Topology and congestion invariant in global internet-scale networks

Huang, Zhijia

Topology and Congestion Invariant in Global Internet-Scale Networks

Zhijia Huang

School of Electronic Engineering and Computer Science

Queen Mary University of London
London E1 4NS, United Kingdom

A thesis submitted for the degree of
Doctor of Philosophy

June 2010

Disclaimer

I hereby declare that the work in this thesis is that of my work alone and that to the best of my knowledge the work is original except where indicated by reference to respective authors.

Zhijia Huang :

Date :

To my loving family.

Acknowledgements

First of all I would like to acknowledge my supervisor Dr Raúl Mondragón for his patient guidance and advice given to me throughout the years of my PhD study and research. Without him I would not achieve the research goals during this period.

I would also like to thank Dr Matthew Woolf for the advice and help on EPSRC (Engineering and Physical Sciences Research Council) network invariant project as research assistant; Dr John Schormans for the discussion on Queueing theory; Dr Xiaonan Yang, Dr Huai Huang, Dr Ling Liu, and Dr Linlin Song for their help through the hard times of getting the research on progress; Dr Vindya Wijeratne and Dr Qiang Yang for their help on Queueing Theory; and all the others on the third and fourth floor for the discussions and fun time together.

I would also take the chance to thank my parents for their love, guidance and encouragement that they have always provided. Thanks Dr Shuxian Chen for her everyday help throughout the years. Last but not least thanks to all my friends.

Financial support from EPSRC (Engineering and Physical Sciences Research Council) is thankfully acknowledged.

Abstract

Infrastructures like telecommunication systems, power transmission grids and the Internet are complex networks that are vulnerable to catastrophic failure. A common mechanism behind this kind of failure is avalanche-like breakdown of the network's components. If a component fails due to overload, its load will be redistributed, causing other components to overload and fail. This failure can propagate throughout the entire network. From studies of catastrophic failures in different technological networks, the consensus is that the occurrence of a catastrophe is due to the interaction between the connectivity and the dynamical behaviour of the networks' elements.

The research in this thesis focuses particularly on packet-oriented networks. In these networks the traffic (dynamics) and the topology (connectivity) are coupled by the routing mechanisms. The interactions between the network's topology and its traffic are complex as they depend on many parameters, e.g. Quality of Service, congestion management (queuing), link bandwidth, link delay, and types of traffic. It is not straightforward to predict whether a network will fail catastrophically or not. Furthermore, even if considering a very simplified version of packet networks, there are still fundamental questions about catastrophic behaviour that have not been studied, such as: will a network become unstable and fail catastrophically as its size increases; do catastrophic networks have specific connectivity properties?

One of the main difficulties when studying these questions is that, in general, we do not know in advance if a network is going to fail catastrophically. In this thesis we study how to build catastrophic

networks. The motivation behind the research is that once we have constructed networks that will fail catastrophically then we can study its behaviour before the catastrophe occurs, for example the dynamical behaviour of the nodes before an imminent catastrophe.

Our theoretical and algorithmic approach is based on the observation that for many simple networks there is a topology-traffic invariant for the onset of congestion. We have extended this approach to consider cascading congestion. We have developed two methods to construct catastrophes. The main results in this thesis are that there is a family of catastrophic networks that have a scale invariant; hence at the break point it is possible to predict the behaviour of large networks by studying a much smaller network. The results also suggest that if the traffic on a network increases exponentially, then there is a maximum size that a network can have, after that the network will always fail catastrophically.

To verify if catastrophic networks built using our algorithmic approach can reflect real situations, we evaluated the performance of a small catastrophic network. By building the scenario using open source network simulation software OMNet++, we were able to simulate a router network using the Open Shortest Path First routing protocol and carrying User Datagram Protocol traffic. Our results show that this kind of networks can collapse as a cascade of failures. Furthermore, recently the failure of Google Mail routers [1] confirms this kind of catastrophic failure does occur in real situations.

Contents

List of Figures

List of Tables

Glossary

3G	Third Generation mobile service
ARPANET	Advanced Research Projects Agency Network
AS	Autonomous System
BBC	British Broadcasting Corporation
BGP	Border Gateway Protocol
CAIDA	Cooperative Association for Internet Data Analysis
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DNS	Domain Name System
DoS	Denial of Service
EGP	Exterior Gateway Protocol
EIGRP	Enhanced Interior Gateway Routing Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronic Engineering
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO	International Standard Organization
ISP	Internet Service Providor
ITU	International Telecommunication Union
LAN	Local Area Network
LSA	Link State Advertisement
MPLS	Multiprotocol Label Switching

MTU	Maximum transmission unit
NS2	Network Simulation 2
NS3	Network Simulation 3
OS	Operating System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PC	Personal Computer
PDNS	Parallel Distributed Network Simulation
PPP	Point to Point Protocol
QoS	Quality of Service
RED	Random Early Discard
RFC	Request For Comment
RIP	Routing Information Protocol
RTP	Real-time Transport Protocol
SIP	Session Initiation Protocol
ToS	Type of Service
TCP	Transport Control Protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
VoIP	Voice over IP
WLAN	Wireless Local Area Network
WWW	World Wide Web

Chapter 1

Introduction

This chapter first introduces the challenges we are facing in designing and managing computer networks. It also states the aim, objectives and contributions of this thesis. Finally, an outline of the thesis is given as a road map for the rest of the content.

1.1 Challenge

1.1.1 The Studies on The Internet

The Internet is a worldwide network of interconnected networks with various devices (routers, switches, servers, and PCs) using different operating systems, that transmits data using a wide range of protocols. It is simply a "network of networks" that consists of millions of domestic, academic, business, and government networks, which together deliver various information and services. The explosive growth of the Internet from the 1990s has led to the emergence of a computational network that covers the globe, with a staggering number of users connected to it, browsing the World Wide Web, transferring data files, making VoIP (Voice over IP) calls and watching bandwidth–consuming videos. The intrinsic heterogeneity of the Internet of various type of devices and connections, added to the unpredictability of traffic on it, makes the Internet a very complex dynamic system. The Internet has dynamic traffic and topology although the changes of traffic and topology have very different time scale.

From the mid-1990s scientists and engineers try to understand what does the Internet look like and what kind of properties it has. For example, how big it is, how well connected it is? Engineers approach this problem from measurement–mapping and visualizing the Internet. Some of them use BGP (Border Gateway Protocol) routing tables, Route Servers or "traceroute" to gather information for the Internet AS (autonomous system) level topology [5; 10; 11; 12]. Others use "traceroute" [5; 13; 14; 15; 16] or "ping" to map Internet router-level topology. But the measurement might not be very accurate for mapping the router-level topology where some layer2 protocols, such as ATM (Asynchronous Transfer Mode) and Multiprotocol Label Switching (MPLS) [17] might hide parts of the Internet topology. These layer 2 topologies are not visible to "ping" and "traceroute". But the data collected from the simple "ping" and "traceroute" is very valuable for us to start understanding the IP-level of the Internet topology. Physicists and mathematicians try to interpret the raw data from measurements [18; 19]. Some also attempted to model the Internet topology [20; 21; 22; 23; 24].

In fact, not only the topology of the Internet, but also the traffic on it has long been studied. The classic queueing theory [25; 26], which is the mathematical study of queues. It enables mathematical analysis of several related processes, including packets' arrivals, waiting in the queue, and being served. There is also a discovery of the self-similar properties of Ethernet traffic [27; 28].

