Network Dynamics and Influence Patterns in the ZackRawrr YouTube Ecosystem

By: Artopoulos Georgios (t8200016)

Course: Social Network Analysis

Coursework: Analysis of a network

Option 2 – Analyze a Youtube Channel Network

Supervising professor: Pournarakis Dimitrios

Table of Contents

1. Introduction

In today's digital age, YouTube is a significant platform where people come together to create and share videos. It's like a vast online community where content creators connect, exchange ideas, and influence one another. This research focuses on Zackrawrr, a YouTube channel that stands out because it offers a wide range of videos, including gaming, personal stories, and community interactions.

Zackrawrr is actually the secondary channel of Zack Hoyt, also known as Asmongold. Asmongold is a popular content creator in the World of Warcraft community and co-owns the gaming organization OTK (One-True-King). The variety of content the creator produces and the personal relationships he has forged within the creator community come together to create an ideal environment for building a diverse network around the Zackrawrr channel.

For the extraction of this network, the tools provided by Bernhard Rieder in the YouTube Data Tools modules were used. It is also important to note that the data was collected at crawl depth 1 and with the subscriptions box checked meaning that both subscription to other channels as well as features were used. In this network, each point (or node) represents a different YouTube channel. The lines (or edges) between these points show how these channels are connected. These connections could be because one channel subscribes to another, or they might be listed together in the 'Creators' section on a YouTube channel’s page.

To analyze this network, we will employ a range of tools, each chosen for its specific strengths in handling different aspects of network analysis. The primary tool for visualizing and analyzing the network will be Gephi, an open-source platform known for its powerful capabilities in graph visualization and exploration. Gephi's intuitive interface and rich set of features make it ideal for representing complex network structures and patterns in a visually accessible manner.Complementing Gephi, we will use NetworkX, a Python library designed for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. NetworkX excels in handling intricate calculations and network algorithms that are beyond Gephi's default capabilities. This combination of Gephi and NetworkX allows us to leverage the strengths of both platforms - the advanced visualization capabilities of Gephi and computational power of NetworkX.

This study has two main goals: Firstly,who are the central creators of the network and how do they influence the spread of information within this YouTube community? Secondly how does a channel’s relatively high subscriber count (more than a million) impact its centrality and connectivity among other content creators?

By looking closely at ZackRawrr and its network, we hope to learn more about how YouTube channels influence each other and form communities. This isn't just about

ZackRawrr; it's about understanding the bigger picture of how information and trends spread in the world of YouTube.

2. Data Overview

After the data has been extracted using the YouTube data tools modules by Bernhard Rieder the file was imported to Gephi where a first view of the raw data was available throught the data laboratory tab.

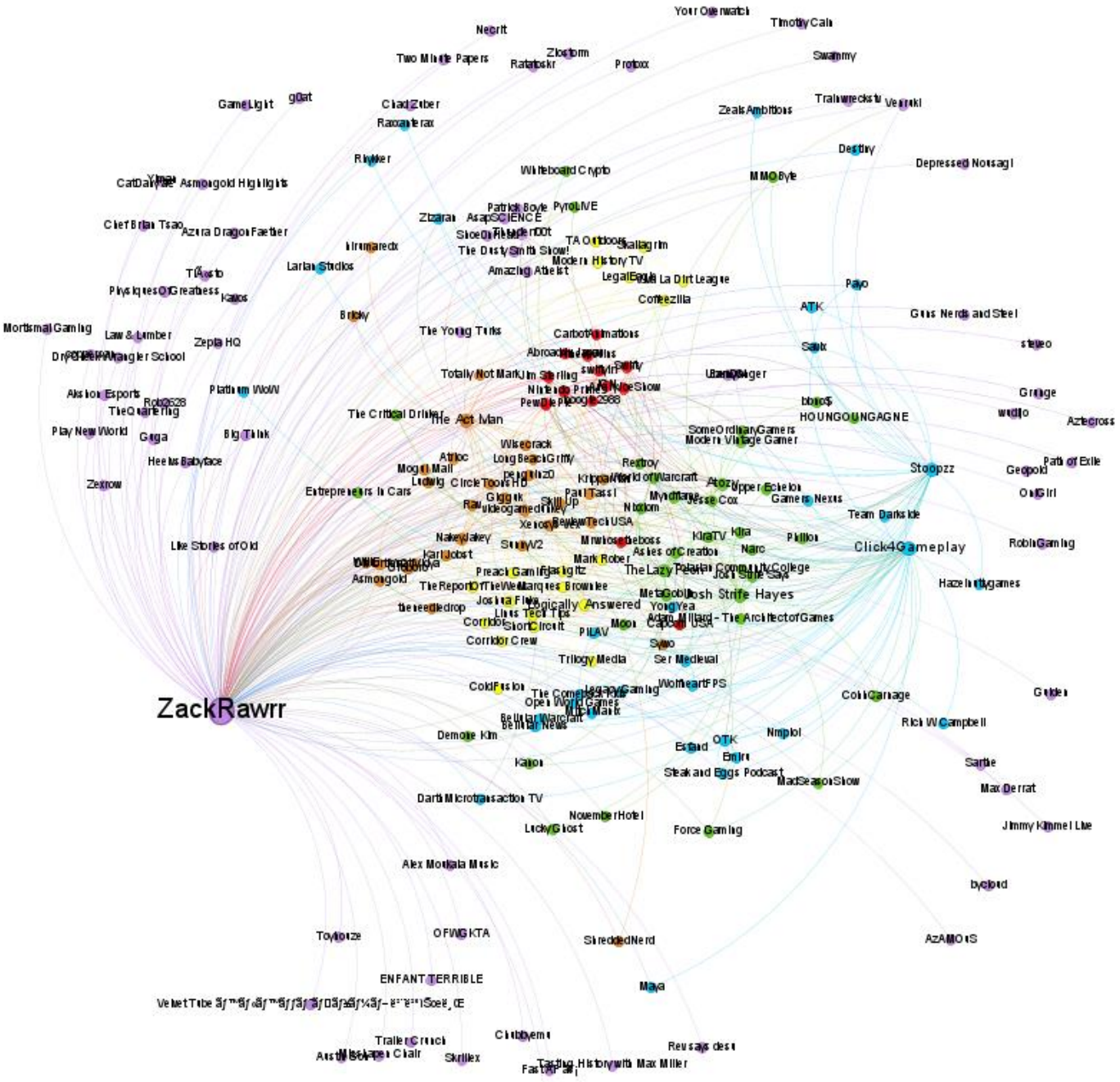
The nodes table provides a detailed snapshot of various YouTube channels, capturing key performance and identification metrics. The columns in the table include 'Id', 'Label', 'Timestamp', 'Interval', 'Isseed', 'Seedrank', 'Subscribercount', 'Videocount', 'Viewcount(100s)', 'Country', 'Publisheddat', and 'Daysactive'. Specifically, the 'Id' column assigns a unique identifier to each channel, while the 'Label' column gives the name of the channel. The 'Subscribercount' shows the number of subscribers, and the 'Videocount' indicates how many videos the channel has posted. 'Viewcount(100s)' represents the total video views in hundreds, providing a sense of the channel's reach and popularity. 'Country' identifies the channel's country of origin, 'Publisheddat' refers to the channel's creation date, and 'Daysactive' reflects the total duration the channel has been operational on YouTube.

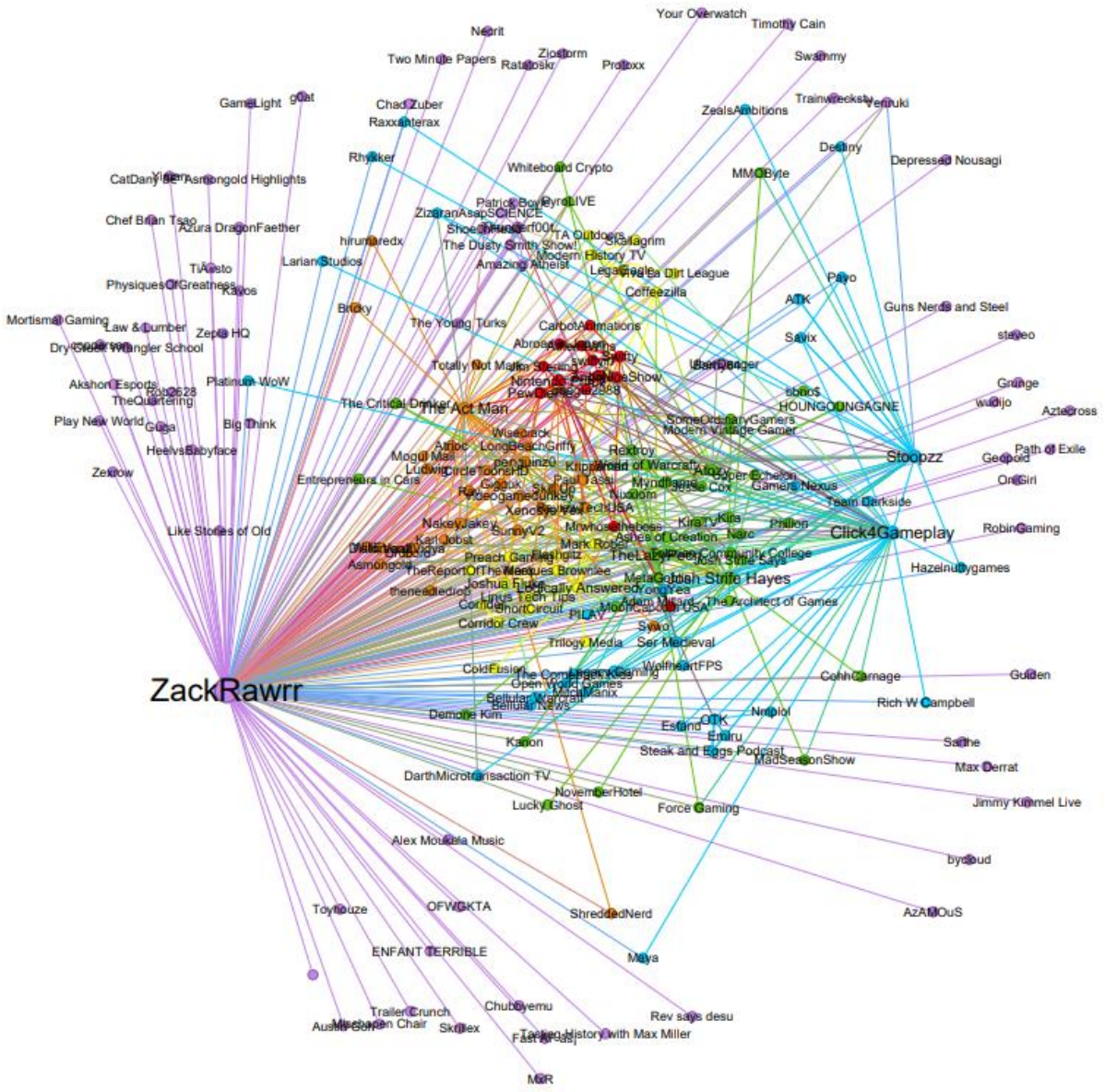
Id	Label	Timestamp	interval	isseed	seedrank	subscribercount	videocount	viewcount(100s)	country	publisheddat	daysactive
UCNnKprAG-MWLsk...	Flashgitz			no		3700000	166	7460557	US	<[2011-03-08T20:08:...	4688
UCpDcj6E73QeFkmq...	ZealsAmbitions			no		377000	568	582804	KR	<[2011-03-11T08:17:...	4686
UCMiJRAwDNSNzuY...	Mrwhosetheboss			no		17600000	1647	48438952	GB	<[2011-04-20T05:10:...	4646
UChYDO3y5H14ow7...	swiftyirl			no		229000	840	656177	not set	<[2011-06-03T06:22:...	4602
UCfwJBTwTgdCj5IH...	ShreddedNerd			no		78300	118	146269	AU	<[2011-07-29T04:06:...	4546
UChpiQy7AuKxVAd...	HOUNGOUNGAGNE			no		726000	540	1449452	not set	<[2011-08-13T21:45:...	4530
UCrPseYLGpNygVi34...	Ludwig			no		5590000	1714	19589624	US	<[2011-08-15T04:32:...	4529
UCchBatdUMZoMfJ3...	Viva La Dirt League			no		6130000	1800	31985851	NZ	<[2011-08-21T01:29:...	4523
UCwiaPYufmQOq5F...	Bellular Warcraft			no		625000	2848	2801825	GB	<[2011-09-17T16:38:...	4496
UCTAfm-YD2M9xzv...	Timothy Cain			no		78500	252	40605	US	<[2011-10-09T20:13:...	4473
UCW3WAb1Empmqf...	Yiman			no		83700	266	324340	US	<[2011-10-15T15:33:...	4468
UCf8BuPKoB2xLIIVg...	Rextroy			no		104000	516	309950	SE	<[2011-10-17T12:27:...	4466
UCY1kMZp36IQSyNx...	Mark Rober			no		29300000	137	46662072	US	<[2011-10-19T23:17:...	4463
UC84PVGjO0Q5rgzK...	MitchManix			no		104000	118	176842	GB	<[2011-10-29T09:59:...	4454
UCi3RdvCWYMzdF-i...	WolfheartFPS			no		209000	884	622706	US	<[2011-11-10T06:50:...	4442
UCCiBMTQv7l99hglz...	Chad Zuber			no		2340000	394	3110894	US	<[2011-12-01T21:03:...	4420
UCh0wrs-8ywjgZO7f...	PhysiquesOfGreatness			no		681000	862	1379076	not set	<[2011-12-12T17:32:...	4410
UCa_TKzLdsJSmwnX...	Azura DragonFaether			no		72600	723	55246	US	<[2012-01-04T00:37:...	4387
UCrrSA2uXHnFom2...	Barny64			no		984000	361	3762064	GB	<[2012-01-25T13:34:...	4366
UCT6iAerLNE-0J1S_E...	YongYea			no		1220000	3781	6300368	US	<[2012-01-26T10:18:...	4365
UCe0DNp0mKMqrY...	VaatiVidya			no		2770000	358	5898670	AU	<[2012-03-01T14:53:...	4330
UCO7dBj4bwaDOMo...	Atozy			no		1470000	832	3352807	US	<[2012-03-09T07:01:...	4322
UCtMVHI3AJD4Qk4h...	SomeOrdinaryGamers			no		3640000	2908	10694663	CA	<[2012-03-15T21:45:...	4315
UCHL9bfHTxCMi-7v...	Abroad in Japan			no		2970000	282	4830945	not set	<[2012-04-07T09:34:...	4293
UC_zmPqauxanOAG...	HeelvsBabyface			no		359000	3790	1552200	GB	<[2012-04-11T06:31:...	4289
UCC552Sd-3nyi_tk2B...	AsapSCIENCE			no		10500000	477	18803903	not set	<[2012-05-28T10:33:...	4242
UCZRoNJu1OszFqAB...	Logically Answered			no		548000	935	1141216	US	<[2012-08-25T15:17:...	4153
UC1uug_uZrVmylfPV...	CarbotAnimations			no		1770000	945	7937921	CA	<[2012-08-28T12:08:...	4150
UCcKSEzhtEkzfCT8_D...	KiraTV			no		481000	597	997121	GB	<[2012-10-13T21:55:...	4103
UCc0AC8pAKI69wq7...	CatDany â€" Asmon...			no		130000	328	792160	not set	<[2012-12-12T04:50:...	4044
UCV6g95OBbVtFmN...	CircleToonsHD			no		2600000	369	7194105	US	<[2012-12-14T19:09:...	4041
UC7WDD6yHgzdqijH...	The Act Man			no		1850000	319	4062959	US	<[2013-01-21T23:54:...	4003
UCMGVp_GnkhHZR...	Hazelnuttygames			no		327000	1981	863112	CA	<[2013-05-29T13:15:...	3876
UC6-ymYjG0SU0jUW...	Wisecrack			no		3140000	1011	5535784	US	<[2013-06-03T15:58:...	3871
UC1kvvdjChB40wuzc...	Asmongold			no		347000	919	814286	not set	<[2013-06-03T18:58:...	3870
UCp9TXGvv2-7JVVdy...	CohhCarnage			no		379000	39129	3631613	US	<[2013-06-06T23:10:...	3867

The edges table depicts the edge list the directed network graph, where each row represents a connection between two nodes. The 'Source' column identifies the starting point of an edge, while the 'Target' column indicates where the edge is directed. The 'Type' column confirms that all relationships are directed, meaning they have a specific orientation from source to target. Each edge is uniquely identified by an 'Id'. The 'Weight' of each edge is given, with all visible weights set to 1.0.

Source	Target	Type	Id	Label	Weight
UCq2jigrlGtupbTXiNjq6Wrw	UC1kvvdjChB40wuzcynmgsCw	Directed	0		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC-91UA-Xy2Cvb98deRXuggA	Directed	1		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC-9C9d1qDDzA2jdH53vSluA	Directed	2		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC-SuvtOypiomM5jX6kJ_1Qw	Directed	5		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC-XXmB9lHCUdteHjgmmsaeg	Directed	7		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC-lHJZR3Gqxm24_Vd_AJ5Yw	Directed	8		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC0aanx5rpr7D1M7KCFYzrLQ	Directed	9		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC1haxYclmhXwa4FKFqYSaRw	Directed	14		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC1uug_uZrVmyIfPVBLBvitQ	Directed	15		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC1yBKRuGpC1tSM73A0ZjYjQ	Directed	16		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC2SMpy2oZV6BoyJEYShw9bw	Directed	20		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC3WlohkLkH4GFoMrrWVZZFA	Directed	22		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC3ltptWa0xfrDweghW94Acg	Directed	24		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC3nPaf5MeeDTHA2JN7clidg	Directed	25		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC4MGZcDG5hnzpi3hDpORkXw	Directed	28		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC4OWLYVuwj55RzgmBbY2afg	Directed	29		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC4QZ_LsYcvcq7qOsOhpAX4A	Directed	30		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC4YSogGVXHZOOBbavfzn9rA	Directed	31		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC4_bwov47DseacR1-ttTdOg	Directed	32		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC554eY5jNUfDq3yDOJYirOQ	Directed	34		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC5wZGwqyPawfmzpve8aJ-hw	Directed	38		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC6-ymVjG0SU0jUWnWh9ZzEQ	Directed	39		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC6Yz9qyayyrYpo_tkc8V14Q	Directed	40		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC6dl8msF1_l9HuDSvMtk_nQ	Directed	41		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC6idiEhlPEreqzCWfQ_GiLw	Directed	42		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC6y5h-dW-A6RBRHXDPlj9Rg	Directed	45		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC7-hR5EfgpM6oHfiGDkxfMA	Directed	46		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC73Us1H9hxv__5oobyLDOyw	Directed	47		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC78d2aaqdrt02RfmtRLmlAw	Directed	49		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC7WDD6yHgzdqijHluCi1z-Q	Directed	50		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC7dF9qfBMXrSlaaFFDvV_Yg	Directed	51		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC7uHWFLx-zVkHxGfA4dbdFw	Directed	52		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC84PVGjO0Q5rgzKXM5aAtvQ	Directed	53		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UC97JZwjhKFJ3HwrGkcJekSw	Directed	57		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UCA7X5unt1JriVReQDUbI_A	Directed	62		1.0
UCq2jigrlGtupbTXiNjq6Wrw	UCAG3CiKOUkQysvKCXSFEBPA	Directed	64		1.0

3. Network Graphical Presentation





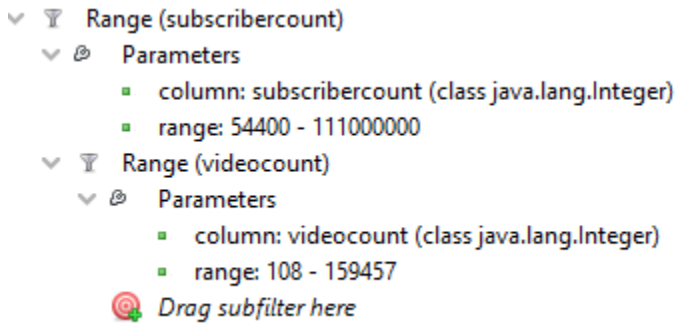
The graphical representation of the Zackrawrr network is designed to illustrate the connections within its structure. This network is depicted as directed and unweighted, with the directed edges signifying the directional flow of relationships between each node. These connections represent whether a channel is subscribed to another or features it in its 'Creators' tab. The choice of unweighted edges comes from the ambiguity in discerning whether a link represents a subscription or is a mention in the featured channels tab. However, should this differentiation become clear, assigning weights to these edges would be advantageous, as certain types of connections, like being featured in a channel's tab, might carry more significance over a simple

subscription. The visualization employs the Force Atlas 2 layout from Gephi, which effectively balances node distribution and edge length, providing a clear view of the network's structure. In this layout, the size of each node is determined by its degree attribute, reflecting the quantity of its incoming and outgoing connections, with a range from a minimum size of 20 to a maximum of 50. This sizing criterion ensures that the most connected nodes, or those with the highest number of inwards and outwards edges, are prominently displayed, thereby highlighting the key influencers or hubs within the Zackrawrr network. The colour of the nodes has to do with each node's modularity class which is something that will be discussed later on.

4. Basic Topological Properties

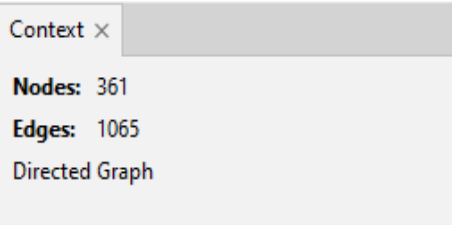
4.1 Number of Nodes and edges

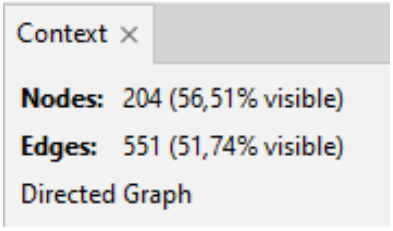
Before starting the basic topology analysis of the network, it's important to mention the use of Gephi's filters. They were used to exclude channels that could potentially distort our analysis results, such as channels that are inactive or channels that have either a subscriber count or video count of 0. This was achieved by eliminating smaller channels that did not meet certain thresholds, namely those with a subscriber count below 50,000 and a video count under 100. By doing this, the analysis now focuses more on channels that are well-established, giving a clearer understanding of the network's main behaviors.



So, as a result there is a decrease in the number of nodes and edges that are a part of the final graph. The starting number of nodes and edges were 361 and 1065 accordingly according to Gephi, but in the final network only 56.51% of the initial nodes and 51.74% and initial edges. So, the final count of nodes and edges comes to 204 nodes and 551 edges.

Which can be seen here:





These results are also confirmed by NetworkX for both the original and the final network.

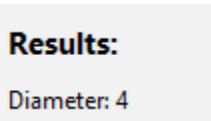
```
In 15
import networkx as nx

#import the gephi files
original= nx.read_graphml('Original.graphml')
final= nx.read_graphml('Final.graphml')

#find the number of nodes and edges in each graph
print("Original Graph")
print(original.number_of_nodes())
print(original.number_of_edges())

print("Final Graph")
print(final.number_of_nodes())
print(final.number_of_edges())
Executed at 2024.01.15 00:38:26 in 49ms

Original Graph
361
1065
Final Graph
204
551
```

Moving forward only the final graph will be analyzed.

