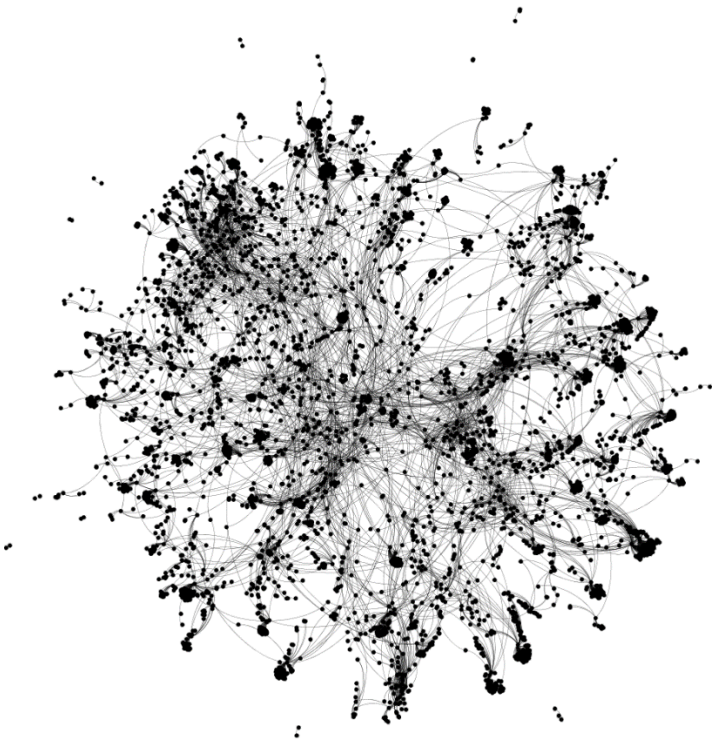
Ευρυζωνικά Δίκτυα
Τμήμα ΗΜΜΥ
Α.Π.Θ.

9ο Εξάμηνο
Φεβ. 2019

Εργασία Σχεδίασης Δικτύου



Παπαδόπουλος Κωνσταντίνος (8677) konserpap@auth.gr

Πίνακας Περιεχομένων

Ερώτημα 1: Preprocessing 3
Java source code 3
Ερώτημα 2: Παρουσίαση του dataset και ανάλυση δικτύου με Gephi 6
I. Παρουσίαση του dataset 6
Εισαγωγή 6
Προκλήσεις 6
Επιλογή μετρήσεων 7
Alias Resolution 8
Γεωγραφική τοποθεσία δρομολογητή και ρόλος 9
Αδυναμίες 9
Rocketfuel 10
Αρχεία “asn.r1.cch” 10
II. Ανάλυση δικτύου με Gephi 11
Τυχαίο δίκτυο vs πραγματικό 12
Outliers – Ακραίες τιμές 15
Διαθεσιμότητα & Communities 16
Hubs 22
Ερώτημα 3: Προεκτάσεις 23
Επαλήθευση αποτελεσμάτων Gephi 23
Matlab 23
R 26
Scale-free networks 28

Ερώτημα 1: Preprocessing

Για την εργασία επιλέχθηκε η μελέτη του ISP Verio από τις ΗΠΑ (2914.r1.cch).

Ως προγραμματιστικό εργαλείο χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java. Με τη χρήση RegEx και άλλων τεχνικών εξάγεται η πληροφορία που χρειαζόμαστε (λαμβάνοντας υπόψη να δημιουργούμε ένα μη-προσανατολισμένο δίκτυο, δηλαδή χωρίς να συμπεριλαμβάνουμε διπλές ακμές) και αποθηκεύεται σε μορφή αρχείου csv.

Στη συνέχεια παρατίθεται ο πηγαίος κώδικα με τα απαραίτητα σχόλια για την επεξήγησή του.

Java source code

```
package isp;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PreprocessingClass {


	public static void main(String[] args) {
		ArrayList <String> string_table = new ArrayList <String>();
	// ArrayList that stores each line of the file (as a string) into an ArrayList
element
		string_table = fileRead("C:\\Users\\Admin\\Desktop\\2914.r1.cch");
	// choose the file to be parsed
		getEdges(string_table);
				// create the list of edges (two nodes)
	}

	// reads the file and puts each line (as a string) into an ArrayList
	public static ArrayList <String> fileRead (String filename){
		List<String> lines = null;
		// creates a list of Strings
		try {
			lines = Files.readAllLines(new File(filename).toPath());   //
reads the file line by line and it stores into the list
		} catch (IOException e) {
```

```
                e.printStackTrace();
            }
            int size = lines.size();
            // the size of the list, the number of lines in the file
            ArrayList <String> str_table = new ArrayList <String>();          //
instantiates a new ArrayList of String type
            for (int i = 0; i < size; i++){
            // stores each list item into an ArrayList;
                str_table.add(lines.get(i));
            }
            return str_table;
      }


      // writes the edges (pairs of two nodes) into a csv file - it's an undirected
graph so no pair of nodes is the same
      public static void getEdges(ArrayList <String> str_table){

            String file_line;                                                 // the
first line of a file
            int index;
      // index used to get the first substring of the file_line String
            String fist_word;                                                 // the
first node of the two of each edge (String)
            int first_node;
      // the first node of the two of each edge (int)

            String regex = "<(.+?)>";                                   // Regex used
to find the nodes with whom the first node is connected (finds type e.g. <123>, <45>
etc)
            Pattern pattern = Pattern.compile(regex);           // creates a
Pattern object
            Matcher match;

            String edges_table[] = new String[20000];           // the final table
of Strings with the edges in csv format, check size!!!!!
            int num_edges = 0;                                                //
number of edges found

            for(int i = 0; i < str_table.size(); i++){          // parses each line
of file

                  int second_nodes[] = new int[20000];          // the second node
of the two of each edge, check size!!!!!
                  int y = 0;
                  file_line = str_table.get(i);
                  index = file_line.indexOf(' ');
                  fist_word = file_line.substring(0, index);
                  first_node = Integer.parseInt(fist_word);
                  if(first_node > 0){                                         // we
ignore the negative nodes
                        match = pattern.matcher(file_line);
                        while (match.find( )) {
                              second_nodes[y] = Integer.parseInt(match.group(1));
                              y++;
```

```
                        }
                              for(int z = 0; z < y; z++){                           //
store the smallest (number) node first
                                    if (first_node < second_nodes[z]){
                                          edges_table[num_edges] = first_node + "," +
second_nodes[z];
                                          num_edges++;
                                    }
                                    else{
                                          edges_table[num_edges] = second_nodes[z] +
"," + first_node;
                                          num_edges++;
                                    }
                              }
                        }
                  }

                   HashSet<String> myVals = new HashSet<String>();     // store in a
HashSet so as to eliminate duplicate edges (we want an undirected graph)

                   for(int i = 0; i < num_edges; i++){
                         myVals.add(edges_table[i]);
                         System.out.println("Found edge: " + edges_table[i] );     //
prints the edges found (from edges_table)
                   }
                   /*
                   TreeSet myTreeSet = new TreeSet();
                   myTreeSet.addAll(myVals);
                   System.out.println(myTreeSet);
                  */
                  BufferedWriter writer = null;                                     //
write the results into a file
                  try {
                        writer = new BufferedWriter(new
FileWriter("C:\\Users\\Admin\\Desktop\\parsed_edges.csv"));
                  } catch (IOException e) {
                        e.printStackTrace();
                  }
                   for(String val : myVals){
                         try {
                               writer.write(val);
                               writer.write("\n");
                         } catch (IOException e) {
                               e.printStackTrace();
                         }
                   }

                   try {
                        writer.close();
                  } catch (IOException e) {
                        e.printStackTrace();
                  }

      }
}
```

