-information from a running ROS 2 system into a JSON file. Besides the nodes of the system and their names, this meta-information contains the publishers and subscribers per node, associated services and clients, and message types for each topic. Publisher/subscriber and service/client connections are the two main communication channels in ROS 2, on which nodes are connected via named topics. The message types are an extension to the presented definition of communication networks, as ROS 2 constraints the format of messages exchanged on a topic to a specific type. To ensure privacy, the tool provides an anonymization option that irreversibly replaces node and topic identifiers with sequential IDs. The extracted datasets are shared open-source on GitHub with consent of the authors [Sch24b].

For the evaluation 8 real-world datasets are used:

- 4 datasets (between 6 and 26 nodes) from *ROS-E*, a social robotics project using ROS2 [Sch19; Sch22].
- A dataset from an open-source self-driving car (7 nodes) [Mor24].
- A ROS tutorial system for showing sensor values on three displays, containing broadcast mechanisms (12 nodes).
- An example system for a motor controller and path-finding in a self-driving car (13 nodes) [Aut24].
- A complex parcel separation system shared by *cellumation GmbH* (61 nodes) [cel25].

Synthetic Data Generation. Synthetic datasets are generated to extend the few gathered real-world datasets to better investigate scalability. There are several well-known feature-driven methods for creating graphs, which Lim *et al.* explains very clearly [Lim+16]. However, existing generators were found to insufficiently cover the connecting behavior of the investigated communication networks, especially regarding the topic-based connections and characteristics like broadcast connections. The **CommGraphGenerator** is designed in this thesis to synthesize communication graphs with a variety of typical structural features:

Pipelines: Generates linear sequences of nodes connected by a path. Each node n_i is connected to its successor n_{i+1}, forming a path $n_1 \to n_2 \to \cdots \to n_k$. The pipeline lengths are determined based on the minimum pipeline length PL_{min}, the mean pipeline length PL_{μ}, and the standard deviation PL_{σ}. This structural feature is the base for most other characteristics.

Forward Edges: Adds additional edges from a node to later nodes in the same pipeline, reinforcing directional flows. Specifically, for the node n_i and a probability P_{fE}, an edge (n_i, n_j) is added with a randomly selected $j > i$.

Backward Edges: Creates connections from a node back to earlier nodes in its pipeline, introducing circularity. Specifically, for the node n_i and a probability P_{bE}, an edge (n_i, n_j) is added with a randomly selected $j < i$.

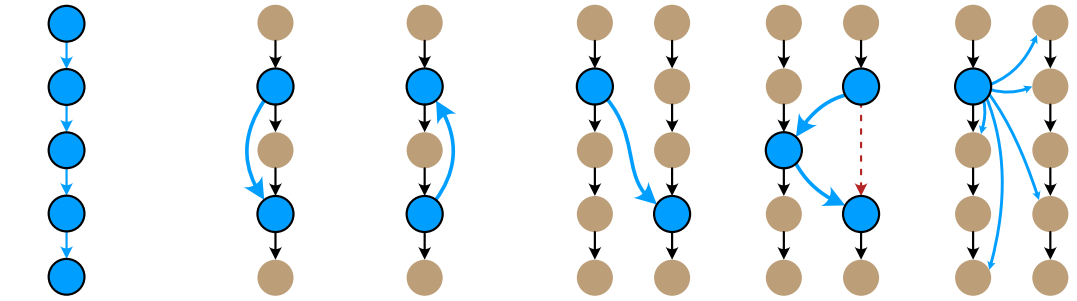
Cross Connections: Introduces random links between nodes from different pipelines to increase inter-pipeline interaction and to form communities. With a probability P_{CC}, a node n_i from one pipeline is connected to a randomly selected node n_j from a different pipeline.

Cross Integrations: Variation of cross connections. With a probability P_{CI}, a direct edge (n_i, n_{i+1}) within a pipeline is replaced by two edges: (n_i, n_k) and (n_k, n_{i+1}), where n_k is a randomly selected node from a different pipeline.

Broadcast Nodes: Establishes one-to-many connections on the same topic to simulate broadcast behavior across the network. With a probability P_B, a node n_i is connected to multiple successor nodes within or across pipelines, creating a broadcast pattern. The amount of nodes targeted t is randomly selected based on the fraction of broadcast target nodes F_B with $t < F_B$.

Node Rewiring: Reassigns connections by redirecting the predecessors and successors of a randomly chosen node to another node, as in the Watts-Strogatz algorithm [WS98]. With a probability P_{NR}, a node n_i has its edges rewired to a randomly selected node n_j.

Each of these features has a certain probability to create a connection on a new topic or to use an existing topic from the node. For the evaluation, 10 datasets with 25, 50, 75, 100 and 150 nodes, each in a sparse and a dense configuration, were synthesized. The exact generation settings are listed in Table 5.1.



(a) *Pipeline* (b) *Forward Edge* (c) *Backward Edge* (d) *Cross Connection* (e) *Cross Integration* (f) *Broadcast*

Figure 5.1: *The different synthesized features by the proposed* ***CommGraphGenerator****. Each feature is applied to a node of a pipeline with the configured probability. In this example, the blue nodes are the selected nodes for the features. The blue edges are those added by a feature. The red edge is removed during the process of a cross integration feature.*

Table 5.1: *The synthesized datasets used in the evaluation with node count n and edge count* $m_{>0.2}$*, only considering edges with a score above* 0.2*. The generator parameters to create the sets are given: Minimum Pipeline Length (*PL_{min}*), Mean Pipeline Length (*PL_{μ}*), Pipeline Length Deviation (*PL_{σ}*), Probability for Forward Edges (*P_{fE}*), Prob. for Backward Edges (*P_{bE}*), Prob. for Cross Connections (*P_{CC}*), Prob. for Cross Integrations (*P_{CI}*), Prob. for Node Rewiring (*P_{NR}*), Prob. for Broadcast (*P_B*), Fraction of Broadcast Target Nodes (*F_B*), Prob. for Topic Reuse (*P_{TR}*). Every generation is instantiated with seed 55.*

n	$m_{>0.2}$	PL_{min}	PL_{μ}	PL_{σ}	P_{fE}	P_{bE}	P_{CC}	P_{CI}	P_{NR}	P_B	F_B	P_{TR}
25	37	3	5	5	0.05	0.1	0.1	0.05	0.05	0.0	/	0.1
50	55	3	5	5	0.05	0.1	0.1	0.05	0.05	0.0	/	0.1
75	80	3	5	5	0.05	0.1	0.1	0.05	0.05	0.0	/	0.1
100	130	3	5	5	0.05	0.1	0.1	0.05	0.05	0.0	/	0.1
150	171	3	5	5	0.05	0.1	0.1	0.05	0.05	0.0	/	0.1
25	66	4	6	7	0.15	0.15	0.15	0.15	0.1	0.1	0.5	0.3
50	98	4	6	7	0.15	0.15	0.15	0.15	0.1	0.1	0.5	0.3
75	165	4	6	7	0.15	0.15	0.15	0.15	0.1	0.1	0.5	0.3
100	209	4	6	7	0.15	0.15	0.15	0.15	0.1	0.1	0.5	0.3
150	280	4	6	7	0.15	0.15	0.15	0.15	0.1	0.1	0.5	0.3

5.3 Evaluated Layout Techniques

For the 18 different datasets, 6 different techniques with a total of 864 computed layouts are evaluated. The following layout techniques have been evaluated for each dataset:

- **12 VisCom** layouts: 3 layouts without community detection (**VisCom Default**), 6 with community detection (**VisCom Comm**), and 3 layouts with community detection and virtual nodes (**VisCom Virtual**),
- **7 Radial** layouts with different configurations of forward and backward circular edges,
- **9 Force Directed** graphs with different settings for the forces regarding link strength, node repulsion, center force and collision [FR91],
- **3 Graphviz** layouts with DOT and CIRCO [Gan11; GKN15],
- **3 Arc** layouts [GBD09], and
- **14 Space Filling** layouts with different space filling curves and orders [MK08].

A selection of the layout configurations is shown in Figure 5.2. To ensure comparability, the linearized techniques all used the same Weighted Flow Sorting from Algorithm 2. Furthermore, all layouts have the same connection filtering with a threshold score $\tau_s = 0.2$ applied.

5.4 Results

Figure 5.3 shows the computed metrics across the layout techniques using boxplots. The proposed *VisCom* approach performs consistently well in most categories:

Aspect Ratio: Due to its radial layout, VisCom Default has a constant aspect ratio of > 0.95. Together with Space Filling and Radial, this makes it one of the layouts with the best aspect ratio value. VisCom Comm and VisCom Virtual have lower aspect ratios, comparable to Force Directed, because the grouped layout due to the introduced communities leads to a less balanced layout. The least balanced layouts are the Arc and GraphViz layouts.

Path Efficiency: isCom Default and VisCom Virtual have path efficiency values of ≈ 1.0. Even though they use smooth spline curves, they are still almost straight lines, due to the anchor refinement process. VisCom Comm has a lower value, because the grouped layout results in longer connections between different groups.

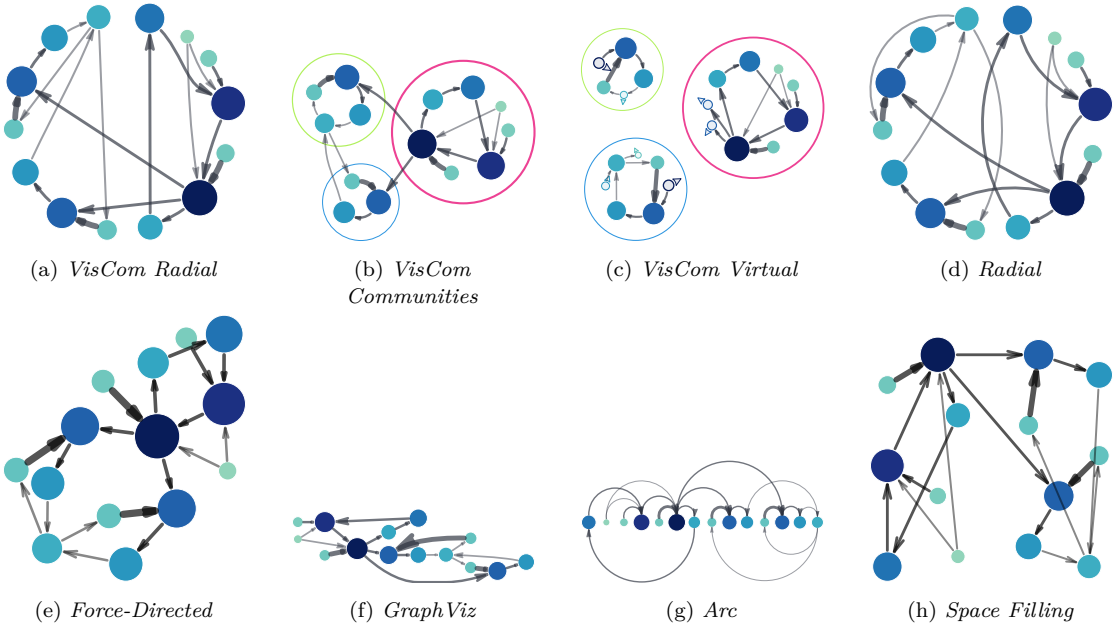
Stress: The stress values for VisCom Default and VisCom Comm are comparable to Radial, GraphViz and Space Filling. Force Directed performs slightly better, because its layouts minimize the simulated forces in the layout, making it more

likely to find layouts with low stress values. However, VisCom Virtual has a significantly lower stress value than all other layouts, because the duplication of specific nodes results in consistently adjacent placements. Arc performs worst in this category, because its strictly linear layout results in potentially very distant nodes.

Edge Crossings: VisCom Default, together with Radial and Arc, performs worst in edge crossings. This can be explained by its strict inner connection layout, which leads to avoidable crossings. The grouped layout of the communities in VisCom Comm slightly improves the edge crossings, because the inter-group connections are less likely to cross the intra-group connections. VisCom Virtual has the best value, comparable to Force Directed and GraphViz.

Node-Edge Overlaps: VisCom Default and VisCom Virtual have no node-edge overlaps, by design due to their strict inner connection layout. The same applies to Radial, GraphViz, and Arc. VisCom Comm may have overlaps if no hierarchical layout is used, but even then they are very small. Since Force Directed and Space Filling use straight line connections in their implemented versions for this evaluation, they have the potential to create node-edge overlaps and thus the worst values.

Node-Node Overlaps: All VisCom layouts have no node-node overlaps by design. FDG and Space Filling have some overlaps, mainly in larger and denser graphs. However, with better parameter tuning, especially for large graphs, such as stronger repulsion forces, the overlaps could be reduced.



(a) *VisCom Radial* (b) *VisCom Communities* (c) *VisCom Virtual* (d) *Radial*

(e) *Force-Directed* (f) *GraphViz* (g) *Arc* (h) *Space Filling*

Figure 5.2: *A graph with 14 nodes and 18 edges, visualized with exemplary configurations of the layout algorithms currently implemented in the frontend.*

(Weighted) Path Continuity: Both metrics show no significant differences between the layouts. The radial placement of nodes in combination with Weighted Flow Sorting aimed to achieve good values in terms of (weighted) path continuity. However, the radial node arrangement inherently exhibits less continuity in its paths. Since the nodes of a path in these layouts cannot be on a straight line by design, which is penalized in the metric, the values are not significantly better or worse than those of other layouts.

Path Prediction: VisCom Default and VisCom Virtual, together with Radial, have the best path prediction values. Their radial node arrangement allows continuous tracking of successive path segments based on the previous segments. Due to the grouped layout of the communities, VisCom Comm has a lower value, because the connections between groups are less likely to be continuous. Force Directed, GraphViz, Space Filling, and Arc have slightly lower scores, as their layout is not optimized for data flow continuity.

While all approaches show good values in some metrics, the proposed *VisCom* approach consistently performs well or even best in most categories. This is especially true for the VisCom Virtual variant, which achieves the best values in stress and edge crossings. However, VisCom Virtual comes with the drawback of duplicating nodes and therefore increasing the number of graphical elements in the layout.

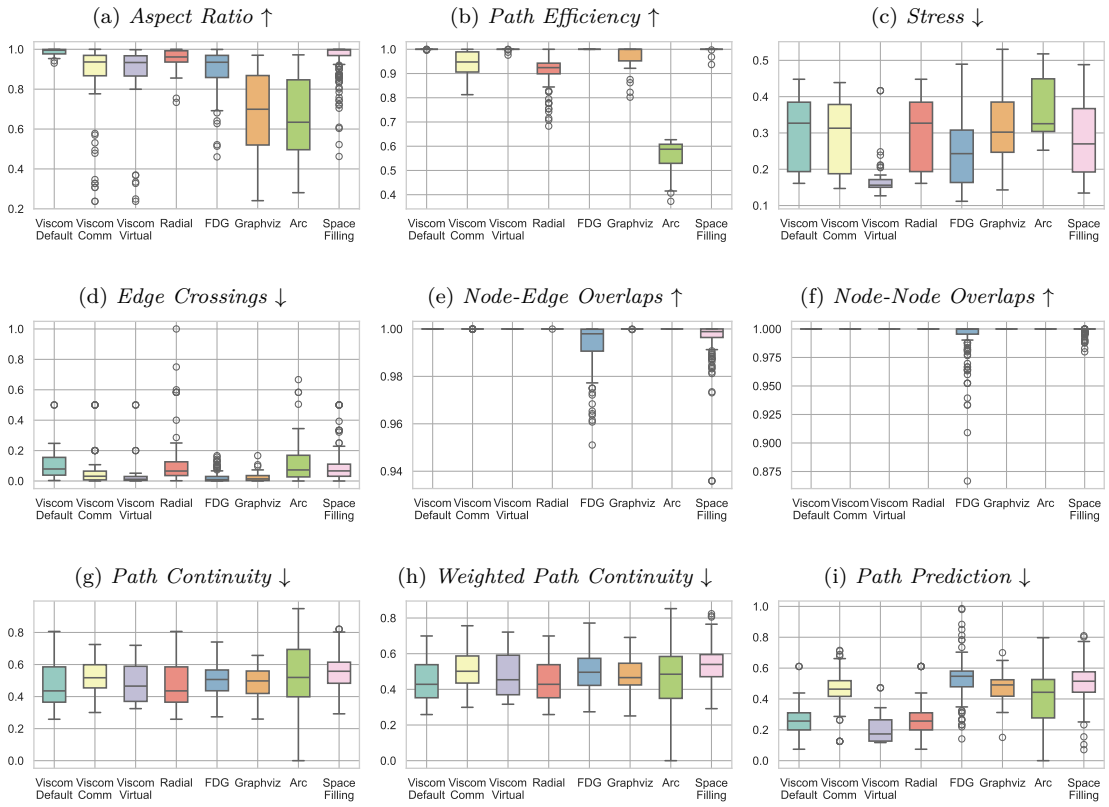


Figure 5.3: *Evaluation results for 9 different metrics. The different layout algorithms have been tested on 18 datasets (8 real-world datasets and 10 synthetic datasets) with system sizes between 6 and 150 nodes. The first three techniques in every plot are the proposed VisCom techniques in different configurations: (1) normal radial layout, (2) with community detection, and (3) with virtual nodes. The other layout techniques are in order: (4) Radial (radial layout with circular arc connections, see Figure 4.6), (5) Force Directed Graphs [FR91], (6) GraphViz with DOT and CIRCO [Gan11; GKN15], (7) arc layouts inspired by Time Arc Trees [GBD09], and (8) Space Filling layouts [MK08].*

Chapter 6

Evaluation Framework

In order to test the presented approach and compare it to other layout algorithms, a graph layout evaluation framework has been implemented as part of this thesis. Using this framework, a user – researcher or visualization designer – can upload own graphs, select pre-loaded ones, or generate synthetic graphs to explore different layouts through their generated layouts. The system allows for the configuration of multiple layouts, which can then be explored by the user through side-by-side comparisons and enlarged views. Further, the framework allows for the computation of layout metrics. Most layout images as well as all metric results in this thesis have been exported from this tool.