But there is a lack of in-depth study of both the traffic dynamics and the underlying network infrastructure, their impact on each other and the overall network performance. Some pioneer researchers have begun the analysis of both traffic and topology from network congestion, where the interaction between them can be measured. Ohira and Sawatari [29] are the first to study the simple model of computer network traffic that can exhibit a phase transition from a low (free flow) to high congestion state. Then Fukś and Lawniczak [30], Solé and Valverde [31; 32], and Woolf *et.al* [33], found that, in a regular network [1], it is possible to predict the traffic load where congestion occurs (a dynamical characteristic) by only considering the average of the shortest path lengths from all sources to all destinations(a topology characteristic).

[1]Network with nodes having the same degree. Please check Chapter 2 for detail explanation.

1.1.2 Network Congestion,Vulnerability and Failures

Network congestions (overloading) in some cases will lead to network failure. We consider network failure as the situation that the network cannot provide normal services to users and hard to recover. Man-made complex networks [20; 34; 35; 36] such as the Internet, power transmission grids and telephone systems are susceptible to catastrophic failures in which the entire network ceases to function [37; 38; 39]. The most common form of a catastrophic failure is an avalanche-like cascade breakdown. This can result from the failure of a single node in a network in which nodes are sensitive to overloading. Redistribution of the load carried by this failed node over the network may cause other nodes to be overloaded and fail as well. Consequently this would trigger an avalanche-like event in which node failures propagate throughout the network. The entire network might in the end fragment into disconnected sub-networks. It is claimed by some researchers that networks with heterogeneous node degree distribution, such as scale-free networks, are much more likely to suffer this type of event [40; 41; 42; 43; 44]. Because a small subset of core nodes are highly connected and handle much of the traffic in this type of network. If one of these heavily loaded core nodes ceases to function, either through malicious attack or random failure, it will have a large impact on other nodes in the network, making subsequent failures very likely.

However, previous researchers do not explain if similar catastrophic failures are possible in networks with more homogeneous degree distributions. Furthermore, they do not show and analyse the properties of the cascade networks, i.e. degree distribution. Nor do they set up network simulations to test the topologies they modeled. Based on the importance of the robustness of the existing network infrastructure and the rapidly increasing demand for resources in the network usage, it is crucial to learn the pattern and properties networks that fail catastrophically. After this, we could explore different ways to prevent the catastrophic failure from happening. The starting point is to follow previous studies on the Internet congestion and failures. Especially, the cascading failures that could lead to a almost complete breakdown of the network.

1.2 Objectives

Our goal is to analysis the properties of catastrophic failures on networks. To achieve this aim we intend to:

1. Develop a new model to build network topologies that will fail catastrophically. The method will take into consideration not only the connectivity of the network but also the traffic it carries.

2. Optimize the method developed in "objective 1". Analyse the topological properties of these generated catastrophic networks.

3. To find out if there is any topology-traffic invariants for catastrophic networks. For example, can it be used to create and study larger size networks?

4. Verify the catastrophic network topology using network simulation. Carry out simulations using different size catastrophic networks to examine if the networks built using the method in "objective 1", do fail catastrophically. Discover more properties related to network failures, especially catastrophic failures.

1.2.1 Methodology Outline

Enlightened by previous researches in evolving complex networks and analysis of complex network robustness, two novel methods are proposed to further investigate the vulnerability of complex networks, especially the Internet. It is described in the following steps:

- Investigate the network congestion and network failures, especially congestion and failures on the Internet.

- Review existing network generation models including cascading network generation studies.

- Propose the "random growth" network generation model to build networks that are designed to fail by steps when the load on it exceed the threshold (critical load).

- Modify and improve the "random growth" model by introducing the "Branch and Bound" growth model to have better control over the generation of catastrophic networks.
- Find out the generated catastrophic networks' properties (using both methods) and compare them with other networks.
- Build the network simulation scenarios to verify catastrophes on topologies generated using proposed method and collect various traffic data for analysis.

1.2.2 Growth Model

A novel way of analysing network failures has been proposed by generating network topologies that are designed to fail (catastrophic networks). The topology generation follows the avalanche-like network failures in reverse, where it starts from a small core network. We considered the node failure/overload in our research using node betweenness centrality. It is also possible to consider link failure/overload using edge betweenesss which is beyond our research. Nodes in the network are sensitive to overloading that the load of nodes are estimated using betweenness centrality. The biggest challenge is the searching of network topology meeting the growing condition in the solution space for possible topology. Because the solution space grows with N (size of the network) at 2^N. The challenge also lies in calculating betweenness centrality for each searched topology to examine if they match the growing conditions. If not, the cascade failure would not happen in the designed topology. The novel random growth method is used to explore the solution space with random walks. Program is coded in C++ and successfully generated the network topology that will fail due to overloading. Two more efficient and novel searching methods are proposed by considering the constraints on generating catastrophic network topology. The network in a growth step is divided into three sub-networks to implement the new methods. New codes are written to implement the new methods with the random search growth program.

1.2.3 Invariant

Invariants in node degree and node betweenness centrality are found in the catastrophic network topologies generated using the proposed methods. We find that there is a special family of catastrophic networks whose connections follow a specific pattern relating to the degree distribution and betweenness centrality properties. This observation hints that it is possible to construct very large catastrophic network by only specifying the degree and betweenness centrality.

1.3 Contributions

The main contributions of this thesis, which distinguish it from prior research are:

1. Developed two novel growth methods to construct catastrophic networks, including "random search" and "branch-and-bound" methods.
2. Found a new topological invariant related to catastrophic networks.
3. Verified that the catastrophic network topologies do fail according to the design.

1.4 Organization of the Thesis

This thesis is organised in six additional chapters as follows. **Chapter 2** introduces different background knowledge used in the thesis. They are graph theory basics and related properties, betweenness centrality, complex network models and the Internet. We also introduce the cascade and catastrophic failures on networks which is the main goal of the research. **Chapter 3** served as a foreground of the research by introducing network congestion and its metrics. It also reviews previous work in predicting congestion in different networks and examines the invariants relate to the onset of congestion.

Chapter 4 takes a closer look at network congestion and the spread of it. Then the causes of a cascading failure of networks and finally a catastrophic failure of the whole network are studied in detail. A novel catastrophic network topology

generation model using "random growth" method is proposed with analysis of the generated networks. With the method introduced, one can build a small size catastrophic network with similar nodes and links. **Chapter 5** introduces the optimized "branch and bound" method to grow catastrophic networks under precise control of the network failure load and failure steps.

Chapter 6 focuses on the network simulation on the open source simulation platform available to validate the catastrophic network topology. We try to see if these networks do fail catastrophically as built under different simulation scenarios.

Finally, a conclusion is made in **Chapter 7** about the research in this thesis and the possible future work that could be carried out to extend the research has been provided.

Chapter 2

Background

The study of network models can be conceptualized as lying at the intersection between graph theory and statistical mechanics [36]. A first step to understand the complicated behaviour of networks lies in the accurate description of the networks' topological properties. Section 2.1 is devoted to the basic notions and properties of networks based on graph theory. It is followed by a brief overview of the development of network models. These models are commonly used to try to reproduce the topological and dynamical properties observed in real world networks. Examples of real world networks are presented together with the analysis of their important topological properties. Special attention is paid to the Internet, whose topological and dynamical properties will be further investigated in following chapters. Previous studies of the robustness of networks are reviewed in Section 2.3. And we focus on introducing network cascading failures and network catastrophes. They have attracted more and more attention recently due to the rapid development of man-made complex networks that we increasingly relied on, i.e. the Internet, Electricity Power Grid.

The present chapter is not supposed to be an exhaustive review of all the recent developments in graph theory, complex networks and the Internet. Detail and in-depth review of complex networks can be found in the following books [45; 46; 47; 48] and articles [20; 34; 35; 36; 49]. Similarly, for an introduction to graph theory the following books are recommended [50; 51; 52]. Last but not least, an overview of computer network technology could be found in [53] while Internet related protocols and standards could be obtained from IETF (Internet

Engineering Task Force), IEEE (Institute of Electrical and Electronics Engineers), ITU (International Telecommunication Union) and other standard organizations.

