Approaches for Quantifying Multicast Complexity

Nora Berg

AW2 - SS2014

Contents

1 Introduction

Network Complexity describes the influence of network elements with all their operations and local configuration in mutual interaction on the functionality of the network. One of the problems of network complexity is that it is not formally defined yet.

This work takes a look at three theoretical environments, which overlap in certain aspects with complexity and offer a way to quantify them. The goal is winning information how to quantify similar aspects of network complexity.

The first topic is graph transformation, which formalizes changes of graphs on basis of so called production rules. Graph transformation overlaps with network complexity, because networks evolve after defined rules, which can lead to unpredictable behavior of a network. This works takes a look to find some theoretical results of graph transformation which can be adapted to network complexity.

Another topic, which shares aspects with network complexity is percolation theory. It introduces the calculation of connectivity of a network on the basis of the probabilistic link existence. It can be used to describe the robustness of a network, which is mainly influenced by complexity. The third topic is a mathematical approach to quantify the change of a multicast distribution tree, if the number of group members changes. It overlaps with network complexity insofar as the change of a network can reflect network complexity.

The complexity of analyzing network complexity increases enormously with the amount of distinct behaving participants. This work is focused on networks, in which all participants follow the same control loop, as in case of many multicast or peer-to-peer algorithms. Complexity influences many characteristics of a network. Two examples are robustness and resilience. While robustness describes how long and under which conditions the functions of a network sustain in the presence of failures, resilience describes the ability of a network to adjust to a changing environment. This work focuses on robustness.

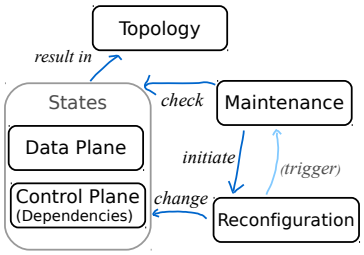


Figure 1: Control loop consist of states and methods for maintenance and reconfiguration

The origin of complexity lies in the control loops of single elements of the network, because they implement the protocols which result in the network operations. The elements of control loops can be matched to recurring components. As visualized in Figure 1, control loops include static information in form of states. States of the data plane include forwarding information, thereby the sum of all data plane states in the routers form the distribution topology. Control plane states may include information for maintenance of networks. The sum of all control plane states form the distribution of the control information. The maintenance methods in control loops check states for inconsistencies, and initiate reconfiguration methods, if necessary. Reconfiguration methods are responsible for gaining all necessary information to calculate and update states.

Note that the topics in this work, often calculate network properties on the basis of graph theory (Section 3 and 4). In this context the words *nodes* and *vertices* are used interchangeably. This comes from the distinct origins of the the words. While *node* is often used in network related topics, *vertex* comes from graph theory. Analogously is the usage of *links*, *edges* and *hops*. In addition, any figures which are not cited in the captions are self-provided.

This report is structured as follows. Section 2 focuses on graph transformation and shows two common graph transformation approaches. Section 3 takes a look at percolation theory and Section 4 is about the calculation of the change of a multicast distribution tree. At the end of each section the relation between the obtained insights and network complexity is discussed. In the last section a short summary and outlook for future work is given.

2 Graph Transformation

Graph transformation is a formal way to describe the change of a graph. There are several approaches to define *graph transformation rules*. In contrast to graphs themselves, which are static and passive structures, graph transformation is interesting for multicast complexity, because complexity and unpredictability do not emerge from static passive graphs, but from changes within the network.

Graph transformation rules describe changes of a graph $G = (V, E, s, t)$ by using subgraphs as input and change them into a new form. With V as the set of vertices, E the set of edges and $s : E \to V$ and $t : E \to V$, which map the edge to its source and target vertex.

Many applications can be modeled as graphs, e.g. multicast networks or software. In the context of such a model the sequential application of the production rules describes the behavior of the model. Graph transformation rules can be expressed in two common forms: *Single Pushout* (SPO) and *Double Pushout* (DPO).

This section approaches graph transformation starting from the paper *Graph Transformation in a Nutshell* [10]. The paper describes graph transformation with single pushout rules, which is an algebraic approach. Thereafter an introduction to the *Double Pushout* approach is given based on the first chapter of *Fundamentals of Algebraic Graph Transformation* [7]. Additional Information comes from *Algebraic Approaches to Graph Transformation, Part I: Basic Concepts and Double Pushout Approach* [6]. Subsequently, these production rules are illustrated by applying them to a graph model of a Pacman game resp. a normal graph.

Since complexity arises from evolving structures, the goal is to obtain information about these structures by looking at basic rules.

2.1 Single Pushout

The *Single Pushout* method (SPO) uses a *production* rule in the form of $p_s = (L, R)$ between the two graphs $L = (V_L, E_L)$ and $R = (V_R, E_R)$. The L represents the precondition and left side of the rule (before application) and the R is the result of applying p_s. To apply a production rule, a *match* of L has to be found in G. A match is a structure preserving graph homomorphism $m : L \to G$. *Structure preserving homomorphism* means, that all adjacent vertices in L stay adjacent after the mapping to graph G. If available, also labels or types of edges and vertices must be preserved.

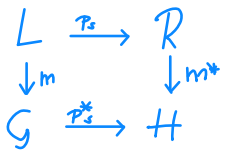


Figure 2: Schematical application of an SPO production rule.