4.2 Network Diameter

Network diameter refers to the greatest distance between any pair of nodes in a network. More formally, it is the longest shortest path between any two nodes in the network. A small diameter indicates that the network is tightly connected, meaning that information or whatever is flowing through the network can spread quickly. A larger diameter suggests a more sparsely connected network, where it takes more steps to go from one part of the network to another.

Using Gephi's imbedded Network Diameter function we get:



Now proceeding to calculate the diameter using NetoworkX:

```
In 6   1  #calculate the the network diameter for the final graph
       2  nx.diameter(final)
          Executed at 2024.01.14 17:12:50 in 54ms

          > Traceback…
            NetworkXError: Found infinite path length because the digraph is not strongly connected
```

From this error message we understand that the graph is not strongly connected, which means there are at least two nodes in the graph such that no path exists from one to the other in the direction respecting the edges. The error will be solved by calculating the **Diameter of the Largest Strongly Connected Component.** The largest strongly connected component (SCC) is the biggest subgraph where there is a directed path from every node to every other node within the subgraph.

From this we get a diameter of 4:

```
In 13  1  largest_scc = max(nx.strongly_connected_components(final), key=len)
       2  subgraph = final.subgraph(largest_scc)
       3  diameter = nx.diameter(subgraph)
       4  diameter
          Executed at 2024.01.14 17:28:22 in 5ms

            4
```

So both methods give a diameter of 4. This means that the network in the is relatively compact. In practical terms, it means that if a shortest path exists, the furthest any two nodes can be from each other, in terms of the number of steps (edges) along the shortest path connecting them, is four steps.

4.3 Average Path Length

The average path length in a network is a measure of the average number of steps along the shortest paths for all possible pairs of network nodes. It's a way of quantifying how interconnected the nodes in a network are, on average.

Using gephi the average path length of the network is 2.64.

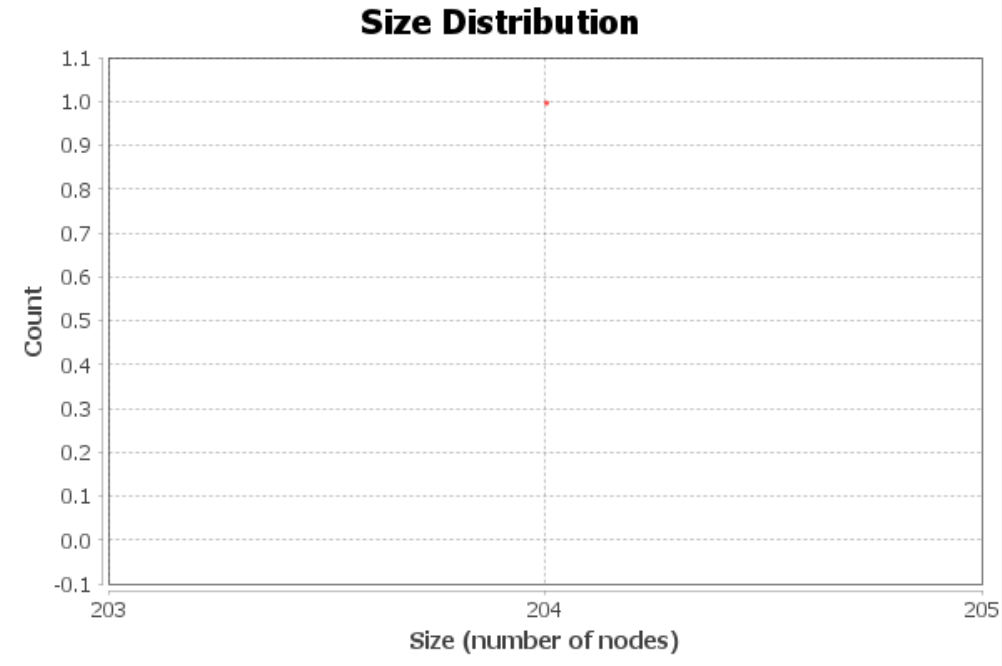
Connected Components Report

Parameters:

Network Interpretation: directed

Results:

Number of Weakly Connected Components: 1
Number of Strongly Connected Components: 176



While using NetworkX,

```
In 16  1  #calculate the average path length
       2  nx.average_shortest_path_length(final)
          Executed at 2024.01.15 00:49:16 in 62ms

          > Traceback…
            NetworkXError: Graph is not strongly connected.

In 14  1  #calculate the average path length for the final graph using the largest strongly connected component
       2  nx.average_shortest_path_length(subgraph)
          Executed at 2024.01.14 17:28:23 in 8ms

            2.466403162055336
```

The same error about weak connectivity is encountered so to get a result, the average path of the **Largest Strongly Connected Component** is calculated, which gives a result of 2.466.

Smaller average path length, such as 2.634 and 2.466 means that, on average, the nodes are closer to each other, indicating a more tightly connected network. This can imply efficient communication or quick spread of information or resources within the network.

5. Component Measures

5.1 Number of Connected Components

The number of components in a network refers to the count of distinct connected subgraphs within the entire graph. In the context of network analysis, these components are groups of nodes that are connected to each other, but not connected to nodes in other components. The concept varies slightly between undirected and directed graphs:

In Undirected Graphs: A component is a set of nodes such that each node is reachable from any other node in the same set by traversing edges, irrespective of their direction. The number of components in an undirected graph is the count of these distinct groups.

In Directed Graphs the concept splits into two:

Strongly Connected Components: A strongly connected component is a subset of nodes where every node is reachable from every other node in the same subset, following the direction of the edges.

Weakly Connected Components: A weakly connected component treats the edges as undirected for connectivity purposes. If you can reach every node from every other node (ignoring edge direction), those nodes are part of the same weakly connected component.

The ZackRawrr network is a directed graph, so it is necessary to analyze both strongly and weakly connected components.

Using the Connected Components tool from Gephi, there are 176 strongly connected components and 1 weakly connected component.

Connected Components Report

Parameters:

Network Interpretation: directed

Results:

Number of Weakly Connected Components: 1
Number of Strongly Connected Components: 176

These results are also confirmed by NetworkX:

This implies a significant level of fragmentation in terms of directed interactions, indicating that while there are many small, tightly-knit groups with mutual reachability, the overall direct interaction across the entire network is limited. However, the presence of only 1 weakly connected component reveals that if the directionality of connections is ignored, the network is entirely interconnected, suggesting a broad, underlying interconnectedness.

5.2 Existence of a Giant Component

A "giant component" in the context of network theory refers to a significantly large connected subgraph (or component) within a given network, especially in comparison to other components in the same network.

With Gephi's tools, it is evident that there is one giant component

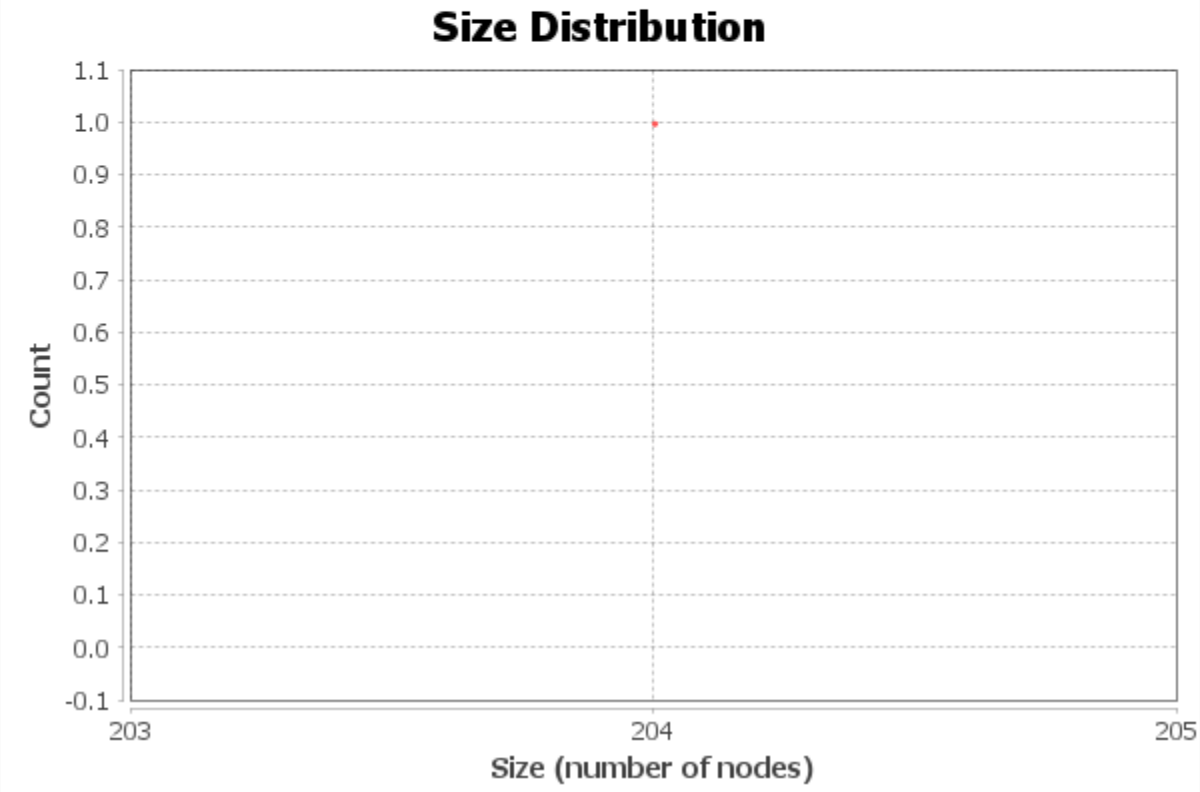
Connected Components Report

Parameters:

Network Interpretation: directed

Results:

Number of Weakly Connected Components: 1
Number of Strongly Connected Components: 176



While using NetworkX the above result is confirmed

```
In 13   1  #calculate giant component
        2  giant=max(nx.weakly_connected_components(final), key=len)
        3  len(giant)
           Executed at 2024.01.15 16:04:31 in 31ms

Out 13     204
```

In a directed network where the largest component encompasses the entire graph, the structure exhibits complete connectivity, indicating that from every node, there is a directed path to every other node. This level of comprehensive interconnectivity reveals the absence of isolated nodes or subgraphs, pointing to a highly cohesive network without structural bottlenecks or disconnected clusters.

5.3 Component Size Distribution

Component size distribution in the context of network analysis refers to the distribution of sizes of the connected components within a network. From Gephi we already know that there is only one Weakly connected component and that it has a size of 204.

This is also backed by NetworkX:

```
In 15
#calculate the component size distribution
import collections
components = nx.weakly_connected_components(final)
component_sizes = [len(c) for c in components]
counter = collections.Counter(component_sizes)
counter
Executed at 2024.01.15 20:22:35 in 5ms

Out 15
Counter({204: 1})
```

Once again it is evident that the network is entirely connected as a single unified structure without any isolated nodes or subgraphs.

https://web.stanford.edu/~jacksonm/Gephi-instruction-1-updatedApr2015.pdf

6. Degree measures

6.1 Maximum Node Degree

The maximum node degree in a network refers to the highest degree (number of connections) of any node within that network. It's a measure of the maximum connectivity that any single node has with other nodes, while taking into account both inwards and outwards connections.

Using Gephi's Average Degree function and by sorting the graph table in the data laboratory tab we can see that the maximum node degree is 209 and the channel that has it is ZackRawrr.

Id	Label	Timestamp	interval	isseed	seedrank	subscriberco...	videocount	viewcount(10...	country	...	Closeness ...	...	...	Ha...	...	...	...	In-Degree	Out-Degree
UCq2jigrIGtupbTXiNjq6Wrw	ZackRawrr			yes	1	424000	169	328231	not set	...	1.0	...	5	1.0	...	0	...	6	203
UCjekkqakBWqPu44aExSDxxA	Click4Ga...			no		86300	470	249828	US	...	0.563889	...	2	0.6...	...	0	...	3	46
UC7WDD6yHgzdqijHluCi1z-Q	The Act M...			no		1850000	319	4062959	US	...	0.539894	...	3	0.5...	...	0	...	5	30
UChGcWDcAbCoRvCc3vaW1m2A	Stoopzz			no		125000	264	158961	US	...	0.539894	...	2	0.5...	...	0	...	5	30
UCRWyPm7MrfotIYF8A8MGV3g	Josh Strife ...			no		835000	685	1484947	GB	...	0.534211	...	1	0.5...	...	0	...	10	26
UCE-f0sqi-H7kuLT0YiW9rcA	TheLazyPe...			no		744000	640	2485732	US	...	0.525907	...	1	0.5...	...	0	...	6	20
UCO7dBj4bwaDOMoX-kKYF2Ww	Atozy			no		1470000	832	3352807	US	...	0.29635	...	1	0.3...	...	0	...	1	19

These results are also confirmed by NetworkX

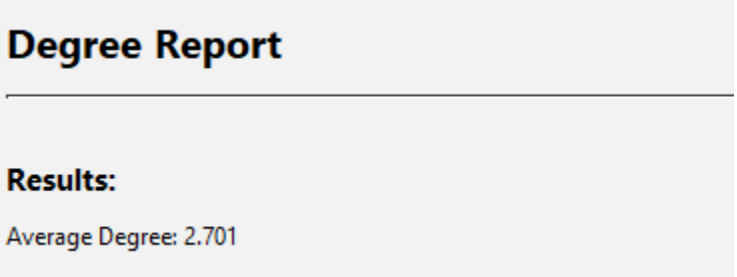
```
In 18
#calculate the maximum degree
sorted(dict(final.degree()).items(), key=lambda x: x[1], reverse=True)
Executed at 2024.01.15 22:04:44 in 9ms

Out 18
[('UCq2jigrIGtupbTXiNjq6Wrw', 209),
 ('UCjekkqakBWqPu44aExSDxxA', 49),
 ('UCRWyPm7MrfotIYF8A8MGV3g', 36),
 ('UC7WDD6yHgzdqijHluCi1z-Q', 35),
 ('UChGcWDcAbCoRvCc3vaW1m2A', 35),
 ('UCE-f0sqi-H7kuLT0YiW9rcA', 26),
 ('UCO7dBj4bwaDOMoX-kKYF2Ww', 20),
 ('UCZRoNJu1OszFqABP8AuJIuw', 17),
 ('UCcGqpQO6y4aB4IaZtLCZMXg', 16),
 ('UC-91UA-Xy2Cvb98deRXuggA', 13),
 ('UCSdma21fnJzgmPodhC9SJ3g', 13),
 ('UC6idiEhlPEreqzCWfQ_GiLw', 13),
 ('UCXuqSBlHAE6Xw-yeJA0Tunw', 11),
 ('UC-lHJZR36qxm24_Vd_AJ5Yw', 11),

In 22
#find the node with the maximum degree
final.nodes['UCq2jigrIGtupbTXiNjq6Wrw']
Executed at 2024.01.15 22:07:34 in 3ms

Out 22
{'label': 'ZackRawrr',
 'isseed': 'yes',
 'seedrank': '1',
 'subscribercount': 424000,
 'videocount': 169,
 'viewcount(100s)': 328231,
 'country': 'not set',
 'daysactive': 5610,
 'Modularity Class': 5,
 'size': 50.0,
 'r': 198,
 'g': 134,
 'b': 233,
 'x': -1216.7535,
```

In a directed network, having a node with a maximum degree of 209, where most are out-degrees, indicates that this node acts predominantly as a broadcaster or influencer, sending out connections to a large number of other nodes. This asymmetry in connectivity suggests that the node is crucial for the dissemination of information and influence, playing a central role in the network's dynamics.

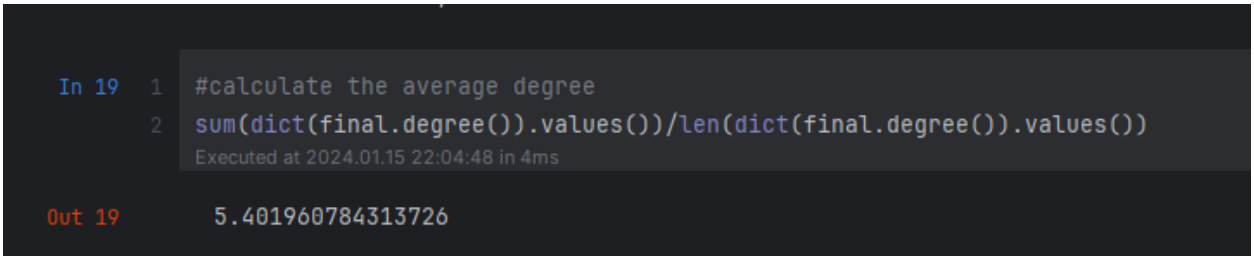
6.2 Average Node Degree

The average node degree in a network is a measure that represents the average number of connections (edges) per node. It is important to note that the average degree calculated by Gephi is approximately half of the average degree of interest, since Gephi define ”average degree” as the average of average in- and out- degrees (source).

With Gephi’s average degree function the result comes to 2.701



While with NetworkX we get 5.402 which is double Gephi’s result

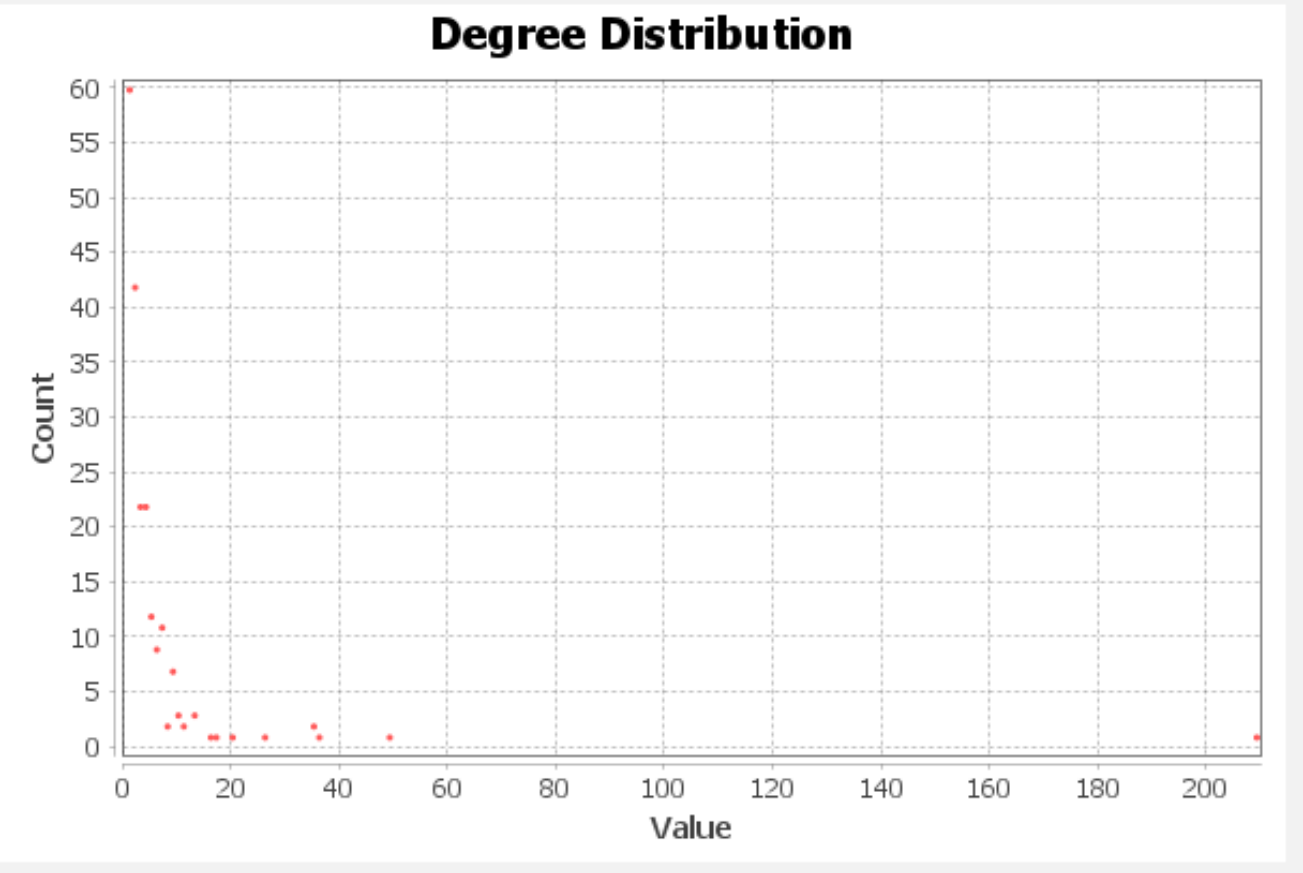


Gephi’s result indicates that each node, on average, has a combination of incoming and outgoing connections that averages to 2.701. It doesn't mean that each node has 2.701 incoming and 2.701 outgoing connections, but rather that the average of these two types of connections is 2.701. While NetworkX approach reveals that each node, on average, directly interacts or is linked with about 5.402 other nodes.

6.3 Degree Distribution

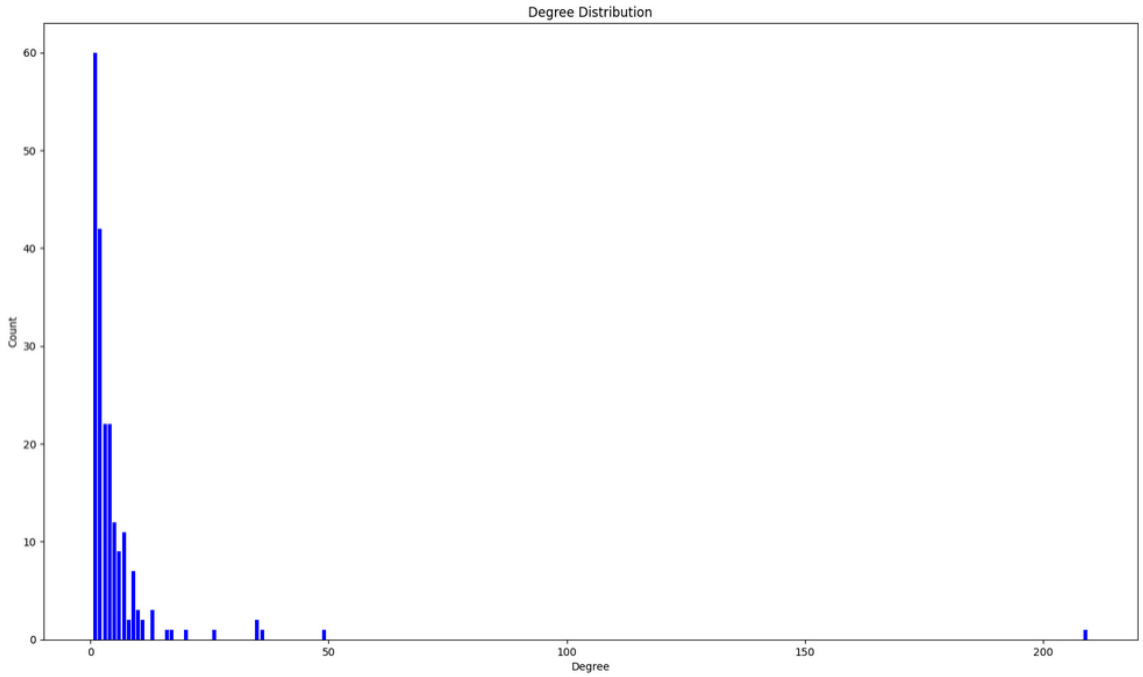
Degree distribution in network analysis is a fundamental concept that describes how the degrees of nodes (i.e., the number of connections each node has) are distributed across the network. In a directed network, nodes have in-degrees (number of incoming edges) and out-degrees (number of outgoing edges).