2.1 Basic Graph Theory

The general concept of a network is a set of nodes (vertices) connected via a number of links (edges). It can describe a wide range of systems either natural or man-made. It is frequently used to study the biological networks (cellular and neural network), food chains, human social relations, citations, power grids, wired and wireless communication networks. In the mathematical terms, a network can be represented as a graph $G = (V, E)$, where V is the set of vertices (or nodes) $v_1, v_2, \ldots, v_n$ and E is the set of links (or edges) $e_1, e_2, \ldots$ that connect two elements of V.

Generally there are four kinds of graphs: directed weighted graphs, directed unweighted graphs, undirected weighted graphs and undirected unweighted graphs. In an undirected graph if two vertices i and j are adjacent, there would be an edge between them represented by unordered pair (i, j) or (j, i), see Figure 2.1(a). On the other hand, in directed graph the edges are only one-way, see Figure 2.1(b). There are two kinds of edges in directed graph, the in–edges and the out–edges.

A graph can be fully characterized by its vertices and edges. Alternatively it is usually described by its adjacency matrix $\hat{\mathcal{A}}$, where $\hat{\mathcal{A}}_{ij}$ represent the connection from node i to node j. In general, $\hat{\mathcal{A}}_{ij} \neq 0$ indicates the existence of an link, while $\hat{\mathcal{A}}_{ij} = 0$ stands for the absence of an link. In weighted graph, every edge has a weight, where the whole graph can be represented by weighted matrix. A subgraph $G' = (V', E')$ of graph $G = (V, E)$ is a graph having all of its vertices and edges in G, $V' \subset V, E' \subset E$ [50]. Networks studied in this thesis are undirected unweighted networks containing no loops or multiple links [50; 54], also known as simple network (graph).

2.1.1 Degree and Degree Distribution

The simplest and the most intensively studied graph characteristic is the node's degree. The degree of a node is the number of links connected to that node (see

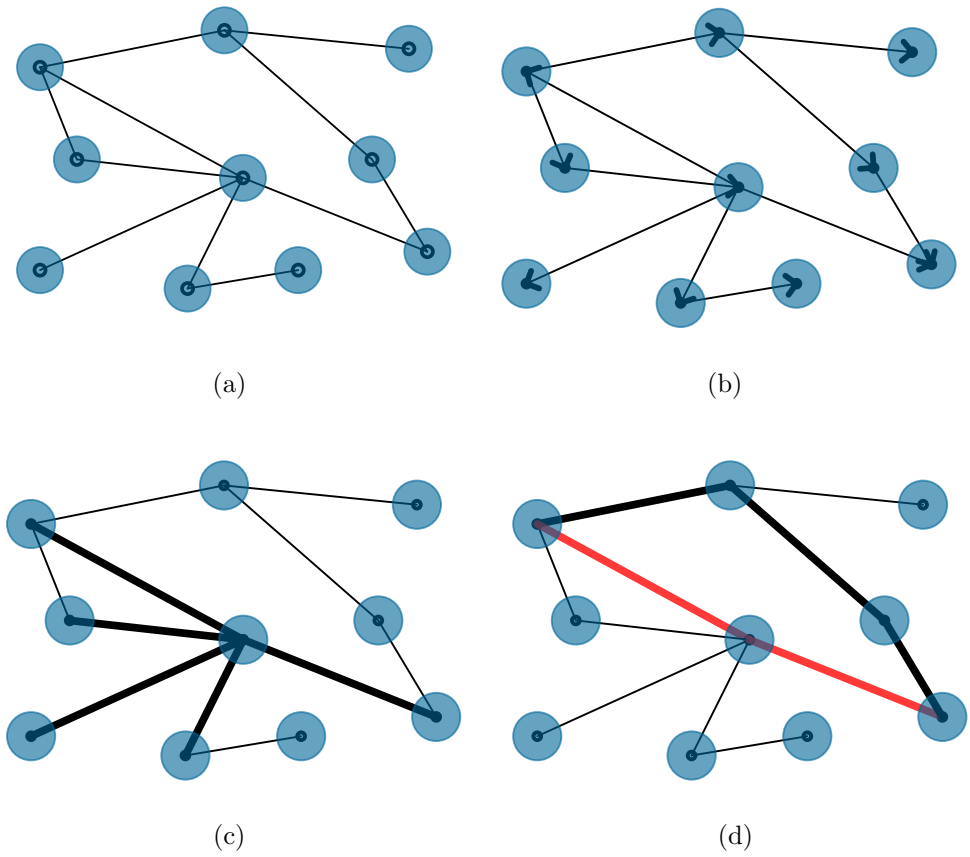


Figure 2.1: Network representations: Graphs are usually represented as a set of dots or circles, each corresponding to a node. Two of these circles being joined by a line if the corresponding nodes are connected or adjacent to each other. (a) Undirected network with nodes and links. (b) Directed links indicated by arrows in a directed network. (c) Degree of the node in the center is the number of links to other nodes it has (shown as thicker lines). (d) The path from source to destination node in black and the shortest path for the pair in red. These networks are drawn using Processing [2].

Figure 2.1 (c)). The spread of the node degrees in the graph can be characterized by the node degree distribution $P(k)$ which is the probability that a node chosen uniformly at random has k connections. Numerically the degree distribution can be exhibited by making a histogram of the degrees' frequency. For a network with N nodes, the degree distribution can be approximated by:

$$P(k) = \frac{n_k}{N}, \tag{2.1}$$

where n_k is the number of nodes with degree k. The degree of a specific node is a local property. However, the node degree distribution gives important information about the global properties of the network, which can be used to characterise different network topologies. Furthermore, the average degree of a network is

$$\overline{k} = \sum_{k=1}^{k_{max}} kP(k). \tag{2.2}$$

2.1.2 Path & Shortest Path

Another essential concept in graph theory is the path. It is defined as a sequence of nodes where there is a link between a node and the next node in the sequence [51]. The first node is called the start node and the last node is called the end node. The other nodes in the path are internal nodes. If a network is unweighted, each link would have weight one. The path length is the number of hops counted from the source node to the destination node.

The path with minimum length that goes from node s to node d is called the shortest-path. The length of this path is $\ell_{s,d}$. Notice that the shortest-path between a pair of nodes in a network is not necessarily unique. Single shortest-path length of a pair of nodes is a local quantity, where the average length of shortest-paths of the whole network $\bar{\ell}$ is a global network property. The $\bar{\ell}$ is also known as the "linear size" of a network which is the average separation of any pair of nodes [35; 36; 55]:

$$\bar{\ell} = \frac{1}{N(N-1)} \sum_{s \in V} \sum_{d \neq s \in V} \ell_{s,d}, \tag{2.3}$$

where N is the total number of nodes and V is the set of nodes in the network.

There are also other properties of path related to different networks:

- A cycle is a path such that the start and end node are the same.
- The path in a directed network is also a sequence of nodes with links being directed from each node to the following one. Often the term directed path is used in this case. The shortest path in directed network is the path with minimum hop count from source s to destination d.
- In a weighted network, every link has an associates weight. In this case, the weight of a path is the sum of the weights of the traversed links. The shortest path in a weighted network is the distance between source node s and destination node d with the smallest weight.