The evaluation framework consists of two parts: (1) the frontend, which provides the user interface for viewing and configuring different layout algorithms, and (2) the backend, which provides computationally intensive functions such as data generation, graph analysis, and layout evaluation through a REST API.

The project is available as open-source on GitHub /vschroeter/VisCom-Thesis with the frontend of the framework available at vschroeter.github.io/VisCom-Thesis.

6.1 Frontend

The frontend is a web-based application that runs in the browser and allows users to interact with the graph layout evaluation framework. It provides an interface for configuring, visualizing, and evaluating different layout algorithms. Its main purpose is to support exploration and comparison of layouts for communication networks. The layout technique proposed in this thesis is implemented in the frontend for proof of concept.

6.1.1 Features

The main features of the frontend include the following:

- Loading of saved graph data, generating and downloading random graphs, and uploading graphs from files.
- Side-by-side comparison of multiple, configurable layout algorithms.
- Presentation and comparison of graph quality metrics.
- Export layout images and metric results for further analysis.

For full functionality, the frontend needs to be connected to the backend of the framework. This includes the generation of graphs, the calculation of metrics, and the analysis of communities and node scores. However, the frontend can also be used in a limited mode without a backend connection, where the user can only load and visualize graphs from files. To assure this limited functionality, all layout algorithms are currently implemented in the frontend and executed client-side. This limited mode can be directly tested at `vschroeter.github.io/VisCom-Thesis`.

6.1.2 Technologies

The web-app uses the *Vue.js* javascript framework [You25] as base. Vue.js is a progressive framework, supporting the development of single-page applications by providing a reactive data model and an architecture based on reusable components. For styling, the CSS-framework *Quasar* [Sto25] is utilized. It builds up on Vue.js and offers a large set of pre-defined components like buttons, sliders, and input fields, which simplifies the development of a responsive user interface. As basis for the graph visualization, the library *D3.js* is used [Bos25a]. D3.js is a library for data-driven SVG-visualizations and provides a large set of functions to create and manipulate SVG elements.

6.1.3 User Interface

The user interface has a three column layout, as shown in Figure 6.1. In the center area, the different configured layouts are shown side-by-side as tiles. The right panel is mainly used to load and generate graphs. In the left panel, the user can configure the settings of a selected layout algorithm.

As soon as a graph is loaded or generated, each configured layout algorithm calculates its layout and displays the result in the corresponding tile. The tiles are grouped row-wise by the type of layout algorithm. Inside each algorithm group, the tiles are arranged side-by-side. The user can interact with the main canvas of each tile to zoom or pan the layout, and select nodes or edges. When a node or an edge is selected, all adjacent elements are highlighted and all other elements are deemphasized. Additionally, users can duplicate, download, or delete layout tiles as needed, illustrated in Figure 6.2.

For a selected tile, the user can configure the parameters of the layout algorithm in the right panel. These parameters – such as forces, distances, and algorithm variants – depend on the selected layout algorithm. As soon as the user changes a parameter, the layout is recalculated live and displayed in the tile.

On the bottom of each tile, the calculated metrics are shown – each color-coded to indicate their relative performance compared to other layouts. To keep the user interface responsive, the metrics are calculated in the background and displayed once they are finished. However, because the metrics calculation for a lot of layouts can be time-consuming, the calculation can be turned on and off for the tool in its general settings.

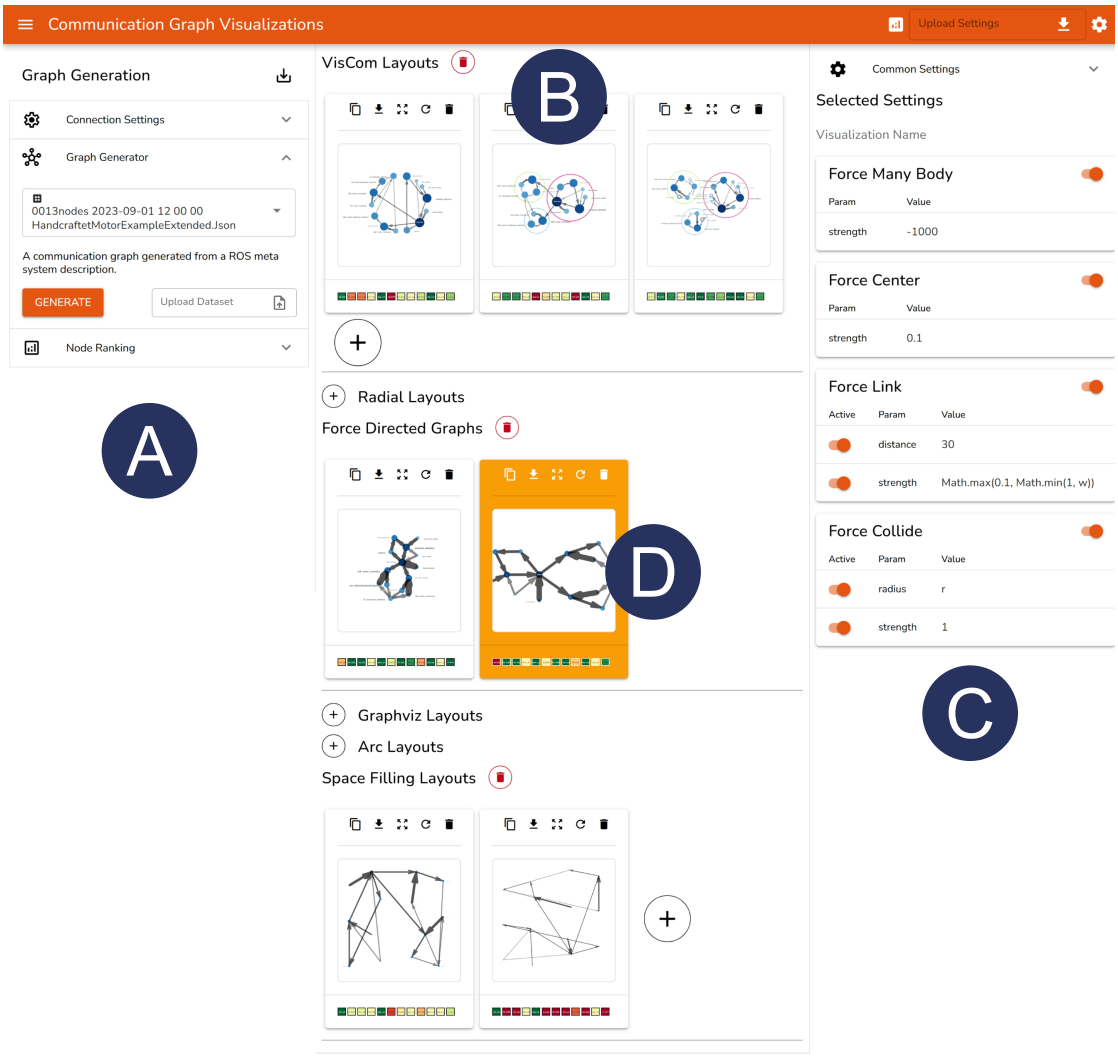


Figure 6.1: *Screenshot of the graph layout evaluation system.* ***(A)*** *The left panel provides access to the API mainly for fetching or uploading of existing graphs, or generation of synthesized graphs.* ***(B)*** *The center area shows the generated layouts of the selected graph. It allows for flexible addition and removal of layout algorithms.* ***(C)*** *The right panel provides access to the general settings of the tool as well as the parameters of a selected layout algorithm.* ***(D)*** *A selected layout algorithm for which the parameters can be configured in the right panel (C).*

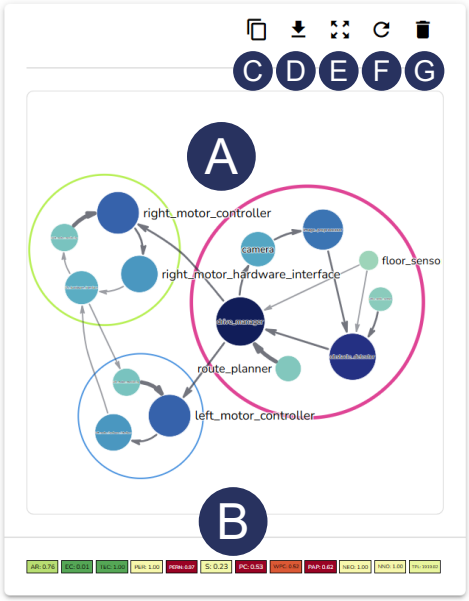


Figure 6.2: *Screenshot of a single layout algorithm tile in the evaluation system.* ***(A)*** *The interactive main canvas for the layout, that allows the user to zoom, pan, and interact with the graph.* ***(B)*** *The calculated metrics for the layout (with more details on user interaction). The color of each metric indicates the relative rank compared to other layouts.* ***(C)*** *Duplicate the layout tile.* ***(D)*** *Download the layout as PDF.* ***(E)*** *View the layout in full screen mode.* ***(F)*** *Rerender the layout.* ***(G)*** *Delete the layout tile.*

6.1.4 Implemented Layout Algorithms

The following graph layout algorithms are implemented in the frontend:

VisCom Layouts: VisCom is the presented layout approach of this thesis. Configurable parameters include the spacings in the radial positioning, the sorting of the nodes, spacing for the edge anchors, community detection, and algorithm variants like hierarchical edge routing and virtual nodes.

Radial Layouts [Bur+21]: Radial layout with strict circular arcs for edges as in Figure 4.6. Configurable parameters include the spacing between nodes, the curvature of the edges and a threshold for routing edges outside the circle.

Force-Directed Graphs [FR91; Kob12]: Force-directed graph layout using simulated forces to position the nodes. For this algorithm the force simulation from D3.js is used [Bos25b]. Configurable parameters include the strength of the different forces, including many-body repulsion, center attraction, link attraction and node collision.

GraphViz Layouts: GraphViz layouts for the engines *DOT* (multi-layered layouts like used in rqt_graph [TB18]) and *CIRCO* (hierarchical radial layouts) [Gan11; GKN15]. As GraphViz is easily usable as command line tool, the graph is converted to a DOT string, generated in the backend from this string as SVG, and parsed in the frontend.

Arc Layouts: Linear arc layouts like in TimeArcTrees [GBD09]. Self-implemented in the frontend. The sorting of nodes, the distance between them, and the orientation of the layout are configurable.

Space Filling Layouts: Layouts based on space filling curves [MK08]. Configurable parameters include the sorting of the nodes, the curve type (Hilbert, Moore, Peano, Gosper, Sierpinski) and its order. Self-implemented in the frontend based on LSystem descriptions for the different curves [Lin68].

The layout algorithms are illustrated in Figure 5.2 as well as in Appendix A.4 in Figures A.6 to A.9.

6.1.5 Adding new Layout Algorithms

The frontend is designed to be extensible with new layout algorithms. This is achieved by separating the layout rendering from the actual algorithm. A new algorithm subclass must provide information about the positioning of the nodes and edges, as well as the parameters for the algorithm – afterward, the system handles the rendering, the styling of the nodes and edges, and the user interaction.
To do so, new algorithms can use the following base classes:

`GraphLayouter` Base class for all layout algorithms. Classes inheriting from this class have access to the graph data, their nodes and connections. Subclasses override the `layout()` method to provide the layout of the graph.

`GraphLayouterSettings` Base class for the settings of a layout algorithm. In this class the settings are defined with types, names, default values, and descriptions, so that they can be automatically displayed in the user interface.

`NodeSorter` Base class for sorting nodes, if needed.

`NodePositioner` Base class for the positioning of nodes. If a sorter class is provided, the positioner already gets the sorted nodes.

`ConnectionLayouter` Base class for the edge layouts. If not given, the edges are drawn as straight lines between the start and end node. For the edges, there are a lot of helper classes to draw different types of edges, like straight lines, circular or elliptical arcs, or bézier curves.

Apart from the inheritance of the `GraphLayouter` class, all other classes are optional and can be omitted if not needed, as long as the `layout()` method is implemented and assigns positions to the nodes of the graph. In case of incremental layout algorithms like force-directed graphs, the `layout()` method can emit update events, which lead to a re-rendering of the layout in the frontend. As soon as the algorithm is finished and emits an *end*-event, the metrics for the layout are calculated and displayed in the layout tile.

Figure 6.3 shows a simplified UML diagram of the relationships and structures involved in the layout algorithms. Listing 6.1 contains a minimal example implementation of a new layout algorithm. In this algorithm, the nodes are placed on a diagonal line based on a user-configurable *distance* setting. The connection layout would be implemented in a similar way to the node positions.