The application of p_s to G is schematically shown in Figure 2. The production p_s is defined only between the exact R-Graph to the exact L Graph. With the application on a graph, the L-Graph is mapped to the graph G with the *match* m and the change in the actual graph G is called *co-production* p_s^*. The homomorphism from R to H is called *co-match* m^*.

To apply a production rule to G, the mapped elements in L, which do not recur in R are removed from G. Subsequently, the elements of R, which did not occur in L are added to G. The new graph can be roughly seen as $H = (G \setminus (L \backslash R)) \cup (R \backslash L)$. The algorithm to transform graph G into H using a SPO-production p_s works as follows:

1. *Find* a match m of L in G
2. *Delete* vertices and edges $m(L \backslash R)$ from G
3. *Copy* new edges and vertices $(R \backslash L)$ to G to derive H and m^*.

An Example is given in Figure 3. L is a graph with the vertices $V_L = \{f1, f2, pacman, ghost\}$ with f1,f2 representing the position where Pacman and the ghost are located. The edge between f1 and f2 indicates that the fields are adjacent and so Pacman and the ghost are located on adjacent fields in the game. With this production rule the ghost is able to kill Pacman by moving forward to the Pacman field f1. Therefore the R side of the rule shows the ghost connected to f1, Pacman's prior location.

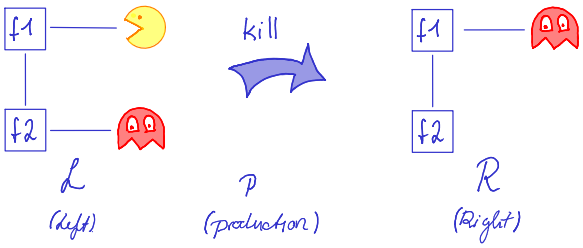


Figure 3: Single pushout rule for killing pacman

The algorithm is applied as shown by example in Figure 4. The upper left figure shows the current stage of the game, in which Pacman is located on a field with the coordinates (x, y) and the ghost right next to it. This complies to the L side of the kill rule with a mapping $m(f1)$ and $m(f2)$ to the coordinates and Pacman and the ghost as adjacent vertices *(Step 1)*. In *Step 2*, all nodes which are removed by the rule $(L \backslash R)$ are deleted in G. This concerns the Pacman vertex, because Pacman dies, so the vertex and the connecting edge will be deleted. Also the connection between the ghost and the underlying field will be deleted, because the ghost moves, and this edge represents the position of the ghost. In *Step 3* the additional Elements $(R \backslash L)$ are added to G. This concerns only the edge between the ghost and the former Pacman field, because the edge represents the new position of the ghost on Pacmans former position (x, y).

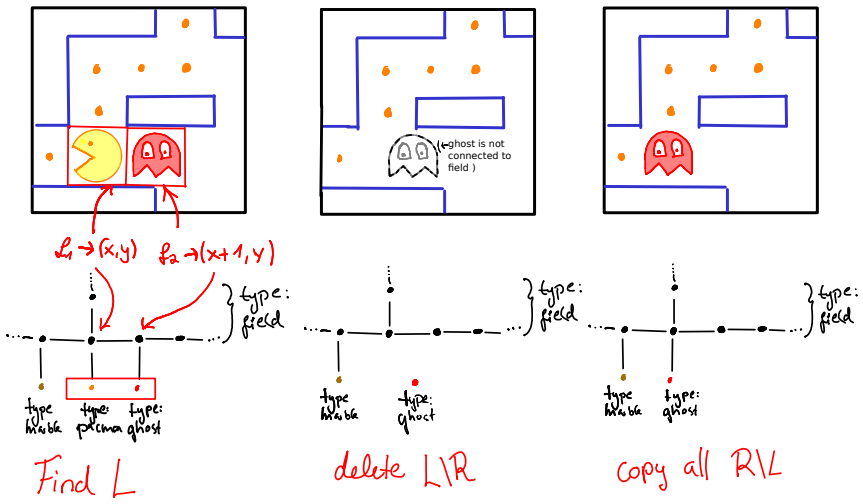


Figure 4: SPO production rule is applied to a graph

The SPO approach does not guarantee, that the outcome is a valid graph. This happens, if a vertex in L, which will be deleted, is mapped to a node in G which has more adjacent edges than specified in the rule. If this node was deleted, the adjacent edge would not have an endpoint. So it can produce *dangling edges*. The reason is that SPO does not check surrounding edges with the rule. Although it is possible to just delete all dangling edges after applying a production rule to obtain a valid graph, there is no way to prevent dangling edges in the first place. Another problem in the SPO-Approach is the conflict, if two nodes of L are mapped to the same node in G, but one of the nodes in L is deleted during the derivation process, while the other one is preserved. In this situation it is not specified, if this node should be deleted or not. The *Double Pushout* method solves these problems of SPO.

2.2 Double Pushout

In the *Double Pushout* method (DPO), *production* rules $p_d = (L, K, R)$ consists of an additional element K including all the vertices which are preserved by the production. Furthermore, the DPO production contains two homomorphism $L \xleftarrow{l} K \xrightarrow{r} R$ based on K .

As represented in Figure 5, there are three mappings to apply productions to a graph. Similar to SPO, the homomorphism $m : L \to G$ maps the elements of L injectively and in a structure-preserving way to G, analogous is the homomorphism $m^* : R \to H$. In addition, a homomorphism $d : K \to D$ exists, from the

interface graph K to the *context graph* D. The interface graph represents all elements which are preserved by the rule.