Ερώτημα 2: Παρουσίαση του dataset και ανάλυση δικτύου με Gephi

Ι. Παρουσίαση του dataset

Εισαγωγή

Σκοπός της παρούσας εργασίας είναι η δημιουργία ρεαλιστικών απεικονίσεων/χαρτών, σε επίπεδο δρομολογητή (router-level), των δικτύων των παρόχων υπηρεσιών δικτύων (Internet Service Providers, ISPs). Συγκεκριμένα προτείνονται καινούργιες τεχνικές μετρήσεων που θα παρέχουν υψηλή ακρίβεια με όσο το δυνατόν λιγότερες μετρήσεις.

Οι ίδιοι οι ISPs θεωρούν τις τοπολογίες τους σε επίπεδο router ως απόρρητες και οι χάρτες τους δεν είναι προσβάσιμοι στην ερευνητική κοινότητα.

Η γνώση της τοπολογίας αυτής είναι χρήσιμη διότι επηρεάζει τη δυναμικότητα των πρωτοκόλλων δρομολόγησης, την επεκτασιμότητα του multicast, την αποτελεσματικότητα έναντι επιθέσεων DoS (Denial-of-Service) και γενικά διάφορες πτυχές της απόδοσης των πρωτοκόλλων.

Προκλήσεις

Όπως και σε παλαιότερες προσπάθειες, οι χάρτες ISP δημιουργήθηκαν χρησιμοποιώντας traceroutes. Η ειδοποιός διαφορά όμως σε αυτήν την εργασία είναι η δημιουργία χαρτών ακριβείας χρησιμοποιώντας λίγες μετρήσεις.

Μία brute-force προσέγγιση συλλέγοντας δεδομένα από κάθε traceroute server προς όλα τα προθέματα στους BGP πίνακες ή σαρώνοντας όλο το IP address space ενός ISP επιβαρύνει το σύστημα (traceroute servers) και είναι χρονοβόρα (η τοπολογία μπορεί να μεταβάλλεται μέχρι να ολοκληρωθεί η αναφερόμενη διαδικασία).

Μία άλλη πρόκληση που εμφανίζεται είναι ότι η δημιουργία ενός ακριβή χάρτη απαιτεί οι διευθύνσεις IP που ανήκουν στον ίδιο router, τα aliases (ψευδώνυμα), να αναγνωρίζονται.

Τέλος, για τη δομική ανάλυση των χαρτών, πρέπει να προσδιοριστούν οι γεωγραφικές τοποθεσίες των routers και ο ρόλος τους στην τοπολογία.

Επιλογή μετρήσεων

Χρησιμοποιούνται δύο βασικές τεχνικές για τη συλλογή των μετρήσεων, το directed probing και το path reductions. Οι τεχνικές αυτές ουσιαστικά αποτελούν τη λύση για τη δημιουργία μιας υλοποίησης με όσο το δυνατό λιγότερες μετρήσεις.

Directed Probing

Με αυτήν την τεχνική παίρνουμε μόνο traceroutes τα οποία αναμένονται να διασχίσουν τον ISP. Κοιτώντας τα BGP tables, που αντιστοιχούν την IP διεύθυνση προορισμού σε ένα σετ μονοπατιών (από ASes) που χρησιμοποιούνται για να φτάσουμε στον προορισμό μας, μπορούμε να αποφύγουμε μονοπάτια που δεν διασχίζουν καν τον ISP.

Ωστόσο, δεν έχουμε τα ακριβή BGP tables που αντιστοιχούν σε κάθε θέση μέτρησης, υπάρχει δυναμική δρομολόγηση και πολλαπλά πιθανά μονοπάτια. Τα παραπάνω μπορεί να οδηγήσουν σε traceroutes μονοπατιών που δεν διασχίζουν τον ISP (μείωση ταχύτητας) και σε μη πραγματοποίηση traceroutes τα οποία διασχίζουν τον ISP, αλλά δε γνωρίζαμε το αληθινό μονοπάτι για να τα συμπεριλάβουμε (μείωση ακρίβειας).

Path Reductions

Η τεχνική αυτή βασίζεται στην υπόθεση ότι οι διαδρομές που καθορίστηκαν με το directed probing δεν είναι όλες μοναδικές για κάθε traceroute.

Συγκεκριμένα το *Ingress Reduction* προσδιορίζει ότι όταν δύο traceroutes από διαφορετικά σημεία προς τον ίδιο προορισμό εισέλθουν στον ISP από το ίδιο σημείο, τότε είναι πολύ πιθανό η διαδρομή τους μέσα σε αυτόν να είναι η ίδια. Σύμφωνα με το *Egress Reduction,* traceroutes που έχουν την ίδια είσοδο στον ISP και κατευθύνονται προς οποιοδήποτε πρόθεμα που βρίσκεται πίσω από την ίδια έξοδο, θα πρέπει να κάνουν την ίδια διαδρομή.