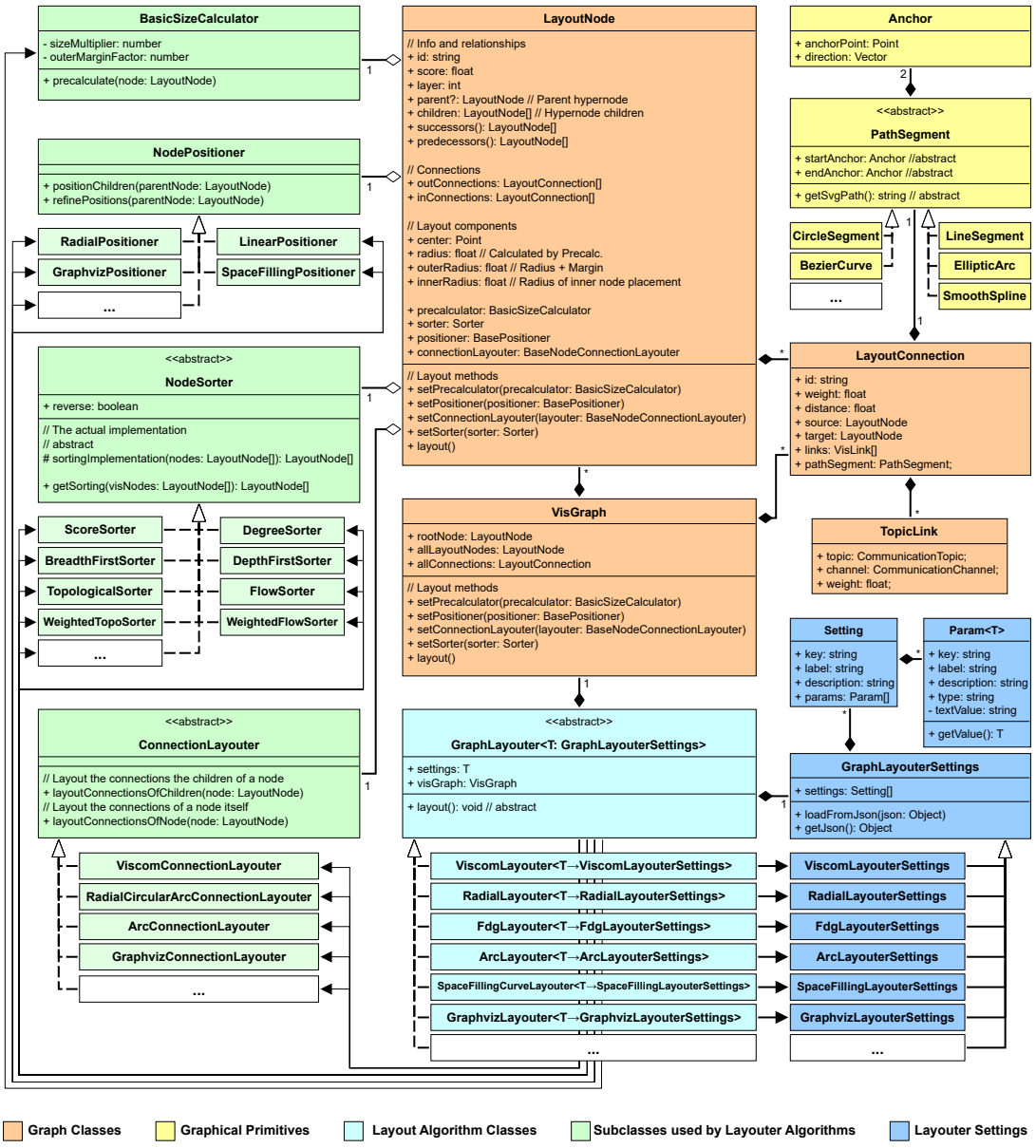


Figure 6.3: *Simplified UML diagram of the classes participating in layout algorithms. New layout algorithms (light blue) inherit from the* `GraphLayouter` *base class. The* `GraphLayouter` *subclasses implement the algorithm by specifying the used algorithms for computing node sizes, node sorting, node positions, and connection layouts (green). These classes manipulate the graph data (orange), so the node and link objects, to realize the different parts of the layout. For the nodes, they compute sizes and positions, and for the links, they compute the connection layout by assigning path segments. The path segments can be graphical primitives, like straight lines, circular arcs, or Bézier curves (yellow), or combinations of them. To allow different configurations of the layout algorithms,* `GraphLayouterSettings` *subclasses can be defined (blue). They can contain multiple settings, each consisting of one or multiple parameters, which are automatically rendered in the frontend. For easy serialization, the settings can be stored to and loaded from JSON-files.*

```
// Setting subclass, can contain multiple parameters.
// These parameters are automatically rendered in the frontend.
export class SizeSetting extends Setting {
    // Distance between nodes
    nodeDistance = new Param({
        key: "nodeDistance", label: "Distance between the nodes.",
        defaultValue: "40",
    });

    constructor() {
        // Defining the information about this setting subclass
        super({
            key: "size", label: "Size",
            description: "Size settings for the layout.",
        });
    }
}

// Settings subclass, can contain multiple setting collections
export class MyLayouterSettings extends GraphLayouterSettings {
    // Size setting subclass
    size = new SizeSetting();
}

// Positioner subclass taking care of the node positions.
export class MyPositioner extends NodePositioner {
    constructor(public settings: MyLayouterSettings) {
        super();
    }

    // Overridden method to position nodes.
    override async positionChildren(parentNode: LayoutNode) {
        parentNode.children.forEach((child, i) => {
            // Use the settings information
            const distance =
                this.settings.size.nodeDistance.getValue() ?? 40;
            // Place the nodes
            child.x = i * distance;
            child.y = i * distance;
        })
    }
}

// The layouter subclass.
// Combines settings, positioner, sorter and connection layouter.
// Settings subclass is passed as generic argument.
export class MyLayouter extends GraphLayouter<MyLayouterSettings> {
    override async layout() {
        // Instantiate positioner and pass the settings
        const positioner = new MyPositioner(this.settings);

        // Set the positioner for the graph
        this.visGraph.setPositioner(positioner);

        // The layout method uses the defined objects
        // like the positioner
        await this.visGraph.layout();
        this.emitEvent("end");
    }
}
```

Listing 6.1: *Example for implementing a new layout algorithm in the evaluation system. A new layout algorithm is represented by a* `GraphLayouter` *subclass that contains other subclasses needed for the layout. This can include subclasses inheriting from GraphLayouterSettings, NodePositioner, NodeSorter, and ConnectionLayouter. Within the overridden* `layout` *method of the* `GraphLayouter` *subclass, these subclasses are assembled and used for the complete layout of the visualization.*

6.2 Backend

Some tasks of the graph evaluation framework can be computationally intensive, especially when calculating metrics for several large graph layouts. By default, the web-based frontend runs in the single main-thread of the browser [MDN24]. While it is possible to run tasks in dedicated web workers [MDN25b], it was decided to outsource the computationally expensive tasks to a backend Python server. Python provides versatile libraries for graph generation and analysis, and an easy way to set up multi-processing.

6.2.1 Features

The main features of the backend include the following:

- Generation of random graph data using different graph generators, as well as loading saved datasets.
- Community detection and calculation of node scores in communication networks.
- Calculation of graph quality metrics of provided layouts.

These functionalities are provided through a REST API, which can be accessed by the frontend or any other client.

6.2.2 Technologies

The backend is based on a *Flask* server [Fla25], a lightweight web framework for Python taking care of the REST API endpoints. Most graph generation and analysis tasks are performed using the *NetworkX* library [Net24a], a widely used library for the creation, manipulation, and analysis of graphs. NetworkX already provides a lot of basic algorithms for graph generation, community detection, and node ranking. The node scoring algorithms for communication graphs proposed in this thesis (Equation (4.8) and Equation (4.9)) are implemented to fit in the NetworkX ecosystem. The same applies to the method for synthetic communication graph generation, used to generate the synthetic datasets used in the evaluation. For the metrics, most methods are self-implemented with the usage of the *svgpathtools* library [Por25] to calculate, for examples, lengths and intersections of SVG-paths, along with the *NumPy* and *SciPy* libraries [Num24; Sci25] for numerical calculations.

6.2.3 REST API

The services provided by the backend are provided through a REST API [MDN25a]. Table 6.1 contains an overview of the API endpoints.

Most endpoints of Table 6.1 directly provide the required information or data in a synchronous manner to the frontend. One exception is the metric calculation endpoint `/metrics/calculate/<method>` which can be called in two different ways:

Table 6.1: *Overview of API endpoints provided by the backend.*

Endpoint	Description	Return Data (Abbr.)
`/generate/methods`	List all available graph generator methods and their parameters.	Generator methods and their parameters.
`/generate/<generator>`	Generate a graph using the specified generator and parameters.	Node-link graph data.
`/analyze/communities/methods`	List all available community detection methods and their parameters.	Community detection methods and their parameters.
`/analyze/communities/<method>`	Detect communities in a given graph using the specified method.	Found communities as node lists.
`/analyze/noderank/methods`	List all available node ranking (score) methods and their parameters.	Node rank methods and their settings.
`/analyze/noderank/<method>`	Calculate node ranking for a given graph using the specified method.	Map of node ID to normalized score.
`/metrics/methods`	List all available metric calculation methods.	List of metric methods.
`/metrics/calculate/<method>`	Calculate a specific metric for a given graph layout (async or sync).	Job info (when async mode) or the metric result (when sync mode).
`/metrics/jobs/<job_id>`	Get the status and result of a metrics calculation job.	Job status and results if finished.

1. Synchronous: The result is directly calculated and returned to the frontend as response to the request.

2. Asynchronous: The backend starts a separate job to calculate the metric. In this case, the request is directly responded to the frontend, containing a job ID, which can be used to check the status of the job. This way, the frontend continues to be responsive while the backend can start multiple metric calculation jobs in parallel. The regular status check of the job returns the metric result back to the frontend as soon as its computation has been finished.

6.2.4 Extending the Backend

Similar to the frontend, the backend is designed to be extensible by developers. The extensibility primarily relates to the addition of **(1)** new graph generators, **(2)** node scoring methods, and **(3)** metrics.

New Graph Data. The backend provides two different types of data: Loaded data from saved datasets, and randomly synthesized data by different graph generator methods. Both types are exposed to the `/generate/methods` endpoint.

To extend the available saved datasets, developers can either export new datasets from running ROS-systems with *RosMetaSys* [Sch24a], or exported synthesized datasets directly in the frontend. With both methods, the exported files in JSON-format can be added to the `./data/datasets` directory of the backend project and are then directly available.

```
"r-ary_tree": {
    "params": [{
            "key": "r", "type": "int",
            "description": "Branching factor of the tree",
            "range": [1, 10], "default": 2
        },
        {
            "key": "n", "type": "int",
            "description": "Number of nodes",
            "range": [1, MAX_NODES], "default": 10
        }
    ],
    "description": "Creates a full r-ary tree of n nodes.",
    "method": get_nx_to_commgraph_method(nx.full_rary_tree)
}
```

Listing 6.2: *Example definition of a new graph generator in the backend. The definition contains a list of parameters, a description, and a callback method from the NetworkX library, with its output converted to the communication network data format.*

To add new graph generators, developers have to register the new synthesizing method in the backend, as shown exemplary in Listing 6.2. In addition to the actual method generating the data, this registration provides information about the configurable parameters. The parameter description contains information like the parameter types – with Integers, Floats, Strings, and a choice of values being supported –, default values and the valid ranges. The frontend uses this information to create appropriate input elements in order to build the correct requests to the backend. The generator method can be any method creating a directed or undirected *NetworkX* graph, that is converted afterwards automatically to the correct format for communication networks containing topics and channels. If new generator methods require fine-grained control about the connections, topics and channels can be directly specified in the edge attributes.

New Node Score Methods. New node scoring methods are registered in the backend in a similar way as the graph generators. The registration contains the name of the method, a description, and the parameters for the method. As return value, the node scoring methods have to return a dictionary with the node ID as key and the score as value. The backend automatically normalizes the scores to a range between 0 and 1.

New Metrics. New metrics are implemented as subclasses of the `MetricCalculator` class, overriding the `calculate()` method. In this class, the name for the metric is defined statically. On program start, every existing metric subclass is registered in the backend and can be used in the API using the `/metrics/calculate/<method>` endpoint. The return value of the metric calculation is an `MetricResult` object, which contains the calculated value and information about the metric, like its name, description, the unit, and the optimum of the metric (for example, if lower or higher values are preferable).

6.3 Run-Time Performance

On a notebook (CPU: Intel i7-1195G7 8-Cores @ 2.9 GHz, RAM: 16 GByte, OS: Windows 11), running both frontend and backend, the following execution times were measured:

- A graph with 150 nodes / 312 edges takes $\approx$ 2 seconds to be rendered with the presented approach (with community detection calculated by the backend and the layout by the frontend). Smaller graphs with less than 50 nodes need $<$ 0.5 seconds.
- For the metric calculation presented in Figure 5.3, 48 layouts have been calculated concurrently. For a graph with 14 nodes / 18 edges, all static layouts have been displayed in $\approx$ 3 seconds. 9 of the 48 layouts are simulated force-directed graphs, which need a total of $\approx$ 12 seconds to finish. For a graph with 100 nodes / 130 edges, the 48 Layouts took $\approx$ 8 seconds to calculate all static layouts, $\approx$ 90 seconds to finish the force-simulated layouts.
- The 12 metrics for a graph with 14 nodes / 18 edges are computed by the backend in $\approx$ 9 seconds for a single layout. The metric calculations scale approximately linearly to the number of layouts, so that the same graph took $\approx$ 7 minutes for 48 layouts. For a graph with 150 nodes / 312 edges, the 12 metrics took $\approx$ 38 seconds to be computed for a single layout and around $\approx$ 31 minutes for 48 layouts.

6.4 Deployment

The source code of the framework is available on GitHub at /vschroeter/VisCom-Thesis. For easy testing, the frontend is available in a limited mode at vschroeter.github.io/VisCom-Thesis. To be able to use all the functions of the system, the backend must also be started locally or in a Docker container.

Using Docker. The project can be started using docker. After cloning the repository, run `docker-compose up` in the root directory to start the backend and the frontend. After the images are built and the containers are started, the frontend is available at `localhost:9000`, with the backend automatically configured to run on `localhost:5000`.

From Source. The frontend is a Node.js project setup with pnpm. To build and run it from source, run `pnpm install` to install the dependencies and `pnpm dev` to start the development server. The backend is a python project setup with *uv*. After installing uv, run `uv run viscom_backend/src/viscom_backend/main.py` to start the backend server. The frontend and the backend are both automatically reloaded to changes in the source code.

Chapter 7

Discussions and Future Work

The presented visualization technique includes a number of heuristic assumptions and situational design choices. These assumptions may work well for the use case of communication networks use case considered in this thesis, but may fail in other scenarios.

7.1 Threats to Validity

Evaluated Data. Currently, the technique has been tested on a very limited amount of data, in particular on only a few different real-world data sets. Although synthetic data sets were included to extend the evaluation, these data sets and their characteristics are based on the knowledge gained from the available real-world data sets and therefore do not cover new scenarios. Data from other domains, such as social networks or other technical domains, could lead to different results and better or worse applicability of the presented approach.

Evaluated Layout Algorithms. The same threats to validity apply to the existing layout algorithms with which the approach was compared. Not only is the number of algorithms limited, but they are also self-implemented in the context of the evaluation framework presented in this thesis. Thus, they might not be as optimized as existing implementations of the algorithms. For example, the force-directed layout is based on the default simulation included in D3.js [Bos25b]. Although the parameters of the implemented comparison algorithms have been carefully configured, there may be significantly better configurations. These, as well as other layout techniques not implemented within the evaluation framework, may lead to better results compared to the proposed approach.

Computed Metrics. The presented approach is only evaluated quantitatively by comparing its graph quality metrics to other layouts. Within this evaluation, a threat to the validity arises due to the selection of metrics computed. Although most of the selected metrics are widely used in the literature, the selection by no means covers all existing metrics [Ben+07; DS09; Moo+24]. There could be other metrics that are more appropriate for the evaluation of other use cases and that yield different results. Furthermore, the presented Path Continuity and Path Prediction metrics are not well covered in the literature and were therefore adapted based on existing approaches with a focus on their applicability in this thesis [DS09].

Qualitative Evaluation. Additionally, a qualitative evaluation in form of a user study, preferably with real developers and users, should be conducted to assess the usability of the layout. The presented visualization evaluation tool is prepared for additional layout algorithms and layout quality metrics, building a starting point further graph layout research. Qualitative user studies could include task-oriented objective measures, such as the time required to find a particular node, edge, or pattern in the graph, or subjective measures, such as the perceived appearance and accessibility of the layout. In the future, the evaluation framework could also be used to support extensive user studies, such as those carried out by Yoghourdjian *et al.* [Yog+21].

7.2 Approach

Comparison to Existing Tools. A direct comparison with existing visualization tools for communication network is not straightforward possible, as they often have a different focus for the layouting. For example, *rqt_graph* [Qui+09], which has introduced in Figure 1.1(a) the problem statement for this thesis, renders all available connections in a system. Details, such as the names of the topics, are also displayed directly in this visualization. Instead, the proposed *VisCom* layout uses edge aggregation as well as filtering to reduce the number of connections shown in the overview. While VisCom shows node labels, the edge aggregation in its current implementation obscures the names of the topics completely.

Nevertheless, a comparison of the two tools is shown in Figure 7.1 and Figure 7.2 for two different datasets, to get an impression of the differences in the visualization approach.

Figure 7.1 visualizes a system, that has almost no multi-connective characteristics. This way, the number of edges in *rqt_graph* and *VisCom*, and thus the layouts are comparable. The system visualized in Figure 7.2 has fewer nodes, but contains a typical broadcast characteristic. It contains a dedicated broadcast node, which is connected to all other nodes in the system (e.g. for global logging purposes), as well as broadcast connections between the other nodes themselves (e.g. for triggering global events).

Both visualizations illustrate the strengths of the approach presented in this thesis. In the first example (Figure 7.1), the *VisCom* layout is able to better visualize the overall structure of the system. Not only are the nodes arranged in a more compact way, but also the community structure and circular communication patterns are emphasized by the layout. The layered arrangement in *rqt_graph* on the other hand leads to a more stretched layout in which nodes are further apart from each other.

The second example (Figure 7.2) shows the strength of the *VisCom* approach in reducing visual clutter by aggregating and filtering details. Due to the broadcast connections, the system is fully connected, and *rqt_graph* displays the system as it is. The rendering of all edges leads to a very cluttered layout. Edges are hard to distinguish and trace, nodes are difficult to identify, and the overall structure of the system is lost. VisCom, on the other hand, is not only reducing the visual clutter by aggregating the

edges, but is also emphasizing the actual structure of the system. Due to the data preprocessing, differences in the significance of nodes and edges are highlighted.