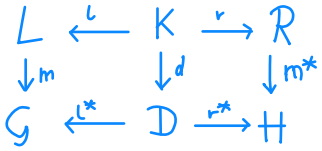


Figure 5: Schematic represantion of the DPO approach.

To prevent the problems described for SPO the *gluing condition* is checked before applying a rule. The gluing condition consist of two parts. The first part of the gluing condition prevents dangling edges with the requirement, that each vertex x in L, such that $m(x)$ is source or target of an edge e in $G \backslash L$ must be in K. Hence no vertex can be deleted which is source or target of an edge from outside the match.

The second part of the gluing condition states that vertices which are deleted during the production must have exactly one pre-image in L. This prevents the conflict whether to delete a vertex or not, if it has two pre-images in L and one of them will be deleted. The algorithm to transforms graph G into H using a DPO-production p_d works as follows:

1. *Find* a match m of L in G which fulfills the *gluing condition*
2. *Replace* $m(L)$ with $m(K)$ to create context Graph $D := (G \backslash m(L)) \cup m(K))$
3. *Copy* new edges and vertices $R \backslash K$ to D, to derive H and m^*

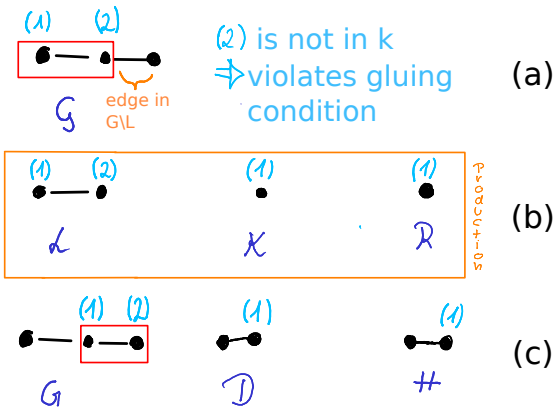


Figure 6: DPO production (b) rule is applied to graph G. In (a) the gluing condition prevents dangling edges. (c) shows the successful application of (b).

The effect of the gluing condition in *Step 1* is illustrated in Figure 6. In (a) the nodes (1) and (2) are mapped to the red highlighted nodes. The elements $G \backslash m(L)$ is the last vertex and the edge connecting to (2). If (2) was to be deleted by the applying the rule, this edge would become a dangling edge. So the condition states that (2) must be in K, because all nodes in K will be preserved by definition. This is why in (a) the gluing condition is not matched and the rule cannot be applied. In (c) the gluing condition is matched because the mapping m is chosen in a way, that $G \backslash m(L)$ does not contain any edges which are not in K.

In *Step 2*, the context graph D is created by removing all elements $m(L)$ and adding all preserved elements from K. In Figure 6c this is vertex (1). In *Step 3*, the final graph H is created by adding new elements $R \backslash K$ to the context graph D. In the production rule of Figure 6 are no new elements, so H stays like D.

2.3 Discussion

The initial motivation to look at Graph Transformation for multicast complexity, is based on the natural way of changing graphs over time with specific rules. Such changes happen in networks, when reconfiguration methods are applied. The goal was to look for theoretical predictions of complexity based on graph transformation rules.

An approach for this comes with the theory of *graph rewriting systems*. It focuses on applying a set of production rules (*graph grammar*) and the expressive power of the resulting graphs (*graph language*)[8].

In this context complexity of graph transformation algorithms is defined over *embedding conditions* of the L and R site of the production [8, 11]. Embedding conditions describe restrictions of the surrounding graph, under which the transformation step can be applied. With the embedding conditions, categories for complexity are defined, based on the Chomsky hierarchy. Examples for categories are *Unrestricted* which allows any description about the embedding graph, and *Depth 1* which allows only direct neighbors of the concerning nodes. As an example, the gluing conditions makes DPO more complex than SPO.

It would be nice to have categories to predict the complexity of network algorithms. The problem is, that graph transformation complexity is defined over the embedding conditions, which do not fit to the main problem for network complexity. The embedding conditions describe the influence (on the L-side) and the change of information (on the R side) for a *single* derivation step. This is relevant for the computation of graph transformation, but does not help with the main question of network complexity. The problem for network complexity is, in which states a network can get by the interplay of multiple transformations. Because this question differs from the orientation of graph transformation complexity, the usage of graph transformation for network complexity seems not to be a promising approach.

3 Measuring Robustness with Percolation Theory

The paper *Catastrophic Cascade of Failures in Interdependent Networks* [4] examines the mutual impact of coupled networks in case of a malfunction. A specific algorithm is used, which simulates the consecutive breakdown of nodes in interdependent networks. This behaviour is examined using percolation theory to determine on which conditions interdependent networks incur a complete breakdown, such as in the giant Italian power blackout of 2003.

In this section an introduction to basic percolation theory for networks is given. It is basis for the subsequent special case of percolation theory for interdependent networks as introduced by the paper [4].

3.1 Percolation Theory

Percolation theory is originated in physics and material science, where it is used to calculate the filtering of liquids through porous materials. The mathematical model for percolation theory is based on graphs and stochastic and can be applied to complex networks. The following aspects of percolation theory for single networks are based on the paper *Statistical mechanics of complex networks* [1].