Το *Next-hop AS Reduction* υποθέτει ότι η διαδρομή μέσα από έναν ISP εξαρτάται μόνο από το AS του επόμενου hop και όχι από το συγκεκριμένο πρόθεμα του προορισμού.

Ο συνδυασμός των τεχνικών αυτών βοηθάει στην επιπλέον μείωση των αναγκαίων μετρήσεων.

Alias Resolution

Ένα άλλο πρόβλημα στο οποίο αναφερθήκαμε και νωρίτερα είναι τα aliases, δηλαδή ο προσδιορισμός των IP διευθύνσεων που ανήκουν στον ίδιο router. Αν δεν λάβουμε υπόψη αυτό το ζήτημα τότε στους χάρτες θα εμφανίζεται μία διαφορετική τοπολογία με περισσότερους routers και συνδέσεις από ότι στην πραγματικότητα.

Η τυπική τεχνική για alias resolution από το Mercator project που βασιζόταν στις “UDP port unreachable” απαντήσεις με συγκεκριμένες διευθύνσεις εξερχόμενων διεπαφών, δεν έβρισκε αρκετά aliases. Συνεπώς, χρησιμοποιήθηκε ένα άλλο εργαλείο για τη δουλειά, το Ally. Το Ally βασίζεται στο γεγονός ότι δύο πακέτα, παρόμοια με αυτά στο Mercator project, που στέλνονται διαδοχικά θα έχουν και διαδοχικά IP IDs. Θέτοντας έτσι ορισμένες προϋποθέσεις για το ποια πρέπει να είναι η σχέση στα IP IDs των επιστρεφόμενων πακέτων, έχουμε μία διαφορετική υλοποίηση και πετυχαίνετε μεγαλύτερη ακρίβεια για το alias resolution.

Γεωγραφική τοποθεσία δρομολογητή και ρόλος

Ως άλλη μία πρόκληση μου αντιμετώπιζαν οι ερευνητές της εργασίας αυτής είχαμε αναφέρει τη δομική ανάλυση των χαρτών, δηλαδή την ανάγκη να προσδιοριστούν οι γεωγραφικές τοποθεσίες των routers και ο ρόλος τους στην τοπολογία.

Για τον εντοπισμό των δρομολογητών που ανήκουν σε έναν ISP χρησιμοποιήθηκαν τα ονόματα DNS έναντι του address space που προβάλλεται από τα ASes, μιας και τα πρώτα επιτυγχάνουν πιο ακριβείς ταυτοποιήσεις.

Όσον αφορά το ρόλο των δρομολογητών και την τοποθεσία τους, αυτή βρίσκεται και πάλι ενσωματωμένη στα ονόματα του DNS. Οι συμβάσεις που έχουν υιοθετηθεί στα DNS ονόματα ανακαλύφθηκαν αναζητώντας τη λίστα των ονομάτων των δρομολογητών που δημιουργήθηκε από τους ερευνητές. Για τους δρομολογητές που δεν είχαν DNS όνομα ή πληροφορίες τοποθεσίας έγινε εκτίμηση για τη θέση τους μέσω των γειτονικών δρομολογητών.

Αδυναμίες

Οι ISP χάρτες αποτελούνται από δρομολογητές δικτύου κορμού, πρόσβασης και απευθείας συνδεδεμένους με γειτονικών domains δρομολογητές μαζί με τη συσχέτιση τους σε επίπεδο IP.

Με την τεχνική του traceroute δεν αποκτάμε γνώση για την τοπολογία του δικτύου σε layer 2 (Data link) επίπεδο. Είναι σημαντικό να σημειώσουμε επίσης ότι όλοι οι γειτονικοί δρομολογητές δε συνδέονται με point-to-point σύνδεση, αλλά μπορεί να συνδέονται και μέσω μιας συσκευής του 2^{ου} επιπέδου (λ.χ. μέσω bridge ή LAN). Αυτές οι περιπτώσεις δεν μπορούν να διαφοροποιηθούν από τις point-to-point συνδέσεις μέσω traceroute.

Οι χάρτες αυτοί επίσης αποκλείουν κομμάτια του address space που αντιπροσωπεύονται από γειτονικά δίκτυα που δεν είναι συμβατά με το πρωτόκολλο BGP ή ευρυζωνικά δίκτυα καταναλωτών και δίκτυα dialup πρόσβασης.

Η φιλοσοφία που αναλύεται και στις παραπάνω τεχνικές, δηλαδή η προσπάθεια να μη ληφθεί υπόψη πληροφορία που μπορεί να είναι περιττή, μειώνει την

ακρίβεια στο βωμό της αποτελεσματικότητας και της ταχύτητας (με το ισοζύγιο βέβαια να είναι στο τέλος θετικό).

Rocketfuel

Τελικά, μέσω της μηχανής χαρτογράφησης Rocketfuel, που αναπτύχθηκε στα πλαίσια της εργασίας, σχηματίζονται οι χάρτες των ISPs χρησιμοποιώντας τις προαναφερθείσες τεχνικές.

Η αρχιτεκτονική του Rocketfuel μπορεί να αναλυθεί σε μερικά βασικά μέρη. Χρησιμοποιείται μία PostgreSQL βάση δεδομένων, 294 δημόσιοι traceroute servers και τα BGP tables από το RouteViews. Επιμέρους τμήματα της αρχιτεκτονικής του Rocketfuel είναι το egress discovery, tasklist generation, path reductions, execution engine, traceroute parser και alias resolution.

Αρχεία “asn.r1.cch”

Στα πλαίσια της αναπαράστασης δικτύου είναι σημαντικό να αναφέρουμε τι αντιπροσωπεύουν οι κόμβοι και οι ακμές στα δεδομένα μας (συγκεκριμένα στα αρχεία “asn.r1.cch”). Οι κόμβοι αποτελούν τους backbone και gateway routers καθώς και routers σε ακτίνα ίση με ένα (ένα επίπεδο) πιο κάτω από αυτούς, στον ISP που μελετάμε. Οι ακμές αντιπροσωπεύουν τη σύνδεση μεταξύ δύο τέτοιων δρομολογητών.

Συγκεκριμένα το *uid* είναι ένα μοναδικό αναγνωριστικό του κόμβου (αρνητικά μοναδικά αναγνωριστικά είναι εξωτερικές συνδέσεις) και το *<nuid-k>* είναι τα *uid* κάθε γείτονα εντός του παρόχου (δηλαδή συνδέσεις σε άλλους δρομολογητές εντός του ISP).