Communication Channels. Communication channels are currently only included in the data model. For the scoring as well as for the visualization, the channels are handled all equally. In the future, channels could be scored differently, based on further system analysis or provided knowledge from developers or users. Similarly, different channels could be visualized by applying different styles (color, width, dash-style) to the edges respectively. This would be especially reasonable for systems like ROS, where the different types of communication (publisher-subscriber connections, client-server connections, and actions), often have a semantically distinct meaning [Qui+09].

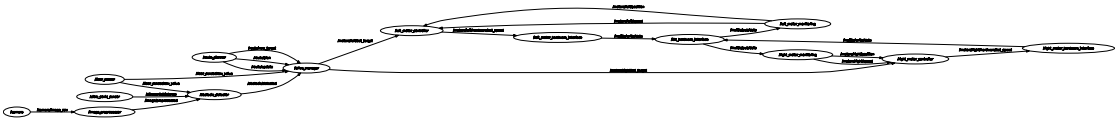
Dynamic System Changes. Currently, the approach focuses on static graphs. However, in communication networks, it is common for nodes to join or leave the system, or for new connections to be established [Qui+09; Mac+22]. In this case of system changes, the layout is currently completely recalculated. Especially if the resulting sorting differs significantly from the previous layout, this can lead to a loss of the mental map [Ead+91; Dog+18]. The same is true for changes in the community structure.

Multiple approaches would improve the preservation of the mental map. For example, the comprehension of changes in the layout and its nodes positions could be supported by smooth animations [EH04]. Furthermore, changes in the system could be integrated into the existing layout without triggering larger restructures up to a given threshold – e.g. by new nodes joining the best fitting community instead of reanalyzing the whole community structure of the graph.

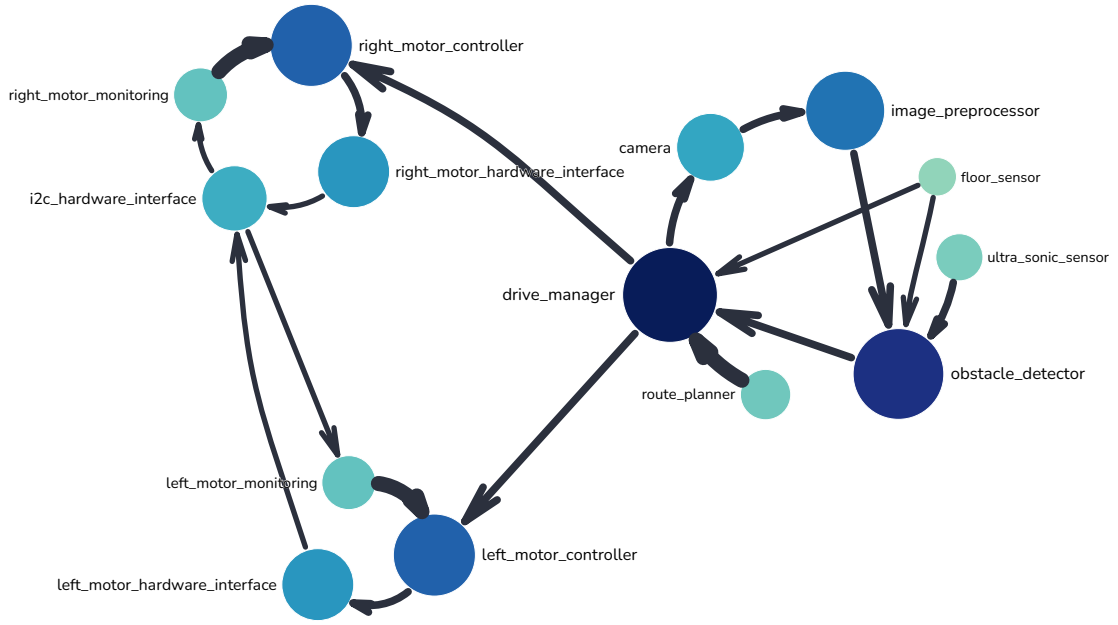
Scalability and Virtual Nodes. The task of obtaining detailed information about the system structure and its individual connections is in clear contrast to a scalable visualization of complex, dense graphs. While the proposed approach uses edge aggregation per node-pair, it still has the potential to suffer from visual clutter in complex systems. For larger graphs, there are multiple approaches for edge bundling in order to reduce edge crossings and thus visual complexity [Hol06; Wei+08; HW09]. While the presented hierarchical edge routing approach follows this idea, it tends to introduce unintuitive edge bends, which in larger systems lead to more visual clutter.

The proposed approach with using Virtual Nodes, however, is an effective method to reduce the visual clutter through extensive edge crossings while at the same time showing all edges to adjacent nodes in detail. In tests, the virtual nodes made it possible to focus entirely on the part of the graph that was currently of interest and ignore all other communities.

However, the unrestricted addition of virtual nodes runs the risk of duplicating a large number of nodes as soon as there are a large number of connections between groups, thereby increasing visual complexity. One potential solution to this challenge is to implement a threshold for when communities are merged instead of adding a lot of virtual nodes. Another compromise for this problem could be the calculation of

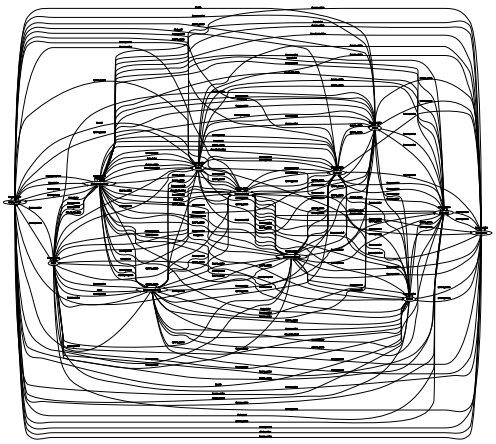


(a) *rqt_graph*

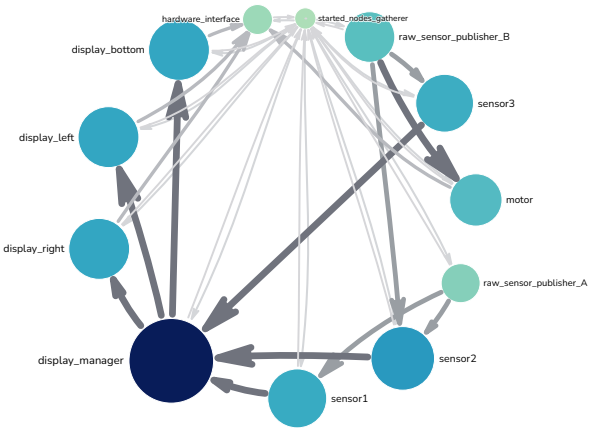


(b) *VisCom*

Figure 7.1: *Comparison visualizating the same system with 14 nodes in rqt_graph and the proposed VisCom layout with community detection. In this example, the system has almost no multi-connective characteristics, so the count of visualized edges is almost the same in both visualizations.*



(a) *rqt_graph*



(b) *VisCom*

Figure 7.2: *Comparison visualizating the same system with 12 nodes in rqt_graph and the proposed VisCom layout. In this example, the system has broadcast connections to an broadcast node as well as between the other nodes themselves.*

overlapping communities, so that only nodes belonging to multiple communities are virtually duplicated [Shi+13; Din+16]. In case of very large communities detected by algorithms like the Louvain algorithm, it would also be interesting to investigate hierarchically applied community detection to further reduce visual clutter.

7.3 Evaluation Framework

The evaluation framework was implemented as a prototype focusing on the proof of concept. It currently offers a limited selection of algorithms and techniques, as well as a limited capability of handling larger graphs.

Supported Graph Formats. In the context of this thesis, the tool has been implemented to handle communication networks and a custom data format uploading graph data, which depicts the introduced data representation for this type of networks. This data format originates in exports of ROS 2 system snapshots, using *rosmetasys* [Sch24a]. The data preprocessing, including the calculation of node and edge score, is also focused on communication networks with topics and channels. However, the layout algorithms are capable of processing and visualizing any graph data with nodes and edges, both with or without scores. Thus, in future, the tool can be extended by accepting other common graph data formats as well, such as GraphML files [Bra+02; Gra07] and other custom JSON files or CSV files. The same applies to the data format of downloaded graphs, which is currently only available in the custom format of the ROS 2 system exporter [Sch24a].

User Interaction and Detail View. The user interaction in the frontend currently only includes highlighting of selected elements and their adjacent nodes and edges. For further steps towards an information visualization, further details could be provided on demand [Shn03]. Regarding communication networks and aggregated topics, this could include the number of included topics or topic names when interacting with an edge.

As the next step, user interactions on nodes or edges can trigger the change to another, more detailed focus+context view [Ger+09]. Within such a detail view centered around a selected node, connected predecessor and successor nodes, as well as a list of all provided topics could be visualized. An example of such a detail view is shown in Figure 7.3.

Implemented Layout Algorithms. The evaluation framework is currently limited to the implemented layout algorithms, including the proposed VisCom layout and a few other algorithms such as force-directed layouts [FR91], circular layouts, hierarchical GraphViz layouts [GKN15], space filling curves [MK08], and arc layouts [GBD09]. The area of layout algorithms and node-link visualizations is very versatile and there are many other algorithms that can be integrated into the framework in the future [Moo+24]. Besides force-directed simulations, other optimization algorithms would be particularly

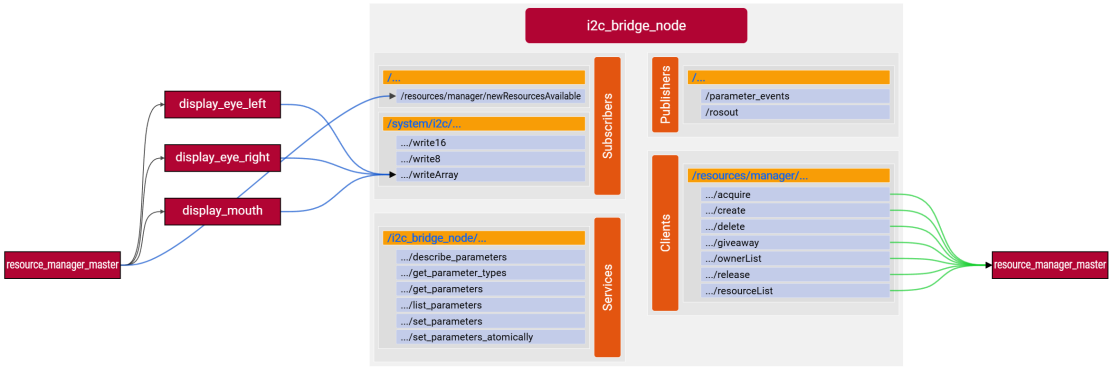


Figure 7.3: *Example of a possible detail view for a selected node of a ROS 2 system. The node in the center is the selected node, while the surrounding nodes are its direct predecessors and successors. The selected center node shows all available topics of the selected node to which the surrounding nodes are connected. The color of the edges indicates the channel of the connection (blue for publisher-subscriber, green for client-server, black for broadcast). This detail view is taken from a preliminary work to this thesis, available at vschroeter.github.io/RosComGraph/.*

interesting. This includes, for example, drawings based on stress majorization [GKN05] or stochastic gradient descent [ZPG19].

Performance. The visualizations in the frontend are based on SVGs. Vue.js for reactive coupling between data and UI could be also used for the visualizations by using its component based approached for nodes and edges, so that each graphical element is realized by a separate Vue component, combining data and SVG primitives. However, displaying a large number of components that are updated at the same time can lead to performance losses. For that reason, the visualizations are implemented using pure D3.js [Bos25a], combined with a stateful object management which guarantees, that only the necessary elements are rendered or updated. The latter is especially necessary for smooth user interactions.

While D3.js provides a powerful API for data-driven visualizations, the SVG- and DOM-based rendering can be slow for visualizations with a large number of graphical elements. Other approaches that are using canvas-based rendering, like *PixiJS* [Gro25], or WebGL-based rendering, like *Three.js* [The25] or *NetV.js* [Han+21], could drastically improve the performance for more complex visualizations.

Dynamic System Changes. Similar to the approach, the frontend focuses on the visualization of static graphs. It could be extended to support dynamic graphs, for example from a running ROS 2 system, so that the layout is updated in real-time. In this case, the comprehension of changes in the layout and its nodes positions could be supported by smooth animations and highlighting changed elements for the user [EH04].

Chapter 8

Conclusions

This thesis introduced *VisCom* as a novel graph visualization technique with better visual scalability compared to existing techniques. Its deterministically computed radial layout, which is based on heuristically guided data preprocessing algorithms, allows the display of a large number of nodes and edges. When combined with community detection algorithms, the layout technique is designed to visually emphasize the difference between intra- and inter-community connections, while avoiding node-node- and minimizing node-edge-overlaps.

VisCom was developed in particular for the visualization of communication networks, as the data preprocessing, which computes scores to quantify the structural significance of nodes and edges, is tailored to the specific characteristics of communication networks. However, it can be applied to any multi-graph structures with named edges, for example from various technical domains such as robotics, Internet of Things, and microservice architectures. Beyond these areas, applicable graphs can also be derived from other domains, such as social networks.

The order of the nodes within the layout is determined using the presented Weighted Flow Sorting algorithm. This algorithm computes topological orderings of directed acyclic graphs or pseudo-topological orderings of directed cyclic graphs and is applicable to any weighted graph, regardless of the underlying structure. Therefore, it can be used as an alternative to existing sorting and graph-traversal algorithms like depth-first search (DFS) or breadth-first search (BFS), with an emphasis on keeping nodes that are connected by shortest paths close together. Thus, the algorithm is suitable as a base for any linearized layout approach, such as TimeArcTrees [GBD09] or Hive Plots [Krz+12; NW23].

Although designed for directed, weighted graphs, the radial layout technique combined with the Weighted Flow Sorting algorithm can be generalized to any graph structure, independent of communication networks. Combined with community detection or any other node grouping strategy, it generates multi-circular views with layout of intra- and inter-group connections.

For demonstration and quantitative evaluation of the presented technique, a versatile evaluation framework was implemented and published as open-source software as part of this thesis. Users and developers can use the framework to compare multiple graph layout techniques side-by-side and evaluate their performance with regard to various graph quality metrics. For this purpose, the framework contains several real-world datasets from ROS 2 systems and also supports the synthetic generation and upload of custom graphs.

The framework is designed to be extensible and as such can be adapted to any graph layout technique or quality metric. When applied to various real-world communication graphs from ROS 2 systems as well as synthetically generated datasets, the proposed visualization technique consistently achieved comparable performance with regard to multiple graph quality metrics. In particular, the use of virtual nodes was shown to be effective in improving local adjacency and removing long, hard to follow inter-group edges.

One specific aim of the research conducted in the scope of this thesis was to provide a better overview than *rqt_graph* [TB18], which is the currently used tool for visualizing ROS 2 communication graphs. Compared to *rqt_graph* or its alternatives [Fox24; Lal24], the proposed VisCom method improves the visualization of a system overview in two main aspects, as illustrated in Figure 7.1 and Figure 7.2:

1. VisCom reduces the visual clutter by aggregating information like multiple topic connections between node pairs and filtering out structurally unimportant edges. Thus, the layout provides a comprehensible and manageable overview, which can be used as a base for subsequent zoom-in and focus+context steps [Ger+09].

2. Within the radial arrangement of nodes in the layout of VisCom, the proximity of adjacent nodes and the traceability of dataflows are optimized by the proposed *Weighted Flow Sorting* algorithm, based on the introduced computation of edge significance scores. The presented technique can effectively handle broadcast characteristics and cycles of communication networks and create structural representations of communication networks.

In order to be usable for ROS 2 or other systems containing communication networks, a visualization method such as the proposed technique would need to be further integrated into their ecosystems and the tasks of developers or users. Providing more detail – both through user interaction and detail views on demand – would be essential for productive use. User interaction at the overview level could make topic names, or at least the number of topics, visible on the aggregated edges. While focusing on a node, the view could switch to a node-centered detail view with further information of topic connections to adjacent nodes around it [Yee+01; JK03], as shown in Figure 7.3.

In smaller systems with simpler connection characteristics, e.g., directed acyclic graphs or trees, solutions like *rqt_graph* create straightforward layered layouts, illustrating the existing structures very plausibly. The same applies to established algorithms like force-directed layouts. It would be valuable to explore integration of these techniques into the presented layout approach [Gan11].

References

[Aut24] AutowareFoundation. *Autowarefoundation/Autoware*. /autowarefoundation/autoware. 2024. (Visited on 04/22/2024).

[BA99] Albert-László Barabási and Réka Albert. *Emergence of Scaling in Random Networks*. https://www.science.org/doi/10.1126/science.286.5439.509. 1999. DOI: 10.1126/science.286.5439.509. (Visited on 04/09/2025).