Percolation theory is based on calculations working on a d-dimensional grid. With a probability p a link exists between two crossing points in the grid. In classical percolation theory the main question is, at which value of p there exists a way from one side to the other. Transferred to networks it is interesting at which value of p the so called *giant component* emerges. A giant component is a connected subgraph which spans nearly every vertex in the graph. As in classical percolation theory, it can be observed that the change of a graph from a number of connected components into a giant component does not increase linearly with p, but converges against a step function. The step function is shown in Figure 7, at a lower value of p the probability that a giant component exist is zero. At a certain value of p the connectivity takes a jump, so that the probability of a giant component is 1. This value is called *percolation threshold* p_c.

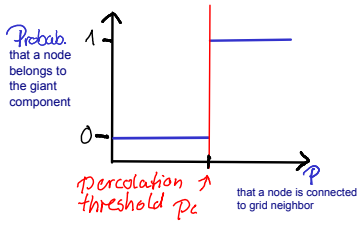
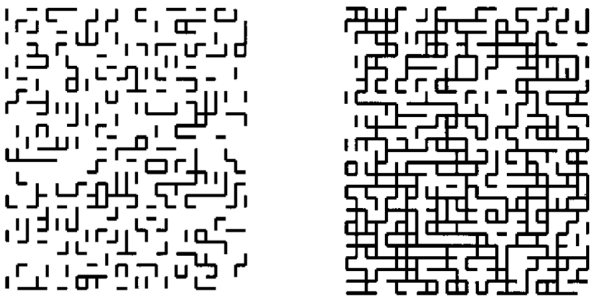


Figure 7: The probability of a node belonging to a giant component converges to a stepfunction with step at the percolation threshold p_c.

An example for the giant component is shown in Figure 8. At Figure 8a, with a low value of p, the graph consists of several small unconnected subgraphs, whereas 8b has connected nearly all existing vertices. This example shows the *bond percolation*. There also exists *site percolation*, which includes all edges in the grid but the nodes are occupied with probability p.



(a) $p = 0.315$ (b) $p = 0.525$

Figure 8: The emergence of a giant component depends on the probability p of existing bonds between two (or more) junctions [1]

.

A major difference between d-dimensional grids and networks is that grids have a static maximum node degree. However, a network graph can be associated with an underlying grid of infinite dimension and most of the results from classical percolation theory can be generalized to infinite dimension. Thereby percolation theory becomes usable for networks.

The analysis of complex network graphs with percolation theory follows two main questions in the *subcritical phase* ($p < p_c$). The first one analyzes the size of the connected components, the other analyzes the probability that a way exist between two randomly chosen nodes. Both proberties decrease exponentially with $p \to 0$ in the subcritical phase. These are interesting results for network, which indicate that below the percolation threshold the probability of two randomly chosen links are able to communicate is rather small. So either their is a good connection between all nodes in the supercritical phase, or nearly none in the subcritical phase.

It leads to the questions which properties in a control loop influence the percolation threshold and which kinds of failures can put the network in the subcritical phase.

3.2 Percolation Theory on Interdependent Networks

The paper *Catastrophic Cascade of Failures in Interdependent Networks* [4] analyzes the decay of networks, in which the proper functioning of network A depends on network B and vice versa. The analysis is based on percolation theory and the percolation threshold for interdependent networks is derived and simulated. The paper concludes with a short overview about an analytical method to describe the specified behavior by using generating functions. However, the focus of this section is on the characteristics of the decay, the measured results and expected effects on network complexity.

Percolation theory in single networks is based on nodes, which do not have any further impact on each other, than the loss of single local links, if a node disappears. In contrast, in interdependent networks two nodes depend on each other to function properly. Each node in network A is connected to a node in network B and will not work properly without it ($A \leftrightarrow B$-links). So a malfunction of one node in network A reduces the functionality of the interdependent node in network B, which in turn reduces the functionality in its own network. This ends in a cascade of failures jumping back and forth between the networks and can fragment them into small disconnected components. Subsequently, this is called the *decay* of the networks.

The following procedure abstractly reproduces the cascading failures following a malfunctioning of nodes. The connected components which emerge in the process are subsequently called *clusters*. Links within the A or B Network are noted as A-links and B-links, links between the two networks are $A \leftrightarrow B$-links.

1. Remove a fraction of $1 - p$ nodes in network A
2. Remove bidirectionally coupled nodes ($A \leftrightarrow B$ links)
3. Remove stepwise links within network B, which connects different clusters in A (and vice versa)

The procedure is illustrated in Figure 9. In (a) a small fraction of nodes in A is removed. Because the $A \leftrightarrow B$ links represent nodes which only work together correctly, in (b) the corresponding neighbor is removed. At this state (*Stage 1*) network A is split into several a_1 clusters (c). In Step 3 (d) the b-links are removed which connect different a_1 clusters, which lead to *Stage 2* with b_2 clusters. Step 3 is now alternately applied to network A and B until no more removal of edges is possible. In the end, the graph is fragmented into so called *mutually connected clusters* (e).