2.2 Betweenness Centrality

A large number of centrality indices have been introduced to measure the "importance" of nodes or links according to different criteria. One of these "importance" measures is the betweenness centrality [56]. Betweenness centrality is the fraction of shortest paths that pass through a node in the network. Given a source s, and destination d, the number of different shortest-paths between them is $g_{s,d}$. Consider the set of nodes W visited by shortest paths from s to d, the number of shortest-path between s and d that contains one of these nodes $w \in W$ is $g_{s,d}(w)$. Then the proportion of shortest-paths, from s to d, which pass through node w is

$$p_{s,d}(w) = \frac{g_{s,d}(w)}{g_{s,d}}. \tag{2.4}$$

Thus the definition of betweenness centrality of a node w is $C_B(w)$[56; 57], where

$$C_B(w) = \sum_{s \in V} \sum_{d \neq s \neq w \in V} p_{s,d}(w) = \sum_{s \in V} \sum_{d \neq s \neq w \in V} \frac{g_{s,d}(w)}{g_{s,d}}. \tag{2.5}$$

A node with high betweenness centrality in communication networks has a high status, because it stands between other nodes on the paths of communications and relays the information.

It is worthy to remark that there are different definitions of betweenness centrality: a factor 1/2 can be included in order not to count twice the paths, meanwhile the paths containing the end nodes (i.e. s and/or d) as initial or ending points can be accepted or discarded (the two cases differing just by a constant contribution). While we focus on analyzing the packet networks, the betweenness centrality used is slightly different from the Freeman's original definition [56]. In his definition, the concerning node w is not counted as either source or destination when summing values of $p_{s,d}(w)$. Other authors do include w as source or destination [56; 57; 58]. For IP-level network, it is important to include single hop routes where traffic from a source node to a directly connected neighbor always contributes to the load on the source. By considering this, the definition of betweenness centrality used in this thesis is:

$$C_B(w) = \sum_{s \in V} \sum_{d \neq s \in V, d \neq w} p_{s,d}(w). \tag{2.6}$$

There is also a property of the shortest path and shortest path length $\ell_{s,d}$ that [55]:

$$\ell_{s,d} = \sum_{w \in W} p_{s,d}(w) = \sum_{w \in W} \frac{g_{s,d}(w)}{g_{s,d}}, \tag{2.7}$$

where W is the set that contains the nodes visited by the shortest paths between s and d. Using this property and Equation (2.3), we could relate the betweenness centrality with the shortest path length in the network:

$$\sum_{w \in V} C_B(w) = \sum_{w \in V} \sum_{s \in V} \sum_{d \neq s \in V, d \neq w} p_{s,d}(w) = \sum_{s \in V} \sum_{d \neq s \in V} \ell_{s,d} = \bar{\ell} N(N-1). \tag{2.8}$$

2.2.1 Examples of Betweenness Centrality

A few examples of specific topologies are shown to further investigate the meaning underlying the notion of betweenness centrality. Figure 2.2(a) shows a typical network with nodes having different betweenness centrality. A star-shape network, Figure 2.2(b), has a unique central node n_f and $N-1$ leaf nodes directly connected to n_f. The node betweenness centrality of n_f is simple to compute

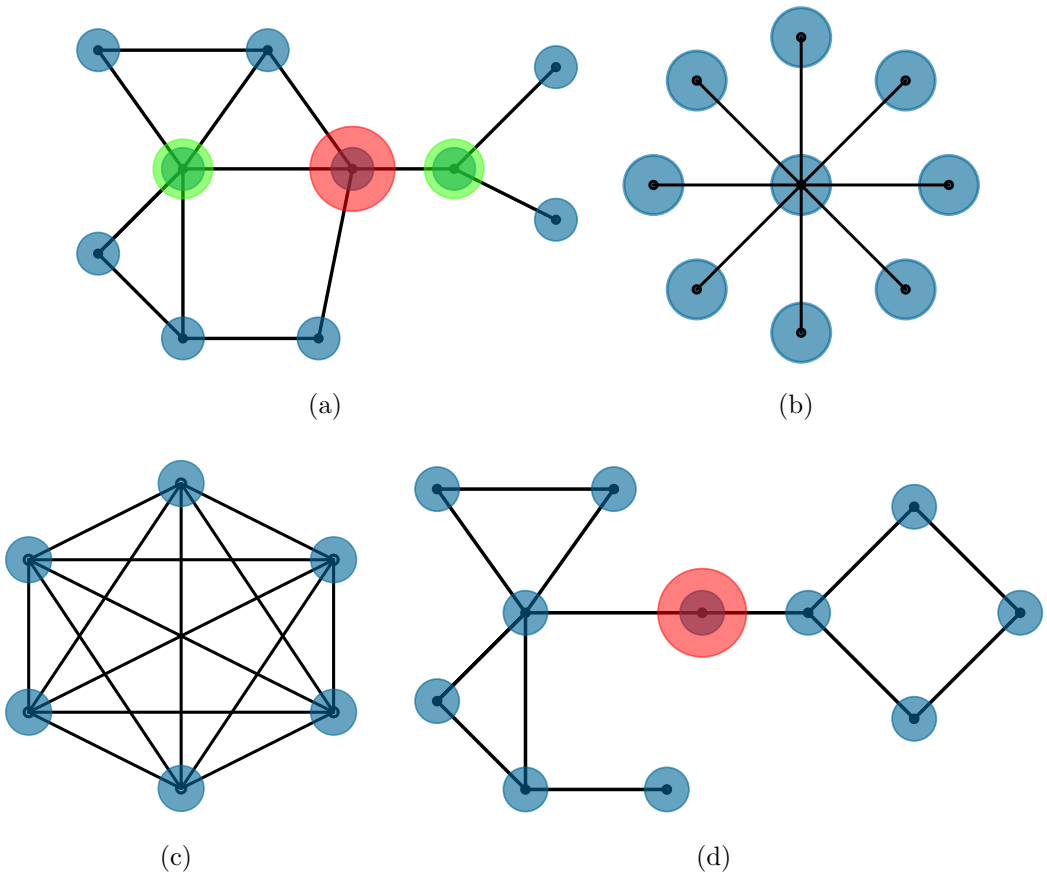


Figure 2.2: (a) Betweenness centrality of the red node is the highest in the network, the green nodes have medium betweenness, and blue nodes have the lowest betweenness. (b) The star-shape network with central node has the highest betweenness. (c) Fully connected network with nodes has the same betweenness. (d) Red node act as the bridge between two sub-networks.

because n_f belongs to all shortest paths between pairs of leaf nodes, therefore, from Equation (2.6)

$$C_B(n_f) = (N-1)(N-2) + (N-1) = (N-1)^2. \tag{2.9}$$

The opposite situation to a star-shape network is a complete graph in Figure 2.2(c). In this graph all nodes, according to the definition in Equation (2.6), have betweenness centrality of $N-1$. Another interesting case is that a node acts as a bridge joining two otherwise disconnected subnetworks shown in Figure 2.2(d). All paths between pairs of nodes belonging to different subnetworks have to pass through that particular bridge node. This bridge node turns out to have very high betweenness even it may have very low degree.

2.2.2 The Calculation of Betweenness Centrality

The computational cost of determining the betweenness centrality for all nodes in the network is very high, since one has to discover all existing shortest paths between every pairs of nodes. Optimized algorithms introduced by Brandes and Newman using different definition can calculate betweenness centrality for all nodes in time $O(NL)$ for a network with N nodes and L links [57; 58]. For sparse networks [1] the algorithm performs in $O(N^2)$ steps. But this is still a high demanding calculation when the size of the network is very large (i.e. $O(1000^2)$). Or when the computation has to be repeated many times such as the topology generation methods introduced in later chapters. Besides, a very different definition and calculation of betweenness centrality has been introduced by Newman by counting how often a node is traversed by a random walk between two other nodes in the network [59]. In this thesis, the Brandes' algorithm is chosen because it is easy to implement and relatively fast to calculate betweenness centrality.

[1] The kind of network in which the number of links is much less than the possible number of links.