With the help of Gephi's functions



With this NetworkX code

```
[14]: #calculate the degree distribution
import collections
degree_sequence = sorted([d for n, d in final.degree()], reverse=True)  # degree sequence
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())

#Plot the degree distribution
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
plt.bar(deg, cnt, width=0.80, color='b')
plt.title("Degree Distribution")
plt.ylabel("Count")
plt.xlabel("Degree")
#make the plot bigger
fig.set_size_inches(18.5, 10.5)
```



Using NetworkX it is also possible to find the nodes with the highest degree

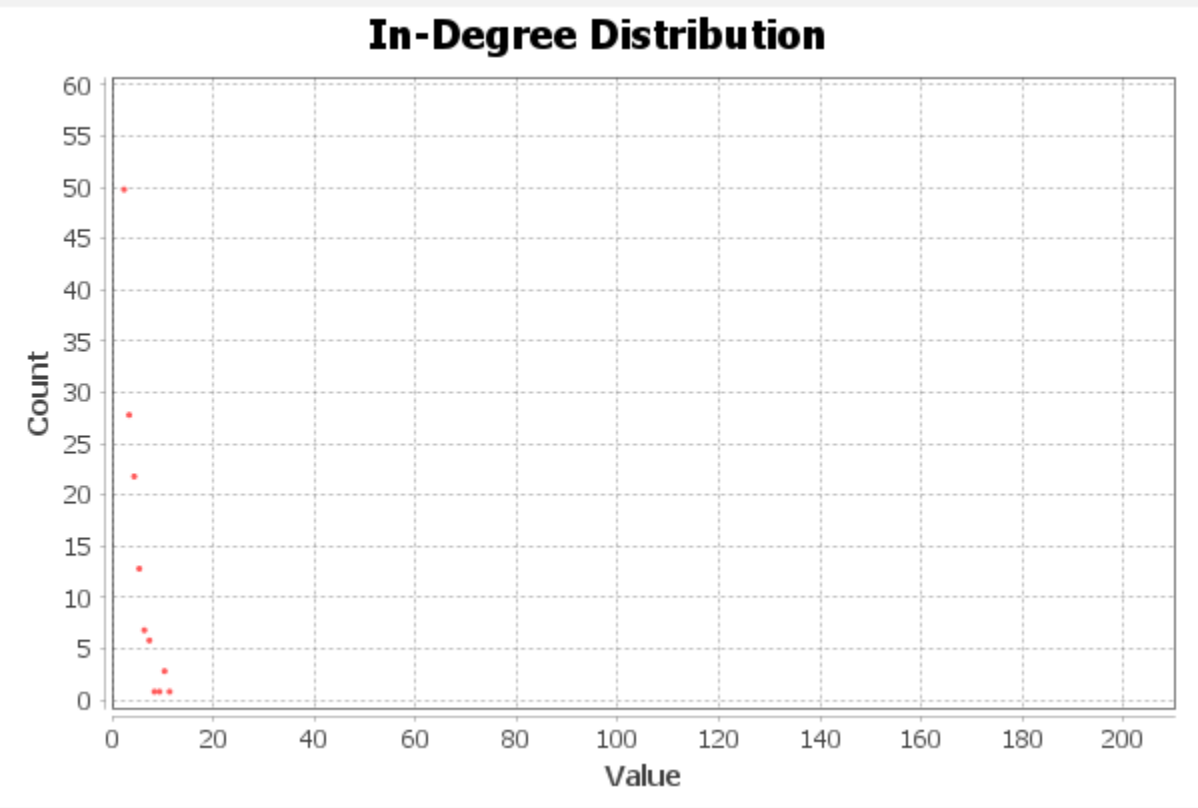
```
In 18  #find the top 10 nodes with the highest degree
       top10_degree = sorted(dict(final.degree()).items(), key=lambda x: x[1], reverse=True)[:10]

       #match the top 10 nodes with the original graph and print the labels and the degree
       for i in top10_degree:
           print(original.nodes[i[0]]['label'], i[1])

       Executed at 2024.01.16 22:01:52 in 4ms

       ZackRawrr 209
       Click4Gameplay 49
       Josh Strife Hayes 36
       The Act Man 35
       Stoopzz 35
       TheLazyPeon 26
       Atozy 20
       Logically Answered 17
       Nintendo Prime 16
       Joshua Fluke 13
```

Nodes like ZackRawrr and Click4Gameplay could be key players or hubs in the network, playing crucial roles in the spread of information and influence. The varying degrees also indicate a heterogeneous network, where some nodes are far more connected than others, which is typical in social media platforms.

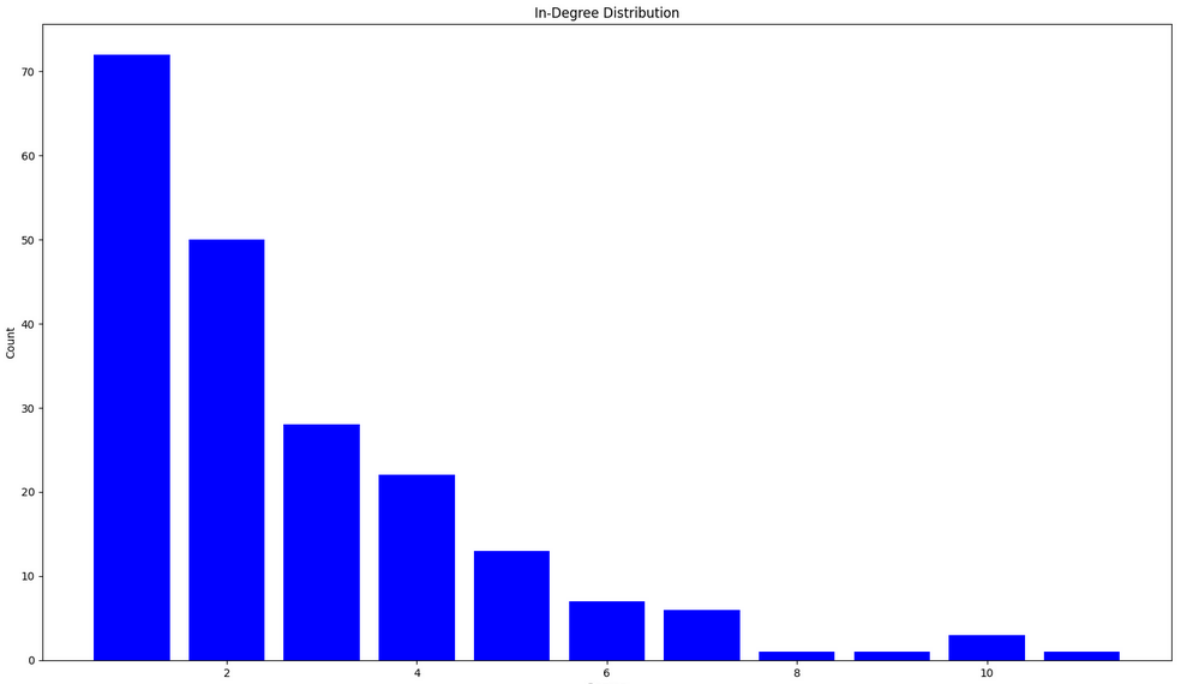
6.4 In-Degree Distribution

In a directed network, the in-degree of a node refers to the number of incoming edges it has in other words the in-degree is a count of how many edges are directed towards a particular node.

The results of Gephi and NetworkX about in-degree



[19]:
```
#Calculate the in-degree distribution
degree_sequence = sorted([d for n, d in final.in_degree()], reverse=True)  # degree sequence
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())

#Plot the degree distribution
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
plt.bar(deg, cnt, width=0.80, color='b')
plt.title("In-Degree Distribution")
plt.ylabel("Count")
plt.xlabel("Degree")
#make the plot bigger
fig.set_size_inches(18.5, 10.5)
```



Finding the nodes with highest in-degree

In 20
```
#Find the top 10 nodes with the highest in-degree
top10_in_degree = sorted(dict(final.in_degree()).items(), key=lambda x: x[1], reverse=True)[:10]

#match the top 10 nodes with the original graph and print the labels and the in-degree
for i in top10_in_degree:
    print(original.nodes[i[0]]['label'], i[1])
```

Executed at 2024.01.16 22:54:29 in 11ms

```
PewDiePie 11
Linus Tech Tips 10
Josh Strife Hayes 10
penguinz0 10
videogamedunkey 9
Ludwig 8
Coffeezilla 7
SunnyV2 7
YongYea 7
Nixxiom 7
```

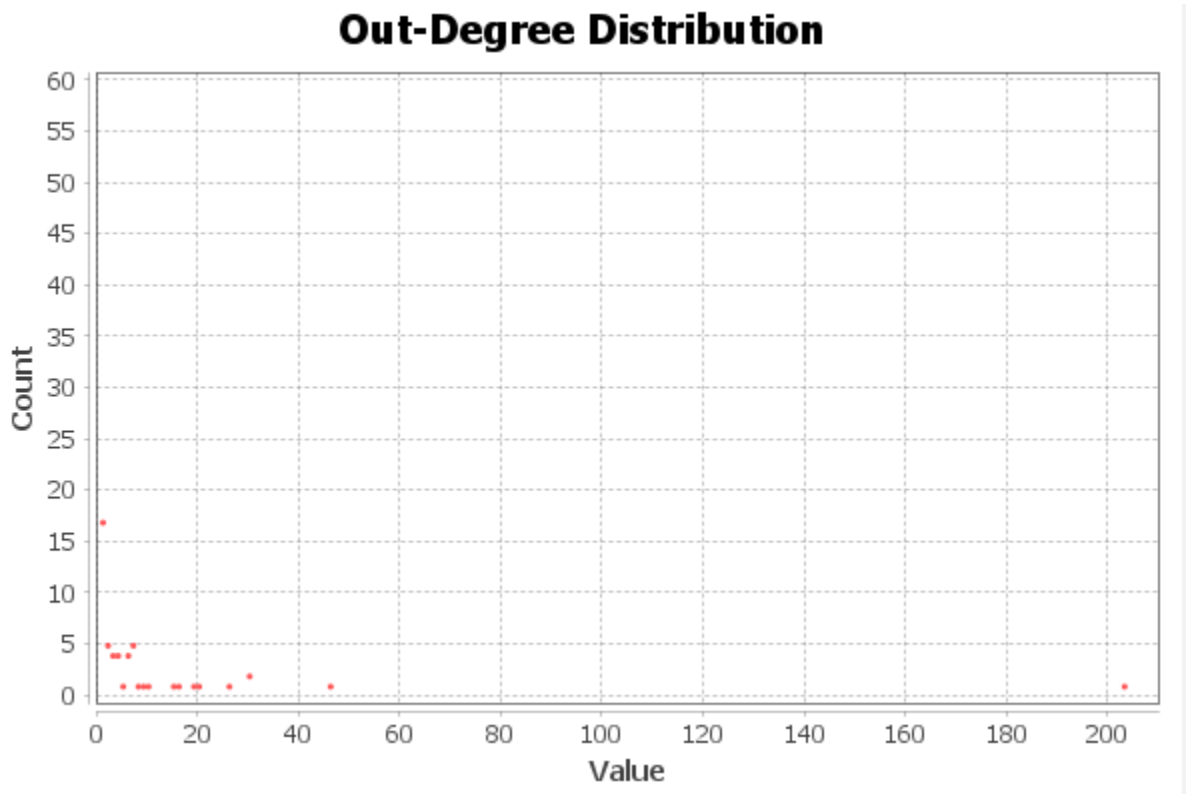
The in-degrees suggest a network where certain nodes (like PewDiePie and Linus Tech Tips) are more central in terms of receiving attention, which could be indicative of their roles as influencers and key figures in their respective domains. The distribution of in-degrees also implies a certain level of hierarchy or asymmetry in the network, with some nodes being more prominent focal points than others.

6.5 Out-Degree Distribution

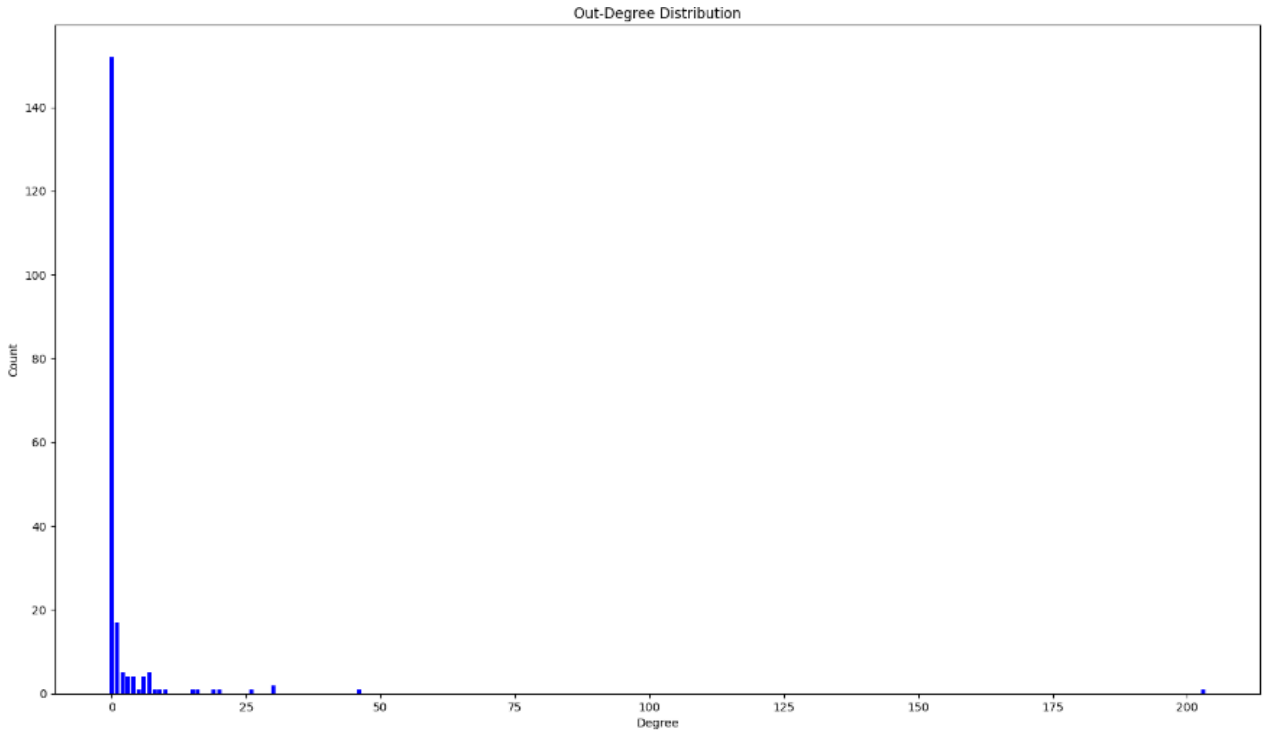
In a directed network, the out-degree of a node refers to the number of outgoing edges it has. In other words, the out-degree is a count of how many edges are directed away from a particular node to other nodes in the network.

Repeating the steps that were followed above

Gephis results



NetworkX results



[20]:
```
#Find the top 10 nodes with the highest out-degree
top10_out_degree = sorted(dict(final.out_degree()).items(), key=lambda x: x[1], reverse=True)[:10]

#match the top 10 nodes with the original graph and print the labels and the out-degree
for i in top10_out_degree:
    print(original.nodes[i[0]]['label'], i[1])
```

```
ZackRawrr 203
Click4Gameplay 46
The Act Man 30
Stoopzz 30
Josh Strife Hayes 26
TheLazyPeon 20
Atozy 19
Logically Answered 16
Nintendo Prime 15
Joshua Fluke 10
```

Overall, these out-degree values suggest a hierarchy of activity or influence in the network, with certain nodes like ZackRawrr and Click4Gameplay being particularly active in reaching out to others.

7. Centrality measures

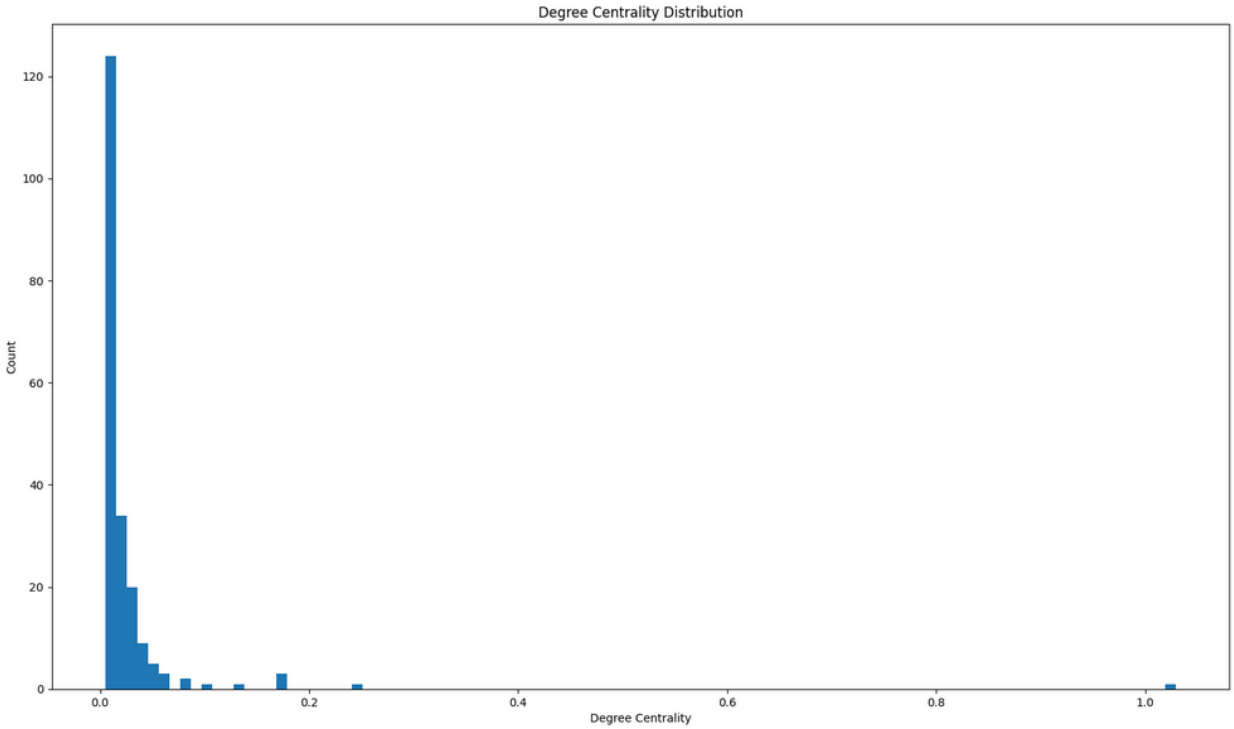
Gephi and NetworkX use different methods to calculate all of the centrality measures. This analysis will focus mainly on the results of NetworkX because, it utilizes a specialized algorithms for the calculation of the measures such as Wasserman and Faust's improved formula for closeness centrality.

7.1 Degree Centrality

Degree centrality is a metric used in network analysis that assesses the importance or influence of a node within a network based on its number of connections. High degree centrality indicates a node with many connections, suggesting it plays a central role in the network's interactions and communication.

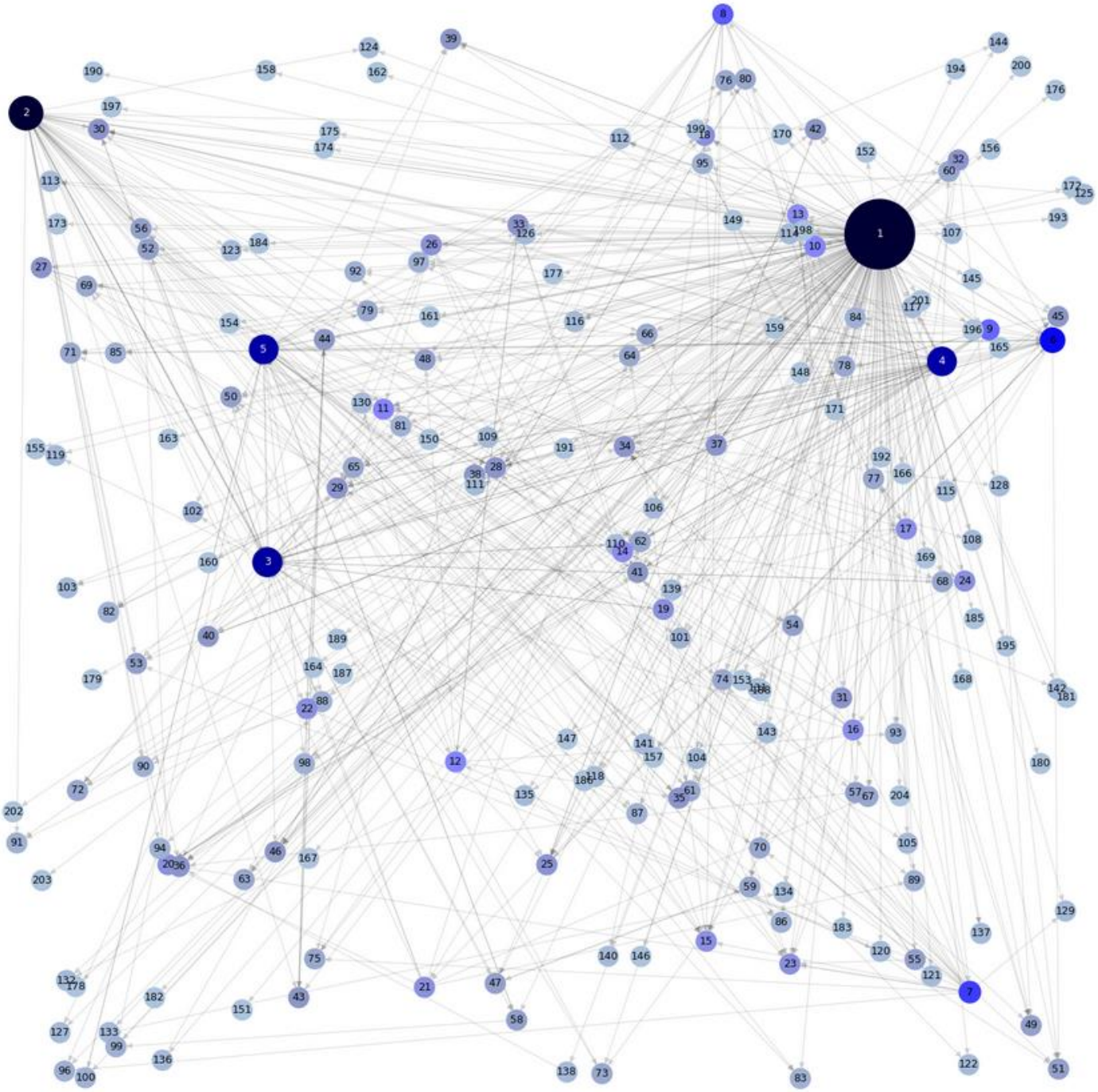
Using NetworkX it is possible to find the Degree Centrality Distribution

[136]:
```
#calculate the centrality degree
centrality = nx.degree_centrality(final)
#plot the centrality degree distribution
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
plt.hist(centrality.values(), bins=100)
plt.title("Degree Centrality Distribution")
plt.ylabel("Count")
plt.xlabel("Degree Centrality")
#make the plot bigger
fig.set_size_inches(18.5, 10.5)
```



Additionally, the network can be visualized based on the degree centrality measure. In this representation, nodes with higher centrality are depicted with larger sizes and darker shades, emphasizing their relative importance in the network. The nodes are also numbered based on their degree centrality in descending order from highest to lowest.