The *mutually connected cluster* is defined as a *set* of a- and corresponding b-nodes which fulfill the following conditions and cannot be enlarged by adding a- or b-nodes without violating them:

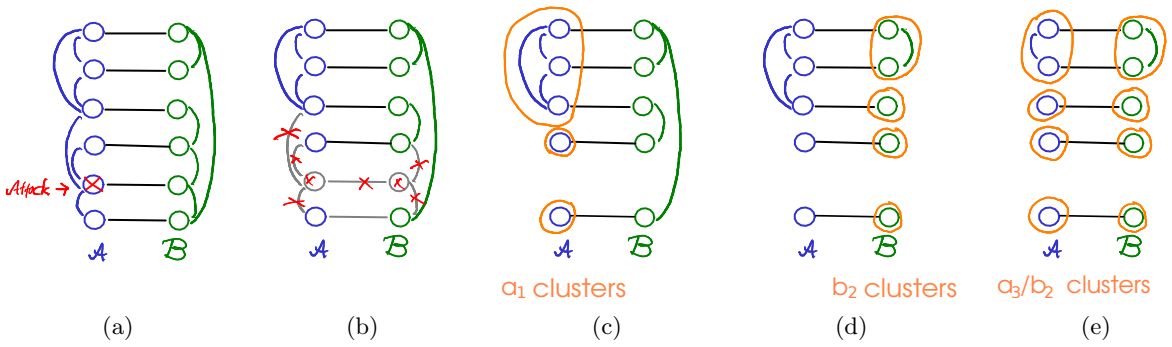
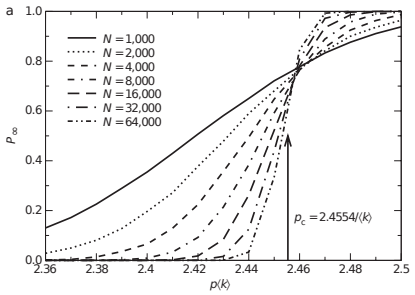


Figure 9: Removal of $1-p$ nodes (a) and their $A \leftrightarrow B$ neighbours (b,c), followed by stepwise decay into mutually connected clusters (d,e)

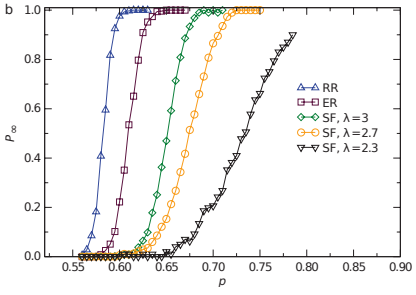
1. each pair of nodes in this set belonging to A is connected via a path consisting only of a-links
2. each pair of nodes in this set belonging to B is connected via a path consisting only of b-links

The main question in the paper [4] is, with which value $p = p_c$ the interdependent networks decays into fragments. Only a *giant mutually connected cluster*, a mutually connected cluster which spans (nearly) the entire network, is considered fully functional. It is possible that one giant mutually connected cluster emerges together with a few very small mutually connected clusters. In this case, only the giant mutually connected cluster is functioning, whereas the small ones stop working. Hence the small mutually connected clusters are not considered any further.

The simulation of the procedure shows probability of the existence of a giant component after the decay procedure converges towards a stepfunction similar to the single networks. Figure 11 shows the probability of the existence of a giant component $P_\infty(p, N)$. N is number of nodes per network and p determines the fraction of $1 - p \cdot N$ removed nodes in the first step of the procedure. After the procedure the existence of a giant component is calculated by determining the fraction of nodes in the largest mutually connected cluster. The probability of the existence of a giant component can be measured by applying the simulation multiple times and counting the fraction of tests, in which a giant component emerges.



(a) The probability of a giant component depends on p and converges with $N \to \infty$ to a step function with a jump at the percolation threshold p_c [4].



(b) The influence of random graph structure (random regular (RR), Erdös-Renyi (ER), scale free (SF)) on the percolation threshold. All networks have an average degree of $\langle 4 \rangle$ and 50000 nodes [4].

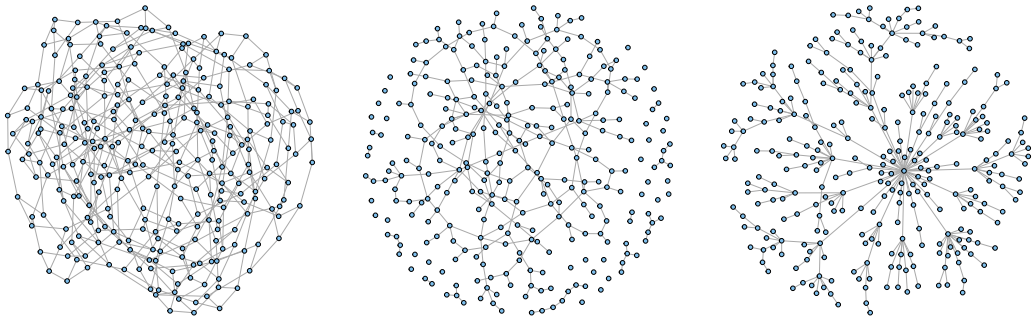
Figure 10: The existence probability of a giant component is influenced by the number of Nodes (a) and the structure of the underlying network (b).

Figure 10a shows the convergence of $P_\infty(p, N)$ towards a stepfunction with percolation threshold p_c while $N \to \infty$. Two interdependent Erdös-Renyi networks with the same average node degree $\langle k \rangle$ were used for the measurement. The x-axis represents p rescaled with the $\langle k \rangle$ because the explicit calculation of p_c reveals a dependency to the average node degree $\langle k \rangle$ as $p_c = 2.4554/\langle k \rangle$ (calculation in [4]).

Another characteristic, which influences the percolation threshold is the structure of the underlying network. Figure 10b shows the change of the percolation threshold subject to the type of the underlying network topology.