2.3 Complex Network Topology

Graph theory has its origins in the eighteenth century by Leonhard Euler, whose early work concentrated on small graphs with a high degree of regularity, for example the famous Königsberg Bridges problem. In the twentieth century graph theory has become more statistical and algorithmic. The topology of a network can be seen as the pattern of links connecting pairs of nodes in the network. Networks with a complex topology and unknown organizing principles were often believed to be random. Thus Solomonoff and Rapoport [60] and independently Erdős and Rényi [61; 62] introduced a simple model of random graph where the node degree distribution is based on random process which is known as "ER model". Later on, it was found out that many large networks do not exactly follow the ER random network model. Their structures are more scale free, that is, the degree distribution follows a power law distribution rather than Poisson. Barabási and Albert introduced the concept of scale-free network which could be used to model the large scale complex networks. Instead of constructing a graph with exact topological features, the model place emphasis on capturing the network dynamics in its growth by using preferential attachment [20].

2.3.1 Regular Network

If in network G all nodes have the same degree, then G is considered as regular [63]. For example, a ring network is always a regular graph with every node having two edges. Further, a lattice – a grid like network and a two branch tree can also be seen as a regular graph (see Figure 2.3 for more examples of regular network). Because of the symmetrical property of regular networks, the degree for all nodes are the same $k_w = K$. So in a regular network the degree distribution is a constant value. Further, by symmetry the betweenness centrality is also constant for all the nodes $C_B(w) = c$ [55].

2.3.2 Random Network

The simplest and most studied network with undirected edges is the ER (Erdős and Rényi) random network. This random network model has been used to

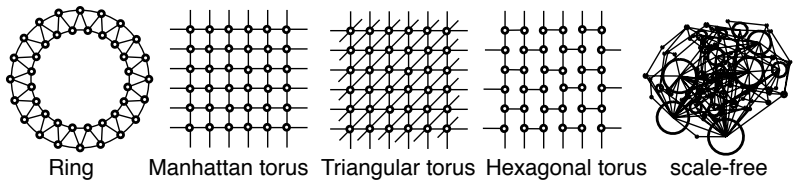


Figure 2.3: Four regular networks and a scale-free network. Nodes in the ring network and manhattan toroidal network, in which the nodes on one edge of the lattice connect to nodes on the opposite edge, have degree four. Nodes in triangular toroidal network and heagonal toroidal network have degree six and three.

study communication networks, in that communication networks tend to have very complex topology and interactions. It is believed that they appear random at very large scale. In this model: the total number of nodes, N, is fixed; and the probability that two arbitrary nodes are connected is p [60; 61; 62]. At small values of p, the system consists of small clusters while at large N and large enough p, the giant connected component (big cluster) appears in the network. On average, the network contains $pN(N-1)/2$ links. Because each edge is present or absent with equal probability $P(k)$. This makes the majority of nodes in random network have approximately similar degree, which is close to the average degree $\overline{k}$. Hence the degree distribution of a random network is a binomial, or Poisson distribution when the number of nodes in the network $N \to \infty$. The degree distribution for random network is,

$$P(k) = \binom{N-1}{k} p^k (1-p)^{N-1-k}. \tag{2.10}$$

so the average degree is $\bar{k} = p(N-1)$. For very large N, the distribution takes the Poisson form,

$$P(k) = \bar{k}^k e^{-\bar{k}}/k!. \tag{2.11}$$

Therefore, the distribution rapidly decreases at large degrees. The shortest-path length of a random network scales with the size of the network is [20]:

$$\bar{\ell}_{rand} \approx \frac{\ln(N)}{\ln(\bar{k})}. \tag{2.12}$$

Example of degree distribution of random network is shown in Figure 2.4(a).

2.3.3 Scale Free Network

2.3.3.1 Compare Random Network with Real Networks

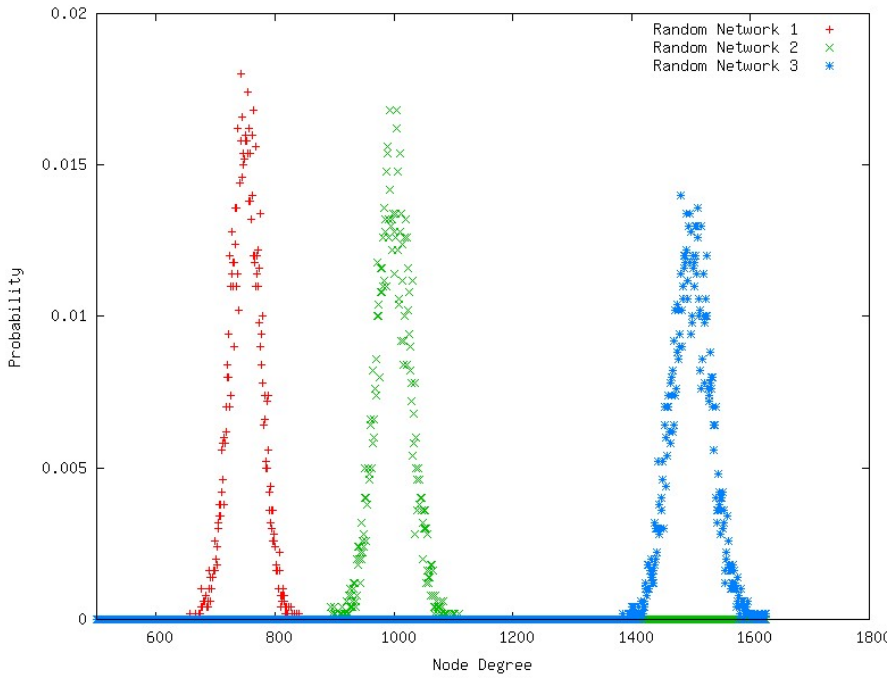
Many natural and man-made communication networks can not be precisely described by a regular network or a random network model. Let's first compare the properties of random network model and real-world networks. A random network is generated as follows:

- Start with a fixed number N of nodes and N is not modified during the network generation.
- Randomly connected or rewired pair of nodes in the network with certain probability p. This is independent of the existing nodes' degree.

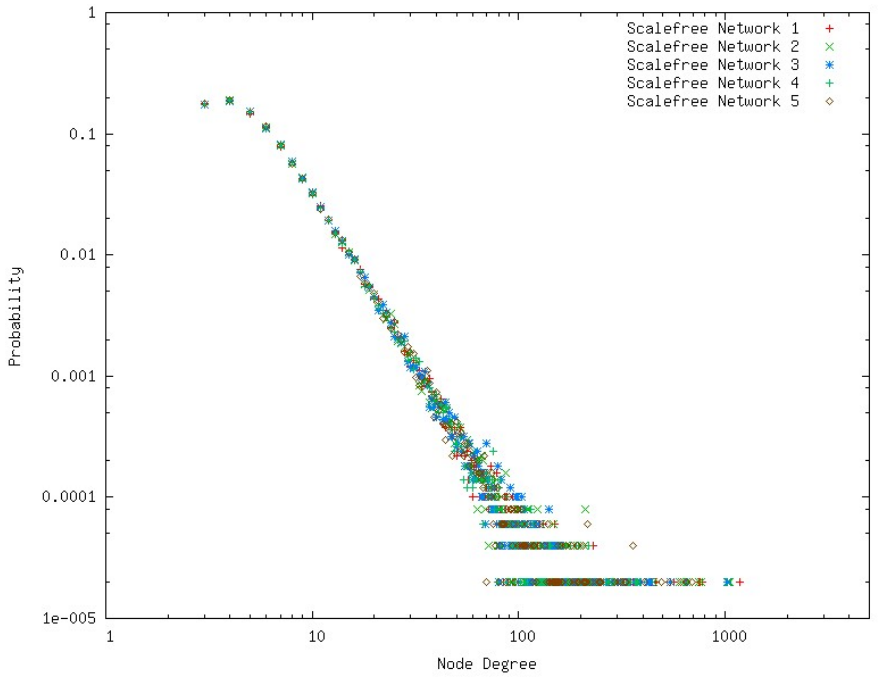
The real-world networks are open systems:

- Starting from a small nucleus of nodes, the number of nodes increases throughout the lifetime of the network by the subsequent addition of new nodes
- Exhibit preferential attachment, such that the likelihood of connecting to a node depends on the node's degree.