(The code for this and the rest of the Centrality measures visualizations will be in the appendix because they were taking a significant amount of space)



Finally, it is possible to check the nodes with the highest degree centrality of the above diagram

```
In 139
#show the labels labels of the nodes and their  degree centrality sorted in descending order and number them
counter = 0
for i in sorted(centrality.items(), key=lambda x: x[1], reverse=True):
    counter += 1
    print(counter,final.nodes[i[0]]['label'], i[1])

Executed at 2024.01.18 18:57:16 in 68ms

1 ZackRawrr 1.0295566502463054
2 Click4Gameplay 0.2413793103448276
3 Josh Strife Hayes 0.17733990147783252
4 The Act Man 0.1724137931034483
5 Stoopzz 0.1724137931034483
6 TheLazyPeon 0.12807881773399016
7 Atozy 0.09852216748768473
8 Logically Answered 0.08374384236453201
9 Nintendo Prime 0.07881773399014778
10 Joshua Fluke 0.06403940886699508
11 NakeyJakey 0.06403940886699508
12 Swifty 0.06403940886699508
13 Linus Tech Tips 0.054187192118226604
14 PewDiePie 0.054187192118226604
```

There are some familiar names in this list like ZackRawrr with 1.02 as well as Click4Gameplay with 0.24 and The Act Man with 0.17 who all scored relatively high in both the degree and the out-degree distributions.

Some additional information about those two channels

The YouTube channel "Click4Gameplay," managed by Marko Glisovic, focuses primarily on RPG (Role-Playing Game) content. The channel has gained popularity for its in-depth coverage, such as reviews, guides and gameplay, of RPG games. Marko has around 87,200 subscribers and has uploaded a total of 471 videos, accumulating over 25 million views.

"The Act Man" is a YouTube channel created by Kelly Van Achte and its known for its game reviews, primarily focusing on gaming videos. The channel has approximately 1,850,000 subscribers and a total of 409.080.674 video views. The Act Man's videos typically include commentary, review and gameplay videos of different types of games such as Halo, Red Dead Redemption and Call of Duty.

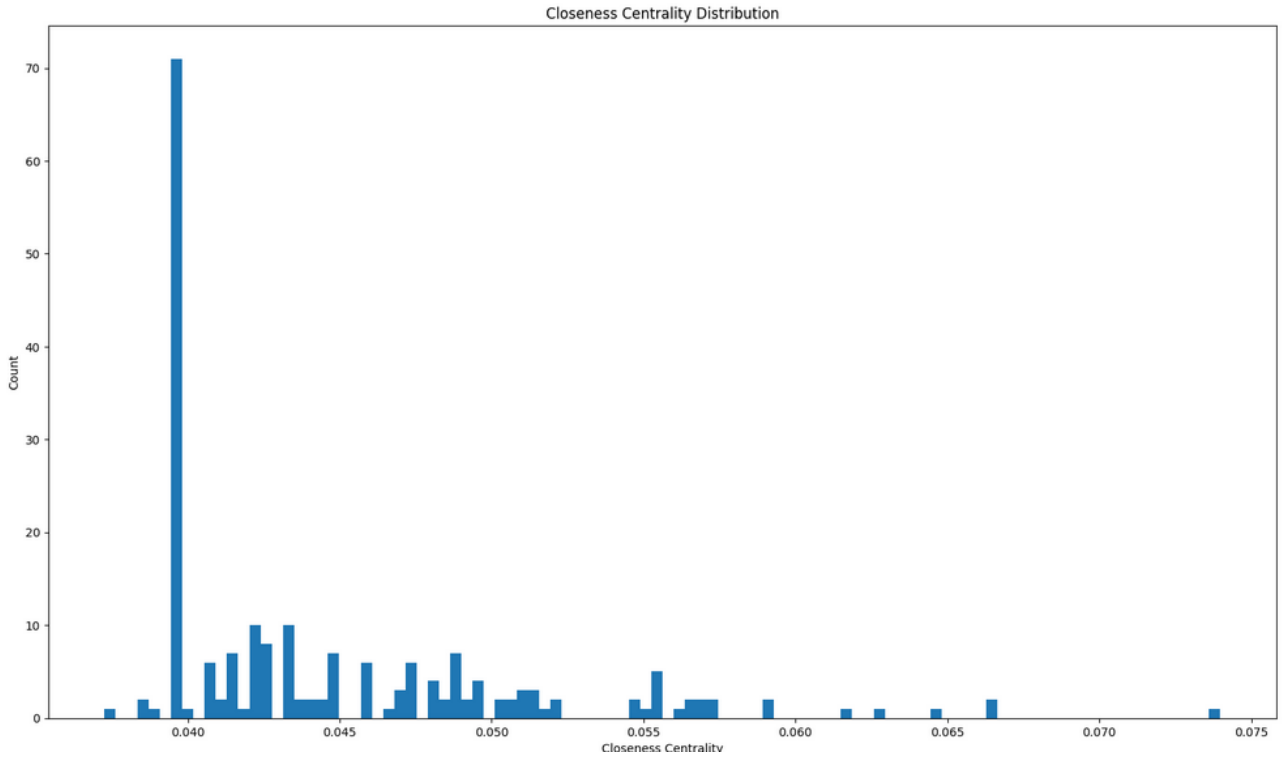
High degree centrality indicates that a node has a large number of connections, signifying an influential or central position within the network's structure. Such nodes are pivotal in disseminating information, ideas, or resources quickly across the network, often representing key players in social networks.

7.2 Closeness Centrality

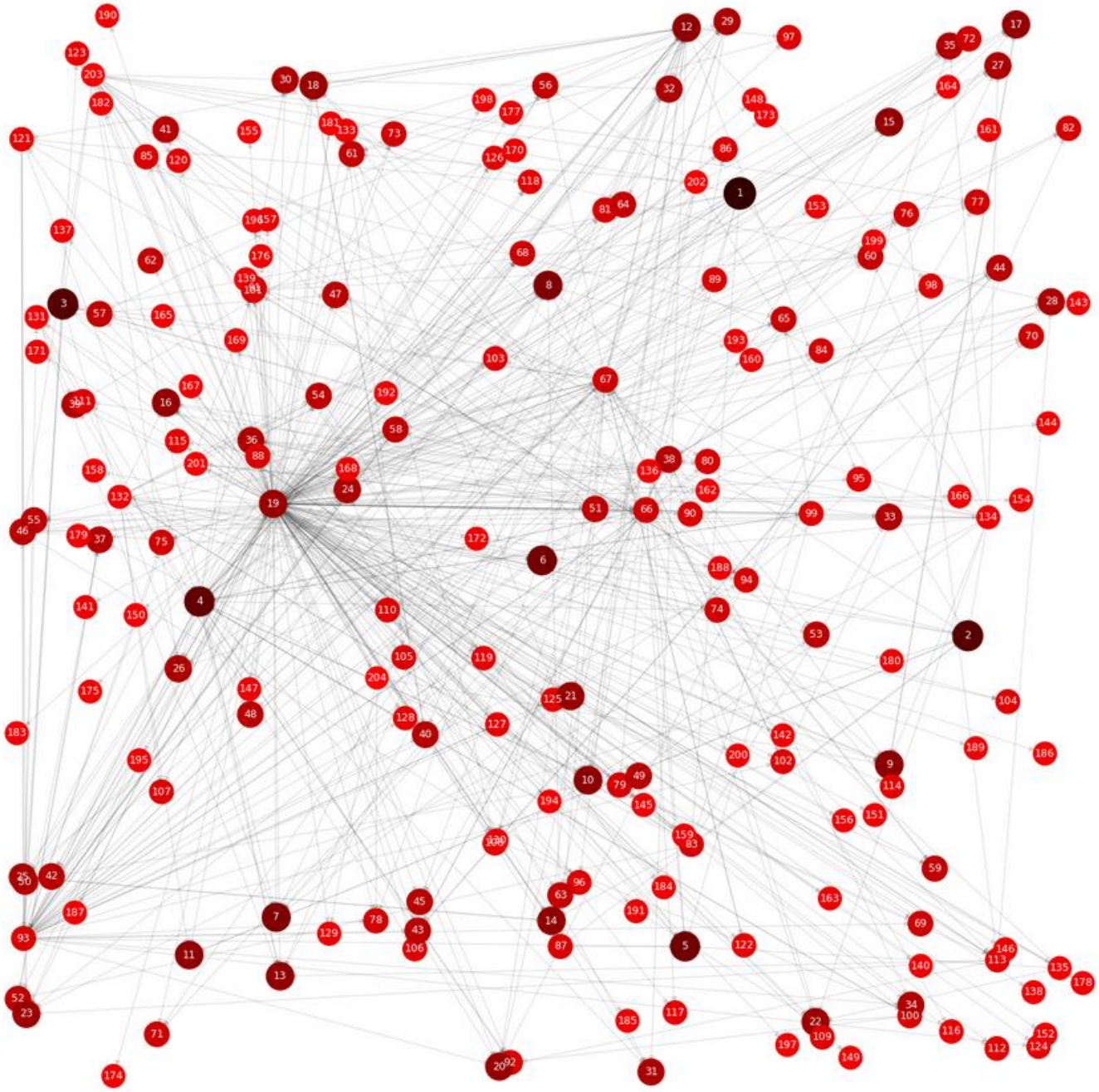
Closeness centrality indicates how close a node is to all other nodes in a network, calculated based on the shortest paths from the node to all others. It measures the average distance of a node to all other nodes, with higher values indicating closer proximity and potential for faster spread of information or resources from the node to the rest of the network.

Starting with the closeness centrality distribution

[134]:
```
  #calculate the closeness centrality
#make the graph undirected

closeness = nx.closeness_centrality(final)

#plot the closeness centrality distribution
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
plt.hist(closeness.values(), bins=100)
plt.title("Closeness Centrality Distribution")
plt.ylabel("Count")
plt.xlabel("Closeness Centrality")
#make the plot bigger
fig.set_size_inches(18.5, 10.5)
```



Continuing to the visualization of the network based on the closeness centrality measure. The graph follows the same rules as the graph of degree centrality but the colour switched from blue to red. This time though because there is a better distribution with lower values, the nodes are not that susceptible to extraordinary size change.



Finaly the top values of the above graph are reviewed.

```
In 169
#show the labels labels of the nodes and their closeness centrality sorted in descending order
counter = 0
for i in sorted(closeness.items(), key=lambda x: x[1], reverse=True):
    counter += 1
    print(counter,final.nodes[i[0]]['label'], i[1])
Executed at 2024.01.18 20:12:03 in 231ms

1 PewDiePie 0.07397959183673469
2 Linus Tech Tips 0.06658739595719382
3 penguinz0 0.06658739595719382
4 Josh Strife Hayes 0.06443882305951272
5 videogamedunkey 0.06277056277056277
6 boogie2988 0.06183368869936033
7 SunnyV2 0.05920803334596438
8 VaatiVidya 0.05918367346938776
9 Ludwig 0.05741464243247835
10 Coffeezilla 0.05741464243247835
11 ShortCircuit 0.056795131845841784
12 TheLazyPeon 0.05676753460004692
13 Nixxiom 0.05665024630541872
```

Again some familiar names are observed such as PewDiePie (0.074) one of the world's biggest YouTube creators, Linus Tech Tips (0.066) one of the biggest tech channels and penguiz0 (0.066) one of YouTube's most popular streamers. Among those channels the channel Josh Strife Hayes can be spotted. This channel is present in the top channels for the degree, in-degree and out-degree distributions as well as the degree centrality distribution.

After diving deeper into this channel, the following information about it can be obtained

Josh Strife Hayes is a popular gaming YouTube channel from the creator of the same name Josh Hayes, known for his insightful MMORPG reviews and guides. He has gained significant attention for his content on games such as RuneScape, World of Warcraft, and Everquest. Since launching his YouTube channel in March 2017, he has attracted a large following, with his channel surpassing 800,000 subscribers and 149.000.000 views.

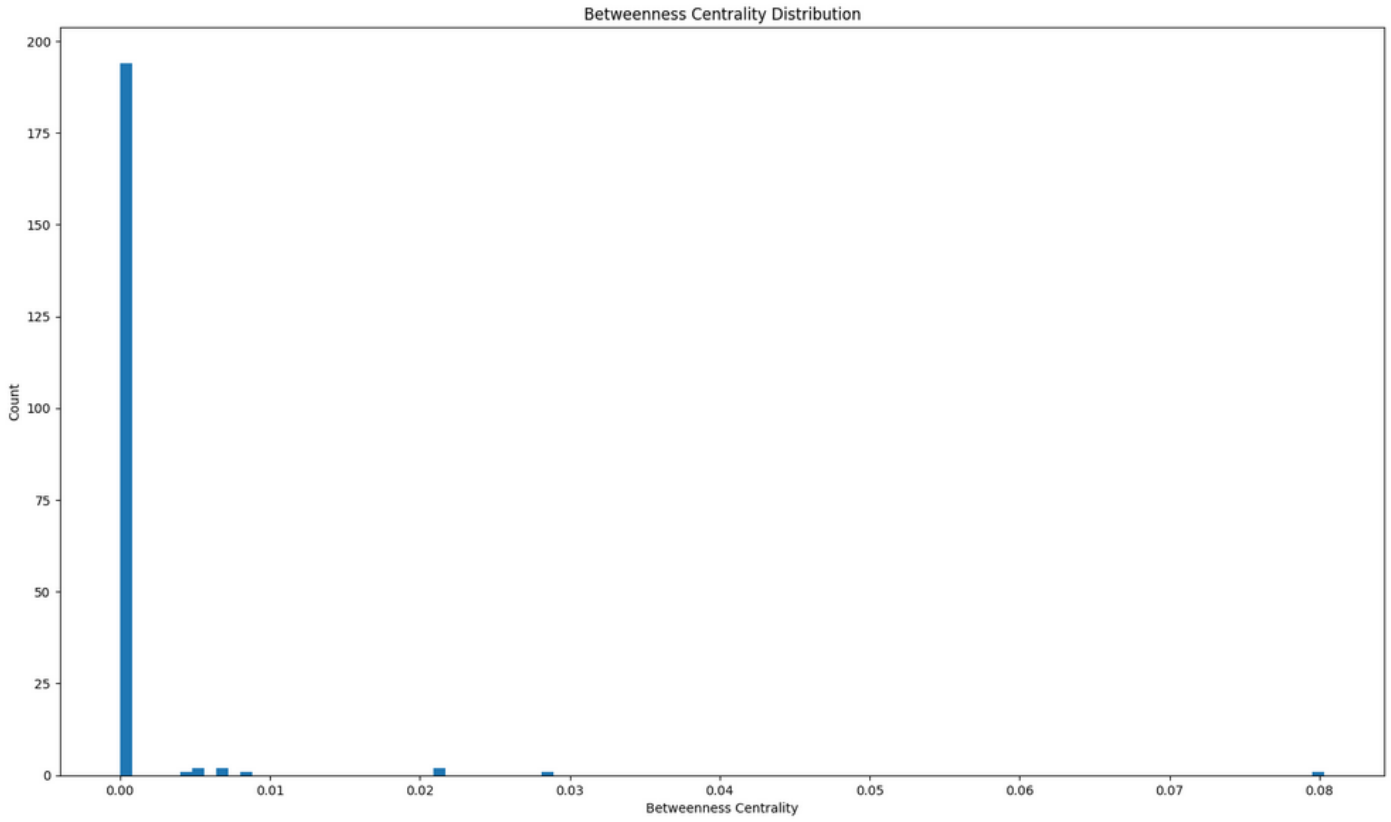
High closeness centrality signifies that a node is, on average, at a short distance from all other nodes, indicating its efficiency in reaching or being reached by others in the network. Nodes with high closeness centrality are crucial for information flow as they can access or be accessed by the rest of the network more efficiently.

7.3 Betweenness Centrality

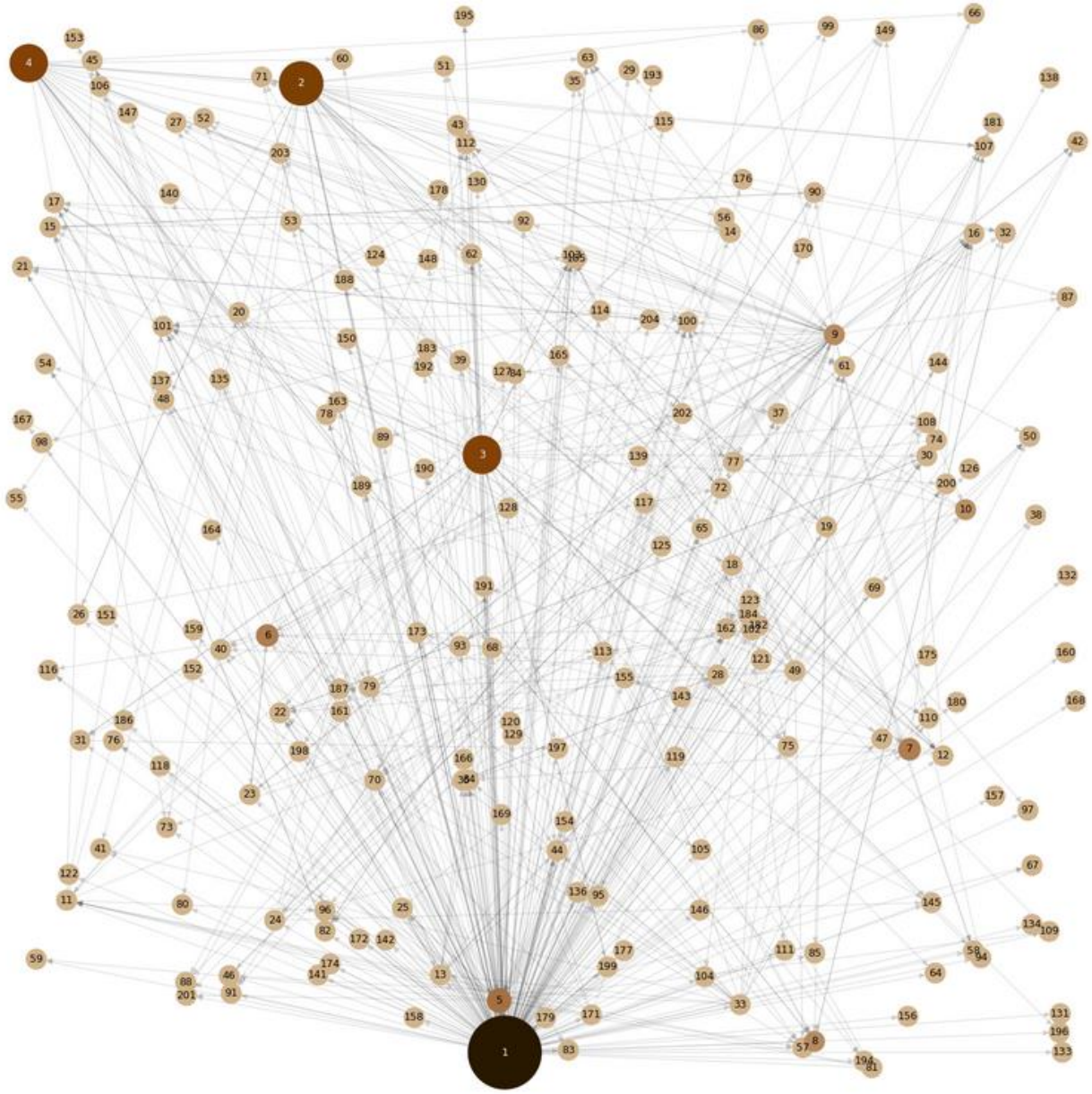
Betweenness centrality measures a node's importance based on its presence in the shortest paths between other nodes in the network. It quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.

Repeating the steps from the previous centrality measures, starting of with the distribution of Betweenness Centrality

[174]:
```
#calculate the betweenness centrality

betweenness = nx.betweenness_centrality(final)

#plot the betweenness centrality distribution
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
plt.hist(betweenness.values(), bins=100)
plt.title("Betweenness Centrality Distribution")
plt.ylabel("Count")
plt.xlabel("Betweenness Centrality")

#make the plot bigger
fig.set_size_inches(18.5, 10.5)
```



Continuing with the graphical representation of the network based on betweenness centrality with the same rules as the previous visualization and with a brown colour pallet



And finally, to check the top values of the graph

```
In 204
#show the labels labels of the nodes and their betweenness centrality sorted in descending order
counter = 0
for i in sorted(betweenness.items(), key=lambda x: x[1], reverse=True):
    counter += 1
    print(counter, final.nodes[i[0]]['label'], i[1])
Executed at 2024.01.18 22:13:10 in 20ms

 1 ZackRawrr 0.08032848851387604
 2 Josh Strife Hayes 0.02860719569493897
 3 Stoopzz 0.021425726316474016
 4 TheLazyPeon 0.02101806239737274
 5 The Act Man 0.008485343608252453
 6 Swifty 0.007047342665301015
 7 Myndflame 0.007042871774862219
 8 KiraTV 0.0050334097449153785
 9 Click4Gameplay 0.004990326618868784
 10 The Comeback Kids 0.004779788323659952
 11 NakeyJakey 0.000715342470207612
 12 Thunderf00t 0.000714123136451576
 13 Joshua Fluke 0.0005771513111902323
 14 Ser Medieval 0.00030076899315547315
```

Among the channel names that have been seen before like ZackRawrr (0.08), Josh Strife Hayes and The Act Man (0.008), the last name that has been seen the most among the top 5 nodes in distributions ,such as degree , out-degree and degree centrality, is Stoopzz (0.021).

"Stoopzz" is an American channel that has been active for over 7 years and is created by Oliver Sabahi. Him and ZackRawrr (Asmongold) have created videos together for the game World of Warcraft As of 2024, it has amassed over 124,000 subscribers and has uploaded 244 videos with over 15.000.000 views. The content of Stoopzz primarily revolves around video game culture and community, with a focus on role-playing games, strategy video games, action games, and action-adventure games.

High betweenness centrality signifies that a node is a crucial connector or bridge along the shortest paths between other nodes, indicating its influence across the network. Such nodes, often seen as gatekeepers or brokers, especially in social networks, hold a position of power and influence due to their ability to mediate interactions between otherwise disconnected groups or individuals.