Considered are three kinds of networks (Figure 11). In random regular graphs, each vertex has the same node degree but random adjacent neighbor vertices [3]. The main characteristic of the Erdös-Renyi

networks is that each possible edge exists with a specific probability p, similar to the ∞-dimensional grid from the previous section [1]. While Erdös-Renyi networks are an easy way to create random graphs, real existing networks like the Internet show different graph characteristics.



(a) (b) (c)

Figure 11: The difference in structure between a random regular network (a), an Erdös-Renyi network (b) and a scale free network with the Albert-Barabasi method ($\gamma \approx 3$) (c) . All graphs consist of 300 Nodes and an average node degree of 3 (a) , 1.933333 (b) and 1.993333 (c)

The internet can be approximated as a *scale free* network, in which the node distribution follows a power law. This means that there exist very few nodes with an exceptionally large node degree (*hubs*), and many nodes with a very small node degree. The probability $P(k)$ for a node being a large hub with k connections to other nodes asymptotically follows $P(k) \sim k^{-\gamma}$. In comparison between a scale free network and an Erdös-Renyi network with an equal number of nodes and the same average node degree, the scale free network contains many more nodes of very low node degree and a few larger hubs (Figure 11).

In most cases (e.g., for Internet-like graphs) random scale free networks are modeled with γ as $2 \leq \gamma \leq 3$. Examples for scale free networks are links on the world wide web ($\gamma \approx 2.1/2.7$), autonomous systems of the Internet ($\gamma \approx 2.5$) or distinct peer-to-peer networks ($\gamma \approx 2.1$) [5].

The decay procedure for interdependent networks has substantial consequences for the stability of the coupled scale free networks. Figur 10b shows the probability that a node is connected to the giant component changes strongly with the structure of the graph [1]. Single networks gain robustness through the scale free structure, so that there exist a giant component for all $p > 0$, if $2 \leq \gamma \leq 3$. The reason for that lies in the broad node degree distribution. If a random number of nodes is removed of a single network, the probability, that a hub is being removed is relatively small, because there are only few hubs. These hubs give increase the stability of single networks.

In interdependent networks, hubs become a vulnerability. There are two reasons for that:
If a node has a malfunction in *Step 1* of the decay procedure, directly a second node (the corresponding node) in the second network is removed (*Step 2*). This increases the probability of removal of a hub significantly, because either the node in network A *or* in network B, can be a hub.

The second reason for hubs as vulnerabilities for scale free networks comes from *Step 3* of the procedure. Hubs become separated, when the corresponding neighbor node in the other networks gets separated into a small cluster during the decay procedure. The vulnerability comes from the high probability, that a hub on one network, is connected to a low degree node over an $A \leftrightarrow B$-link. In general, low degree nodes get easier separated into very small clusters, because they have only few links to connect to other nodes. If the corresponding low degree node to a hub is separated, the hub will directly loose most of its links. Since the hubs are the nodes, which hold the network together and only a few of them exist, the network decays into very small clusters at once. Hence hubs become a vulnerability in coupled scale free networks.

As a consequence, random regular networks, with their lack of large degree nodes, turn out to be most stable (Figure 10b), while broader degree distributions are more vulnerable.

3.3 Discussion

Percolation theory for single and interdependent networks is an interesting topic from the network complexity point of view. There are two reasons for that.

The first one is that the percolation threshold is a number which describes robustness of a network in a formally defined way. Robustness is one of the main characteristics which is influenced by complexity. By

[1] The paper does not mention how regular random and Erdös-Renyi algorithms were used to create connected graphs. The algorithms themselves do not guarantee connected graphs, but there are several possible workarounds.

observing the change of the percolation threshold, the influence from specified properties on robustness can be made visible. This can be done similar to the measurement of the percolation threshold subject to the different network topologies.

The second reason derives from the decay procedure of interdependent networks. In multicast networks, the change of one state can affect a large number of nodes, if it starts a chain of reconfiguration methods which spreads through the network. An example for this can be found in the NICE-algorithm [2], in which the clusters are periodically checked for their sizes. If a cluster is too small, its nodes will be devided and integrated into other clusters. Since there is now one clusterleader less in layer above, this cluster is also checked for its size, and possibly devided. This behavior can spread through the network.

The procedure of decay for interdependent network provides a similar chain and can possibly be adapted to represent the effects of reconfiguration methods.

4 Stability of a Multicast Tree

The paper *Stability of a multicast tree* [13] uses graph theory and stochastics to quantify the change of multicast distribution trees if a group member joins or leaves. The paper states that in large graphs the number of changed links follows a Poisson distribution. Furthermore, it devises a formula to approximate the change of the tree depending on the number of all nodes and the number of members in the multicast group. The simulation supports the calculations and in addition the influence of the link weight distribution on the stability is discussed.

This section concentrates on the main result, which is the change of links as a Poisson distribution, the explanation for this characteristic and the simulation results. These topics of the paper are the basics to get an idea how the change of a multicast tree can be quantified. Further results like the influence of the link weight distribution or the exact formulas for the calculation can be found in the paper.

A multicast distribution tree is constructed as a shortest path tree with m group members, on a connected random graph $G_p(N)$ with N vertices and independently chosen links with probability p (similar to Erdös-Renyi graphs). On the basis of such a graph $g_N(m)$ describes the expected hopcount in the shortest path tree from the source to m distinct group members.