A common feature of real world networks is that the network continuously expands by the addition of new nodes that are connected to the existing nodes already in the system. And there are also examples of preferential connectivity, such as a newly created web page will more likely include links to well known, popular documents with already high connectivity, or new edge routers would more likely to connect their uplinks to an existing aggregation router. These



(a)



(b)

Figure 2.4: (a) Degree distribution of three ER-Model Random Networks of 5000 nodes. Different connection probability are used for different networks: $p_1 = 0.15$, $p_2 = 0.2$ and $p_3 = 0.3$. So that the mean degree values vary. (b) Degree distribution of five BA-Model Scale-free Networks of 10000 nodes with $P(k) = k^{-\gamma}$ and $\gamma = 2.3$ [3]. Note that both X and Y axes are in logscale.

examples indicate that the probability with which a new node connects to the existing nodes is not uniform, but there is a higher probability to be linked to a node that already has a large number of connections.

2.3.3.2 Model of Scale-free Network

It is shown that many real world networks have degree distributions that deviates from the Poisson distribution. In particular the Internet, World Wide Web, cellular networks, and ecological networks, their degree distributions have a power-law tail [18; 20; 64]:

$$P(k) \sim Ck^{-\gamma}, \tag{2.13}$$

where $\gamma > 1$ is the scale exponent and C is a constant. Even for those networks for which $P(k)$ has an exponential tail, the degree distribution significantly deviates from a Poisson distribution. The example of scale-free network degree distribution is shown in Figure 2.4(b).

The model of the power-law degree distribution observed in networks was addressed by Barabási and Albert [20]. They argued that the scale-free nature of real networks is rooted in two generic mechanisms growth and preferential attachment. Therefore these two ingredients, inspired the introduction of the BA-model scale-free network.

1. Growth: Starting with a small number m_0 of nodes, at every time step, one adds a new node with $m < m_0$ edges that link the new node to m different nodes already present in the system.

2. Preferential attachment: When choosing the nodes to which the new node connects, one assumes that the probability p that a new node will be connected to node i depends on the degree k_i. Such that, $P(k_i) = k_i / \sum_j k_j$.

After t time steps this procedure results in a network with $N = t + m_0$ nodes and mt edges. Numerical simulations indicated that this network evolves into a scale invariant state with the probability that a node has k edges following a power law with an exponent $\gamma_{BA} = 3$ for Barabasi and Albert scale-free network [20]. The scaling exponent is independent of m, the only parameter in the model. In practical terms a power law distribution means that the majority of the nodes

will have very few neighbors, but there is a very small set of nodes with very large number of neighbors. Networks with this property are known as scale-free because power-law are free of a characteristic scale, that is, there is no characteristic node degree. The average shortest path length of a scale-free network increases approximately logarithmically with N [20]

$$\bar{\ell} = A \times ln(N - B) + C, \tag{2.14}$$

where A,B and C are constants. Some other researches' analytical results indicate that there might be a double logarithmic correction to the logarithmic N dependence [65; 66], i.e.,

$$\bar{\ell} \propto \ln(N)/\ln\ln(N). \tag{2.15}$$

2.3.3.3 Compare Scale-free network with Random Network

An example of the scale-free network and random network is shown in Figure 2.5. The scale-free network shown in Figure 2.5 (b) is a sample of artists network with 133 nodes of "myspace.com" [67; 68]. Its degree distribution in Figure 2.5 (d) follows the power-law distribution. Figure 2.5 (a) shows the random network of 133 nodes generated using the ER model. Its degree distribution is also illustrated below in Figure 2.5 (c), which reveals a different distribution from scale-free network. The random network is homogeneous, in which most nodes have approximately the same number of links. On the other hand, the scale-free network is heterogeneous where the majority of the nodes have one or two links but a few nodes have a large number of links. In Figure 2.5 (a) and (b), the **red** nodes are highly connected with a high number of links; the **green** nodes are nodes with medium number of connections.

2.4 The Internet

A good example of manmade complex network is the Internet. From the Oxford Advanced Learner's dictionary [69], the Internet is an international computer network connecting other networks and computers from companies, universities, etc. In fact, the advance of technology enables users to connect to the Internet using

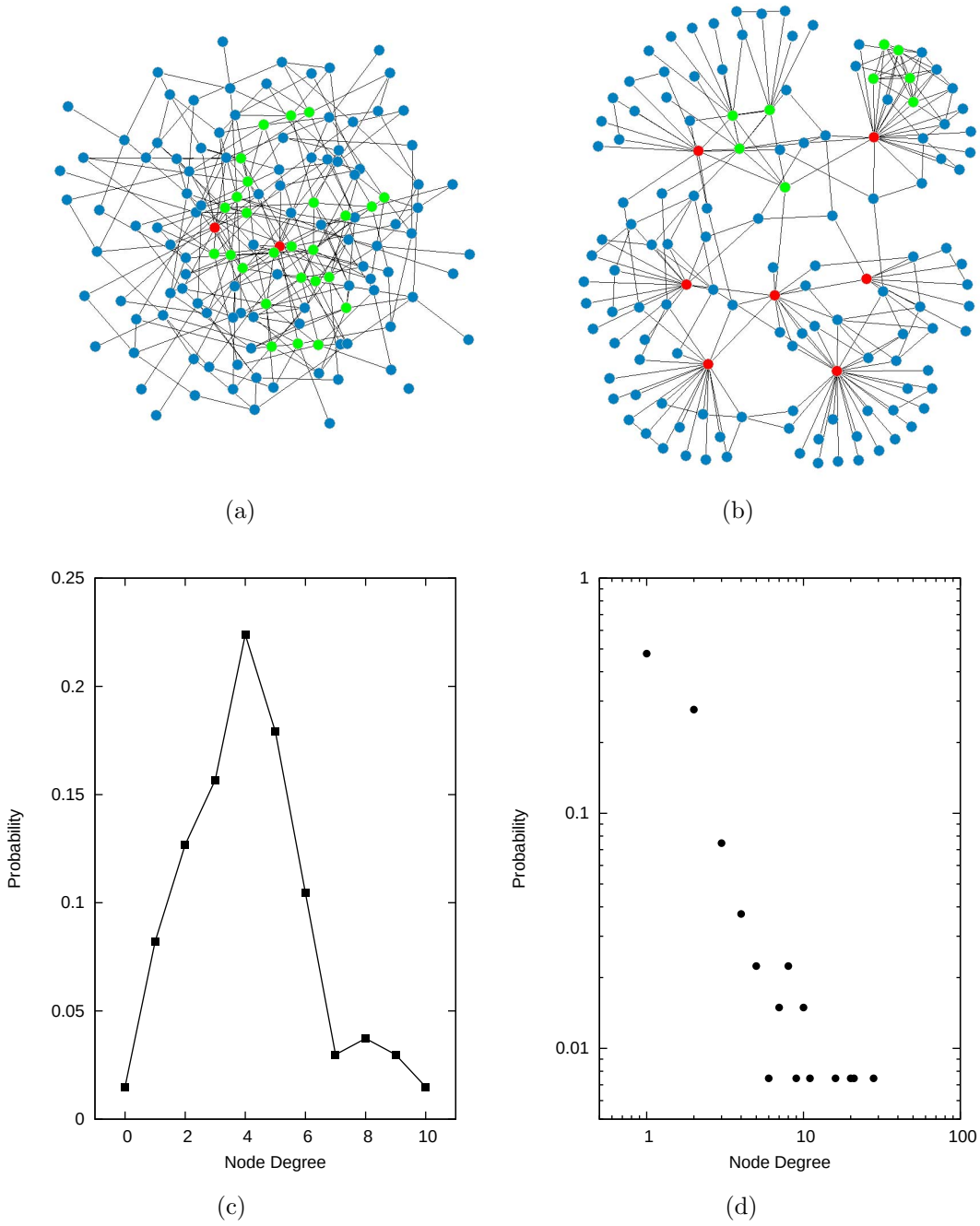


Figure 2.5: (a) and (c) are topology and degree distribution of the 133-node ER model random network (generated using Pajek [4]). (b) and (d) are topology and degree distribution of 133-node scalefree network.