[Ban+13] Michael J. Bannister, David Eppstein, Michael T. Goodrich, and Lowell Trott. "Force-Directed Graph Drawing Using Social Gravity and Scaling". In: Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-36763-2. DOI: 10.1007/978-3-642-36763-2_37. (Visited on 04/03/2025).

[Bas25] Mathieu Bastian. *Gephi - The Open Graph Viz Platform*. https://gephi.org/. 2025. (Visited on 04/27/2025).

[Bav50] Alex Bavelas. "Communication Patterns in Task-Oriented Groups." In: *Journal of the acoustical society of America* (1950). DOI: 10.1121/1.1906679.

[BB08] Michael Baur and Ulrik Brandes. "Multi-Circular Layout of Micro/Macro Graphs". In: *Graph Drawing*. Ed. by Seok-Hee Hong, Takao Nishizeki, and Wu Quan. Vol. 4875. Springer Berlin Heidelberg, 2008, pp. 255–267. ISBN: 978-3-540-77536-2 978-3-540-77537-9. DOI: 10.1007/978-3-540-77537-9_26. (Visited on 04/07/2025).

[Bea65] Murray A. Beauchamp. "An Improved Index of Centrality". In: *Behavioral Science* 10.2 (1965), pp. 161–163. ISSN: 00057940, 10991743. DOI: 10.1002/bs.3830100205. (Visited on 02/25/2025).

[Bec+17] Fabian Beck, Michael Burch, Stephan Diehl, and Daniel Weiskopf. "A Taxonomy and Survey of Dynamic Graph Visualization". In: *Computer Graphics Forum* 36.1 (2017), pp. 133–159. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/cgf.12791. (Visited on 04/22/2024).

[Ben+07] Chris Bennett, Jody Ryall, Leo Spalteholz, and Amy Gooch. "The Aesthetics of Graph Visualization". In: *Computational Aesthetics in Graphics, Visualization, and Imaging*. EG, 2007, pp. 57–64. ISBN: 978-3-905673-43-2. DOI: 10.2312/COMPAESTH/COMPAESTH07/057-064.

[BHJ09] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. "Gephi: An Open Source Software for Exploring and Manipulating Networks". In: *Proceedings of the International AAAI Conference on Web and Social Media* 3.1 (2009), pp. 361–362. ISSN: 2334-0770, 2162-3449. DOI: 10.1609/icwsm.v3i1.13937. (Visited on 04/27/2025).

[Bik+16] Nikos Bikakis, John Liagouris, Maria Krommyda, George Papastefanatos, and Timos Sellis. *graphVizdb: A Scalable Platform for Interactive Large Graph Visualization*. 2016. DOI: 10.1109/ICDE.2016.7498340.

[Bir+06] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. "Mining Email Social Networks". In: *Proceedings of the 2006 International Workshop on Mining Software Repositories*. ACM, 2006, pp. 137–143. ISBN: 978-1-59593-397-3. DOI: 10.1145/1137983.1138016. (Visited on 04/06/2025).

[BJT23] Francis Bloch, Matthew O. Jackson, and Pietro Tebaldi. "Centrality Measures in Networks". In: *Social Choice and Welfare* 61.2 (2023), pp. 413–453. ISSN: 0176-1714, 1432-217X. DOI: 10.1007/s00355-023-01456-4. (Visited on 04/03/2025).

[Blo+08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. "Fast Unfolding of Communities in Large Networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (2008), P10008. ISSN: 1742-5468. DOI: 10.1088/1742-5468/2008/10/P10008. (Visited on 03/30/2025).

[Bos23] Mike Bostock. *D3 Gallery / D3*. https://observablehq.com/@d3/gallery. 2023. (Visited on 04/08/2025).

[Bos25a] Mike Bostock. *D3 by Observable | The JavaScript Library for Bespoke Data Visualization*. https://d3js.org/. 2025. (Visited on 04/18/2025).

[Bos25b] Mike Bostock. *Force Simulations | D3 by Observable*. https://d3js.org/d3-force/simulation. 2025. (Visited on 04/18/2025).

[Bra+02] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt, and M. Scott Marshall. "GraphML Progress Report Structural Layer Proposal". In: *Graph Drawing*. Ed. by Gerhard Goos, Juris Hartmanis, Jan Van Leeuwen, Petra Mutzel, Michael Jünger, and Sebastian Leipert. Vol. 2265. Springer Berlin Heidelberg, 2002, pp. 501–512. ISBN: 978-3-540-43309-5 978-3-540-45848-7. DOI: 10.1007/3-540-45848-4_59. (Visited on 04/29/2025).

[BST00] Ulrik Brandes, Galina Shubina, and Roberto Tamassia. "Improving Angular Resolution in Visualizations of Geographic Networks". In: *Data Visualization 2000*. Ed. by Willem Cornelis De Leeuw and Robert Van Liere. Springer Vienna, 2000, pp. 23–32. ISBN: 978-3-211-83515-9 978-3-7091-6783-0. DOI: 10.1007/978-3-7091-6783-0_3. (Visited on 03/13/2025).

[Bur+21] Michael Burch, Kiet Bennema Ten Brinke, Adrien Castella, Ghassen Karray Sebastiaan Peters, Vasil Shteriyanov, and Rinse Vlasvinkel. "Dynamic Graph Exploration by Interactively Linked Node-Link Diagrams and Matrix Visualizations". In: *Visual Computing for Industry, Biomedicine, and Art* 4.1 (2021), pp. 1–14. DOI: 10.1186/s42492-021-00088-8.

[BV14] Paolo Boldi and Sebastiano Vigna. "Axioms for Centrality". In: *Internet Mathematics* 10.3-4 (2014), pp. 222–262. DOI: 10.1080/15427951.2013.865686. arXiv: 1308.2140 [cs]. (Visited on 02/23/2025).

[cel25] cellumation GmbH. *cv.SINGULATE*. https://cellumation.com/de/produkte/cv-singulate/. 2025. (Visited on 04/09/2025).

[Cer+22] Tomas Cerny, Amr S. Abdelfattah, Vincent Bushong, Abdullah Al Maruf, and Davide Taibi. "Microservice Architecture Reconstruction and Visualization Techniques: A Review". In: *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2022, pp. 39–48. DOI: 10.1109/SOSE55356.2022.00011. (Visited on 04/06/2025).

[Che+12] Roman Chernobelskiy, Kathryn I. Cunningham, Michael T. Goodrich, Stephen G. Kobourov, and Lowell Trott. "Force-Directed Lombardi-Style Graph Drawing". In: *Graph Drawing*. Ed. by Marc Van Kreveld and Bettina Speckmann. Vol. 7034. Springer Berlin Heidelberg, 2012, pp. 320–331. ISBN: 978-3-642-25877-0 978-3-642-25878-7. DOI: 10.1007/978-3-642-25878-7_31. (Visited on 03/13/2025).

[CHS21] Wenjun Chen, Anwar Haque, and Kamran Sedig. "Design of Interactive Visualizations for Next-Generation Ultra-Large Communication Networks". In: *IEEE Access* 9 (2021), pp. 26968–26982. DOI: 10.1109/ACCESS.2021.3057803.

[Cor09] Thomas H. Cormen, ed. *Introduction to Algorithms*. 3rd ed. MIT Press, 2009. ISBN: 978-0-262-03384-8 978-0-262-53305-8.

[CP96] Michael K. Coleman and D. Stott Parker. "Aesthetics-Based Graph Layout for Human Consumption". In: *Software: Practice and Experience* 26.12 (1996), pp. 1415–1438. ISSN: 0038-0644, 1097-024X. DOI: 10.1002/(SICI)1097-024X(199612)26:12<1415::AID-SPE69>3.0.CO;2-P. (Visited on 04/02/2025).

[Din+16] Zhuanlian Ding, Xingyi Zhang, Dengdi Sun, and Bin Luo. "Overlapping Community Detection Based on Network Decomposition". In: *Scientific Reports* 6.1 (2016), p. 24115. ISSN: 2045-2322. DOI: 10.1038/srep24115. (Visited on 10/15/2024).

[Dog+18] Ugur Dogrusoz, Alper Karacelik, Ilkin Safarli, Hasan Balci, Leonard Dervishi, and Metin Can Siper. "Efficient Methods and Readily Customizable Libraries for Managing Complexity of Large Networks". In: *PLoS ONE* 13.5 (2018),

e0197238. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0197238. (Visited on 04/27/2025).

[DS09] Cody Dunne and Ben Shneiderman. "Improving Graph Drawing Readability by Incorporating Readability Metrics: A Software Tool for Network Analysts". In: *University of Maryland, HCIL Tech Report HCIL* 13 (2009).

[Dun+11] Christian A. Duncan, David Eppstein, Michael T. Goodrich, Stephen G. Kobourov, and Martin Nöllenburg. "Lombardi Drawings of Graphs". In: Springer Berlin Heidelberg, 2011, pp. 195–207. ISBN: 978-3-642-18469-7. DOI: 10.1007/978-3-642-18469-7_18. (Visited on 03/13/2025).

[Ead+91] Peter Eades, Wei Lai, Kazuo Misue, and Kozo Sugiyama. *Preserving the Mental Map of a Diagram*. Tech. rep. Technical Report IIAS-RR-91-16E, Fujitsu Laboratories, 1991.

[ED07] G. Ellis and A. Dix. "A Taxonomy of Clutter Reduction for Information Visualisation". In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1216–1223. ISSN: 1077-2626. DOI: 10.1109/TVCG.2007.70535. (Visited on 03/13/2025).

[EH04] Peter Eades and Mao Lin Huang. "Navigating Clustered Graphs Using Force-Directed Methods". In: *Graph Algorithms And Applications 2*. 2004, pp. 191–215. DOI: 10.1142/9789812794741_0010.

[ELS93] Peter Eades, Xuemin Lin, and W.F. Smyth. "A Fast and Effective Heuristic for the Feedback Arc Set Problem". In: *Information Processing Letters* 47.6 (1993), pp. 319–323. ISSN: 00200190. DOI: 10.1016/0020-0190(93)90079-O. (Visited on 04/11/2025).

[ER60] P Erdős and A Rényi. "On the evolution of random graphs". In: *Publ. math. inst. hung. acad. sci* 5.1 (1960), pp. 17–60.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. "On Power-Law Relationships of the Internet Topology". In: *ACM SIGCOMM Computer Communication Review* 29.4 (1999), pp. 251–262. ISSN: 0146-4833. DOI: 10.1145/316194.316229. (Visited on 04/27/2025).

[Fla25] Flask. *Welcome to Flask — Flask Documentation (3.1.x)*. https://flask.palletsprojects.com/en/stable/. 2025. (Visited on 04/18/2025).

[Fox24] Foxglove Studio. *Foxglove Studio - Topic Graph*. https://docs.foxglove.dev/docs/visualization/panels/topic-graph. 2024.

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. "Graph Drawing by Force-directed Placement". In: *Software: Practice and Experience* 21.11 (1991), pp. 1129–1164. ISSN: 0038-0644, 1097-024X. DOI: 10.1002/spe.4380211102. (Visited on 12/15/2024).

[Fre77] Linton C. Freeman. "A Set of Measures of Centrality Based on Betweenness". In: *Sociometry* 40.1 (1977), pp. 35–41. ISSN: 0038-0431. DOI: 10.2307/3033543. JSTOR: 3033543. (Visited on 02/25/2025).

[Fre78] Linton C. Freeman. “Centrality in Social Networks Conceptual Clarification”. In: *Social Networks* 1.3 (1978), pp. 215–239. ISSN: 03788733. DOI: 10.1016/0378-8733(78)90021-7. (Visited on 02/23/2025).

[Gan11] Emden R Gansner. “Drawing Graphs with Graphviz”. In: *Technical report, AT&T Bell Laboratories* (2011).

[GBD09] Martin Greilich, Michael Burch, and Stephan Diehl. “Visualizing the Evolution of Compound Digraphs with TimeArcTrees”. In: *Computer Graphics Forum* 28.3 (2009), pp. 975–982. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/j.1467-8659.2009.01451.x. (Visited on 04/22/2024).

[Ger+09] Jens Gerken, Mathias Heilig, Hans-Christian Jetter, Sebastian Rexhausen, Mischa Demarmels, Werner A. König, and Harald Reiterer. “Lessons Learned from the Design and Evaluation of Visual Information-Seeking Systems”. In: *International Journal on Digital Libraries* 10.2-3 (2009), pp. 49–66. ISSN: 1432-5012, 1432-1300. DOI: 10.1007/s00799-009-0052-6. (Visited on 03/14/2025).

[GK07] Emden R. Gansner and Yehuda Koren. “Improved Circular Layouts”. In: *Graph Drawing*. Ed. by Michael Kaufmann and Dorothea Wagner. Vol. 4372. Springer Berlin Heidelberg, 2007, pp. 386–398. ISBN: 978-3-540-70903-9 978-3-540-70904-6. DOI: 10.1007/978-3-540-70904-6_37. (Visited on 04/29/2025).

[GKN05] Emden R. Gansner, Yehuda Koren, and Stephen North. “Graph Drawing by Stress Majorization”. In: *Graph Drawing*. Ed. by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, and János Pach. Vol. 3383. Springer Berlin Heidelberg, 2005, pp. 239–250. ISBN: 978-3-540-24528-5 978-3-540-31843-9. DOI: 10.1007/978-3-540-31843-9_25. (Visited on 04/27/2025).

[GKN15] Emden Gansner, Eleftherios Koutsofios, and Stephen North. “Drawing Graphs with Dot”. In: *Technical report, AT&T Bell Laboratories* (2015).

[GNV88] E. R. Gansner, S. C. North, and K. P. Vo. “DAG—a Program That Draws Directed Graphs”. In: *Software: Practice and Experience* 18.11 (1988), pp. 1047–1062. ISSN: 0038-0644, 1097-024X. DOI: 10.1002/spe.4380181104. (Visited on 04/16/2025).

[Gos+11] David Gossow, Adam Leeper, Dave Hershberger, and Matei Ciocarlie. “Interactive Markers: 3-D User Interfaces for ROS Applications [ROS Topics]”. In: *IEEE Robotics & Automation Magazine* 18.4 (2011), pp. 14–15. ISSN: 1070-9932. DOI: 10.1109/MRA.2011.943230. (Visited on 04/22/2024).

[Gra07] GraphML. *GraphML Specification*. http://graphml.graphdrawing.org/specification.html. 2007. (Visited on 04/29/2025).

[Gro25] Mat Groves. *PixiJS | The HTML5 Creation Engine | PixiJS*. https://pixijs.com/. 2025. (Visited on 04/25/2025).

[Guc+24] Kathrin Guckes, Lisa Eisenhardt, Margit Pohl, and Tatiana von Landesberger. *GuidelineExplorer – Navigating through the Forrest of Actionable Guidelines on Node-Link Graph Visualization*. 2024. DOI: 10.48550/arXiv.2406.05558. arXiv: 2406.05558 [cs]. (Visited on 04/22/2025).

[Han+21] Dongming Han, Jiacheng Pan, Xiaodong Zhao, and Wei Chen. "NetV.Js: A Web-Based Library for High-Efficiency Visualization of Large-Scale Graphs and Networks". In: *Visual Informatics* 5.1 (2021), pp. 61–66. ISSN: 2468502X. DOI: 10.1016/j.visinf.2021.01.002. (Visited on 04/25/2025).

[HB03] Mark Harrower and Cynthia A Brewer. "ColorBrewer. Org: An Online Tool for Selecting Colour Schemes for Maps". In: *The Cartographic Journal* 40.1 (2003), pp. 27–37. DOI: 10.1179/000870403235002042.

[Hea92] Christopher G Healey. "Visualization of Multivariate Data Using Preattentive Processing". In: (1992).

[HMM00] I. Herman, G. Melancon, and M.S. Marshall. "Graph Visualization and Navigation in Information Visualization: A Survey". In: *IEEE Transactions on Visualization and Computer Graphics* 6.1 (2000), pp. 24–43. ISSN: 10772626. DOI: 10.1109/2945.841119. (Visited on 04/15/2025).

[Hol06] D. Holten. "Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data". In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 741–748. ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.147. (Visited on 12/15/2024).

[Hol24] Yan Holtz. *Arc Diagram | the D3 Graph Gallery*. https://d3-graph-gallery.com/arc.html. 2024. (Visited on 04/27/2025).

[HW09] Danny Holten and Jarke J. van Wijk. "Force-Directed Edge Bundling for Graph Visualization". In: *Computer Graphics Forum* 28.3 (2009), pp. 983–990. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/j.1467-8659.2009.01450.x. (Visited on 03/14/2025).