II. Ανάλυση δικτύου με Gephi

Για την εργασία επιλέχθηκε η μελέτη του ISP Verio από τις ΗΠΑ (2914.r1.cch). Με την προ-επεξεργασία των δεδομένων οι ακμές του μη-προσανατολισμένου δικτύου εξήχθησαν σε μορφή αρχείου csv και χρησιμοποιηθήκαν για την ανάλυση του δικτύου στο πρόγραμμα Gephi.

Η παρακάτω εικόνα είναι μία γραφική απεικόνιση του μελετώμενου δικτύου χρησιμοποιώντας τον αλγόριθμο *Force Atlas* (στις default τιμές του).

** Ο αλγόριθμος ForceAtlas κάνει τα γραφήματα πιο συμπαγή και αναγνώσιμα. Βελτιώνει επίσης τη σύγκλιση στα άκρα του σχεδίου.*

Στη συνέχεια παρατίθενται μερικά από τα στατιστικά που δίνονται από το Gephi.

Statistics	
Average Degree	3.263
Avg. Weighted Degree	3.263
Network Diameter	15
Graph Density	0.001
Modularity	0.86
Connected Components	12
Avg. Clustering Coefficient	0.182
Avg. Path Length	6.224

Nodes: *4670*
Edges: *7618*
Undirected Graph

Τυχαίο δίκτυο vs πραγματικό

Αν και γνωρίζουμε ότι δεν υπάρχουν δίκτυα στη φύση ή στον πραγματικό κόσμο τα οποία περιγράφονται από το μοντέλο του τυχαίου δικτύου, θα συγκρίνουμε ορισμένα χαρακτηριστικά του δικτύου μας με αυτά ενός random network.

Το τυχαίο δίκτυο θα λειτουργήσει ως μοντέλο αναφοράς για το δικό μας δίκτυο και θα μας βοηθήσεις να καταλάβουμε, μέσω της σύγκρισης με διάφορες μετρικές του, σε ποιο βαθμό μια συγκεκριμένη ιδιότητα είναι αποτέλεσμα μίας τυχαίας διεργασίας.

	Random Net.	Real Net.
Avg. Path Length	$\frac{logN}{log<k>}$ = 7.1440	6.224
Avg. Clustering Coefficient	$\frac{<k>}{N}$ = 6.98×10^{-4}	0.182
Degree Distribution	Προσεγγίζει Poisson	Άλλη

- Το πραγματικό μας δίκτυο παρατηρούμε ότι έχει μικρές αποστάσεις όπως θα είχε ένα τυχαίο δίκτυο, κάτι που συμφωνεί με τη θεωρία. Αυτό φανερώνει και την ύπαρξη *small world* φαινομένου, δηλαδή ότι το *avg. path length* εξαρτάται λογαριθμικά από το μέγεθος του συστήματος, είναι αντίστοιχο του *logN* αντί του *N*.

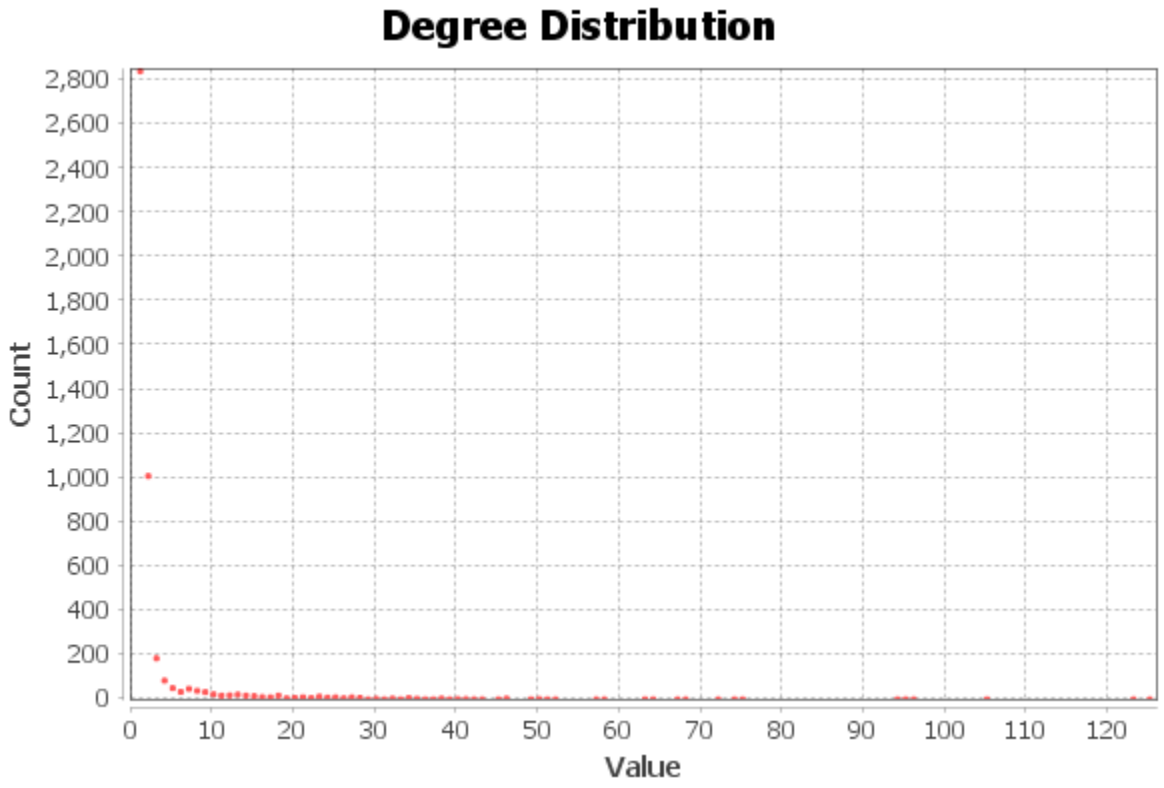
- Το clustering coefficient ενός τυχαίου δικτύου είναι μικρό. Βλέπουμε πράγματι ότι το C_{rand} είναι μερικές τάξεις μεγέθους μικρότερο από το C_{real}. Στα τυχαία δίκτυα αναμένουμε από ένα σημείο και μετά ένα και μοναδικό μεγάλο cluster.