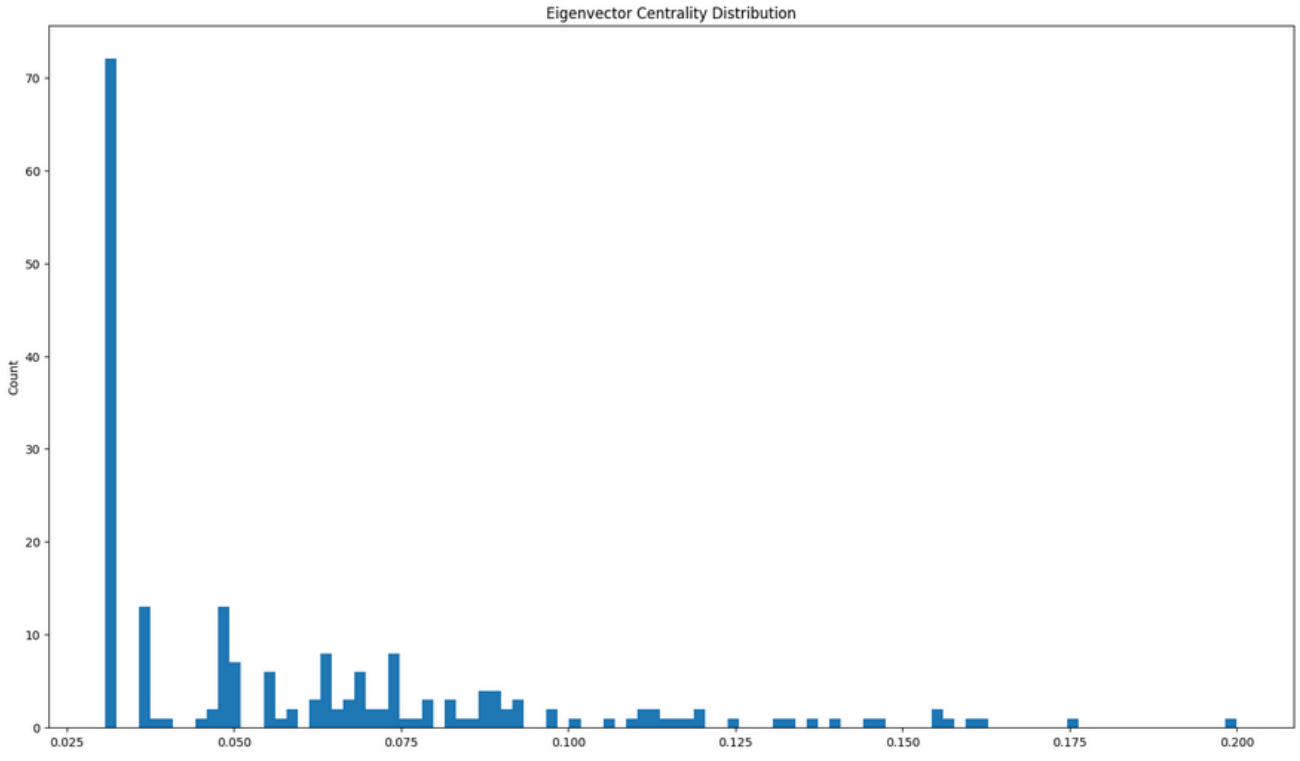
7.4 Eigenvector Centrality

Eigenvector centrality evaluates a node's influence not just by the number of its direct connections, but also by the significance of these connections. It takes into account the centrality of a node's neighbors, meaning that connections to highly influential nodes contribute more to a node's score.

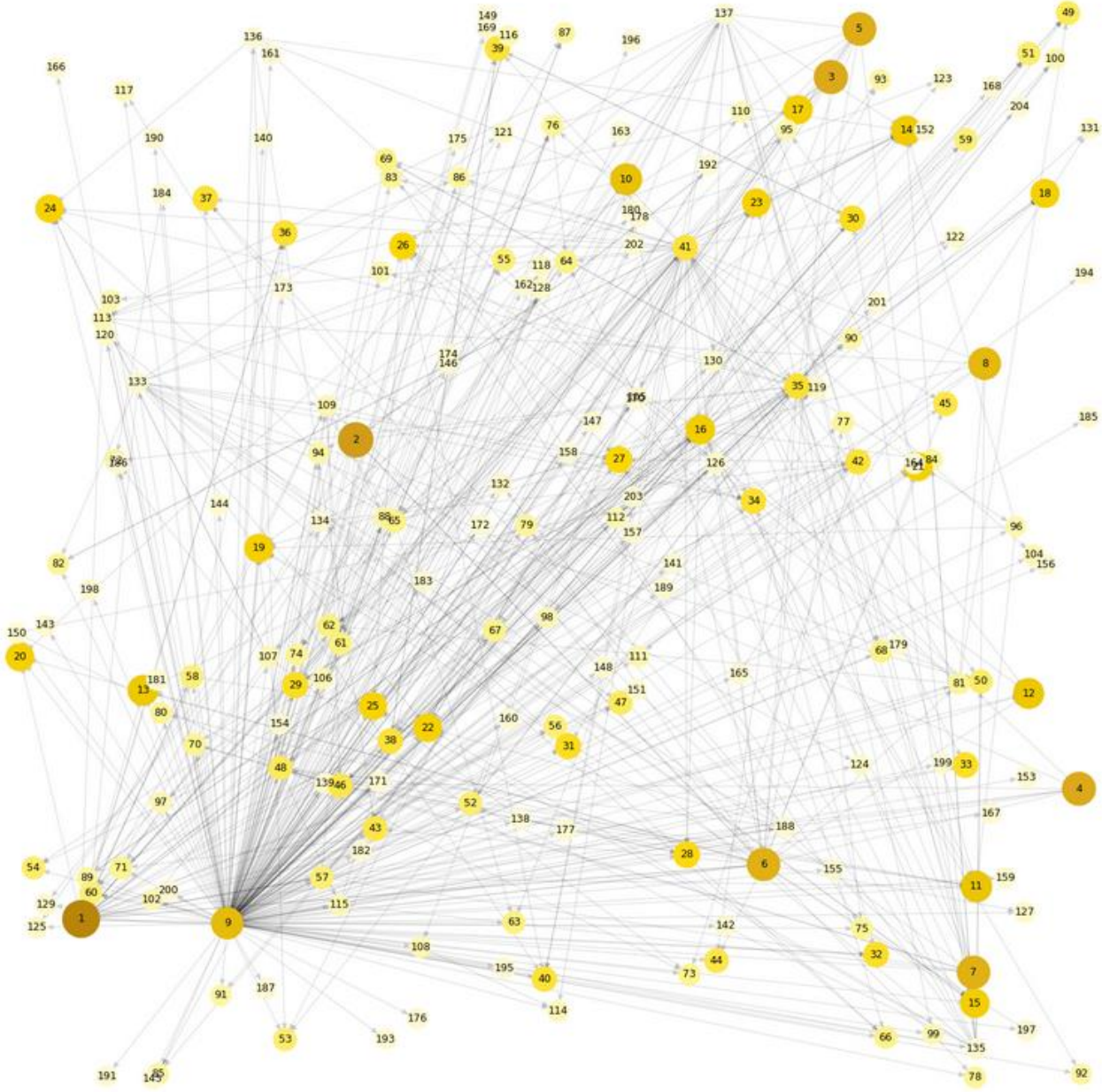
Distribution Graph

[171]:
```
#calculate the eigenvector centrality
eigenvector = nx.eigenvector_centrality(final)
```

[172]:
```
#plot the eigenvector centrality distribution
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
plt.hist(eigenvector.values(), bins=100)
plt.title("Eigenvector Centrality Distribution")
plt.ylabel("Count")

#make the plot bigger
fig.set_size_inches(18.5, 10.5)
```



Graph visualization with a yellow palette



And finally to check the best nodes of the graph and distribution

```
In 222
#find the labels of the nodes and their eigenvector centrality sorted in descending order
counter = 0
for i in sorted(eigenvector.items(), key=lambda x: x[1], reverse=True):
    counter += 1
    print(counter, final.nodes[i[0]]['label'], i[1])
Executed at 2024.01.19 00:55:14 in 77ms

1 Josh Strife Hayes 0.2000061971914833
2 KiraTV 0.1750689817142868
3 Kira 0.16187630981119946
4 SunnyV2 0.16016041467593112
5 PewDiePie 0.15746117220249695
6 TheLazyPeon 0.1556091140816621
7 penguinz0 0.15545488427085868
8 YongYea 0.1471364060720851
9 ZackRawrr 0.14465758213329716
10 Nixxiom 0.13988685147616348
11 videogamedunkey 0.13700409319804757
12 MitchManix 0.13260310481442106
13 The Critical Drinker 0.13155696218689672
14 Linus Tech Tips 0.12447540510328928
```

As for Eigenvector Centrality it is clear that the familiar channel Josh Strife Hayes has the highest value with 0.2 followed by KiraTV,with 0.17, a reaction and analysis channel, Kira with 0.16 a channel focus on music production and SunnyV2 with 0.16 a big analysis channel. Lastly a name to remember is TheLazyPeon with 0.15 which has been consistently in the top ten of the previous measures (both degree and centrality).

High eigenvector centrality in a network denotes that a node is not only well-connected but also linked to other highly connected or influential nodes, reflecting its significant role in the network. This measure goes beyond simply counting connections, emphasizing the quality of these connections, as a node's importance is amplified by being connected to other central nodes.

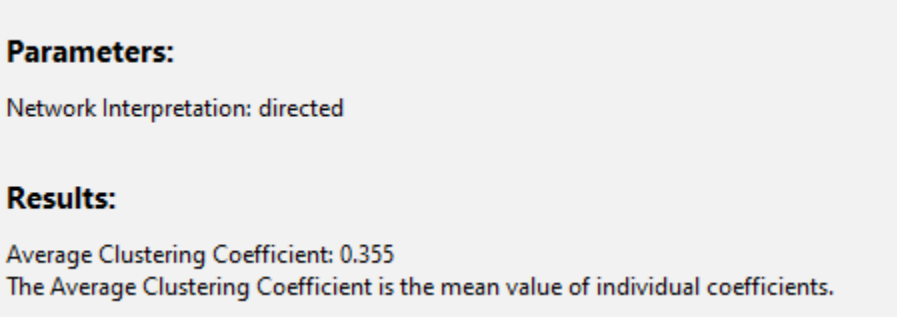
8. Clustering effects

8.1 Average Clustering Coefficient

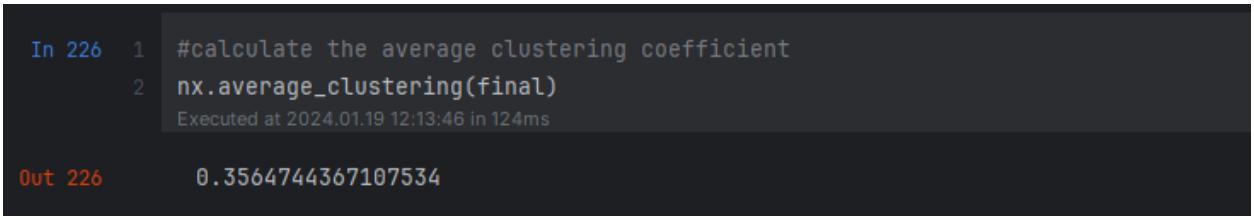
The average clustering coefficient in a network is a measure that captures the overall tendency of nodes to form tightly knit groups, calculated as the mean of the individual clustering coefficients of all the nodes. A high average clustering coefficient suggests that the network has a pronounced structure of local clusters or communities, where nodes tend to create dense interconnections, indicative of small-world properties. Conversely, a low average clustering coefficient implies a more tree-like structure with sparser connections, where neighbors of a node are less likely to be connected to each other.

Both NetworkX and Gephi give similar results for the average clustering coefficient

Gephi's Avg. Clustering Coefficient function gives a result of 0.355.



NetworkX's average_clustering() function gives a result of 0.356.



A clustering coefficient of 0.35 in a social network, indicates a moderate to high level of local clustering, signifying that the network's nodes tend to form interconnected groups or communities to a reasonable extent.

The presence of a short average shortest path length of 2.46 combined with a moderate to high average clustering coefficient of 0.35 in the network suggests the presence of the small-world effect. The network is characterized by efficient information transfer due to the short paths, along with a moderate to high degree of local interconnectedness or community-like structure due to the clustering.

8.2 Number of Triangles

Triangles refer to a set of three nodes that are all interconnected. A triangle in a network is formed when there are three edges connecting three nodes, creating a closed loop. Each node in the triangle has a direct connection (or edge) to the other two nodes.

With NetworkX it is evident that the number of triangles in the directed network is 31.

```
In 237
import pandas as pd
#Finding triangles and sorting nodes within each cycle
triangles = list(nx.simple_cycles(final.to_directed(), length_bound=3))
triangles = [sorted(cycle, key=lambda x: final.nodes[x]['label']) for cycle in triangles if len(cycle) == 3]

#Remove duplicate triangles
triangles = list(set(tuple(cycle) for cycle in triangles))

#Create a dataframe to display the triangles with node labels
triangles_df = pd.DataFrame([tuple(final.nodes[node]['label'] for node in triangle) for triangle in triangles], columns=['Node 1', 'Node 2', 'Node 3'])
len(triangles_df)
Executed at 2024.01.19 14:30:11 in 410ms

Out 237
31
```

It is also possible to take a look at these triangles.

```
triangles_df
```

[241]:

	Node 1	Node 2	Node 3
0	Josh Strife Hayes	Ser Medieval	ZackRawrr
1	KiraTV	TheLazyPeon	ZackRawrr
2	Click4Gameplay	The Act Man	ZackRawrr
3	Atrioc	The Act Man	ZackRawrr
4	Savix	Stoopzz	ZackRawrr
5	Stoopzz	Swifty	ZackRawrr
6	Josh Strife Hayes	Stoopzz	ZackRawrr
7	Josh Strife Hayes	Josh Strife Says	ZackRawrr
8	Josh Strife Hayes	Tolarian Community College	ZackRawrr
9	Josh Strife Hayes	Josh Strife Says	Tolarian Community College
10	Click4Gameplay	Josh Strife Hayes	Stoopzz
11	Click4Gameplay	TheLazyPeon	ZackRawrr
12	Emiru	OTK	Steak and Eggs Podcast
13	Stoopzz	TheLazyPeon	ZackRawrr
14	Josh Strife Hayes	Myndflame	ZackRawrr
15	Josh Strife Hayes	The Act Man	ZackRawrr
16	Click4Gameplay	Ser Medieval	TheLazyPeon
17	The Comeback Kids	TheLazyPeon	ZackRawrr
18	Click4Gameplay	Ser Medieval	ZackRawrr
19	Josh Strife Hayes	Xenosys Vex	ZackRawrr
20	The Act Man	Xenosys Vex	ZackRawrr
21	Click4Gameplay	Stoopzz	ZackRawrr
22	Click4Gameplay	Josh Strife Hayes	The Act Man
23	Click4Gameplay	Josh Strife Hayes	TheLazyPeon
24	Kira	KiraTV	TheLazyPeon
25	Josh Strife Hayes	Ser Medieval	TheLazyPeon
26	Josh Strife Hayes	TheLazyPeon	ZackRawrr
27	Josh Strife Hayes	The Act Man	Xenosys Vex
28	Click4Gameplay	Josh Strife Hayes	ZackRawrr
29	Josh Strife Hayes	KiraTV	TheLazyPeon
30	ATK	Stoopzz	ZackRawrr

It is clear that Josh Strife Hayes and ZackRawrr take part in the majority of them. These triangles are formed based on different types of connections between the nodes. For example, the triangle containing Josh Strife Hayes, Stoopzz and ZackRawrr is formed because all the channels have a connection to MMO RPG types of games. Another example is the triangle that consists of Emiru, OTK and Stakes and Eggs Podcast, which are all channels that are connected to the gaming organization OTK, emiru is a member of OTK, OTK is the main YouTube channel of the organization and Steak and Eggs Podcast is a podcast produced by the organization in which both Emiru and ZackRawrr take part in.

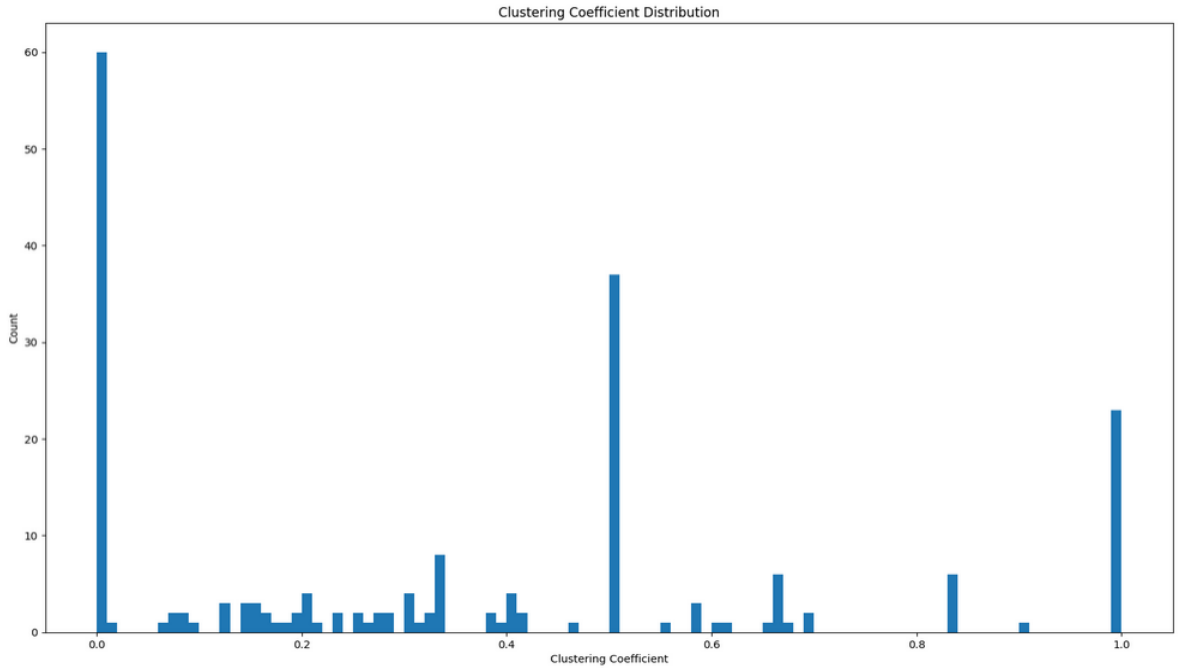
In a network comprising 204 nodes, the existence of 31 unique triangles, particularly with two nodes appearing in most of them, indicates a moderate level of local clustering and points to the significant roles of these two nodes. Their involvement in forming multiple triangles implies they have strong direct relationships with various nodes, influencing the network's information flow, resilience to disruptions, and overall cohesion.

8.3 Clustering Coefficient Distribution

The clustering coefficient distribution in a network is a statistical representation that shows how the clustering coefficients are distributed across all nodes in the network.

Using NetworkX we can plot the distribution.

[243]:
```
#calculate the  clustetering coefficient distribution
clustering = nx.clustering(final)
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
plt.hist(clustering.values(), bins=100)
plt.title("Clustering Coefficient Distribution")
plt.ylabel("Count")
plt.xlabel("Clustering Coefficient")
#make the plot bigger
fig.set_size_inches(18.5, 10.5)
```



The clustering coefficient distribution depicted in the histogram showcases three distinct peaks at 0, 0.5, and 1, indicating a network with varied levels of local node clustering. A substantial number of nodes have a clustering coefficient of 0, suggesting that many nodes act as connecting points without their neighbors being interconnected, possibly indicative of a network with bridge-like nodes or end points. The peak at 0.5 represents a moderate level of clustering, where exactly half of the neighbors of these nodes are interconnected, pointing towards the presence of some semi-tight communities or nodes that act as interfaces between clusters. Finally, the peak at 1 signifies that there is also a significant number of nodes within the network that are part of complete cliques, where each of their neighbors is connected to one another, reflecting the presence of tightly knit groups or fully interconnected subnetworks. This trichotomy within the clustering coefficient distribution implies a complex network structure characterized by a mix of isolated nodes, nodes that partially integrate into their communities, and nodes that are central to highly clustered communities.

8.4 Triadic Closure

Triadic closure is a concept that describes the tendency for people who share a common friend to become friends themselves. In the context of graph theory, which is often used to model social networks, triadic closure occurs when three nodes (representing individuals or entities) form a closed triangle of edges (representing relationships or connections). To calculate it the previous metrics calculated will be used as well as transitivity.

Transitivity is a measure that captures the overall likelihood of triangles being formed in a network. It's a global metric that represents the degree to which nodes in a network tend to create closed triads (triangles).

To calculate it

```
In 37   1  #calculate transitivity
        2  nx.transitivity(final)
           Executed at 2024.01.20 17:56:09 in 56ms

Out 37     0.01423194045000846
```

In a directed network of 204 nodes with 31 unique triangles, predominantly involving two central nodes, a transitivity of 0.014, and an average clustering coefficient of 0.355 with peaks at 0, 0.5, and 1 in its distribution, the results suggest a moderate presence of triadic closure. The low transitivity indicates that, while triangles exist, they are not widespread throughout the network, implying that closed triads are not a dominant structure. The clustering coefficient reveals a network with diverse connectivity: a significant number of nodes do not participate in any triangles (peak at 0), some are part of partially connected triads (peak at 0.5), and others are in highly interconnected

clusters (peak at 1). This structure points to a network with a heterogeneous composition, containing highly connected subgroups around key nodes, as well as nodes with fewer or no such local connections, highlighting the centrality of certain nodes in facilitating triadic closures.

9. Bridges and Local Bridges

9.1 Bridges

In the context of network theory, bridges are edges (or nodes) that, if removed, would increase the number of disconnected components of the network. In other words, a bridge connects two parts of a network that would otherwise be separate components.

Using NetworkX it is possible to check if there are any bridges in the network

```
In 38   1  #make final graph undirected
        2  final_undirected = final.to_undirected()
        3
        4  nx.has_bridges(final_undirected)
           Executed at 2024.01.20 17:56:10 in 37ms

Out 38     True
```

As well as how many are there

```
In 39   1  #find bridges
        2  len(list(nx.bridges(final_undirected)))
           Executed at 2024.01.20 17:56:12 in 82ms

Out 39     60
```

Diving deeper into the list of available bridges

[45]:
```
list(nx.bridges(final_undirected))
#show the label of each node that corresponds to the node id
for i in list(nx.bridges(final_undirected)):
    print(final.nodes[i[0]]['label'], final.nodes[i[1]]['label'])
```

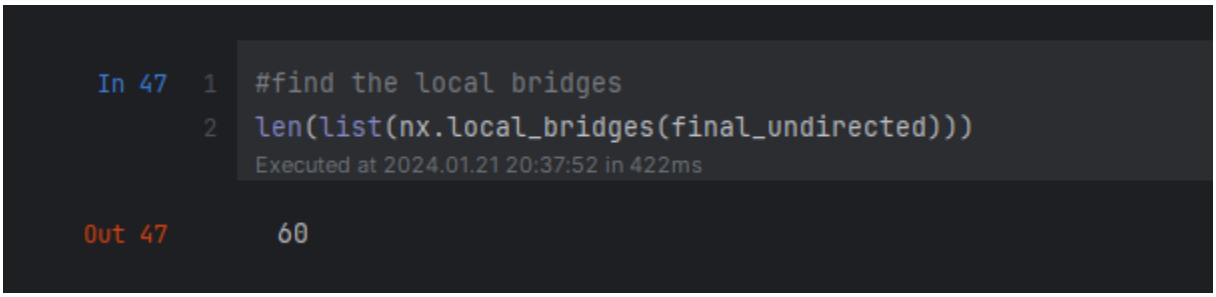
```
Swammy ZackRawrr
Kavos ZackRawrr
Guiden ZackRawrr
Trailer Crunch ZackRawrr
ENFANT TERRIBLE ZackRawrr
Grunge ZackRawrr
Yiman ZackRawrr
OFWGKTA ZackRawrr
Big Think ZackRawrr
RobinGaming ZackRawrr
Law & Lumber ZackRawrr
Timothy Cain ZackRawrr
steveo ZackRawrr
coppersan ZackRawrr
Zexrow ZackRawrr
Play New World ZackRawrr
Rob2628 ZackRawrr
Tasting History with Max Miller ZackRawrr
Like Stories of Old ZackRawrr
Guns Nerds and Steel ZackRawrr
Protoxx ZackRawrr
Austin Goh ZackRawrr
ZackRawrr MxR
ZackRawrr Depressed Nousagi
ZackRawrr Geopold
ZackRawrr GameLight
ZackRawrr Akshon Esports
ZackRawrr wudijo
ZackRawrr g0at
ZackRawrr OniGiri
ZackRawrr Path of Exile
ZackRawrr AzAMOuS
ZackRawrr Velvet Tube ãƒ™ãƒ«ãƒ™ãƒƒãƒˆãƒ•ãƒ¥ãƒ¼ ë²¨ë²³íŠœë¸Œ
ZackRawrr Dry Creek Wrangler School
ZackRawrr Chad Zuber
ZackRawrr Alex Moukala Music
ZackRawrr Mortismal Gaming
ZackRawrr Your Overwatch
ZackRawrr Fast AF âš¡
ZackRawrr Sarthe
ZackRawrr Zepla HQ
ZackRawrr Chubbyemu
ZackRawrr TiÃ¼sto
ZackRawrr Skrillex
ZackRawrr HeelvsBabyface
ZackRawrr Jimmy Kimmel Live
ZackRawrr Rev says desu
ZackRawrr Azura DragonFaether
ZackRawrr Misshapen Chair
ZackRawrr Two Minute Papers
ZackRawrr CatDany â€" Asmongold Highlights
ZackRawrr Toyhouze
ZackRawrr TheQuartering
ZackRawrr bycloud
ZackRawrr PhysiquesOfGreatness
ZackRawrr Necrit
ZackRawrr Max Derrat
ZackRawrr Chef Brian Tsao
ZackRawrr Aztecross
ZackRawrr Guga
```

It is evident that ZackRawrr is in all the bridges within the network, being involved in each either through incoming or outgoing connections. This signifies that ZackRawrr is a critical connector, serving as the primary link between various network components or clusters, and holding substantial control over the flow of information, resources, or interactions. This central position implies that the node is a gatekeeper, crucial for maintaining network integrity, as its removal or failure could lead to significant network fragmentation and disrupt connectivity.