[JK03] T.J. Jankun-Kelly and Kwan-Liu Ma. "MoireGraphs: Radial Focus+context Visualization and Interaction for Graphs with Visual Nodes". In: *IEEE Symposium on Information Visualization 2003 (IEEE Cat. No.03TH8714)*. IEEE, 2003, pp. 59–66. ISBN: 978-0-7803-8154-4. DOI: 10.1109/INFVIS.2003.1249009. (Visited on 04/22/2024).

[Kah62] Arthur B Kahn. "Topological Sorting of Large Networks". In: *Communications of the ACM* 5.11 (1962), pp. 558–562. DOI: 10.1145/368996.369025.

[KMM24] Nicolas Kruchten, Andrew M. McNutt, and Michael J. McGuffin. "Metrics-Based Evaluation and Comparison of Visualization Notations". In: *IEEE Transactions on Visualization and Computer Graphics* 30.1 (2024), pp. 425–435. DOI: 10.1109/TVCG.2023.3326907. (Visited on 04/07/2025).

[Kob12] Stephen G. Kobourov. *Spring Embedders and Force Directed Graph Drawing Algorithms*. 2012. DOI: 10.48550/arXiv.1201.3011. arXiv: 1201.3011 [cs]. (Visited on 04/03/2025).

[Koo+17] Roozbeh Haghnazar Koochaksaraei, Ivan Reinaldo Meneghini, Vitor Nazário Coelho, and Frederico Gadelha Guimarães. "A New Visualization Method in Many-Objective Optimization with Chord Diagram and Angular Mapping". In: *Knowledge-Based Systems* 138 (2017), pp. 134–154. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2017.09.035.

[KR12] Santosh Kumar and Sonam Rai. "Survey on Transport Layer Protocols: TCP & UDP". In: *International Journal of Computer Applications* 46 (2012).

[Kru64] J. B. Kruskal. "Nonmetric Multidimensional Scaling: A Numerical Method". In: *Psychometrika* 29.2 (1964), pp. 115–129. ISSN: 0033-3123, 1860-0980. DOI: 10.1007/BF02289694. (Visited on 03/28/2025).

[Krz+09] Martin Krzywinski, Jacqueline Schein, Inanç Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J. Jones, and Marco A. Marra. "Circos: An Information Aesthetic for Comparative Genomics". In: *Genome Research* 19.9 (2009), pp. 1639–1645. ISSN: 1088-9051. DOI: 10.1101/gr.092759.109. (Visited on 12/15/2024).

[Krz+12] M. Krzywinski, I. Birol, S. J. Jones, and M. A. Marra. "Hive Plots–Rational Approach to Visualizing Networks". In: *Briefings in Bioinformatics* 13.5 (2012), pp. 627–644. ISSN: 1467-5463, 1477-4054. DOI: 10.1093/bib/bbr069. (Visited on 08/01/2024).

[Lal24] Chris Lalancette. *Ros2/Ros_network_viz*. /ros2/ros_network_viz. 2024. (Visited on 12/15/2024).

[Li+10] Jianpo Li, Xuning Zhu, Ning Tang, and Jisheng Sui. "Study on ZigBee Network Architecture and Routing Algorithm". In: *2010 2nd International Conference on Signal Processing Systems*. Vol. 2. IEEE, 2010, pp. V2–389. DOI: 10.1109/ICSPS.2010.5555486.

[Lim+16] Seung-Hwan Lim, Sangkeun Lee, Sarah S Powers, Mallikarjun Shankar, and Neena Imam. *Survey of Approaches to Generate Realistic Synthetic Graphs*. Tech. rep. ORNL/TM–2016/3, 1339361. 2016, ORNL/TM–2016/3, 1339361. DOI: 10.2172/1339361. (Visited on 05/17/2024).

[Lin68] Aristid Lindenmayer. "Mathematical Models for Cellular Interactions in Development I. Filaments with One-Sided Inputs". In: *Journal of theoretical biology* 18.3 (1968), pp. 280–299. DOI: 10.1016/0022-5193(68)90079-9.

[Mac+22] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. "Robot Operating System 2: Design, Architecture, and Uses in the Wild". In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074.

[Man+23] Mohammad Mansour, Amal Gamal, Ahmed I. Ahmed, Lobna A. Said, Abdelmoniem Elbaz, Norbert Herencsar, and Ahmed Soltan. "Internet of Things: A Comprehensive Overview on Protocols, Architectures, Technologies, Simulation Tools, and Future Directions". In: *Energies* 16.8 (2023), p. 3465. ISSN: 1996-1073. DOI: 10.3390/en16083465. (Visited on 04/06/2025).

[MDN24] MDN. *Main Thread - MDN Web Docs Glossary: Definitions of Web-related Terms | MDN*. https://developer.mozilla.org/en-US/docs/Glossary/Main_thread. 2024. (Visited on 04/18/2025).

[MDN25a] MDN. *REST - MDN Web Docs Glossary: Definitions of Web-related Terms | MDN*. https://developer.mozilla.org/en-US/docs/Glossary/REST. 2025. (Visited on 04/18/2025).

[MDN25b] MDN. *Web Workers API - Web APIs | MDN*. https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API. 2025. (Visited on 04/18/2025).

[MK08] C. Muelder and Kwan-Liu Ma. "Rapid Graph Layout Using Space Filling Curves". In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1301–1308. ISSN: 1077-2626. DOI: 10.1109/TVCG.2008.158. (Visited on 08/19/2024).

[Moo+24] Gavin J Mooney, Helen C Purchase, Michael Wybrow, and Stephen G Kobourov. "The Multi-Dimensional Landscape of Graph Drawing Metrics". In: (2024). DOI: 10.1109/PacificVis60374.2024.00022.

[Mor24] Sergey Morozov. *Ser94mor/Self-Driving-Car-Using-Ros*. /ser94mor/self-driving-car-using-ros. 2024. (Visited on 04/22/2024).

[MTB72] Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, eds. *Complexity of Computer Computations*. Springer US, 1972. ISBN: 978-1-4684-2003-6 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2. (Visited on 04/11/2025).

[Nad+16] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, and Mike Amundsen. *Microservice Architecture*. O'Reilly Media, Inc., 2016. ISBN: 1-4919-5625-9.

[Net24a] NetworkX. *NetworkX 3.4.2 Documentation*. https://networkx.org/documentation/stable/index.html. 2024. (Visited on 04/18/2025).

[Net24b] NetworkX. *Topological_generations — NetworkX 3.4.2 Documentation*. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.dag.topological_generations.html. 2024. (Visited on 04/11/2025).

[Net25] NetworkX. *Centrality — NetworkX 3.4.2 Documentation*. https://networkx.org/documentation/stable/reference/algorithms/centrality.html. 2025. (Visited on 02/25/2025).

[NG04] M. E. J. Newman and M. Girvan. “Finding and Evaluating Community Structure in Networks”. In: *Physical Review E* 69.2 (2004), p. 026113. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.69.026113. (Visited on 04/07/2025).

[Num24] NumPy. *NumPy*. https://numpy.org/. 2024. (Visited on 04/18/2025).

[NW23] Martin Nöllenburg and Markus Wallinger. “Computing Hive Plots: A Combinatorial Framework”. In: *Graph Drawing and Network Visualization*. Springer Nature Switzerland, 2023, pp. 153–169. ISBN: 978-3-031-49275-4. DOI: 10.1007/978-3-031-49275-4_11. (Visited on 08/01/2024).

[Par18] June Sung Park. *Service-Oriented Architecture (SOA) and Microservice Architecture (MSA)*. https://jp-institute-of-software.com/439889679. 2018. (Visited on 04/27/2025).

[Pfe13] JÜrgen Pfeffer. “Fundamentals of Visualizing Communication Networks”. In: *China Communications* 10.3 (2013), pp. 82–90. ISSN: 1673-5447. DOI: 10.1109/CC.2013.6488833. (Visited on 03/14/2025).

[PK07] David J. Pearce and Paul H. J. Kelly. “A Dynamic Topological Sort Algorithm for Directed Acyclic Graphs”. In: *ACM Journal of Experimental Algorithmics* 11 (2007). ISSN: 1084-6654, 1084-6654. DOI: 10.1145/1187436.1210590. (Visited on 04/22/2024).

[Por25] Andrew Port. *Mathandy/Svgpathtools*. /mathandy/svgpathtools. 2025. (Visited on 04/18/2025).

[Pur02] Helen C Purchase. “Metrics for Graph Drawing Aesthetics”. In: *Journal of Visual Languages & Computing* 13.5 (2002), pp. 501–516. ISSN: 1045-926X. DOI: 10.1006/jvlc.2002.0232.

[Qui+09] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. “ROS: An Open-Source Robot Operating System”. In: *ICRA Workshop on Open Source Software* 3.3.2 (2009), p. 5.

[ROS24a] ROS. *Ros-Visualization/Rviz*. /ros-visualization/rviz. 2024. (Visited on 04/22/2024).

[ROS24b] ROS. *Understanding Topics — ROS 2 Documentation: Iron Documentation*. https://docs.ros.org/en/iron/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html. 2024. (Visited on 04/22/2024).

[Sam69] John W. Sammon. “A Nonlinear Mapping for Data Structure Analysis”. In: *IEEE Transactions on Computers* C-18.5 (1969), pp. 401–409. DOI: 10.1109/T-C.1969.222678.

[San24] André Santos. *Git-Afsantos/Haros*. /git-afsantos/haros. 2024. (Visited on 04/22/2024).

[Sch+20] Willy Scheibel, Matthias Trapp, Daniel Limberger, and Jürgen Döllner. "A Taxonomy of Treemap Visualization Techniques:" in: *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. SCITEPRESS - Science and Technology Publications, 2020, pp. 273–280. ISBN: 978-989-758-402-2. DOI: 10.5220/0009153902730280. (Visited on 04/07/2025).

[Sch19] Valentin Schroter. *Prototypal Construction of a Table Robot as a Future Development and Research Platform in the Area of Ambient Assisted Living and as an Alternative for the NAO-Robot*. University of Applied Science Wildau. 2019. DOI: 10.15771/BA_2019_1.

[Sch22] Valentin Schroter. *ROS-E - Ein sozialer Roboter | iCampus Wildau*. https://icampus.th-wildau.de/cms/roboticlab/projekte/ros-e-tischroboter . 2022. (Visited on 04/09/2025).

[Sch24a] Valentin Schröter. *Vschroeter/Rosmetasys*. /vschroeter/rosmetasys. 2024. (Visited on 04/02/2025).

[Sch24b] Valentin Schröter. *Vschroeter/Rosmetasys-Datasets*. /vschroeter/rosmetasys-datasets. 2024. (Visited on 04/02/2025).

[Sci25] SciPy. *SciPy*. https://scipy.org/. 2025. (Visited on 04/18/2025).

[SCM19] Andre Santos, Alcino Cunha, and Nuno Macedo. "Static-Time Extraction and Analysis of the ROS Computation Graph". In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2019, pp. 62–69. ISBN: 978-1-5386-9245-5. DOI: 10.1109/IRC.2019.00018. (Visited on 04/22/2024).

[Sha64] Marvin E. Shaw. "Communication Networks". In: ed. by Leonard Berkowitz. Vol. 1. Advances in Experimental Social Psychology. Academic Press, 1964, pp. 111–147. DOI: 10.1016/S0065-2601(08)60050-7.

[Shi+13] Chuan Shi, Yanan Cai, Di Fu, Yuxiao Dong, and Bin Wu. "A Link Clustering Based Overlapping Community Detection Algorithm". In: *Data & Knowledge Engineering* 87 (2013), pp. 394–404. ISSN: 0169023X. DOI: 10.1016/j.datak.2013.05.004. (Visited on 09/16/2024).

[Shn03] Ben Shneiderman. "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations". In: *The Craft of Information Visualization*. Interactive Technologies. Morgan Kaufmann, 2003, pp. 364–371. ISBN: 978-1-55860-915-0. DOI: 10.1016/B978-155860915-0/50046-9.

[SMK24] Kiran Smelser, Jacob Miller, and Stephen Kobourov. *"Normalized Stress" Is Not Normalized: How to Interpret Stress Correctly*. 2024. DOI: 10.48550/arXiv.2408.07724. arXiv: 2408.07724 [cs]. (Visited on 03/29/2025).

[SR14] Santiago Segarra and Alejandro Ribeiro. "Stability and Continuity of Centrality Measures in Weighted Graphs". In: *IEEE Transactions on Signal Processing* 64.3 (2014), pp. 543–555. DOI: 10.1109/TSP.2015.2486740.

[ST15] Addythia Saphala and Prianggada Indra Tanaya. “Implementation and Reconfiguration of Robot Operating System on Human Follower Transporter Robot”. In: *CommIT (Communication and Information Technology) Journal* 9.2 (2015), pp. 59–65. DOI: 10.21512/commit.v9i2.1646.

[Sto25] Razvan Stoenescu. *Quasar Framework - Build High-Performance VueJS User Interfaces in Record Time*. https://quasar.dev/. 2025. (Visited on 04/18/2025).

[STT81] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. “Methods for Visual Understanding of Hierarchical System Structures”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 11.2 (1981), pp. 109–125. ISSN: 0018-9472. DOI: 10.1109/TSMC.1981.4308636. (Visited on 04/22/2024).

[Sug02] Kozo Sugiyama. *Graph Drawing and Applications for Software and Knowledge Engineers*. Vol. 11. World Scientific, 2002. ISBN: 978-981-02-4879-6. DOI: 10.1142/4902.

[TB18] Dirk Thomas and Aaron Blasdel. *Rqt_graph - ROS Wiki*. https://wiki.ros.org/rqt_graph. 2018. (Visited on 04/22/2024).

[TDT16] Quoc-Dinh Truong, Taoufiq Dkaki, and Quoc-Bao Truong. “Graph Methods for Social Network Analysis”. In: 168 (2016), p. 286. DOI: 10.1007/978-3-319-46909-6_25.

[The25] TheejsOrg. *Three.Js – JavaScript 3D Library*. https://threejs.org/. 2025. (Visited on 04/25/2025).

[TK19] Nicole Todtenberg and Rolf Kraemer. “A Survey on Bluetooth Multi-Hop Networks”. In: *Ad Hoc Networks* 93 (Oct. 2019), p. 101922. ISSN: 15708705. DOI: 10.1016/j.adhoc.2019.101922. (Visited on 04/06/2025).

[van05] Jarke J van Wijk. “The Value of Visualization”. In: *VIS 05. IEEE Visualization* (2005), pp. 79–86. DOI: 10.1109/VISUAL.2005.1532781.

[Ven21] Dheera Venkatraman. *Introducing Rosboard: Web-based Visualizations for ROS1 and ROS2*. 2021.

[Wal+22] Markus Wallinger, Daniel Archambault, David Auber, Martin Nöllenburg, and Jaakko Peltonen. “Edge-Path Bundling: A Less Ambiguous Edge Bundling Approach”. In: *IEEE Transactions on Visualization and Computer Graphics* 28.1 (2022), pp. 313–323. DOI: 10.1109/TVCG.2021.3114795.

[WB94] Douglas R. White and Stephen P. Borgatti. “Betweenness Centrality Measures for Directed Graphs”. In: *Social Networks* 16.4 (1994), pp. 335–346. ISSN: 03788733. DOI: 10.1016/0378-8733(94)90015-9. (Visited on 04/03/2025).

[Wei+08] Weiwei Cui, Hong Zhou, Huamin Qu, Pak Chung Wong, and Xiaoming Li. “Geometry-Based Edge Clustering for Graph Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1277–1284. ISSN: 1077-2626. DOI: 10.1109/TVCG.2008.135. (Visited on 03/13/2025).

[Wel91] Emo Welzl. “Smallest Enclosing Disks (Balls and Ellipsoids)”. In: *New Results and New Trends in Computer Science*. Ed. by Hermann Maurer. Vol. 555. Springer-Verlag, 1991, pp. 359–370. ISBN: 978-3-540-54869-0. DOI: 10.1007/BFb0038202. (Visited on 02/10/2025).

[WS98] Duncan J Watts and Steven H Strogatz. “Collective Dynamics of ‘Small-World’ Networks”. In: *Nature Publishing Group* 393.6684 (1998), pp. 440–442. DOI: 10.1038/30918.

[Xu+12] Kai Xu, Chris Rooney, Peter Passmore, Dong-Han Ham, and Phong H. Nguyen. “A User Study on Curved Edges in Graph Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2449–2456. DOI: 10.1109/TVCG.2012.189. (Visited on 04/08/2025).