- Η κατανομή των τυχαίων δικτύων είναι διωνυμική και για μεγάλα N και μικρά k προσεγγίζει την κατανομή Poisson. Η κατανομή του πραγματικού μας δικτύου μοιάζει περισσότερο με εκθετική κατανομή και φαίνεται στο παρακάτω διάγραμμα.

Degree Report

Results:

Average Degree: 3.263



* Το <*k*> είναι σχετικά μικρό, γεγονός που είναι λογικό μιας και δε συνδέονται όλοι οι routers με όλους τους routers, έχουμε ένα sparse δίκτυο. Το γεγονός αυτό μας ωθεί να μελετήσουμε και τη διαθεσιμότητα του συστήματός μας.

Outliers – Ακραίες τιμές

Παρατηρούμε ορισμένες δυάδες και τριάδες κόμβων που είναι αποκομμένες από τον υπόλοιπο γράφο. Ψάχνοντας αυτές τις τιμές στο datasheet παρατηρούμε ότι μάλλον πρόκειται για edge routers διότι συνδέονται μόνο με έναν κόμβο που ανήκει στον ISP και με έναν εξωτερικό κόμβο. Παράδειγμα οι κόμβοι με *uid* 992 και 7091:

- ✓ 992 @Mountain+View,+CA (1) &2 -> <7091> {-32} {-9} =131.103.121.158 r1
- ✓ 7091 @Mountain+View,+CA + (1) -> <992> =d1-5-0-0-20.a03.mtvwca01.us.ra.verio.net r0

Ενδιαφέρον έχει και η περίπτωση του Smithtown,+NY, όπου έχουμε αρκετούς κόμβους να συνδέονται μεταξύ τους, αλλά να είναι αποκομμένοι από τον υπόλοιπο γράφο.

Αφαιρούμε αυτά τα στοιχεία και ξαναβλέπουμε τα στατιστικά.

	Statistics	
	New	**Old**
Average Degree	3.282	3.263
Avg. Weighted Degree	3.282	3.263
Network Diameter	15	15
Graph Density	0.001	0.001
Modularity	0.868	0.86
Connected Components	1	12
Avg. Clustering Coefficient	0.181	0.182
Avg. Path Length	6.225	6.224

* Παρατηρούμε ότι ο αριθμός των connected components τώρα που έφυγαν τα outliers προφανώς μειώνεται.

Διαθεσιμότητα & Communities

Όπως αναφέραμε και πριν, στο αραιό (sparse) σχετικά δίκτυό μας, μας ενδιαφέρει να μελετήσουμε τη διαθεσιμότητά του σε ακραίες περιπτώσεις. Σκοπός είναι να εντοπίσουμε τους σημαντικούς κόμβους (ορίζοντας τη σημαντικότητα με κάποιες συγκεκριμένες μετρικές), να τους αφαιρέσουμε και να δούμε ορισμένα χαρακτηριστικά του γράφου μετά.

Στη θεωρία γράφων και στην ανάλυση δικτύων, οι δείκτες centrality προσδιορίζουν τους πιο σημαντικούς κόμβους μέσα σε ένα γράφημα.

Betweenness centrality

Αρχικά θα θεωρήσουμε ως δείκτη centrality το betweenness centrality. Το betweenness centrality έχει πολύ μεγάλη σημασία στο γράφο μας, όπου οι κόμβοι του είναι δρομολογητές και οι ακμές μεταξύ τους είναι οι συνδέσεις τους. Η μετρική αυτή αντιπροσωπεύει το βαθμό στον οποίο οι κόμβοι στέκονται μεταξύ τους, έτσι, στο δίκτυο μας (ένα τηλεπικοινωνιακό δίκτυο), ένας κόμβος με υψηλότερη κεντρική θέση θα έχει περισσότερο έλεγχο στο δίκτυο, επειδή περισσότερες πληροφορίες θα περάσουν από αυτόν τον κόμβο.

Παρακάτω δίνουμε το γράφο χρωματισμένο ανάλογα με το modularity class και τους κόμβους μεγεθυμένους ανάλογα με το betweenness centrality.

Το modularity είναι μία μετρική για τη δομή των δικτύων (ή των γράφων). Μετράει την ισχύ της διαίρεσης ενός δικτύου σε modules ή communities.

Οι τέσσερεις κόμβοι με το μεγαλύτερο betweenness centrality ήταν όλοι backbone routers και συγκεκριμένα:

- 4067 @McLean,+VA + bb
- 4134 @Palo+Alto,+CA + bb
- 4064 @McLean,+VA + bb
- 4017 @Chicago,+IL + bb

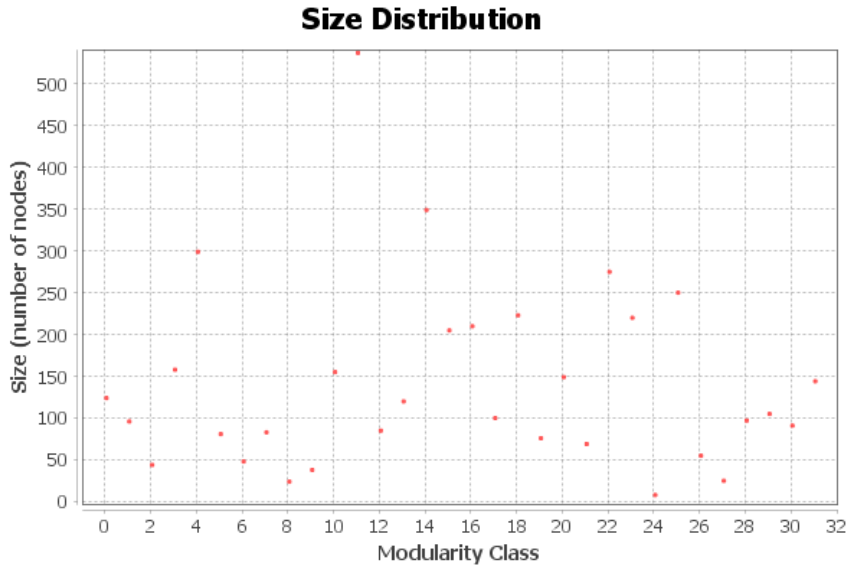
Με την αφαίρεση του κόμβου με το μεγαλύτερο betweenness centrality το δίκτυο έσπασε σε 2 connected components, αλλά το ένα component αποτελούνταν μόνο από δύο κόμβους. Ακόμα και με την αφαίρεση των 10 μεγαλύτερων σε betweenness centrality κόμβων το 98% των κόμβων άνηκε στο ίδιο component (το υπόλοιπο μοιράστηκε σε 73 components – avg. path length αυξήθηκε στο 7.21).