not only conventional computers but also many other devices. It has become a worldwide interconnected networks of hosts (PCs, laptops, netbooks, mobile phones, sensors and many other terminals), servers (a computer that provides certain network services), switches (connect network segments), routers (arrange traffic across networks) and other network hardware. The Internet has also become a critical infrastructure system. Because human beings rely heavily on it for communication, banking, entertainment, and other infrastructure systems' control. At the same time, the Internet also depends on other infrastructure systems, such as the electrical power grid and water supply system for energy supply and cooling. So it is crucial to understand the Internet's topology and its behavior, in order to avoid any severe malfunction which may have a huge impact on our lives.

The Internet is designed to have a layer architecture, which is the most important and successful feature. There are two important reference models: Open Systems Interconnection (OSI) Reference Model by International Standard Organization (ISO) [70] and TCP/IP Reference Model created in the 1970s by DARPA of the United States Department of Defense, but now maintained by the Internet Engineering Task Force (IETF) [71]. In the top layer (application layer) of the Internet, one of the well studied topology is the World Wide Web (WWW), where the nodes are web pages and the links are directed hyperlinks. One can also find Email network, peer-to-peer network, and many other networks in this layer.

2.4.1 Different Levels of Internet Topology

In the Network layer (Internet layer), researchers are focusing on three different levels: AS-level(Autonomous System), IP-level and router-level. At AS-level, each administration domain is composed of a set of routers and hosts and represented by a single node in the AS network. There are links between the ASs if they are connected with each other. The AS-level topology is of interest to those analyzing inter-domain networking, routing and QoS (Quality of Service). Examples of AS-level Internet IPv4 and IPv6 topology measured by CAIDA are shown in Figure 2.6 [5]. Although having the aggregation, the AS-level network is still a very large network in scale. The IP-level network take the IP addresses

as network nodes and the connections of these IPs are obtained from routing table entries. And the Router-level network considers routers and end-systems IP interfaces and their connections. The IP-level network and router-level network are similar to each other, but they are not the same. In the router-level network, a node often represents a single router and the links are the connections between them. Note that this is an assumption for studying the IP-level network. Studies at this level mainly focus on the impact of router and link failures, and the optimization of network planning to avoid such failures. The IP-level and router-level network of the Internet is much bigger and more complex than the AS-level network (see Figure 2.7 [72]).

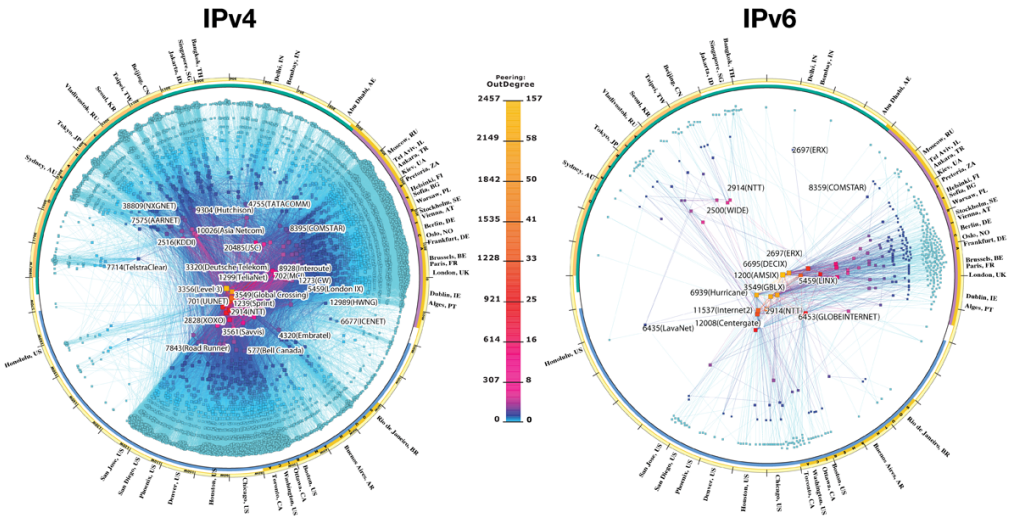


Figure 2.6: IPv4 and IPv6 AS-level topology produced by CAIDA 2009 [5].

The main differences in the AS and router level of the Internet from a technical point of view are the constraints associated with routing protocols. Routing protocol is a set of messages, rules and algorithms used by routers for the overall purpose of learning and calculating routes [73]. This process includes the exchange and analysis of routing information. Each router chooses the best route to each subnet (path selection) and finally places those best routes in its IP routing table. Routers are subject to administrative routing policies and make

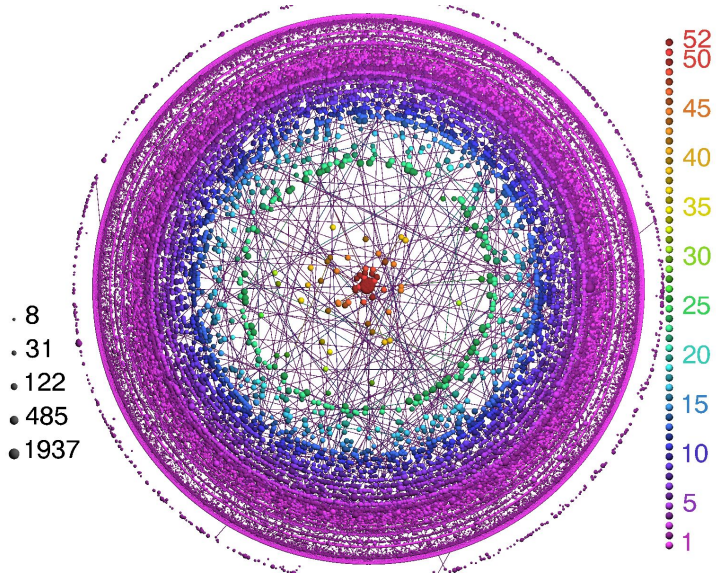


Figure 2.7: IP address Graph by LaNet-vi [6].

decisions according to the routing protocols used and information in the routing table. There are two different classes of routing protocols: the intra-domain routing protocols and the inter-domain routing protocols. The intra-domain protocols are used to pass information within the same AS, while the inter-domain protocols serve for exchanging topological, path, and condition information between peer routers in different AS. For example, the intra-domain protocols are Routing Information Protocol (RIP) [74], Enhanced Interior Gateway Routing Protocol (EIGRP) [75; 76], Open Shortest Path First (OSPF) [77; 78], Intermediate system to intermediate system IS-IS [75]; and the inter-domain protocols are Exterior Gateway Protocol (EGP) [75], Border Gateway Protocol (BGP) [79].