[Yee+01] Ka-Ping Yee, Danyel Fisher, Rachna Dhamija, and Marti Hearst. “Animated Exploration of Dynamic Graphs with Radial Layout”. In: *Proc. IEEE InfoVis 2001* (2001), pp. 43–50. DOI: 10.1109/INFVIS.2001.963279.

[Yog+21] Vahan Yoghourdjian, Yalong Yang, Tim Dwyer, Lee Lawrence, Michael Wybrow, and Kim Marriott. “Scalability of Network Visualisation from a Cognitive Load Perspective”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2021), pp. 1677–1687. ISSN: 1077-2626. DOI: 10.1109/TVCG.2020.3030459.

[You25] Evan You. *Vue.Js*. https://vuejs.org/. 2025. (Visited on 04/18/2025).

[YS16] Tetsuya Yokotani and Yuya Sasaki. “Comparison with HTTP and MQTT on Required Network Resources for IoT”. In: *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*. IEEE, 2016, pp. 1–6. DOI: 10.1109/ICCEREC.2016.7814989.

[yWo25] yWorks. *yEd Graph Editor*. https://www.yworks.com/products/yed. 2025. (Visited on 04/28/2025).

[ZH21] Antonio Zea and Uwe D. Hanebeck. “Iviz: A ROS Visualization App for Mobile Devices”. In: *Software Impacts* 8 (2021), p. 100057. ISSN: 26659638. DOI: 10.1016/j.simpa.2021.100057. (Visited on 04/22/2024).

[ZL17] Junlong Zhang and Yu Luo. “Degree Centrality, Betweenness Centrality, and Closeness Centrality in Social Network”. In: *Proceedings of the 2017 2nd International Conference on Modelling, Simulation and Applied Mathematics (MSAM2017)*. Atlantis Press, 2017. ISBN: 978-94-6252-324-1. DOI: 10.2991/msam-17.2017.68. (Visited on 02/25/2025).

[ZMC05] Shengdong Zhao, Michael J McGuffin, and Mark H Chignell. “Elastic Hierarchies: Combining Treemaps and Node-Link Diagrams”. In: *IEEE Symposium on Information Visualization* (2005). DOI: 10.1109/INFVIS.2005.1532129.

[ZPG19] Jonathan X. Zheng, Samraat Pawar, and Dan F. M. Goodman. “Graph Drawing by Stochastic Gradient Descent”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.9 (2019), pp. 2738–2748. ISSN: 1077-2626, 1941-0506, 2160-9306. DOI: 10.1109/TVCG.2018.2859997. (Visited on 04/27/2025).

Appendix A

Supplemental Material

A.1 Evaluation Results

The plots in Figure 5.3 contain the combined evaluation results for 9 different metrics. The following Figure A.1 shows the same evaluation results, but with the metrics being separated for real-world and synthetic datasets.

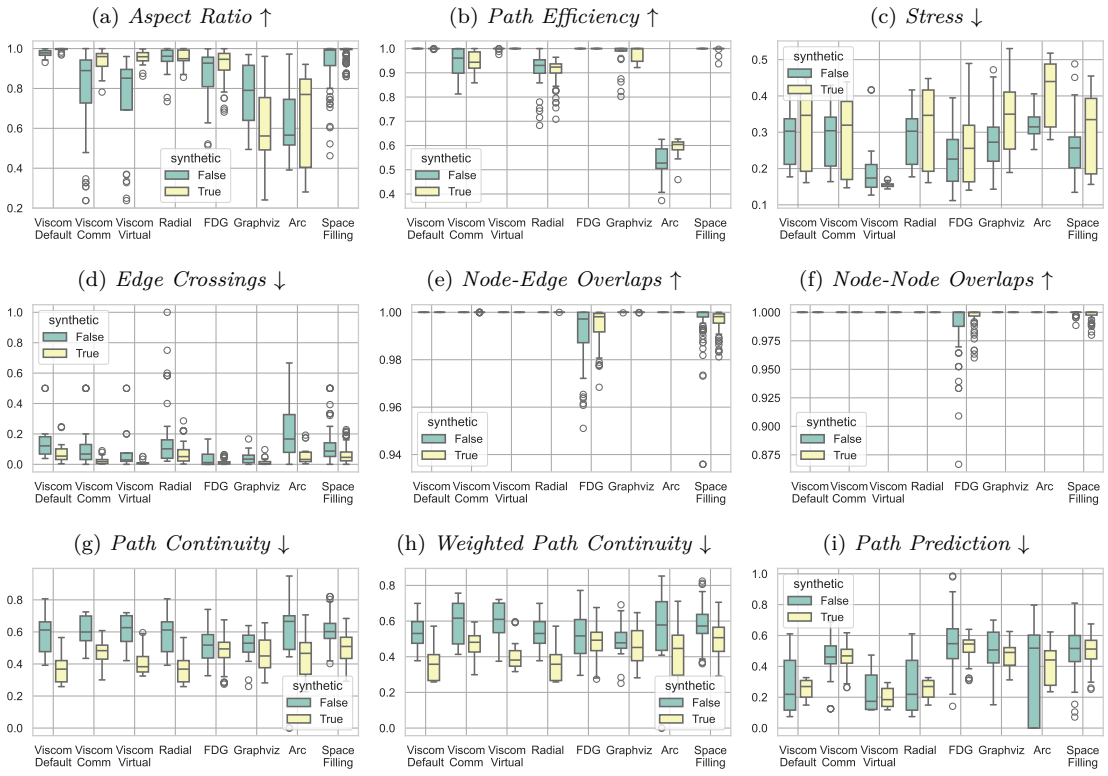


Figure A.1: *Evaluation results for 9 different metrics, each illustrated separately for real-world and synthetic datasets. The different layout algorithms have been tested on 18 datasets (8 real-world datasets and 10 synthetic datasets) with system sizes between 6 and 150 nodes.*

A.2 Detailed Node Scores

Table A.1: *Node score calculation methods for the graph used in Figure 4.3, each calculated for the graph without the broadcast node B in column w/o B, and for the graph with the broadcast node B in column w/ B. Our proposed centrality measures are Communication Path Centrality (**CPC**) from Equation (4.9) and the Harmonic Communication Centrality (**HCC**) from Equation (4.8).*

Centrality Method	D2		p3		p7		D1		p2		p1		M		p6		p5		p4		S		B
	w/o B	w/ B	w/o B	w/ B	w/o B	w/ B	w/o B	w/ B	w/o B	w/ B	w/o B	w/ B	w/o B	w/ B	w/o B	w/ B	w/o B	w/ B	w/o B	w/ B	w/o B	w/ B	w/ B
Betweenness	0.00	0.00	0.11	0.03	0.05	0.01	0.00	0.00	0.24	0.06	0.13	0.03	1.00	0.26	0.09	0.02	0.09	0.02	0.09	0.02	0.00	0.00	1.00
Closeness	1.00	0.86	0.62	0.63	0.11	0.42	0.72	0.69	0.62	0.63	0.62	0.63	0.70	0.58	0.11	0.42	0.11	0.42	0.11	0.42	0.00	0.39	1.00
CPC	**0.32**	**0.31**	**0.40**	**0.40**	**0.34**	**0.33**	**0.22**	**0.21**	**0.50**	**0.49**	**0.43**	**0.42**	**1.00**	**1.00**	**0.45**	**0.45**	**0.45**	**0.45**	**0.45**	**0.45**	**0.15**	**0.16**	**0.12**
HCC	**0.55**	**0.55**	**0.58**	**0.57**	**0.43**	**0.42**	**0.37**	**0.37**	**0.60**	**0.61**	**0.58**	**0.57**	**1.00**	**1.00**	**0.56**	**0.56**	**0.56**	**0.56**	**0.56**	**0.56**	**0.26**	**0.27**	**0.16**
Degree	0.50	0.60	0.33	0.58	0.33	0.58	0.33	0.58	0.50	0.60	0.33	0.58	1.00	0.67	0.33	0.58	0.33	0.58	0.33	0.58	0.67	0.63	1.00
Harmonic	0.00	0.13	0.14	0.25	0.27	0.37	0.00	0.13	0.27	0.36	0.14	0.25	1.00	1.00	0.82	0.82	0.82	0.82	0.82	0.82	0.55	0.53	0.23
Katz	0.60	0.60	0.70	0.67	0.71	0.70	0.60	0.60	0.80	0.73	0.70	0.67	1.00	0.95	0.78	0.76	0.78	0.76	0.78	0.76	0.94	0.90	1.00
Laplacian	0.41	0.72	0.40	0.72	0.38	0.73	0.41	0.70	0.48	0.77	0.43	0.73	0.83	0.99	0.43	0.75	0.43	0.75	0.43	0.75	1.00	0.93	1.00
Pagerank	0.12	0.43	0.16	0.48	0.16	0.51	0.12	0.43	0.21	0.55	0.18	0.50	0.59	0.87	0.29	0.58	0.29	0.58	0.29	0.58	1.00	0.87	1.00
Percolation	0.00	0.00	0.11	0.03	0.05	0.01	0.00	0.00	0.24	0.06	0.13	0.03	1.00	0.26	0.09	0.02	0.09	0.02	0.09	0.02	0.00	0.00	1.00

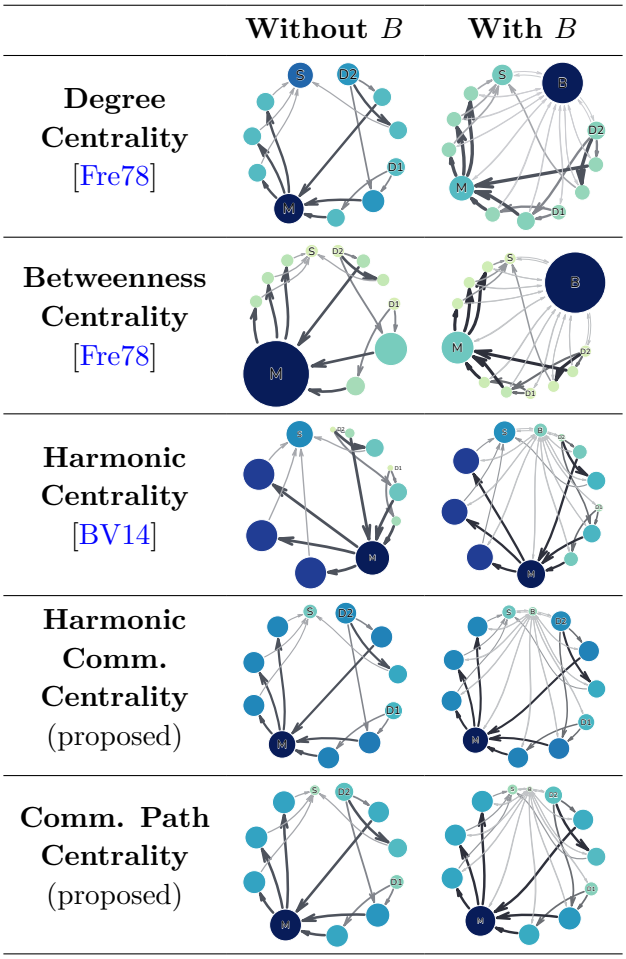
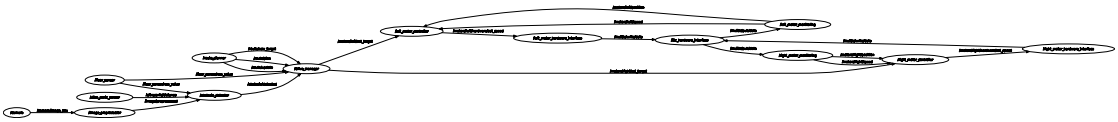
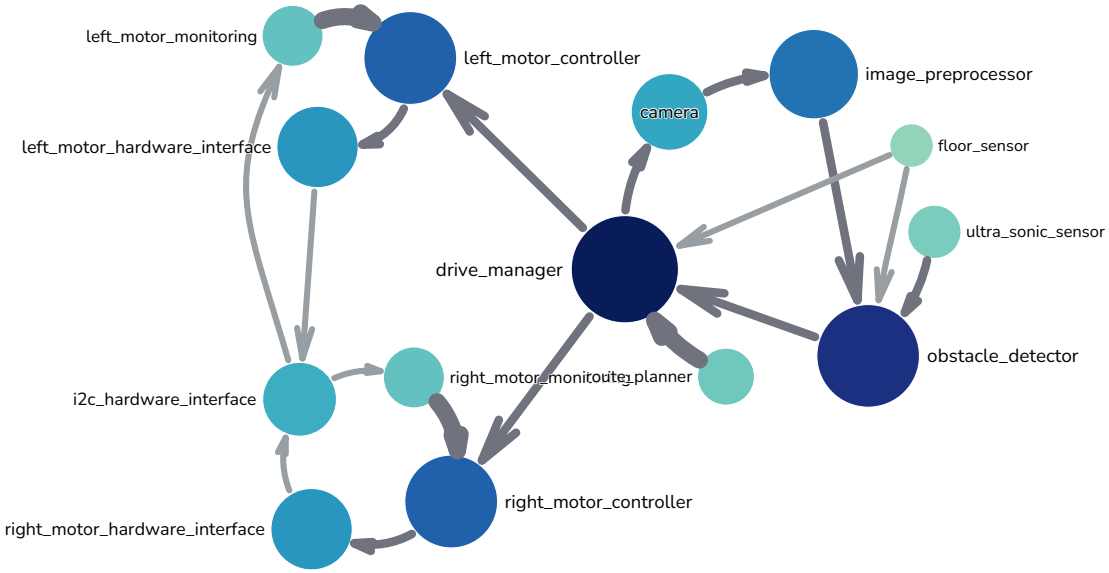


Figure A.2: *The same figure as Figure 4.3, visualizing different node centrality calculation methods.*

A.3 Comparison of *VisCom* and *rqt_graph*

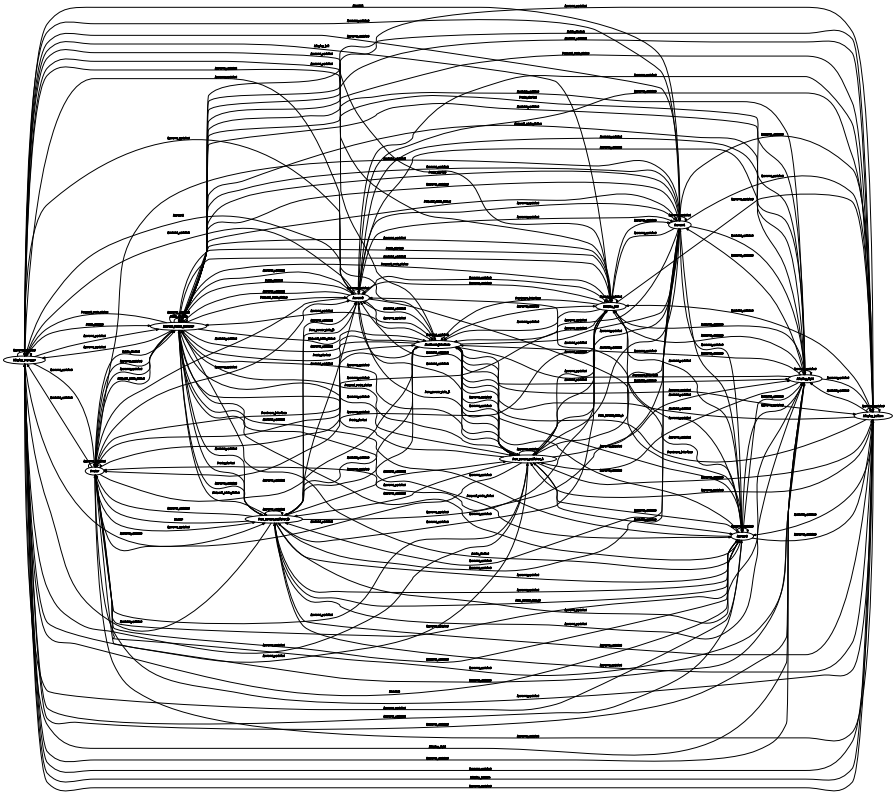


(a) *rqt_graph*

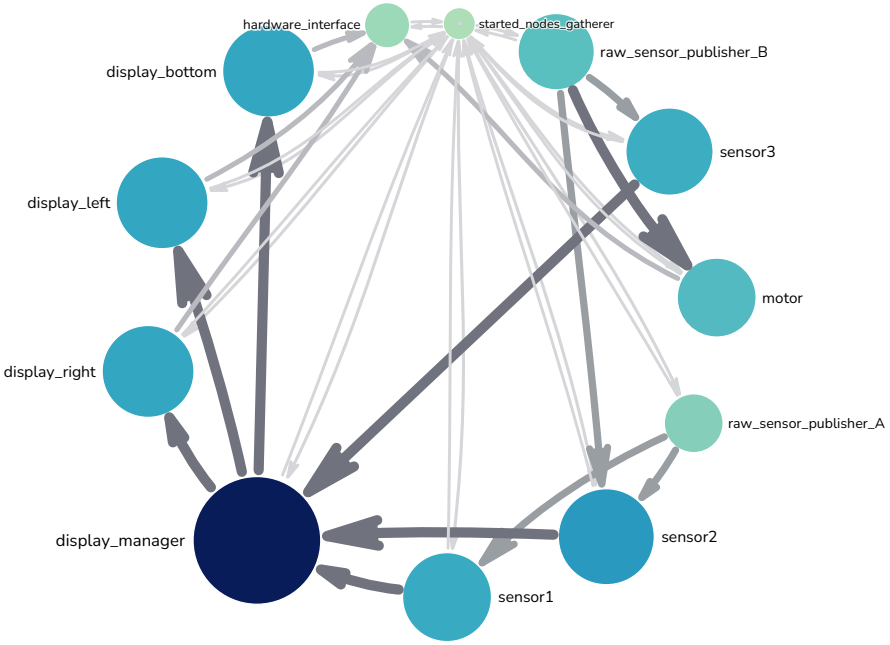


(b) *VisCom with Community Detection*

Figure A.3: *Comparison of rqt_graph and VisCom with community detection (proposed method). System with 14 nodes.*