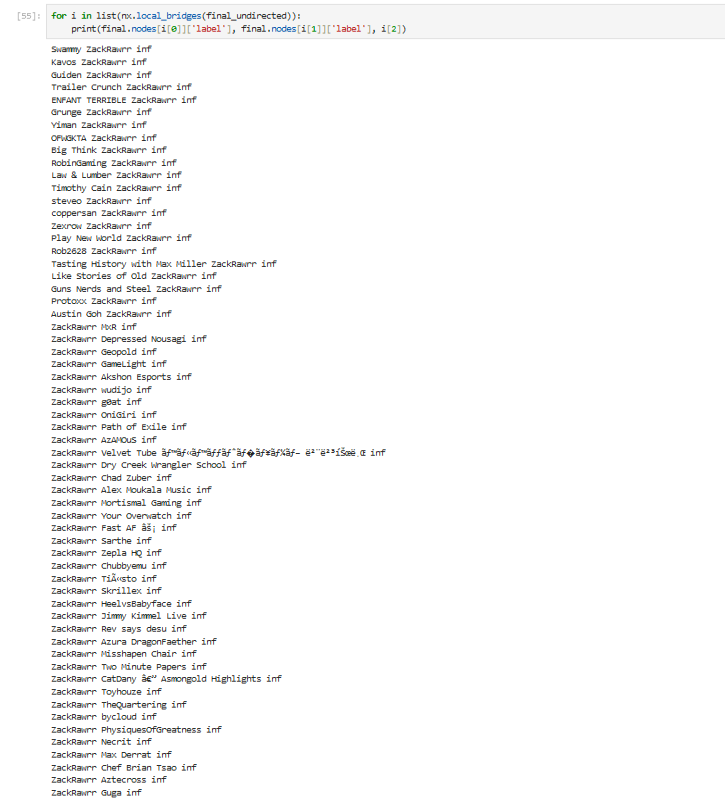
9.2 Local Bridges

Local bridges in a network are unique edges that connect two nodes without shared neighbors, distinguishing them from regular bridges which might connect nodes with mutual connections. These local bridges serve as exclusive pathways between distinct parts or communities within the network, playing a crucial role in the structure of the network by indicating the presence of loosely connected groups.

Through NetworkX it is possible to find the number of the local bridges in the network



It is evident that the local bridges are the same number as the regular bridges.



Also all the bridges have inf in their length meaning that there is no alternative path for those nodes. Lastly it is important to check if the local bridges are the same as the normal bridges

[52]:
```python
# Find bridges and local bridges
bridges = set(nx.bridges(final_undirected))
local_bridges = set(nx.local_bridges(final_undirected, with_span=False))  # with_span=False to get edges only

# Check if they are the same
are_same = bridges == local_bridges

# Find differences
bridges_not_local = bridges - local_bridges  # Bridges that are not local bridges
local_not_bridges = local_bridges - bridges  # Local bridges that are not bridges

# Output results
print("Are bridges and local bridges the same:", are_same)
print("Bridges that are not local bridges:", bridges_not_local)
print("Local bridges that are not bridges:", local_not_bridges)
```

```
Are bridges and local bridges the same: True
Bridges that are not local bridges: set()
Local bridges that are not bridges: set()
```

Indeed, this reaffirms the crucial role of ZackRawrr in maintaining the structural integrity and overall connectivity of the network. ZackRawrr's position as a gatekeeper is pivotal, controlling the flow of information across a significant portion of the network.

10. Homophily - Assortativity

Homophily - Assortativity, in the context of network theory, refers to a network's tendency to connect nodes with similar characteristics or properties. It's a measure of the correlation in the attributes, especially the degrees, of connected nodes within the network.

10.1 Attribute Assortativity - Homophily

Attribute assortativity quantifies the preference for connections between nodes with similar attribute values. These attributes can be anything defined on the nodes, such as age, language, type, category, etc.

The mesure that will be used to calculate the homophily of the network is the Assortativity Coefficient. The Assortativity Coefficient for attributes is a number between -1 and 1 that quantifies the likelihood of nodes in a network to connect with other nodes that have similar attributes or characteristics. A positive coefficient indicates that nodes with similar attributes tend to connect with each other (assortative mixing), while a negative coefficient suggests that nodes with different attributes tend to connect (disassortative mixing). A coefficient near zero implies no particular preference for connections based on the attribute.

To calculate the homophily of the attributes of the ZackRawrr network the function attribute_assortativity_coefficient() from NetworkX, will be used.

```
In 130
#calculate the assortativity with all attributes

as_subcount = nx.attribute_assortativity_coefficient(final, 'subscribercount')

as_videocount = nx.attribute_assortativity_coefficient(final, 'videocount')

as_viewcount = nx.attribute_assortativity_coefficient(final, 'viewcount(100s)')

as_country = nx.attribute_assortativity_coefficient(final, 'country')

as_publ = nx.attribute_assortativity_coefficient(final, 'daysactive')

print("Subscriber Count Assortativity:", as_subcount)
print("Video Count Assortativity:", as_videocount)
print("View Count Assortativity:", as_viewcount)
print("Country Assortativity:", as_country)
print("Days Active Assortativity:", as_publ)
```

The above code calculates the assortativity for all the attributes of the network and gives the following results

```
Subscriber Count Assortativity: -0.0071151078897606 95
Video Count Assortativity: -0.007336052359536371
View Count Assortativity: -0.00827272425359503
Country Assortativity: -0.0005237037109546565
Days Active Assortativity: -0.008272724253595034
```

The assortativity coefficients for Subscriber Count, Video Count, View Count, Days Active and Country all being very close to zero and slightly negative, indicate a general lack of correlation in these attributes among connected nodes. Specifically, the slightly negative values in subscriber count (-0.007), video count (-0.007), country(-0.0005), days active(-0.008) and view count (-0.008) assortativities suggest that there is no significant tendency for nodes to connect with others having similar numbers in these categories. Overall, these results point to the fact that the formation of connections in the network is not strongly influenced by similarities or differences in these specific attributes, suggesting that other, unmeasured factors might play a more significant role in how connections are established.

10.2 Attribute Homophily based on the Number of Cross Edges

Cross-edges are the edges that connect nodes from different groups. Count the number of edges in the network that link nodes belonging to different groups.

One way to quantify homophily is by using the proportion of within-group edges as opposed to cross-group edges. The formula is:

$$\text{Homophily} = 1 - \frac{\text{Number of Cross-Edges}}{\text{Total Number of Edges}}$$

In this context, cross-edges refer to links between nodes that are not similar, and the total number of edges is the total links in the network. The formula calculates the proportion of edges that are not between similar nodes (cross-edges) and subtracts this from 1 to give a measure of homophily

This formula yields a value between 0 and 1, where a value closer to 1 indicates higher homophily (more within-group connections), and a value closer to 0 indicates lower homophily (more cross-group connections)

Using NetworkX it is possible to calculate the above formula of homophily for categorical attributes.

It is clear that some of the attributes of the network are not categorical, so groups will be created to make the numerical values into cagegorical.

10.2.1 Subscriber Count Attribute

With the help of python the channels' subscriber count is divided into three categories, high-range subscriber count with channels that have subscribers above 10 million, low-range subscriber count that contains channels with a subscriber count below 1million and mid-range subscriber count that has the channels with values between 1million and 10 million.

The below code snippet was used to categorize the channels as well as find the homophily based on the above formula.

```
In 104
#calculate the homophily based on the number of cross-subscription edges
import pandas as pd

# Create a dataframe with the subscriber count of each node
sub_df = pd.DataFrame({'subscribercount': [final.nodes[node]['subscribercount'] for node in final.nodes()]})

# Define thresholds for low, mid, and high subscriber counts
low_threshold = 1000000
high_threshold = 10000000

# Categorize subscriber counts
def categorize_subscriber_count(count):
    if count <= low_threshold:
        return 'low'
    elif count <= high_threshold:
        return 'mid'
    else:
        return 'high'

# Add the category to each node in the graph
for node in final.nodes():
    final.nodes[node]['category_sub'] = categorize_subscriber_count(final.nodes[node]['subscribercount'])

# Calculate the number of cross-category edges
cross_sub_edges = len([edge for edge in final.edges() if final.nodes[edge[0]]['category_sub'] != final.nodes[edge[1]]['category_sub']])

# Calculate the total number of edges
total_edges = final.number_of_edges()

# Calculate the homophily
homophily = 1 - (cross_sub_edges / total_edges)

# Output the results
print("Number of cross-sub edges:", cross_sub_edges)
print("Number of edges:", total_edges)
print("Homophily:", homophily)
Executed at 2024.01.23 14:26:30 in 364ms
```

The results of the code can be seen here

```
Number of cross-sub edges: 222
Number of edges: 551
Homophily: 0.5970961887477314
```

A homophily index of 0.59 indicates that there is a moderate level of homophily in the network based on the categorization of subscriber counts into 'low-range', 'mid-range', and 'high-range'.

Also it is possible to investigate which of those categories tend to form connections with nodes of the same category using the formula 2pq where p is the proportion of all edges in the network that connect nodes of the same type (internal connections) and q is he proportion of all edges in the network that connect nodes of different types (external connections).

```
p = sum(1 for node, attr in final.nodes(data=True) if attr.get('category_sub') == 'low') / final.number_of_nodes()
q = 1 - p
expected_cross_edges = 2 * p * q * total_edges

# Calculate the actual number of cross-edges
actual_cross_edges = sum(1 for u, v in final.edges() if final.nodes[u]['category_sub'] != final.nodes[v]['category_sub'])

# Implement the homophily measure
homophily = actual_cross_edges < expected_cross_edges

print(f'Expected number of cross-edges: {expected_cross_edges}')
print(f'Actual number of cross-edges: {actual_cross_edges}')
print(f'Evidence of homophily: {homophily}')
```

And the results

```
Expected number of cross-edges: 243.06175509419455
Actual number of cross-edges: 222
Evidence of homophily: True
```

Since the actual number of cross-edges is less than the expected number, the conclusion is that there is evidence of homophily. This means that nodes in the low-range category tend to connect with other nodes within the same category more often than would be expected by chance.

10.2.2 Video Count Attribute

With the same method it is also possible to categorize the videocount attribute to check for homophily in the network. So the attribute is split into three different categories similar to the subscriber count, with low-count that contains the channels with a lower video count than 300, high-count that has the channels with a video count higher than 1000 and lastly the mid-count with the channels with video count between 300 and 1000.

So, after modifying the subscriber count attribute code to fit this attribute

```
In 142
#calculate the homophily based on the number of cross-video edges
import pandas as pd

# Create a dataframe with the subscriber count of each node
video_df = pd.DataFrame({'videocount': [final.nodes[node]['videocount'] for node in final.nodes()]})

# Define thresholds for low, mid, and high subscriber counts
low_threshold = 300
high_threshold = 1000

# Categorize subscriber counts
def categorize_videos_count(count):
    if count <= low_threshold:
        return 'low'
    elif count <= high_threshold:
        return 'mid'
    else:
        return 'high'

# Add the category to each node in the graph
for node in final.nodes():
    final.nodes[node]['category_video'] = categorize_videos_count(final.nodes[node]['videocount'])

# Calculate the number of cross-category edges
cross_category_edges = len([edge for edge in final.edges() if final.nodes[edge[0]]['category_video'] != final.nodes[edge[1]]['category_video']])

# Calculate the total number of edges
total_edges = final.number_of_edges()

# Calculate the homophily
homophily = 1 - (cross_category_edges / total_edges)

# Output the results
print("Number of cross-video edges:", cross_category_edges)
print("Number of edges:", total_edges)
print("Homophily:", homophily)
Executed at 2024.01.23 19:24:12 in 360ms
```

The results are

```
Number of cross-video edges: 377
Number of edges: 551
Homophily: 0.3157894736842105
```

The network has a certain level of homophily, but it is not high. This value suggests that while there is a small a preference for nodes to connect within the same category, a significant number of connections still occur between nodes of different categories.

10.2.3 View Count Attribute

Once again the view into three different categories, but this time percentiles will be used for the categorization because there is a very broad distribution of views. The category low viewed channels contains all the channels below the 33rd percentile, while the categotry high viewed channels includes all the channels above the 66th percentile. Lastly the moderate viewed category has all the channels in between those percentiles.

Once again the modified code,

```
In 143
#calculate the homophily based on the number of cross-view edges
import pandas as pd

# Create a dataframe with the subscriber count of each node
views_df = pd.DataFrame({'viewcount(100s)': [final.nodes[node]['viewcount(100s)'] for node in final.nodes()]})

# Define thresholds for low, mid, and high view counts with the percentiles
low_threshold = views_df['viewcount(100s)'].quantile(0.33)
high_threshold = views_df['viewcount(100s)'].quantile(0.66)

# Categorize subscriber counts
def categorize_view_count(count):
    if count <= low_threshold:
        return 'low'
    elif count <= high_threshold:
        return 'mid'
    else:
        return 'high'

# Add the category to each node in the graph
for node in final.nodes():
    final.nodes[node]['category_views'] = categorize_view_count(final.nodes[node]['viewcount(100s)'])

# Calculate the number of cross-category edges
cross_views_edges = len([edge for edge in final.edges() if final.nodes[edge[0]]['category_views'] != final.nodes[edge[1]]['viewcount(100s)']])

# Calculate the total number of edges
total_edges = final.number_of_edges()

# Calculate the homophily
homophily = 1 - (cross_views_edges / total_edges)

# Output the results
print("Number of cross-views edges:", cross_views_edges)
print("Number of edges:", total_edges)
print("Homophily:", homophily)
Executed at 2024.01.23 19:25:05 in 353ms
```

And its results

```
Number of cross-views edges: 551
Number of edges: 551
Homophily: 0.0
```

A homophily value of 0 shows that there is no observable preference for nodes to connect with other nodes that are similar to them in terms of the view count attribute. It means that connections between nodes are as likely to occur between dissimilar nodes as they are between similar nodes, suggesting a completely random pattern of connectivity.

10.2.4 Days Active Attribute

By repeating the process of the previous attributes, three categories for days active are created with the lowest days active being the nodes that have been active for less than 2500 days, with the highest days active being the nodes that have been active for more than 4500 days and finally the moderate days active being the nodes that have been active between 2500 and 4500 days.

The code

```
In 147
#calculate the homophily based on the number of days active edges
import pandas as pd

# Create a dataframe with the subscriber count of each node
daysactive_df = pd.DataFrame({'daysactive': [final.nodes[node]['daysactive'] for node in final.nodes()]})

# Define thresholds for low, mid, and high view counts with the percentiles
low_threshold = 2500
high_threshold = 4500

# Categorize subscriber counts
def categorize_subscriber_count(count):
    if count <= low_threshold:
        return 'low'
    elif count <= high_threshold:
        return 'mid'
    else:
        return 'high'

# Add the category to each node in the graph
for node in final.nodes():
    final.nodes[node]['category_days'] = categorize_subscriber_count(final.nodes[node]['daysactive'])

# Calculate the number of cross-category edges
cross_days_edges = len([edge for edge in final.edges() if final.nodes[edge[0]]['category_days'] != final.nodes[edge[1]]['daysactive']])

# Calculate the total number of edges
total_edges = final.number_of_edges()

# Calculate the homophily
homophily = 1 - (cross_days_edges / total_edges)

# Output the results
print("Number of cross-views edges:", cross_days_edges)
print("Number of edges:", total_edges)
print("Homophily:", homophily)
Executed at 2024.01.23 19:31:03 in 357ms
```

The results

```
Number of cross-views edges: 551
Number of edges: 551
Homophily: 0.0
```

Once again homophily value of 0 this means that connections between nodes are as likely to occur between dissimilar nodes as they are between similar nodes, suggesting a completely random pattern of connectivity.

10.2.5 Country Attribute

Lastly the country attribute is categorical so the is no need for categorization.

Starting, with the python code

```
In 81
#calculate homophily based on the number of cross-country edges
import pandas as pd

# Create a dataframe with the country of each node
country_df = pd.DataFrame([final.nodes[node]['country'] for node in final.nodes()], columns=['country'])

# Calculate the number of cross-country edges
cross_country_edges = len([edge for edge in final.edges() if final.nodes[edge[0]]['country'] != final.nodes[edge[1]][
    'country']])

#calculate the total number of edges
total_edges = final.number_of_edges()

# Calculate the homophily
homophily = 1 - (cross_country_edges / total_edges)

# Output the results
print("Number of cross-country edges:", cross_country_edges)
print("Number of edges:", total_edges)
print("Homophily:", homophily)
Executed at 2024.01.23 13:35:02 in 344ms
```

And the results

```
Number of cross-country edges: 371
Number of edges: 551
Homophily: 0.3266787658802178
```

The network has a certain level of homophily, but it is not high. This value suggests that while there is a very small preference for nodes to connect within the country, a significant number of connections still occur between nodes of different categories.

```
p = sum(1 for node, attr in final.nodes(data=True) if attr.get('country') == 'US') / final.number_of_nodes()
q = 1 - p
expected_cross_edges = int(2 * p * q * total_edges)

# Calculate the actual number of cross-edges
actual_cross_edges = sum(1 for u, v in final.edges() if final.nodes[u]['country'] != final.nodes[v]['country'])

# Implement the homophily measure
homophily = actual_cross_edges < expected_cross_edges

print(f'Expected number of cross-edges: {expected_cross_edges}')
print(f'Actual number of cross-edges: {actual_cross_edges}')
print(f'Evidence of homophily: {homophily}')
```

Using the 2*p*q formula for the countries with the most nodes we can see that none of them mainly prefer to form connections with nodes of the same country

US homophily

```
Expected number of cross-edges: 273
Actual number of cross-edges: 371
Evidence of homophily in US nodes: False
```

AU homophily

```
Expected number of cross-edges: 31
Actual number of cross-edges: 371
Evidence of homophily in AU nodes: False
```

GB homophily

```
Expected number of cross-edges: 106
Actual number of cross-edges: 371
Evidence of homophily in GB nodes: False
```

CA homophily

```
Expected number of cross-edges: 65
Actual number of cross-edges: 371
Evidence of homophily in CA nodes: False
```

10.3 Degree Assortativity - Homophily

Degree assortativity measures the correlation between the degrees of connected nodes within a network. It evaluates whether high-degree nodes tend to be connected to other high-degree nodes and whether low-degree nodes tend to be connected to other low-degree nodes.

The degree assortativity is quantified by a coefficient that ranges from -1 to 1. A positive coefficient indicates assortative mixing, meaning that nodes preferentially attach to similar nodes in terms of degree. A negative coefficient indicates disassortative mixing, where nodes tend to connect with others that have a different degree. A coefficient around zero suggests no particular preference for connections based on degree.

To calculate the homophily of the attributes of the ZackRawrr network the function degree_assortativity_coefficient() from NetworkX, will be used.

```
In 39
#calculate the assortativity for all degrees

as_in = nx.degree_assortativity_coefficient(final, x='in', y='in')

as_out = nx.degree_assortativity_coefficient(final, x='out', y='out')

as_out_in = nx.degree_assortativity_coefficient(final, x='out', y='in')

as_in_out = nx.degree_assortativity_coefficient(final, x='in', y='out')

print("In-Degree Assortativity:", as_in)
print("Out-Degree Assortativity:", as_out)
print("Out-In-Degree Assortativity:", as_out_in)
print("In-Out-Degree Assortativity:", as_in_out)
```

The above code calculates the assortativity for all the attributes of the network and gives the following results

```
In-Degree Assortativity: -0.3396311302305901
Out-Degree Assortativity: -0.09000906601265317
Out-In-Degree Assortativity: -0.4552031893700823
In-Out-Degree Assortativity: 0.019889618820661776
```

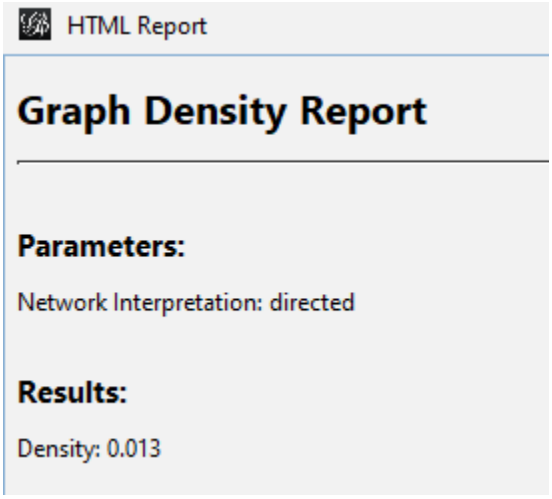
The degree assortativity coefficients in the network reveal distinct patterns of node connectivity based on in-degrees and out-degrees. With an in-degree assortativity of -0.339, there's a clear disassortative mixing trend, showing that nodes receiving many connections (high in-degrees) typically connect to nodes with fewer incoming connections (loq in-degrees). The out-degree assortativity at -0.09, although less pronounced, similarly suggests a slight disassortative pattern where nodes making numerous connections (high out-degrees) tend to link to nodes with fewer outgoing

connections(lower out-degree). The out-in-degree assortativity coefficient of -0.455 shows a stronger disassortative pattern, where nodes actively connecting to many others (high out-degree) predominantly connect to those that receive connections from fewer nodes(low in-degree). Conversely, the in-out-degree assortativity at 0.0198, being slightly positive, indicates a very weak assortative mixing, where nodes with many incoming connections (high in-degree) have a slight tendency to connect to other nodes that also make many connections (high out-degree). Overall, these coefficients suggest that ZackRawrr's network generally exhibits disassortative mixing in terms of node degrees, with the most significant tendency being for active nodes (high out-degree) connecting to less active nodes (low in-degree).