Τα παραπάνω δείχνουν ότι το δίκτυο διατηρεί μια καλή διαθεσιμότητα ακόμα και αν πέσουν μερικοί κεντρικοί κόμβοι του.

Μία μικρή αύξηση παρατηρήθηκε και στον αριθμό των communities. Πιο συγκεκριμένα:

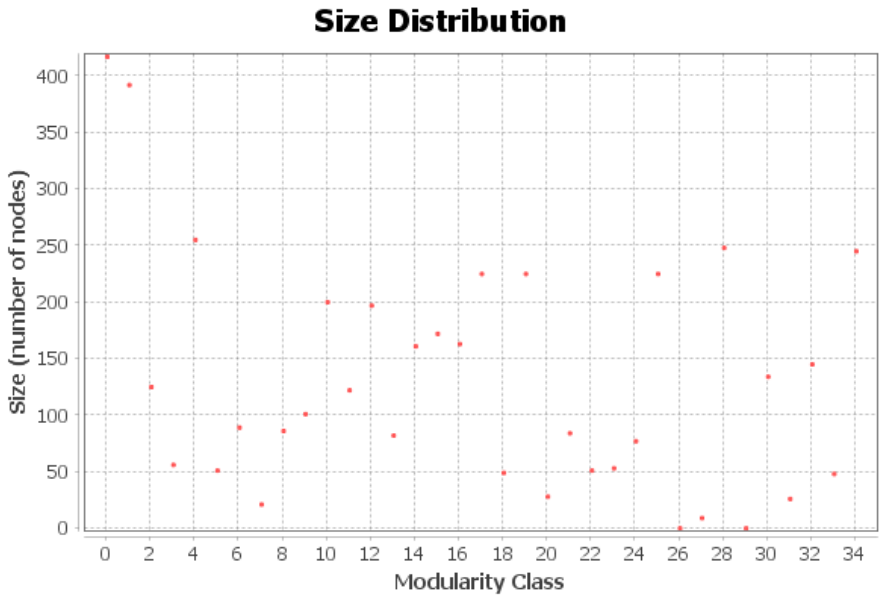
Modularity Report

Πριν την αφαίρεση των 10 μεγαλύτερων σε betweenness centrality κόμβων:
Modularity: 0.872
Modularity with resolution: 0.872
Number of Communities: 32



Modularity Report

Μετά την αφαίρεση των 10 μεγαλύτερων σε betweenness centrality κόμβων:
Modularity: 0.895
Modularity with resolution: 0.895
Number of Communities: 35



Closeness Centrality

Το closeness centrality αποτελεί μία άλλη μετρική του centrality. Υπολογίζεται ως το αντίστροφο του αθροίσματος του μήκους των συντομότερων διαδρομών μεταξύ ενός κόμβου και όλων των άλλων κόμβων του γραφήματος. Έτσι, όσο πιο κεντρικός είναι ένας κόμβος, τόσο πιο κοντά είναι σε όλους τους άλλους κόμβους.

Μετά την αφαίρεση των 10 μεγαλύτερων σε closeness centrality κόμβων, πάλι το μεγαλύτερο ποσοστό των κόμβων ανήκει στο ίδιο component id (με ποσοστό 99.63%). Ομοίως είχαμε αύξηση του αριθμού των communities σε 41 (με Modularity: 0.891).

Παρακάτω δίνουμε το γράφο χρωματισμένο ανάλογα με το modularity class και τους κόμβους μεγεθυμένους ανάλογα με το closeness centrality.

Eigenvector centrality

Στη θεωρία γράφων, το eigenvector centrality είναι ένα μέτρο της επίδρασης ενός κόμβου σε ένα δίκτυο. Αποδίδονται σχετικές βαθμολογίες σε όλους τους κόμβους του δικτύου βάσει της έννοιας ότι οι συνδέσεις με κόμβους υψηλής βαθμολογίας συμβάλλουν περισσότερο στην βαθμολογία του εν λόγω κόμβου από ό, τι οι ίδιες συνδέσεις με κόμβους χαμηλής βαθμολογίας. Ένα υψηλό eigenvector score σημαίνει ότι ένας κόμβος συνδέεται με πολλούς κόμβους οι οποίοι έχουν και οι ίδιοι υψηλή βαθμολογία (score).

Με την αφαίρεση των 10 μεγαλύτερων σε eigenvector centrality κόμβων παρατηρούμε μεγαλύτερο σπάσιμο των connected components από ό,τι στις δύο προηγούμενες μετρικές centrality. Το μεγαλύτερο ποσοστό των κόμβων ανήκει στο ίδιο component id με ποσοστό 95.43% και το υπόλοιπο ποσοστό

διαμοιράζεται σε 193 components (με μικρό ποσοστό). Ομοίως είχαμε αύξηση του αριθμού των communities σε 230 (με Modularity: 0.894).

Παρακάτω δίνουμε το γράφο χρωματισμένο ανάλογα με το modularity class και τους κόμβους μεγεθυμένους ανάλογα με το eigenvector centrality.

Hubs

Στην επιστήμη των δικτύων, ένα hub είναι ένας κόμβος με έναν αριθμό συνδέσμων που υπερβαίνει κατά πολύ τον μέσο όρο. Η εμφάνιση των κόμβων είναι συνέπεια της scale free ιδιότητας των δικτύων. Ενώ τα hubs δεν μπορούν να παρατηρηθούν σε τυχαίο δίκτυο, αναμένεται να εμφανιστούν σε scale free δίκτυα. Περισσότερα για τα scale free δίκτυα θα αναφέρουμε στο Ερώτημα 3.

Από την οπτικοποίηση του δικτύου μας παρατηρούμε αρκετές «βεντάλιες», δηλαδή κόμβους με τους οποίους συνδέονται πολλοί άλλοι, με άλλα λόγια hubs. Αυτό φαίνεται και στο degree distribution που σχολιάσαμε προηγουμένως και παραθέσαμε το διάγραμμά του.

Ερώτημα 3: Προεκτάσεις

Επαλήθευση αποτελεσμάτων Gephi

Matlab

Χρησιμοποιώντας το εργαλείο προγραμματισμού Matlab θα επαληθεύσουμε την ορθότητα ορισμένων αποτελεσμάτων που έδωσε το Gephi.