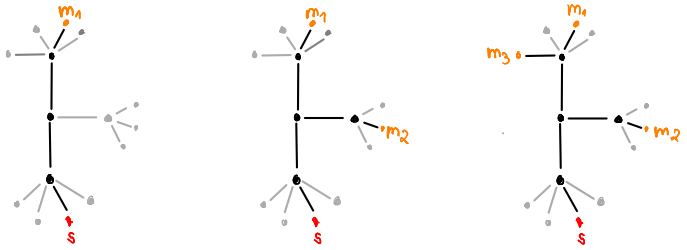


Figure 12: In a finite network the expected number of new links with a joining node decreases with the increasing number of group members.

The number of new links after a node joins, depends on the fraction of nodes in the network which already joined the network. The more nodes of the network already joined the distribution tree, the less new links are needed to connect an addition node to the distribution tree. So the mapping $m \to g_N(m)$ is considered concave. An example is shown Figure 12. The join of the first member m_1 leads to 4 new links while the m_2 and m_3 need only a few new links until they reach an already joined routing node on the shortest path.

The number of new links if the m^{th} members joins in a network with N nodes is denoted as $\Delta_N(m)$. If the number of members changes by one, the expected change of the tree is $E[\Delta_N(m)] = g_N(m) - g_N(m-1)$ links. It is the difference of links between a distribution tree with m and $m-1$ members. Since $m \to g_N(m)$ is concave, $E[\Delta_N(m)]$ is always decreasing. A multicast tree is considered stable, if $E[\Delta_N(m)] < 1$), in other words that an average of less then one link changes, if a node joins or leaves. This happens, if about $\frac{1}{3}$ of all nodes in a graph have joined the multicast group. Since this is an highly unrealistic setting, the behavior below this step is analyzed.

The paper states that below a number of $\frac{N}{3}$ group members, the distribution of changed links in a tree with m members over a graph $G_p(N)$ with uniformly distributed link weights and large N, follows a Poisson distribution. A property of the Poisson distribution is the independence of the number of events per fixed interval [9]. In this case the number of events are the number of new hops due to the joined node m. Considering two members m and the nearest joined member $m-1$ (Figure 13), and the random

variables $H_N(m)$ resp. $H_N(m-1)$ counting the hops between the source and the receiver nodes. $H_N(m)$ and $H_N(m-1)$ are not independent, and m and $m-1$ have an overlapping path to the source. This changes, if we consider the subtree rooted at the last node on this path (node B in Figure 13).

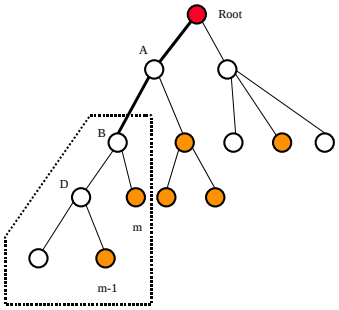
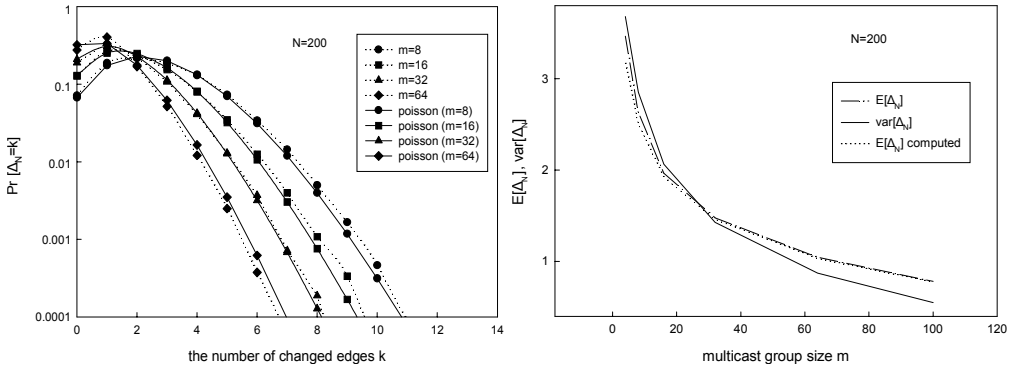


Figure 13: The nodes m and $m-1$ are independently chosen in the subtree rooted at b (dotted line). [13]

Shortest path trees of $G_p(N)$-Graphs with sufficiently large N are *uniform recursive trees* [12]. A uniform recursive tree is a tree on which each node n is connected to one of the $n-1$ nodes randomly with uniform probability. A subtree in a uniform recursive tree is also a uniform recursive tree.

Since $G_p(N)$-graphs are used in this case, the subtree rooted at B, has this exact property. So the paths and hopcount from the subtree root to m resp. $m-1$ can be considered independent and within the subtree m and $m-1$ are uniformly chosen. The probability distribution has a mean of $E[\Delta_N(m)]$, which gives the Poisson distribution formula $Pr[\Delta_N(m)=k] \sim \frac{(E[\Delta_N(m)])^k}{k!} \cdot e^{-E[\Delta_N(m)]}$. The conclusive proof and a formula to calculate $E[\Delta_N(m)]$ can be found in the paper [13].