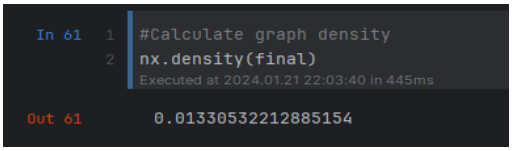
11. Graph Density

Graph density is a measure used in network analysis to quantify how close a network is to being fully connected. It reflects the proportion of potential connections in a network that are actual connections. The density of a graph is calculated as the ratio of the number of actual edges in the network to the number of possible edges.

With Gephi's graph density function, the result is 0.013



With NetworkX the same result is also found



In ZackRawrr's Network of 204 nodes and 551 edges, a density of 0.013 highlights that the network is far from fully interconnected. This low density value, indicates that while there are connections present, they are relatively few compared to the total possible

connections in a fully connectednetwork of this scale. The fact that many nodes only connect to one other node further accentuates the network's sparse structure, confirming a prevalence of peripheral or isolated nodes.

12 Community structure

Gephi will be used as the main tool for community detection and its visualization. The main mesure in which community detection will be based on, is modularity, which can be fount in the Community Detection sections of gephi

Modularity is a measure used in network analysis to quantify the strength of division of a network into modules, also called communities or clusters. In essence, modularity measures the density of links inside communities as compared to links between communities. Modularity is a scalar value between -1 and 1 that measures the degree to which a network can be divided into clearly delineated groups or communities. A high modularity indicates a structure where there are dense connections between the nodes within modules but sparse connections between nodes in different modules.

Running Gephi’s algorithm we get a modularity of 0.326

Results:

Modularity: 0.326

But for this analysis modularity with resolution will be used because higher resolution parameter makes the algorithm more sensitive to smaller communities. It effectively reduces the tendency to merge smaller communities into larger ones, allowing for the identification of more, but smaller, groups within the network. And in this situation modularity with resolution provides a clearer result which is also indicated by the fact that modularity with resolution is 0.402.

Modularity Report

Parameters:

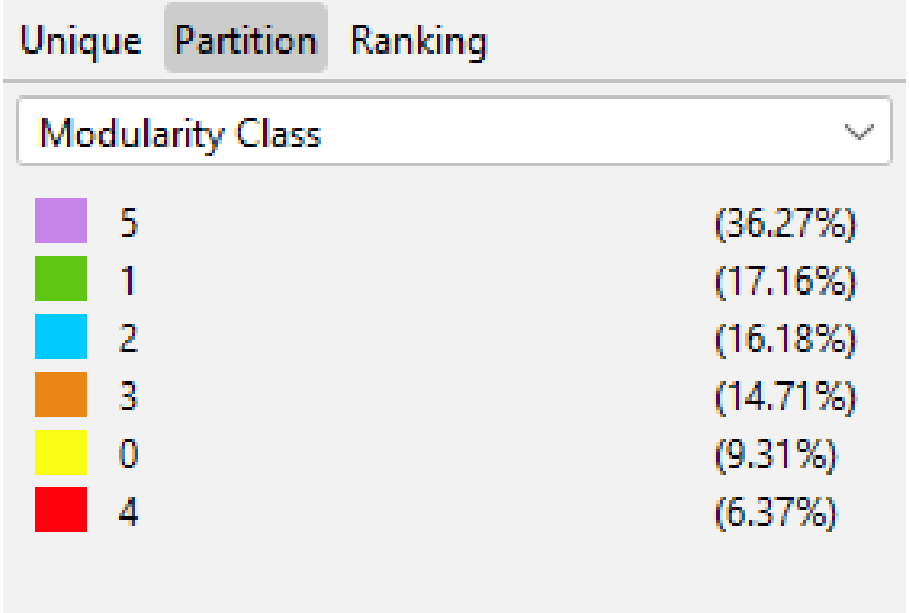
Randomize: On
Use edge weights: On
Resolution: 1.15

Results:

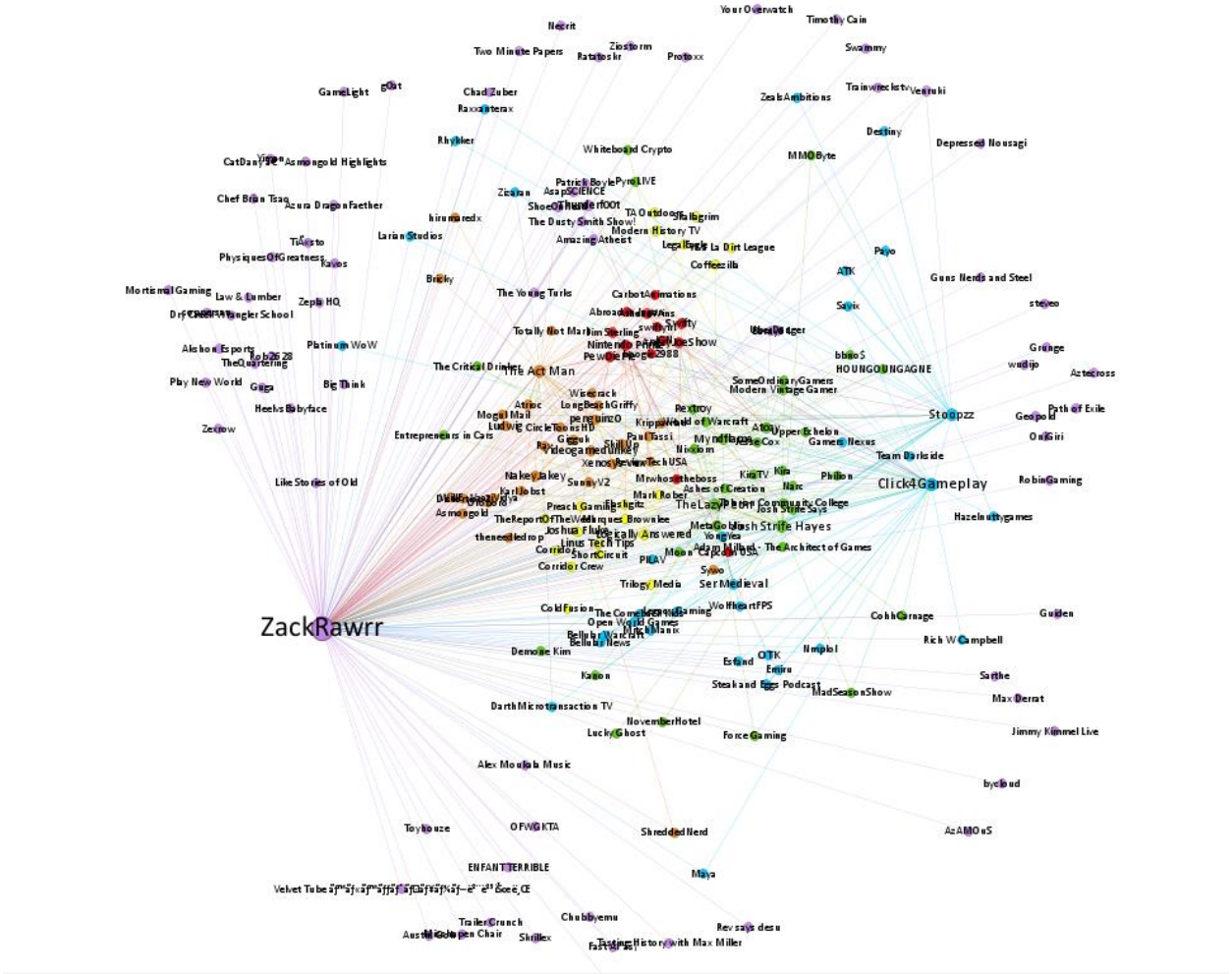
Modularity: 0.326
Modularity with resolution: 0.404
Number of Communities: 6

The function returns 6 communities with this distribution

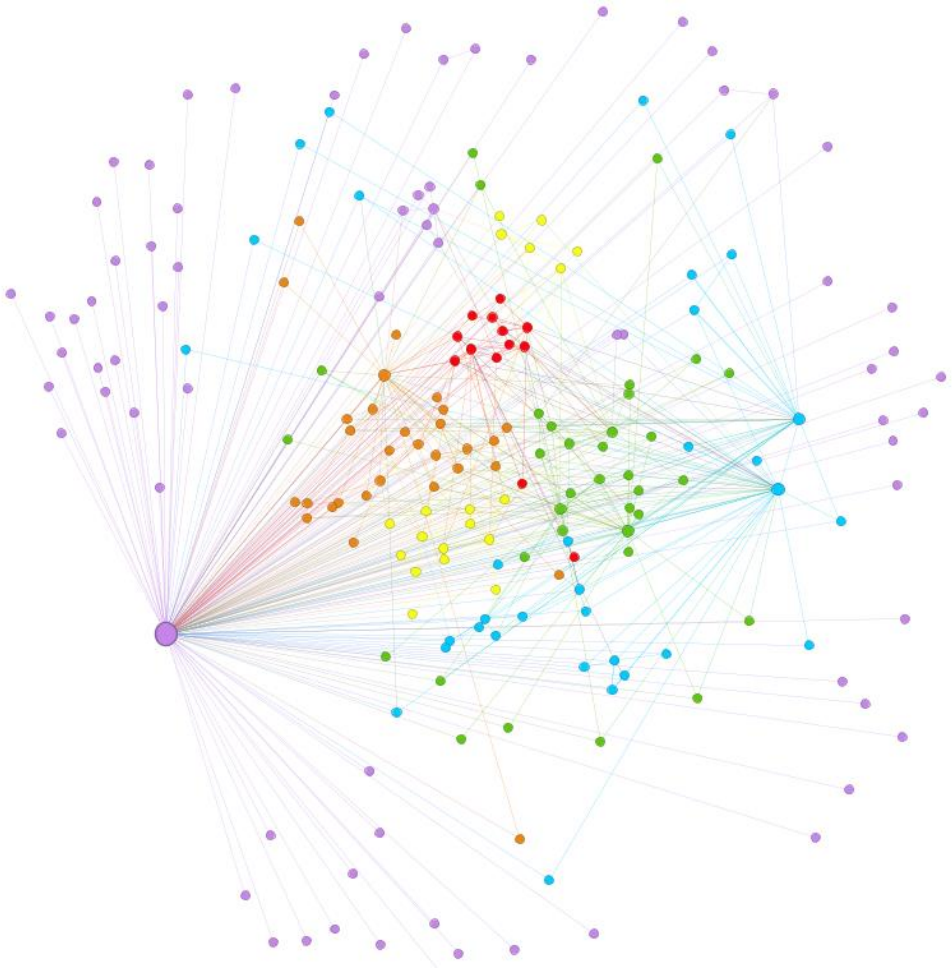

Size Distribution
Size (number of nodes)
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
-1
0
1
2
3
4
5
6
Modularity Class


Unique
Partition
Ranking
Modularity Class
5 (36.27%)
1 (17.16%)
2 (16.18%)
3 (14.71%)
0 (9.31%)
4 (6.37%)

By colouring each node based on modularity class we get a visualization with all the communities



And one without the labels so the communities can be seen clearer



12.1 Community analysis

From the above analysis it is clear that there are 6 communities in ZackRawrr's network. It is also possible to take a better look at each cluster and find important information about the structure of them and for the network as a whole.

12.1.1 Insightful Analysis Cluster

The first cluster of the network is the cluster 0 with the yellow colour and a percentage of 9.31% or 19 nodes of the total 204. This cluster is called insightful analysis because the channels within this it are characterized by their deep dives into subjects like technology, history, legal matters, and consumer goods. Some on those channels are Marques Brownlee, LegalEagle, Modern History TV, Coffeezilla and ShortCircuit. They often approach their topics with a critical eye and aim to offer their audience a thorough understanding of the subject matter. The content these channels produce is not just informative but also thought-provoking, encouraging viewers to consider the wider implications of the products, events, and phenomena they cover.

There are 3 channels that are at the forefront of this cluster based on the measures that have been calculated so far.

Linus Tech Tips is the channel with the highest in-degree (10), closeness centrality (0.66) and eigenvector centrality (0.12) of the cluster, they also have the second highest betweenness centrality (0.002) of the cluster as well as in-degree and closeness centrality overall . Linus Tech Tips stands out in the tech community for offering comprehensive reviews and expert analysis on a wide array of technology products. With a keen eye for detail, Linus Sebastian and his team dissect complex tech topics into engaging and understandable content, making it a go-to source for both enthusiasts and professionals. The channel's blend of in-depth hardware evaluations, tutorials, and tech discussions embodies the essence of insightful analysis in tech.

Logically Answered has the highest degree (17), out-degree (16) and degree centrality (0.083) of the cluster while also holding the eighth place overall in those categories. Logically Answered is a channel that delivers analytical content focused on dissecting and understanding various topics from smaller ones, such as why people love the Lenovo think pad to more complex ones such as why EU is against big tech companies. Through breakdowns and clear explanations, the channel caters to viewers seeking to enhance their critical thinking and problem-solving skills. Its content serves as a resource for those looking to delve deeper into the realm of logic and reasoning within an analytical framework.

Joshua Fluke has the second highest degree(13), in-degree, out-degree(10) and degree centrality (0,064) while also being the node with the highest betweenness centrality of the cluster (0,0005). Joshua Fluke's channel provides a candid look into the tech industry, offering career advice, coding tutorials, and personal insights into the life of a software developer. His content is known for its direct approach to career development, job hunting strategies, and the realities of working in tech.

An honorable mention goes Coffeezilla who score high in all of the measures except out-degree and degree centrality but doesn’t have the highest spot in the measures of the cluster.

12.1.2 Strategic Gameplay Cluster

The second cluster of the network is the cluster 1 of the diagram with the colour green that has 17.16% or 35 nodes of the total 204. It's named Strategic Gameplay because it primarily comprises channels that are dedicated to gaming, with a significant emphasis on strategy and commentary on game mechanics and design, such as Tolarian Community College, Rextroy, MetaGoblin, Demone Kim and Adam Millard - The Architect of Games. The channels in this cluster cater to an audience that appreciates tactical depth in games, whether it be through comprehensive guides, discussions on game development, or walkthroughs that reveal the strategic layers of complex gameplay.

The most important node of the Cluster is Josh Strife Hayes which is a channel that has been emphasized before in this analysis. Josh Strife Hayes is first in each measure that has been calculated so far for his cluster as well as holding top positions in those measures overall, including a first place in the eigenvector centrality (0.2), a second place in betweenness centrality(0.02) and in-degree(10) and a third place in degree(36), and in degree centrality (0.17) as well as fourth in closeness centrality(0.064).

Another important node of this cluster is The Lazy Peon which comes second after Josh Strife Hayes in every measure in his cluster except Eigenvector centrality where he comes fourth . This node also holds top positions (5^{th} and 6^{th}) in the overall measures of the network, except closeness centrality . Their content includes critical reviews, gameplay strategies, and previews of upcoming titles, all delivered with a deep understanding of the intricacies and tactical nuances of the games they explore. The channel not only entertains but also educates its viewers on the strategic elements that could define and elevate their gaming experience.

Another honorable mention is Atozy, who is the third largest node in the cluster in terms of measures, but doesn't hold the highest position in any of them. The node also holds a position in the top 10 nodes in most measures in the overall network.

12.1.3 Gaming Nexus Cluster

The third cluster of the network is the cluster 2 of the visualization with the colour blue that has 16.18% of the overall network or 33 nodes. The name of Gaming nexus reflects the clusters focus on fostering a gaming community and providing instructional content such as, guides and playthroughs to help players through various aspects of gaming. Some main examples of the above purpose are Destiny, Rhykker, Click4Gameplay and Hazelnuttygames. The channels of this cluster cater to an audience that values deep engagement with gaming content, from those seeking the excitement of shared gaming experiences to individuals looking for detailed guides to enhance their play. This cluster serves as a collective resource for both casual and hardcore gamers who wish to connect with the community and improve their mastery of games.

The two nodes that are top of this cluster have already been touched upon before in this analysis. The first one is Click4Gameplay, who holds the top position in metrics such as degree (49), out-degree(46), degree centrality(0.24) while also having the second highest position as far as betweenness centrality is concerned (0.0049). It is also important to note that Click4Gameplay holds the second highest position, for the degree measures mentioned above, in the network as a whole. The second one is Stoopzz who holds the second position in all degree metrics as Click4Gameplay, meaning degree (35),in-degree (5), out-degree (30), degree centrality (0.17) as well as the first position in betweenness centrality (0.054). Stoopzz also is also holding top positions in the overall network being third in out-degree as well as betweenness centrality.

An important honorable mention is LongYea that has some top positions in the cluster in terms of measures but fails to reach top positions overall.

12.1.4 Commentary and Review Cluster

The fourth cluster of the network is cluster 3 with the colour orange that has 14,17% or 30 of the 204 overall nodes of the network. This cluster is called Commentary and Review due to its members' focus on providing reactions and commentary to a wide array of content, from video games and technology to pop culture and current events, coupled with in-depth reviews. Some prime examples are Ludwig, Mogul Mail, penguinz0, The Act Man and Karl Jobst.

The most important node of the cluster is channel The Act Man, which has already been seen before, because it holds the top position in the network in all of the degree measures (degree(35),out-degree(30),degree centrality(0.17)), except in-degree, as well as degree and betweenness centrality (0.0084). The Act Man also is a top candidate in terms of the overall network, by having the fouth position overall in degree and the third in out-degree and degree centrality, as well as the fifth highest betweenness centrality.

The second position in the network is held by penguinz0 who has the highest in-degree(10) and closeness centrality (0.066) while also being second in eigenvector centrality(0.15). Additionally penguinz0 is tied with with Linus Tech Tips and Josh Strife hays for the second highest in-degree overall as well as with Linus Tech Tips for the second highest centrality overall. Within the React & Review cluster, penguinz0 also known as Cr1TiKaL stands as a channel that thrives on a blend of humorous reactions and candid reviews across a spectrum of subjects, from gaming and internet culture to real-life events. The channel's content is characterized by its spontaneous and often comedic commentary, resonating with viewers who appreciate a mix of satire and sincerity.

12.1.5 Outlets for News and Information Cluster

The fifth cluster of the network is cluster 4, marked with the red color. It is the smallest cluster in the network and contains 6.37% or 13 of the 204 overall nodes. It is called the Outlets for News and Information because it primarily consists of channels that specialize in delivering news and updates various domains, such as gaming, world events, and entertainment. Some of the channels from the cluster that share this common ideology are Nintendo Prime, IGN, AngryJoeShow, Capcom USA and Jim Sterling.

There are two main channels that score the highest cluster-wide, in 3 different measures. Those are Nintendo Prime and Pewdiepie.

Nintendo prime holds the top spot in all of the degree measures except in degree with a degree of 16, out degree of 15 and a degree centrality of 0.078. The channel is also 9th overall position in the network in those measures. Nintendo Prime is a channel that zeroes in on the world of Nintendo, offering news and in-depth analysis of games, consoles, and broader information within the Nintendo ecosystem. The content is tailored for fans of the brand, from casual players to die-hard enthusiasts, providing a mix of informational updates and engaging commentary. It stands as a dedicated source for all things Nintendo, capturing the excitement and nostalgia that the brand evokes among its audience.

Pewdiepie is in the top spot of in-degree (11), closeness centrality(0.073) and eigenvector centrality(0.15), in terms of its cluster, while also holding the top spot overall in in-degree and closeness centrality and the fifth spot in eigenvector centrality. PewDiePie, one of the biggest creators on the YouTube platform, is known for its incredibly diverse array of content, ranging from Let's Play video game commentaries to comedic vlogs and satirical shows. Among his various series, Pew News became particularly notable, where PewDiePie donned the hat of a newscaster to discuss and humorously comment on various topics and current events from around the world, often bringing his unique perspective to the forefront of internet culture.

12.1.6 The Streaming Bubble Cluster

The final and largest cluster of the network is cluster 5 with the purple colour, it has 75 nodes or 36.27% of the total graph. It is named The Streaming Bubble because it primarily consists of channels that despite uploading regular videos they also engage in live streaming content on platforms like Twitch and YouTube, with some examples being Trainwreckstv, UberDanger, ZackRawrr, Ziostorm and ZexRow. This cluster represents a significant segment of the network where creators connect with their audience in real-time, offering a mix of gameplay, commentary, and interactive experiences.

Even though this is a big cluster of nodes only a handful of them have a big impact on the cluster and on the network as a whole.

The first one being ZackRawrr, who has already been seen before is this analysis. ZackRawrr scores the highest in all of the measures that have been analyzed so far while also having the highest overall degree(209), out-degree(203), degree centrality (1.029) and betweenness centrality (0.080).

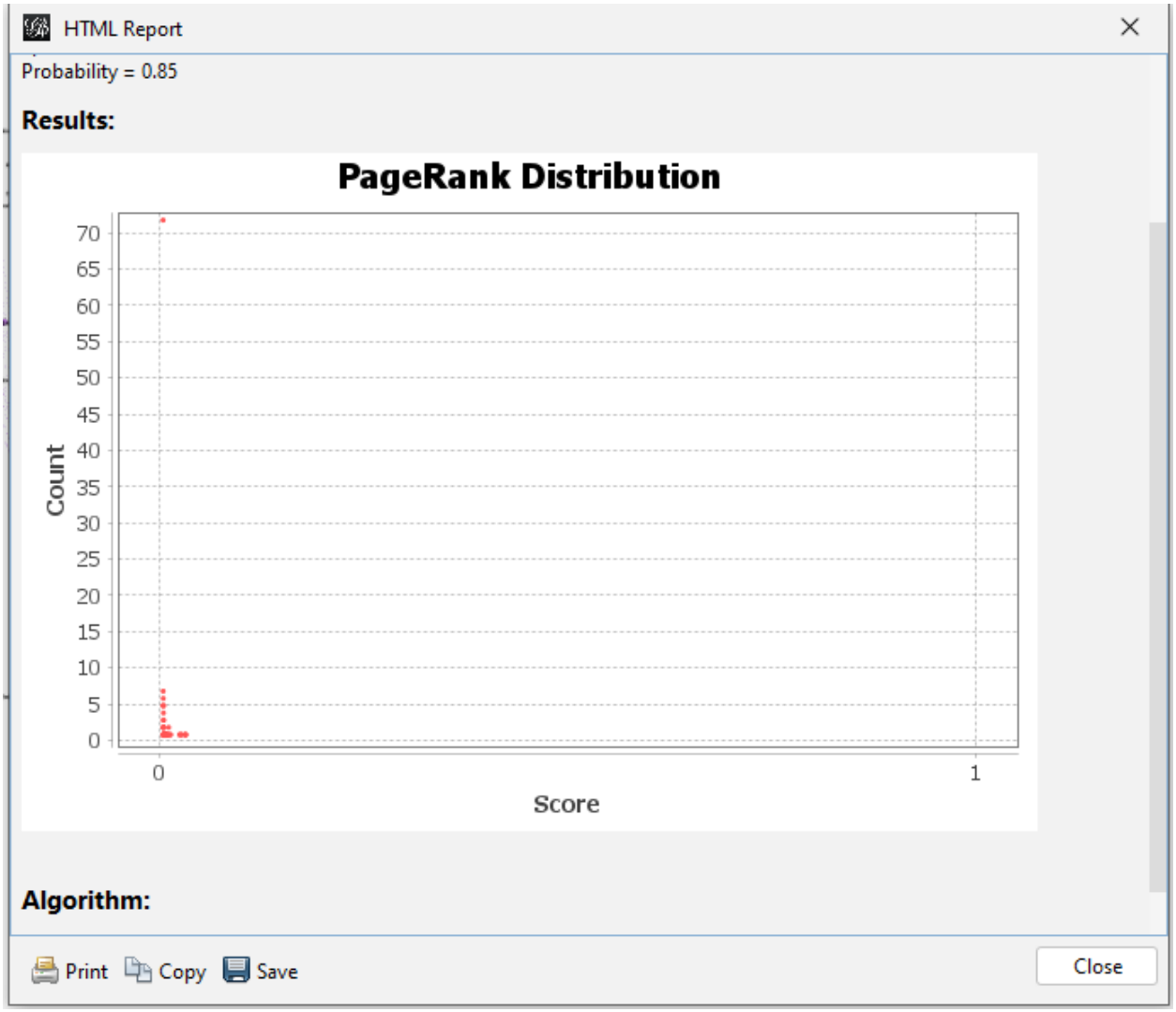
The second channel is Thunderf00t which holds the second position in every measure of the cluster after ZackRawrr, except in-degree where Thunderf00t places third. It is also important to note that this node does not hold any top positions overall.Thunderf00t, a creator known for his scientific analyses and debunking videos, occasionally ventures into the realm of streaming. In these streams, he extends his signature approach of critical examination and discussion to a live format, engaging with his audience in real-time. The content often mirrors the themes of his main channel of in-depth analysis and commentary on a variety of scientific claims, pseudoscience, and popular media.