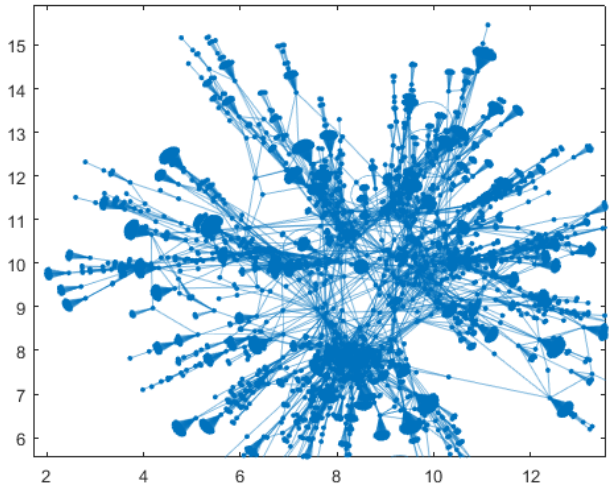
Matlab code

Σχεδίαση του γράφου και υπολογισμός του μέσου βαθμού (πριν την αφαίρεση των outliers):

```
nodes1 = VarName1;      % import the data
nodes2 = VarName2;      % every one of the two nodes of an edge is in nodes1,
nodes2 respectively
G = graph(nodes1,nodes2);
D = degree(G);          % find the degree of each node
index = find(D == 0);   % find the nodes with degree 0
H = rmnode(G,index);    % remove the nodes with degree zero and create a new
graph
Deg = degree(H);
avg_degree = mean(Deg)
plot(H);
h = plot(H,'Layout','force'); % change layout of the graph
```

Avg. degree Matlab: 3.2625

Avg. degree Gephi: 3.263



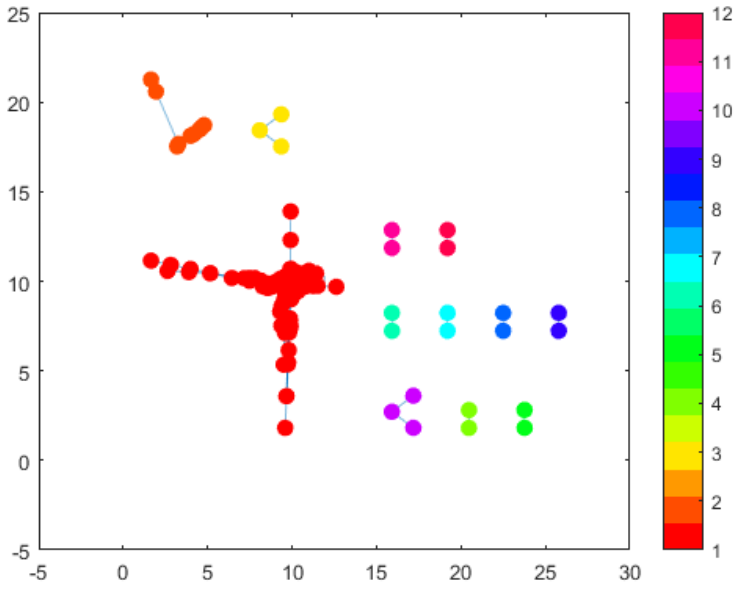
Απόσπασμα γράφου σχεδιασμένου στο Matlab

Ο αριθμός των nodes και edges είναι προφανώς ακριβώς ο ίδιος:

```
numnodes(H);
numedges(H);
```

Βλέπουμε τα connected components (πριν την αφαίρεση των outliers):

```
% Create and plot a graph that contains several strongly connected
components.
% Highlight the strongly connected components.
p = plot(H);
bins = conncomp(H);
p.MarkerSize = 7;
p.NodeCData = bins;
colormap(hsv(20))
```



Γράφος με τα connected components στο Matlab

Connected components Matlab: 12

Connected components Gephi: 12

R

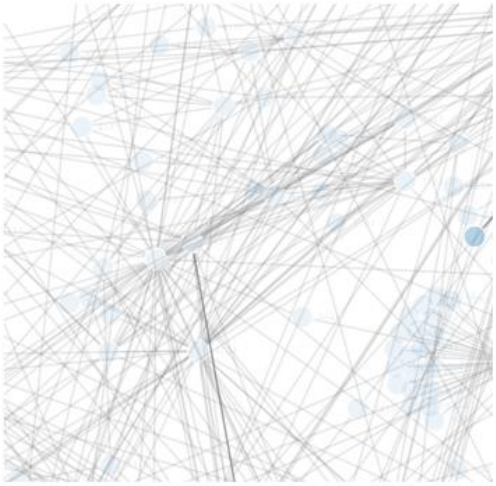
Χρησιμοποιώντας το εργαλείο προγραμματισμού R (iGraph, networkD3) θα επαληθεύσουμε την ορθότητα ορισμένων αποτελεσμάτων που έδωσε το Gephi.

Η χρήση της R και συγκεκριμένα του πακέτου iGraph αποδείχθηκε πιο αποδοτική σε σχέση με το Matlab.

R code

```
edges=read("edges.csv")
rm(g)
e = c()
g <- make_empty_graph(n = 2*nrow(edges),directed=FALSE)
for(i in 1:nrow(edges)){                        # create a matrix with every
vetrice of an edge side by side
  #e = c(c(e), c(edges[i,1]), c(edges[i,2]))
  g <- g + edges(edges[i,1],edges[i,2])
}

E(g)[[]]
plot(g)
plot(g, edge.arrow.size=0.4,vertex.label=NA)

length(V(g))  # number of vertices in graph
gsize(g)      # number of edges in graph

deg = degree(g) # find the degrees of each node
deg = list(deg) # put them into a list
deg = lapply(deg, function(x) {x[x!=0]}) # get rid of the degrees of 0

deg = as.numeric(unlist(deg)) # unlist and make values numeric (to apply
mean())
mean = mean(deg)              # ans = 3.262099
diameter(g)                   # ans = 15
edge_density(g)               # ans = 6.564695e-05
transitivity(g)              # ans = 0.05983204. Transitivity measures the
probability
                              # that the adjacent vertices of a vertex are
connected. This is
                              #  sometimes  also  called  the  clustering
coefficient.
```

```
mean_distance(g)                # ans = 6.229228. Avg. path length

library("networkD3")            # library for better layout of plots
simpleNetwork(edges)
```