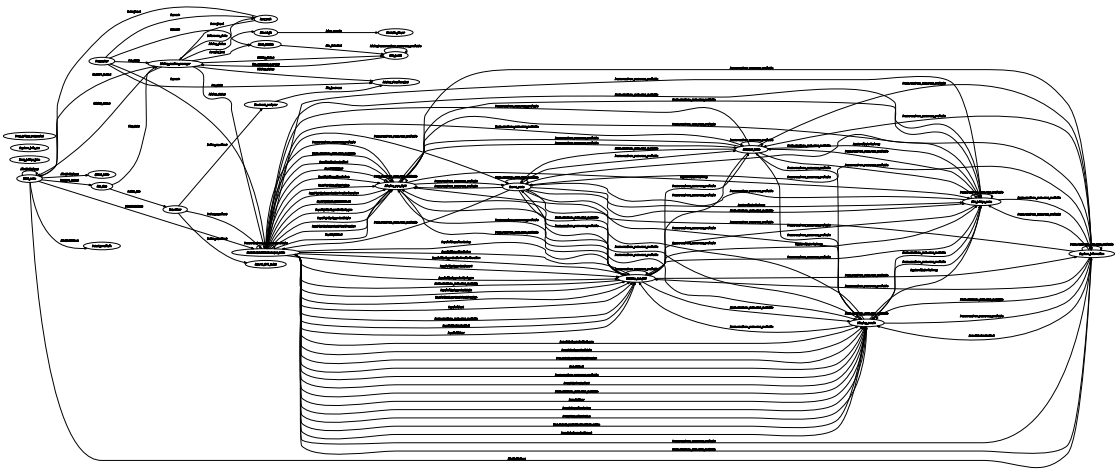


(a) *rqt_graph*

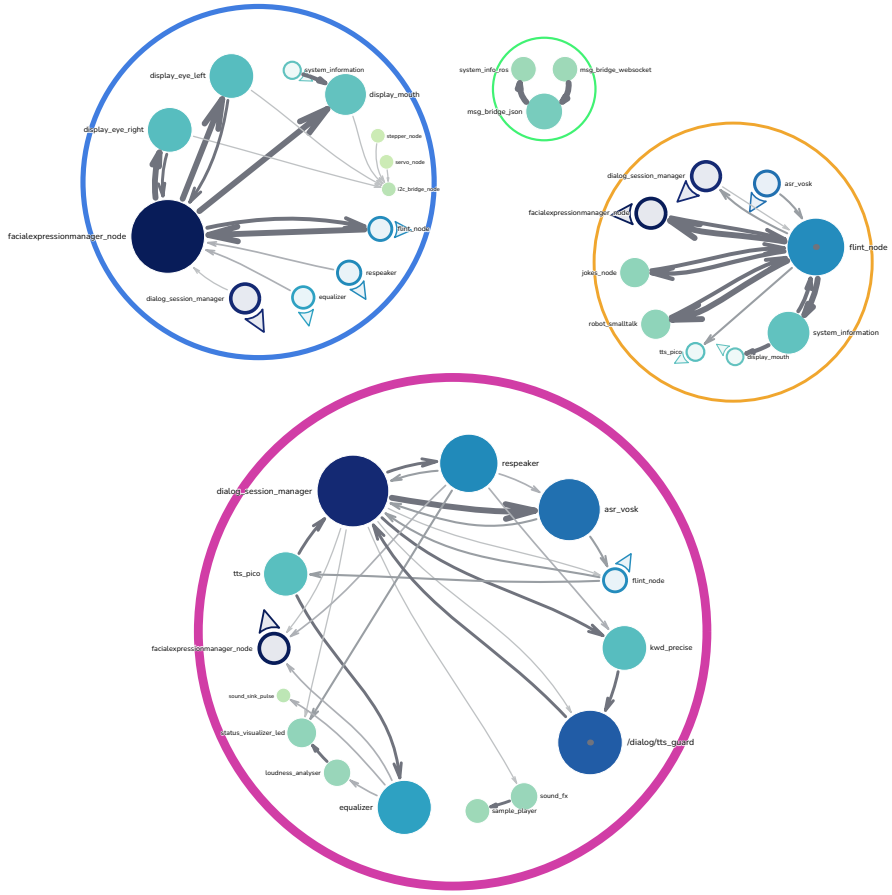


(b) *Default VisCom*

Figure A.4: *Comparison of rqt_graph and VisCom (proposed method). System with 12 nodes and broadcast characteristics.*



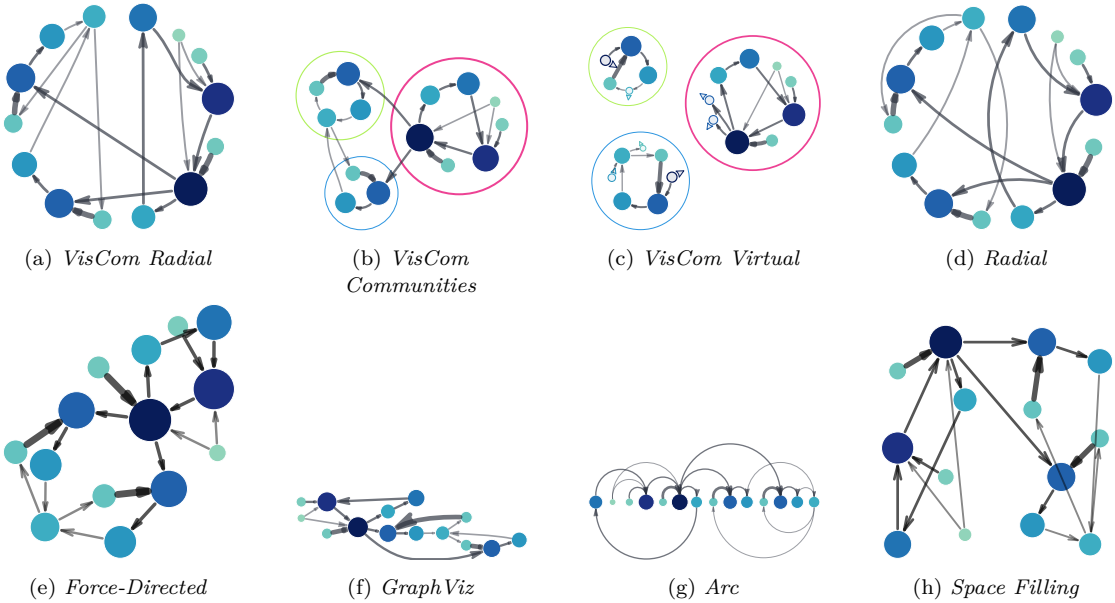
(a) *rqt_graph*



(b) *VisCom with Virtual Nodes*

Figure A.5: *Comparison of rqt_graph and VisCom with Virtual Nodes (proposed method). Complex system with 26 nodes. In VisCom, 12 extra Virtual Nodes are added to the graph.*

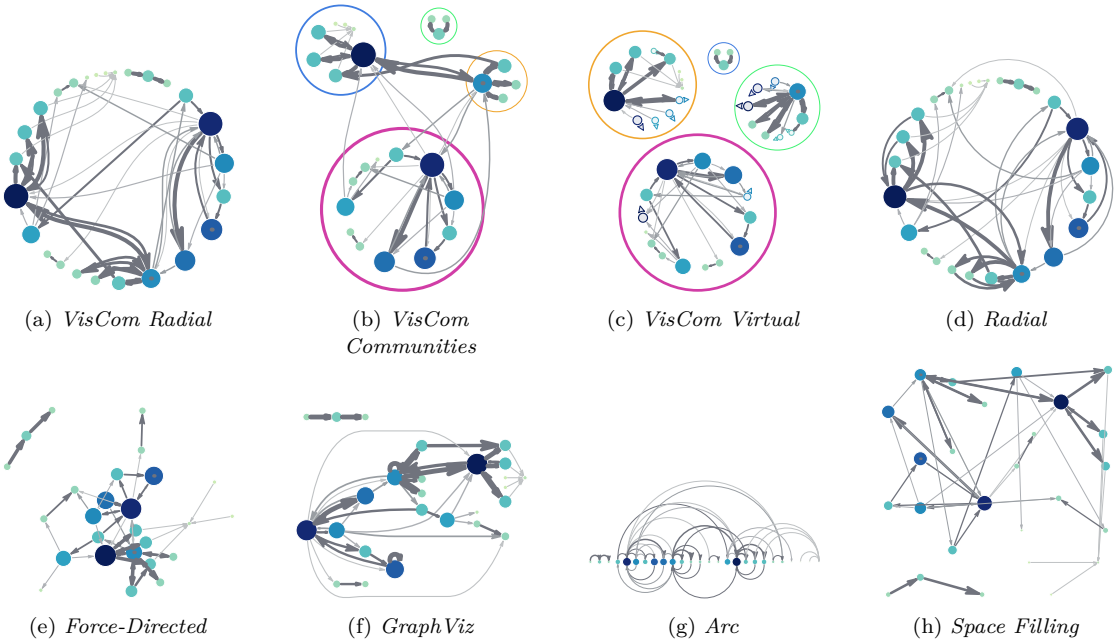
A.4 Layout Gallery



(a) *VisCom Radial* (b) *VisCom Communities* (c) *VisCom Virtual* (d) *Radial*

(e) *Force-Directed* (f) *GraphViz* (g) *Arc* (h) *Space Filling*

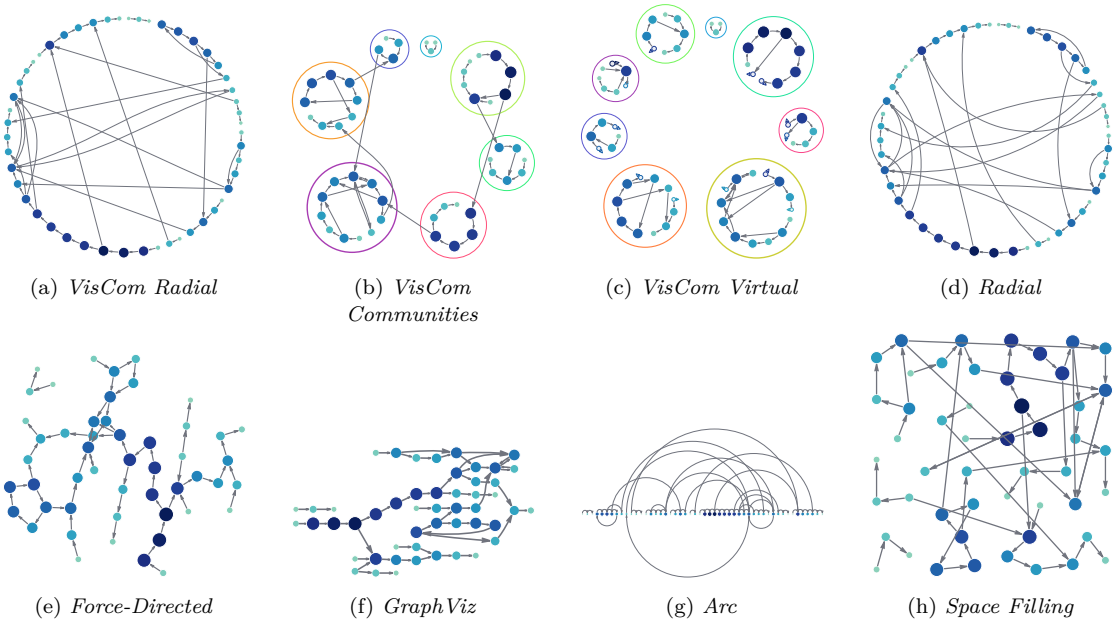
Figure A.6: *A graph with 14 nodes and 18 edges visualized with the currently implemented layout algorithms in the evaluation tool.*



(a) *VisCom Radial* (b) *VisCom Communities* (c) *VisCom Virtual* (d) *Radial*

(e) *Force-Directed* (f) *GraphViz* (g) *Arc* (h) *Space Filling*

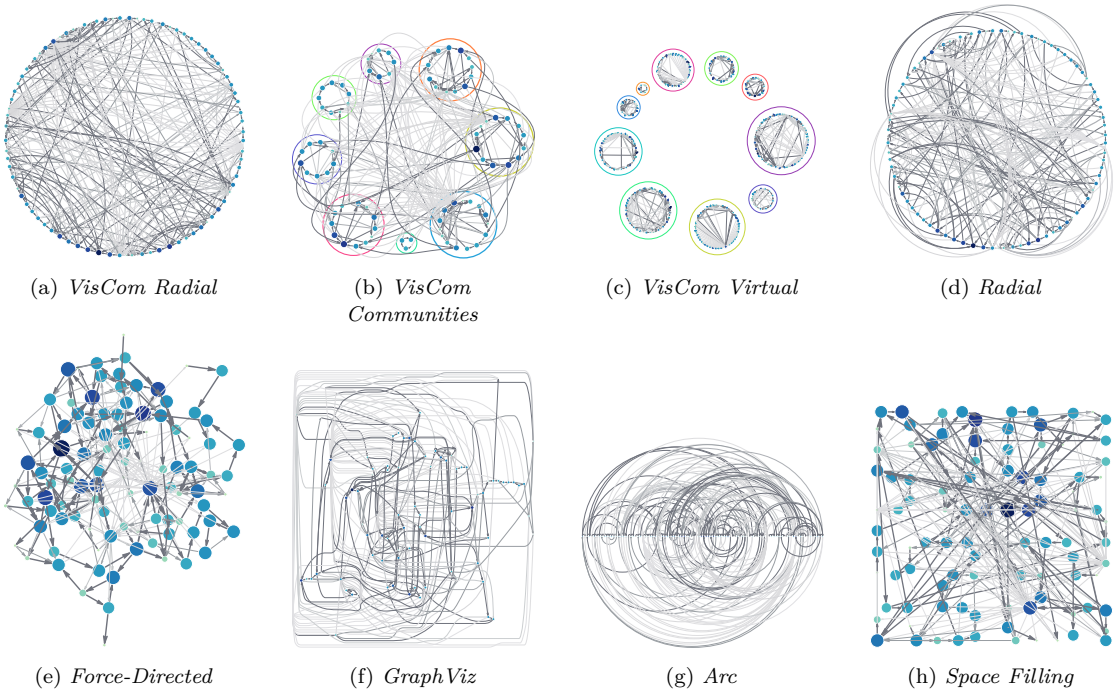
Figure A.7: *A graph with 26 nodes and 49 edges visualized with the currently implemented layout algorithms in the evaluation tool.*



(a) *VisCom Radial* (b) *VisCom Communities* (c) *VisCom Virtual* (d) *Radial*

(e) *Force-Directed* (f) *GraphViz* (g) *Arc* (h) *Space Filling*

Figure A.8: *A graph with 50 nodes and 55 edges visualized with the currently implemented layout algorithms in the evaluation tool.*



(a) *VisCom Radial* (b) *VisCom Communities* (c) *VisCom Virtual* (d) *Radial*

(e) *Force-Directed* (f) *GraphViz* (g) *Arc* (h) *Space Filling*

Figure A.9: *A graph with 100 nodes and 198 edges visualized with the currently implemented layout algorithms in the evaluation tool.*