(a) SPT: Pdf $Pr[\Delta_N = k]$ and $m < N/3$ (b) Expectation Value and Variance of Δ_N

Figure 14: Simultation results for Δ_N over a graph with 200 Nodes [13]

The validity of the Poisson distribution for the change of links is supported by measurements. Figure 14a shows the changed number of links between a shortest path tree (SPT) with m receivers and the same tree with one member less. The change of links is shown as probability density function (pdf) with m joined nodes and measured below the critical stability property of $\frac{N}{3}$ nodes. The measured pdf (dotted line) and the calculated Poisson distribution based on the formula for $E[\Delta_N(m)]$ show nearly the same values.

A property of the Poisson distribution is, that expectation value and variance tend to the same value. They are compared in Figure 14b and show this behavior especially for the lower values of m. At values of $m > \frac{N}{3}$ expectation value and variance drift apart. At this time, the tree is considered to fulfill the above-mentioned stability criteria $E[\Delta_N(m)] \leq 1$. So the measurement supports that Δ_N for $m < \frac{N}{3}$ follows a Poisson distribution.

4.1 Discussion

In summary, the number of changed links initiated by the departure and arrival of the multicast receivers is quantified. It is shown that the number of changed links follow a Poisson distribution for "unstable" considered networks.

The paper develops the calculation for networks which run IP-Multicast algorithms. Some assumptions are made, so that the results cannot be transferred one-to-one to other multicast algorithms. The first assumption is that multicast trees are considered stable, if the expected link change is < 1 per join/leave. This is only possible, if the last links to the receiver endpoints are not counted. Otherwise the criteria value for stability would have to be > 1. This means for other multicast algorithms (e.g. overlay algorithms) that m does not correlate to the number of receiver nodes, but the number of forwarding nodes.

Furthermore, the formula, which calculates the expectation values includes the number of nodes in the underlying network. For overlay algorithms there is not necessarily an underlying network, where the nodes can change its state between "member" and "not-member" of the distribution tree. So the the stability and change of a network created with such an algorithm would depend on other things than the ratio between the number of members and the number of all nodes.

From the network complexity point of view the paper is interesting, because it shows how many nodes are effected by a joining or leaving node. These nodes would be the ones, which are subsequently responsible for the optimization of the network, and therefore the starting points for possible chains of reconfiguration which can lead to unpredictable behavior in the network.

5 Summary and Outlook

In this work three approaches were introduced to formalize and evaluate distinct aspects of graphs and networks.

The *graph transformation* approach formalized the change of a graph on the basis of production rules. Although production rules have a direct mapping to the reconfiguration methods of distributed algorithms, the focus of the evaluation with graph grammars and graph languages do not fit to the goals of network complexity.

In *percolation theory* the cohesion and decay of networks is quantified for single and interdependent networks. Cohesion and decay of a network is influenced by complexity, insofar as some complexity is needed to create a robust network, but with too much complexity, a networks robustness decreases. So by comparing distinct reconfiguration methods, the impact on the network can be measured by the change of the percolation threshold.

The *stability of a multicast tree* is quantified by the number of changed links when a group member joins or leaves. It is shown that the number of changed links follows a Poisson distribution and the average value can be approximated via a formula. This quantification shows the reach of the reconfiguration methods *join* and *leave*. The reach is important to analyze the impact on other nodes and the interplay between control loops on multiple nodes.

The next step is to analyze a range of distributed algorithms. The constraint of this work, to analyze only multicast algorithms turns out to allow too much variety in the algorithms. A better restriction would be to analyze algorithms which build spanning trees and in which all nodes follow the same control loop. This increases the comparability of the algorithms. Within the control loops the reconfiguration methods will be analyzed for the influence on robustness. The goal is to find recurring characteristics which influence the robustness.

References

[1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[2] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable Application Layer Multicast. In *Proc. of SIGCOMM '02*, pages 205–217, New York, NY, USA, 2002. ACM Press.

[3] Béla Bollobás. *Random Graphs*. Cambridge Univ. Press, 2006.

[4] Sergey V Buldyrev, Roni Parshani, Gerald Paul, H Eugene Stanley, and Shlomo Havlin. Catastrophic cascade of failures in interdependent networks. *Nature*, 464(7291):1025–1028, 2010.

[5] Reuven Cohen and Shlomo Havlin. *Complex Networks: Structure, Robustness and Function*. Cambridge University Press, 2010.

[6] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. Algebraic Approaches to Graph Transformation-Part I: Basic Concepts and Double Pushout Approach. In *Handbook of Graph Grammars*, pages 163–246, 1997.

[7] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*, volume 373. Springer, 2006.

[8] Hoda Fahmy and Dorothea Blostein. A Survey of Graph Grammars: Theory and Applications. In *Pattern Recognition, 1992. Vol. II. Conference B: Pattern Recognition Methodology and Systems, Proceedings., 11th IAPR International Conference on*, pages 294–298. IEEE, 1992.

[9] Peter Hartmann. Mathematik für Informatiker: Ein praxisbezogenes Lehrbuch. Vierte Auflage, 2006.

[10] Reiko Heckel. Graph Transformation in a Nutshell. *Electronic Notes in Theoretical Computer Science*, 148:187–198, 2006.

[11] Manfred Nagl. A Tutorial and Bibliographical Survey on Graph Grammars. In *Graph-Grammars and Their Application to Computer Science and Biology*, pages 70–126. Springer, 1979.

[12] Piet Van Mieghem. *Performance Analysis of Communications Networks and Systems*. Cambridge University Press, Cambridge, New York, 2006.

[13] Piet Van Mieghem and Milena Janic. Stability of a Multicast Tree. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 1099–1108. IEEE, 2002.