Απόσπασμα γράφου σχεδιασμένου με το networkD3

	Comparison	
	R (iGraph)	**Matlab**
Average Degree	3.262099	3.263
Network Diameter	15	15
Transitivity (R) - Avg. Clustering Coefficient (Matlab)	0.05983204	0.182
Avg. Path Length	6.229228	6.224

** Η χρήση του iGraph μπορούσε να γίνει και μέσω της Python, αλλά μετά από σχετική δοκιμή δεν κρίθηκε αναγκαίο. Επίσης, έγινε χρήση του πακέτου NetworkX στην Python, αλλά προσωπική άποψη του συγγραφέα είναι ότι δεν είναι προτιμότερη αυτή η λύση από τα άλλα προγραμματιστικά εργαλεία που εξετάστηκαν.*

Scale-free networks

Στις προεκτάσεις θα αναλυθεί μία ιδιότητα που κέντρισε το ενδιαφέρον, το scale-free property (Power law).

Στη στατιστική, ένας νόμος δύναμης (power law) είναι μια συναρτησιακή σχέση μεταξύ δύο ποσοτήτων, όπου μια ποσότητα μεταβάλλεται ως δύναμη της άλλης.

Ένα χαρακτηριστικό της κατανομής νόμου δύναμης είναι η σταθερότητα της κλίμακας. Αν ισχύει $f(x) = ax^k$, για τον υπολογισμό της συνάρτησης με τη μεταβλητή *x* πολλαπλασιασμένη κατά ένα σταθερό παράγοντα *c* παρατηρούμε ότι δημιουργείται απλά μία ανάλογη απεικόνιση της ίδιας της συνάρτησης. Δηλαδή $f(cx) = a(cx)^k = c^k f(x)$.

Αυτή η συμπεριφορά είναι που παράγει τη γραμμική σχέση όταν λογαριθμίζουμε και τα δύο μέρη της συνάρτησης f (η ευθεία γραμμή στο γράφημα των λογαρίθμων αποκαλείται *υπογραφή* του νόμου δύναμης).

Στα πραγματικά δεδομένα, η γραμμική σχέση είναι αναγκαία, αλλά όχι και ικανή συνθήκη για δεδομένα που συνδέονται με κατανομή νόμου δύναμης. *(πηγή: https://el.wikipedia.org/wiki/Κατανομή_νόμου_δύναμης)*

Στην ανάλυση μας θα χρησιμοποιήσουμε γραφικές μεθόδους για την διαπίστωση ύπαρξης power law. Αν και αυτές οι μέθοδοι δεν αποτελούν αυστηρή απόδειξη, θα μας δώσουν μία ένδειξη ύπαρξης κατανομής νόμου δύναμης.

R function

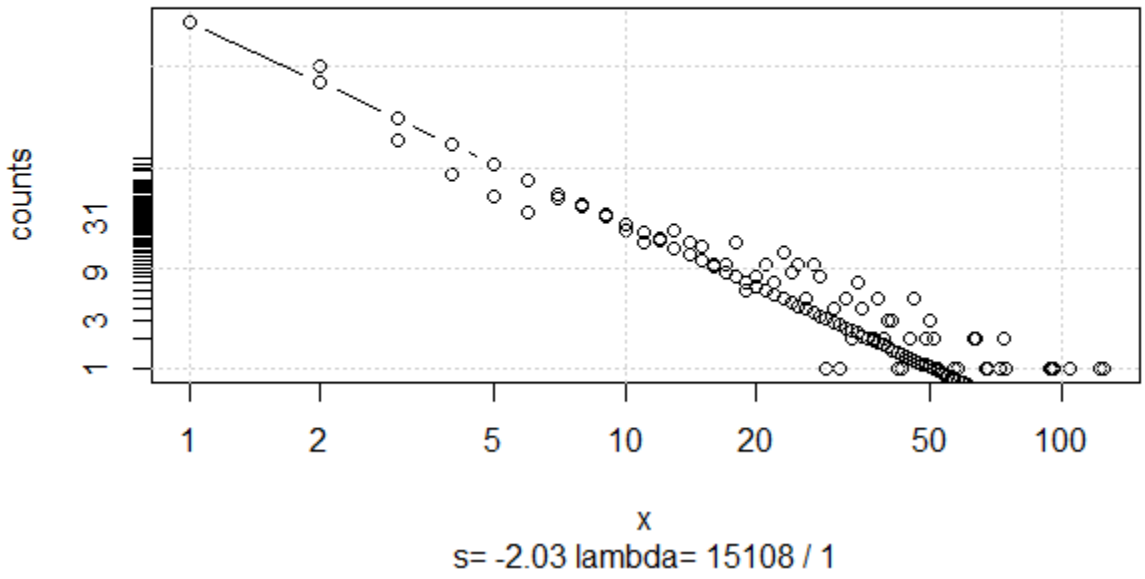
Η παρακάτω συνάρτηση στην R υπολογίζει τον εκθέτη, αποτυπώνει τα δεδομένα σε γράφημα log-log και εντοπίζει τη γραμμή που κάνει fitting.

```
pwrdist <- function(u,...) {
        # u is vector of event counts, e.g. how many
        # crimes was a given perpetrator charged for by the police
        fx <- table(u)
        i <- as.numeric(names(fx))
        y <- rep(0,max(i))
        y[i] <- fx
        m0 <- glm(y~log(1:max(i)),family=quasipoisson())
        print(summary(m0))
        sub <-
paste("s=",round(m0$coef[2],2),"lambda=",sum(u),"/",length(u))
        plot(i,fx,log="xy",xlab="x",sub=sub,ylab="counts",...)
        grid()
        lines(1:max(i),(fitted(m0)),type="b")
        return(m0)
    }
```

Έχοντας εξάγει σε μορφή csv όλους του βαθμούς των κόμβων του γράφου μας, τρέχουμε την παραπάνω συνάρτηση στην R.

```
> degree = read.csv("degree.csv")
> pwrdist(degree)
```

Τα αποτελέσματα δίνονται παρακάτω:



Φαίνεται να έχουμε μία ένδειξη ύπαρξης power law καθώς τα δεδομένα κάθονται σε μία ευθεία πάνω στο log-log γράφημα.