13. PageRank

PageRank is an algorithm used for ranking the importance of elements in a network, particularly well-known for its initial use by Google to rank web pages in their search engine results. Developed by Larry Page and Sergey Brin, co-founders of Google, the PageRank algorithm essentially measures the importance of each node within the network based on the number and quality of links to each node.

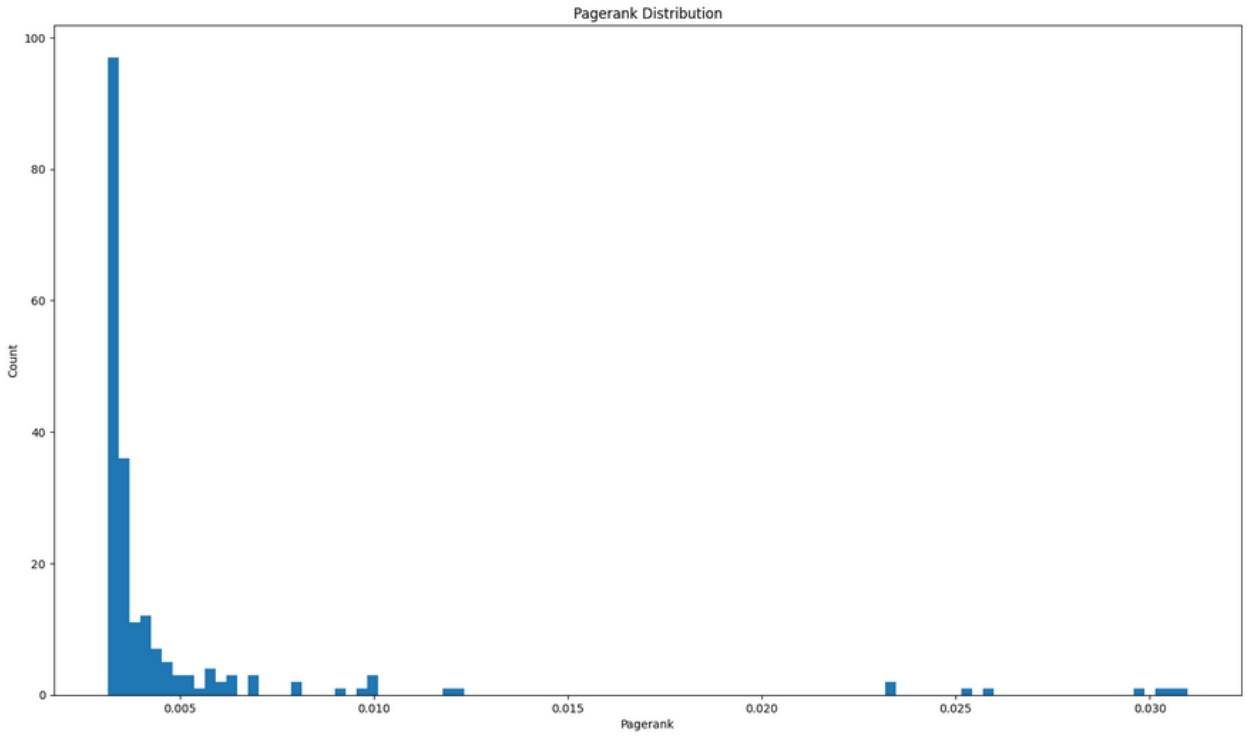
Both Gephi and NetworkX are able to calculate PageRank and both will use a damping factor of 0.85, which means that there's a 15% chance at any given moment that the user will jump to a random web page instead of following a link from the current page.

Gephi


HTML Report
Probability = 0.85
Results:
PageRank Distribution
Count
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
Score
Algorithm:
Print
Copy
Save
Close

NetworkX

```
[52]: #plot the pagerank distribution
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
plt.hist(pagerank.values(), bins=100)
plt.title("Pagerank Distribution")
plt.ylabel("Count")
plt.xlabel("Pagerank")
#make the plot bigger
fig.set_size_inches(18.5, 10.5)
```



Taking a better look at the nodes with the highest PageRank

```
In 53  #show the labels labels of the nodes and their pagerank sorted in descending order
       counter = 0
       for i in sorted(pagerank.items(), key=lambda x: x[1], reverse=True):
           counter += 1
           print(counter, final.nodes[i[0]]['label'], i[1])
```

Executed at 2024.01.28 02:50:03 in 105ms

```
1 Linus Tech Tips 0.030981061072604267
2 Ludwig 0.03068840647799553
3 Mogul Mail 0.03026467807846753
4 ShortCircuit 0.029639308648543975
5 Bellular Warcraft 0.0257425119288963
6 Bellular News 0.025235105107363556
7 Corridor Crew 0.023364478432296363
8 Corridor 0.02332600539539332
9 KiraTV 0.012316108639447286
10 OTK 0.01184818163173311
11 Josh Strife Hayes 0.010022820438561633
12 Emiru 0.009956481612283038
13 Steak and Eggs Podcast 0.009956481612283038
14 TheLazyPeon 0.009601281485778447
```

A high PageRank signifies that a webpage is deemed significant within the web's vast network, primarily because it accumulates a substantial number of inbound links, including those from other high-ranking pages, enhancing its authority and centrality. This central position in the network implies that the page serves as a pivotal hub of information or a crucial connector, thereby potentially increasing its visibility in search engine results. In this situation the familiar name Linus Tech Tips, of the Insightful analysis cluster holds the top position indicating that it is a highly influential and central node within the network it's part of.

14. Correlation between a relatively high Subscriber Count and Centrality

Python will be used to check if there is any correlation between a high subscriber count (more than 1,000,000) and the centrality measures of a node.

First of all, it is crucial to check if the measures follow the normal distribution, to ensure that the most precise correlation method is used. It is evident from the distribution of the measures, as seen above, that they don’t follow the normal distribution, but the Shapiro-Wilk test will also be performed to check for normality.

Using python and Scipy

```
In 95
#keep only the centrality measures and the subscriber countS
cor = centrality_df[['degree_centrality', 'closeness_centrality', 'betweenness_centrality', 'eigenvector_centrality', 'subscribercount']]

#use the Shapiro-Wilk test to check for normality
from scipy.stats import shapiro

#check for normality
for column in cor.columns:
    stat, p = shapiro(cor[column])
    print(f'{column} - p={p}')
Executed at 2024.02.15 23:11:33 in 71ms
```

```
degree_centrality - p=1.6489365344043072e-28
closeness_centrality - p=9.185485216057123e-15
betweenness_centrality - p=1.375575088203521e-29
eigenvector_centrality - p=1.6310110157250284e-14
subscribercount - p=3.2428244747819553e-28
```

It is evident that the p-value of each measure is lower than 0.05 meaning that there are not any signs of normality in the measures. As a result, the spearman correlation method will be used.

Using Python it is possible to calculate the spearman correlation between a moderate to high subscriber count and for each of the metrics

```
In 92
#keep only the centrality measures and the subscriber count
spear = centrality_df[['degree_centrality', 'closeness_centrality', 'betweenness_centrality', 'eigenvector_centrality', 'subscribercount']]

#keep only the subscribers count greater than 1000000
spear = spear[spear['subscribercount'] > 1000000]

spearman_corr_matrix = spear.corr(method='spearman')
spearman_correlations = spearman_corr_matrix['subscribercount'].drop('subscribercount')

print("Spearman Correlation Coefficients between 'subscribercount' and each centrality measure:")
print(spearman_correlations)
Executed at 2024.02.15 22:57:58 in 102ms
```

```
Spearman Correlation Coefficients between 'subscribercount' and each centrality measure:
degree_centrality         0.081324
closeness_centrality      0.216589
betweenness_centrality   -0.065537
eigenvector_centrality    0.073349
Name: subscribercount, dtype: float64
```

These coefficients suggest that there are only weak correlations between subscriber count and the centrality measures examined. All the relationships being relatively weak, indicates that a moderate to high subscriber count is not a strong predictor of high centrality withing the network.

15. Conclusions

Among the vast network of youtube channels that ZackRawrr is related to, there are 6 channels that stand out as the most influential nodes of the network. First it's the channel that the network is named after, ZackRawrr, who is also the most influential node in his cluster, The Streamer Bubble, having the highest overall degree (209), out-degree(203), degree centrality (1.029) and betweenness centrality (0.080). These degree metrics highlight the node's vast number of connections and its strong broadcasting influence, while the high betweenness centrality as well as the node's involvement with all the bridges and local bridges signify that ZackRawrr is a crucial connector or bridge along the shortest paths between other nodes, while also being the gatekeeper of his cluster, because it holds a position of power and influence due to its ability to mediate interactions between the nodes of his cluster with the rest of the network. Having an outwards connection with all 203 nodes of the network indicates that ZackRawrr can broadcast information directly to every cluster and to every node. The second highly influential node of the channel is the forefront of the Strategic

Gameplay cluster, Josh Strife Hayes, a channel that holds a lot of top positions in the network. Holding third place in both degree (36) and degree centrality (0.17) and second in betweenness centrality (0.02) and fourth in closeness centrality (0.064),the channel Josh Strife Hayes demonstrates a broad reach and centrality within the network, suggesting its ability to connect with numerous other channels effectively and fast. Additionally, securing second place in in-degree (10) emphasizes the channel's role as a prominent receiver of information or references from other nodes, underscoring its importance within the community. Moreover, Josh Strife Hayes leads in eigenvector centrality with a score of 0.2, highlighting its connections to other highly influential nodes, which amplifies its overall impact in the network. Overall, the node Josh Strife Hayes, has strong connections, reaches all the clusters of a network fast, effectively connecting them, and takes the role of being a significant recipient of information, allowing it to quickly learn from and adapt to the evolving interests of the community while also swiftly spreading that information to rest of the nodes. From the Gaming Nexus cluster, the channel Click4Gameplay is the third important node of the ZackRawrr network with the second highest position in metrics such as degree (49), out-degree (46), degree centrality(0.24), while also being connected with nodes that belong in every cluster of the network, a high degree signifies a large number of direct connections, making it a well-connected node, the large out-degree points to its active role in reaching out to other nodes, suggesting that Click4Gameplay frequently has interactions or shares content with a wide array of channels. The notable degree centrality further emphasizes its central position in the network, indicating that Click4Gameplay is a key player in the flow of information and trends. From the same cluster, Stoopzz is also a significant node, although it may not connect to as many nodes as Click4Gameplay, its ranking in the top positions across several centrality measures speaks to its pivotal role. Holding the fourth overall position in degree with 35 connections and the third in out-degree with 30 connections, indicates a strong network presence, with a good number of channels, from every cluster, directly linked to Stoopzz. Its degree centrality score of 0.1724 , which is also the fourth highest overall, further emphasizes its central position within the network, highlighting its importance in the flow of information and interactions. Additionally, ranking third in out-degree with 30 connections suggests that Stoopzz actively engages with other nodes of the network, while the notable third position in betweenness centrality, with a score of 0.0214, suggests that Stoopzz acts as a key bridge within the network, facilitating the flow of information between different nodes and clusters. The fifth important node of the network is The Act Man from the Commentary and Review cluster, who holds the same positions with Stoopzz in some measures, such as fourth with degree (35) and degree centrality (0.1724) and third in out-degree (35), while also being connected to every cluster. This showcases the nodes extensive connections across the network and its role in bridging diverse clusters. This interconnectivity not only underscores The Act Man's influence but also its ability to facilitate the flow of information reinforcing the network's cohesive structure. Finally the last important node is Linus Tech Tips from the Insightful Analysis cluster, this node is tied in second place with Josh Strife Hayes for in-degree(10), has the second highest closeness centrality (0.0666) and the highest PageRank (0.031) meaning that, even though the node doesn’t have many out going connections or many connections in general, it can take information in a quick way

because of that high in-degree and closeness centrality and spread it just as quick to other important nodes that will share it with the rest of the network, due to its high PageRank.

Furthermore, it is evident that from the 6 top nodes only two have a moderate to high subscriber count of more than 1 million, with Linus Tech Tips having more than 15 million and The Act Man reaching 1.8 million subscribers, while the rest of the channels have a lower subscriber count with Josh Strife Hayes having 835,000, ZackRawrr having 424000, Stoopzz having 125,000 and Click4Gameplay having a count only just 86,300. Such a diversity in subscriber counts indicates, that even though two of the central nodes have a high subscriber count, it is not a predominant feature of all of them, suggesting that there isn’t a significant connection between the high subscriber count and the centrality of a node. This observation aligns with two key findings from the study. Firstly the lack of homophily, particularly among channels with moderate to high subscriber counts, suggests that the network does not create an environment that exclusively favors channels with large subscriber bases to become central within the network by creating tight-knit communities between them. Secondly, the low Spearman correlation coefficient results suggest that subscriber count does not influence a channel's centrality within the network. Despite the intuitive assumption that channels with higher subscriber counts might hold more central positions due to their potential influence and connectivity, the weak correlations indicate that a channel's structural importance or role within the network—as measured by centrality metrics—is not predominantly dictated by its subscriber count. These results underscore the complexity of network dynamics, where a channel's position and role within the network are likely shaped by a multitude of external and internal factors beyond just its subscriber base.

Bibliography

1. Bernhard Rieder Youtube data tools : https://ytdt.digitalmethods.net/index.php
2. Average node degree: https://web.stanford.edu/~jacksonm/Gephi-instruction-1-updatedApr2015.pdf
3. Assortativity : https://en.wikipedia.org/wiki/Assortativity
3. ZackRawrr's/Asmongold's information : https://youtube.fandom.com/wiki/Asmongold_TV
4. Josh Strife Hayes : https://youtube.fandom.com/wiki/Josh_Strife_Haye
5. Linus Tech Tips: https://youtube.fandom.com/wiki/Linus_Tech_Tips
6. Click4Gameplay: https://www.youtube.com/@click4gameplay
7. Stoopzz: https://www.youtube.com/@Stoopzz , https://www.method.gg/stoopzz-joins-method
8. The Act Man: https://youtube.fandom.com/wiki/The_Act_Man
9. Logically Answered: https://youtube.fandom.com/wiki/Logically_Answered
10. Joshua Fluke: https://youtube.fandom.com/wiki/Joshua_Fluke, https://www.youtube.com/@JoshuaFluke1
11. The Lazy Peon: https://www.youtube.com/channel/UCE-f0sqi-H7kuLT0YiW9rcA
12. Penguinz0: https://en.wikipedia.org/wiki/Cr1TiKaL
13. Pewdiepie: https://en.wikipedia.org/wiki/PewDiePie
14. Nintendo Prime: https://www.youtube.com/channel/UCc0qpQO6y4aB4IaZtLCZMXg
15. Thunderf00t: https://en.wikipedia.org/wiki/Thunderf00t

Appendix

Code for Degree Centrality figure.

```
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap

# Set the size of the plot
plt.figure(figsize=(20,20))

# Set the planar layout
pos = nx.random_layout(final)

# Define a minimum node size
min_node_size = 300  # Adjust this value as needed

# Set the size of the nodes based on centrality with a minimum size
node_size = [max(v * 3500, min_node_size) for v in centrality.values()]

# Normalize centrality values
sorted_centrality = sorted(centrality.values(), reverse=True)
second_largest_centrality = sorted_centrality[1]
normalized_centrality = {node: min(centrality[node] / second_largest_centrality, 1.0) for node in final.nodes()}

# Create a custom blue color map with more shades
colors = ["#b3cde3", "#8c96c6", "#8c8cff", "#4c4cff", "#0c0cff", "#0000cc", "#000099", "#000066", "#000033"]  # Gradient of blues
cmap = LinearSegmentedColormap.from_list("custom_blue", colors, N=100)

# Map normalized centrality to the custom color map
node_color = [cmap(normalized_centrality[node]) for node in final.nodes()]

# Draw the nodes
nx.draw_networkx_nodes(final, pos, node_size=node_size, node_color=node_color)

# Draw the edges
nx.draw_networkx_edges(final, pos, edge_color='black', alpha=0.1)

# Identify top five nodes
top_five_nodes = sorted(centrality, key=centrality.get, reverse=True)[:5]

# Label the nodes based on their centrality value in descending order
channel_labels = {node: str(i) for i, node in enumerate(sorted(centrality, key=centrality.get, reverse=True), start=1)}

# Draw labels at the nodes' positions, make top 5 labels white
for node, label in channel_labels.items():
    if node in top_five_nodes:
        nx.draw_networkx_labels(final, pos, labels={node: label}, font_size=9, font_color='white', font_family='sans-serif')
    else:
        nx.draw_networkx_labels(final, pos, labels={node: label}, font_size=9, font_family='sans-serif')

# Hide axis
plt.axis('off')

# Show the plot
plt.show()
```

Code for Closeness Centrality figure

[168]:
```python
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap

# Set the size of the plot
plt.figure(figsize=(20,20))

# Set the random layout
pos = nx.random_layout(final)

# Define a minimum node size
min_node_size = 100  # Adjust this value as needed

# Set the size of the nodes based on centrality with a minimum size and make it more susceptible to change
node_size = [max(v * 10000, min_node_size) for v in closeness.values()]

# Normalize centrality values
sorted_closeness = sorted(closeness.values(), reverse=True)
largest_closeness = sorted_closeness[0]
normalized_closeness = {node: min(closeness[node] / largest_closeness, 1.0) for node in final.nodes()}

# Define a gradient of reds
red_colors = ["#ffcccc", "#ff9999", "#ff6666", "#ff3333", "#ff0000", "#cc0000", "#990000", "#660000", "#330000"]

# Create a custom red color map
cmap = LinearSegmentedColormap.from_list("custom_red", red_colors, N=100)

# Map normalized centrality to the custom color map
node_color = [cmap(normalized_closeness[node]) for node in final.nodes()]

# Draw the nodes
nx.draw_networkx_nodes(final, pos, node_size=node_size, node_color=node_color)

# Draw the edges
nx.draw_networkx_edges(final, pos, edge_color='black', alpha=0.1)

# Label the nodes based on their centrality value in descending order
channel_labels = {node: str(i) for i, node in enumerate(sorted(closeness, key=closeness.get, reverse=True), start=1)}

nx.draw_networkx_labels(final, pos, labels= channel_labels, font_size=9, font_color='white', font_family='sans-serif')

# Hide axis
plt.axis('off')

# Show the plot
plt.show()
```

Code for Betweenness Centrality figure

[203]:
```
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap

# Set the size of the plot
plt.figure(figsize=(20,20))

# Set the random layout
pos = nx.random_layout(final)

# Define a minimum node size
min_node_size = 300  # Adjust this value as needed

# Set the size of the nodes based on centrality with a minimum size
node_size = [max(v * 50000, min_node_size) for v in betweenness.values()]

# Normalize centrality values
sorted_centrality = sorted(betweenness.values(), reverse=True)
largest_betweeness = sorted_centrality[0]
normalized_centrality = {node: min(betweenness[node] / largest_betweeness, 1.0) for node in final.nodes()}


from matplotlib.colors import LinearSegmentedColormap

# Create a custom brown color map with a gradient from lighter to darker shades
light_to_dark_brown_colors = ["#d2b48c", "#8b4513", "#804000", "#6b4423", "#553000", "#3d2000", "#291800"]  # Gradient of browns
light_to_dark_brown_cmap = LinearSegmentedColormap.from_list("custom_light_to_dark_brown", light_to_dark_brown_colors, N=100)

# Map normalized centrality to the custom darker green color map
node_color = [light_to_dark_brown_cmap(normalized_centrality[node]) for node in final.nodes()]


# Draw the nodes
nx.draw_networkx_nodes(final, pos, node_size=node_size, node_color=node_color)

# Draw the edges
nx.draw_networkx_edges(final, pos, edge_color='black', alpha=0.1)

# Identify top five nodes
top_five_nodes = sorted(betweenness, key=betweenness.get, reverse=True)[:4]

# Label the nodes based on their centrality value in descending order
channel_labels = {node: str(i) for i, node in enumerate(sorted(betweenness, key=betweenness.get, reverse=True), start=1)}

# Draw labels at the nodes' positions, make top 5 labels white
for node, label in channel_labels.items():
    if node in top_five_nodes:
        nx.draw_networkx_labels(final, pos, labels={node: label}, font_size=9, font_color='white', font_family='sans-serif')
    else:
        nx.draw_networkx_labels(final, pos, labels={node: label}, font_size=9, font_family='sans-serif')

# Hide axis
plt.axis('off')

# Show the plot
plt.show()
```

Code for Eigenvector Centrality graph

[225]:
```
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap

# Set the size of the plot
plt.figure(figsize=(20,20))

# Set the random layout
pos = nx.random_layout(final)

# Define a minimum node size
min_node_size = 300  # Adjust this value as needed

# Set the size of the nodes based on centrality with a minimum size
node_size = [max(v * 5000, min_node_size) for v in eigenvector.values()]

# Normalize centrality values
sorted_centrality = sorted(eigenvector.values(), reverse=True)
largest_eigenvector = sorted_centrality[0]
normalized_centrality = {node: min(eigenvector[node] / largest_eigenvector, 1.0) for node in final.nodes()}

# Create a custom yellow color map with a gradient from lighter to darker shades
light_to_dark_yellow_colors = ["#ffffe0", "#fffacd", "#fff68f", "#ffd700", "#eec900", "#daa520", "#b8860b"]  # Gradient of yellows
light_to_dark_yellow_cmap = LinearSegmentedColormap.from_list("custom_light_to_dark_yellow", light_to_dark_yellow_colors, N=100)

# Map normalized centrality to the custom darker green color map
node_color = [light_to_dark_yellow_cmap(normalized_centrality[node]) for node in final.nodes()]

# Draw the nodes
nx.draw_networkx_nodes(final, pos, node_size=node_size, node_color=node_color)

# Draw the edges
nx.draw_networkx_edges(final, pos, edge_color='black', alpha=0.1)

# Identify top five nodes
top_five_nodes = sorted(eigenvector, key=eigenvector.get, reverse=True)[:4]

# Label the nodes based on their centrality value in descending order
channel_labels = {node: str(i) for i, node in enumerate(sorted(eigenvector, key=eigenvector.get, reverse=True), start=1)}

# Draw labels at the nodes' positions
nx.draw_networkx_labels(final, pos, labels=channel_labels, font_size=9, font_family='sans-serif')

# Hide axis
plt.axis('off')

# Show the plot
plt.show()
```