Faloutsos *et.al* [18] are the first to study what does Internet look like. Many other researches follow the trend of measuring and mapping the Internet topology. In fact, the Internet is hard to map: with no central control organization, there are economic incentives for ISPs to obscure network structure, making it almost impossible to directly inspect. So one depends on measurement to find out what the Internet looks like with a mismatch between what one want to measure and can measure. At the AS-level, it is very difficult to obtain a consistent map of the actual AS topology due to the dynamic nature of the Internet and the

reluctance of AS Operators to reveal their ASs connections. However, one could still map the AS topology by collecting BGP tables data from different routers. The IP-level and router-level networks are also changing all the time for multiple reasons, such as network equipment failure, link failure and network maintenance. Most of the ISPs are not willing to reveal their network detail either, which is considered as commercial secret and high security information. The most widely used method for IP and router level topology discovery is the "traceroute" [80], although it is not very efficient and accurate, because it might report a router with several interfaces of different IP addresses more than once, miss some routes, and misinterpret the Layer2 structure and MPLS (Multiprotocol Label Switching) network as a router. This is known as aliasing problem which can be partly solved by aggregating a router's IP interfaces under a single identifier using "alias resolution" [81]. It is also important to point out that for router-level topologies, there are constraints on the current technology of how many links one router can have [23]. In another word, there is an upper limit for the node degree in the router-level network.

2.4.2 Shortest Path, Betweenness Centrality and Routing

In communication networks, some users would always want to send information as fast as possible through the network provided with certain reliability. While network service providers try to deliver the information as quickly as possible through the shortest-path in the mean time, considering QoS, cost, etc. In IP networks, the routing protocols are designed to choose the best available path to send packets to their destinations. One of the functions of any routing protocols is to calculate the shortest path for the packets in the router based on the information in the router's routing table. The first important algorithms for calculating shortest paths is the Bellman-Ford single-source shortest-paths algorithm [82; 83]. Some of the early Internet routing protocols, such as RIP [74], use a distributed form of this algorithm. RIP uses hop count as the metric for routes, in which all links have the same weight (unweighted) for shortest path calculation. Another well known algorithm is the Dijkstra's single-source shortest-paths algorithm. It finds all the shortest paths from the source node to every other

node in the network. It is used by the OSPF routing protocol [77]. So that each router running OSPF would calculate its own shortest paths to all routers in the same network. The implementation of these two shortest-path algorithm could be found in Boost Graph Library [84] and programming algorithm books [85; 86].

But the journey time for a packet to go through two shortest-paths with the same length can be very different, due to different traffic patterns and rate of usage of the routes (paths). Therefore, in the network, there are nodes that are highly utilized to transfer packets, while the others spend most of the time idle or transferring small amount of packets. The betweenness centrality can be used to measure the importance of nodes in different traffic situations and find out possible loads of the nodes. The nodes with high betweenness centrality are relatively more important in the network as they are visited by more routes (shortest paths). The queues in these nodes can be built up more easily and quickly. It is expected that the removal of these nodes will worsen the network performance to a certain level for different kinds of network topologies. In the worst situation failure of a node with high betweenness centrality could cause severe congestion. In some cases the congestion could spread out through the network and trigger a cascade breakdown of nodes in the network.

2.5 Cascade Failures and Catastrophes

Examples of very large-scale complex network can be easily found in everyday life. In the human body, the brain is a network of nerve cells connected by axons (nerve fibers), and the cells themselves are networks of molecules connected by biochemical reactions. In telecommunication, the Internet is a network of hosts, switches, and routers linked by various communication lines (telephone line, twist-pair line, optical fibre) with different bandwidths. Despite of the importance of these networks, scientists have had limited understanding of their structure, properties and interactions. Humanity becomes increasingly dependent on ubiquitous communication networks, electricity provided by power grids, and other critical infrastructure networks. A much-voiced concern arises: exactly how reliable are these networks, especially the Internet that everybody now rely on so much? Why

does diffusion of information occur so rapidly in certain social and communications systems, leading to epidemics of diseases and computer viruses? Why do the Power Grids sometimes breakdown so quickly and the failure spread through a very large area? Is there a pattern for this kind of breakdown? All these questions lead to the research and analysis of the robustness of networks' structures and their resilience to failures and attacks.

Some researchers argue that scale-free complex systems can be amazingly resilient against accidental failures [40; 87], due to the existence of 'hub' like nodes: small number of nodes with very high connectivity. Further, random failures or attacks are more likely to be on one of the low-degree nodes, that the removal of one of these nodes has minimal impact on the network's overall connectivity or functionality. But there are still some unexpected failures on critical networks. For example in 2007, the online telephone service Skype was not working for two days from 16th August, leaving its 220 million users disconnected, some of them were small businesses that had given up their land lines, without a way to call colleagues, customers and friends [88]. A flaw in a crucial piece of software that connects users to central servers appears to have been the source of the problem, where overloaded servers were brought down one by one. The denial-of-service (DoS) attack on the Internet root name servers in 2002 and 2007 could easily bring down the Internet rather than several web sites. In 2003, the Northeast Electricity Blackout was a massive widespread power outage that occurred throughout parts of the Northeastern and Midwestern United States and Ontario, Canada. The blackout affected an estimated 10 million people in the Canadian province of Ontario and 45 million people in eight U.S. states. But some of the most serious, even potentially devastating, problems with networks arise from sources with no intentional attack from Hackers. It is simply, complex system breaks down in a complex way. As a result, the focus of this thesis is trying to answer some of these very complex questions. The starting point is analyzing the communication networks which are the man-made networks designed typically for distribution of information. But the theory could also be extended to other complex systems.

2.5.1 Vulnerability of Complex Network

Previous researches show that network components (nodes, links) can fail due to several events. For example, an accident happened in the network, internal failure of a node or link, or an intentional attack to network components. The attack is most destructive in a complex network according to Albert *et.al* [87] who claim that scale-free networks possess the "robust-yet-fragile" property. They are robust against random failures of nodes but fragile to intentional attacks on the highly connected nodes. However, the term fragility here means that a scale-free network can become disconnected under attacks on a small but still appreciable set of nodes that include a substantial fraction of links in the network [89; 90]. An attack on a single or a very few random selected nodes will, in general, not bring down the network. This interesting result was actually obtained based purely on the scale-free structure of the network. In other words, it is considering only the topological structure, but the network dynamics has not been taken into account [43].

It is assumed that after an attack, a node will lose the ability to function. This type of node failure would not be necessarily the same for different types of real-world networks. To maximize the destructive effect, one can target the important nodes or links. In this case, the two mainly used metrics to measure the importance of a node (or edge) in the network are node degree and betweenness centrality. That is the degree characterises the number of neighboring nodes and betweenness measures the number of shortest paths pass through the node.

2.5.2 Cascading Failures

In some circumstances, in a complex network, small-scale local activities of individual nodes may cause the effect to propagate to other parts of the network, and sometimes a global (system wide) event occurs. These avalanche-like failures are known as cascading failures. Researchers are focusing on the dynamic response of the network to the initiating disturbance and examining the subsequent cascading failures that are induced when nodes' operating thresholds are exceeded [37; 39]. Imagine a network that transports some physical quantities or load. Nodes with large numbers of links receive a relatively heavier load. Each node, however, has

a finite capacity to process or transport load. In order for a node to function properly, its load must be less than the capacity it could handle at all times; otherwise it would not meet the demand and causes failures. In these cases, its load will be redirected to other nodes, causing a redistribution of load in the network. If the failing node deals with a small amount of load, there will be little effect on the network because the amount of load that needs to be redistributed is small. However, if the failing node carries a large amount of load, the consequence could be serious because this amount of load needs to be redistributed and it is possible that for some nodes, the new load exceeds their capacities. These nodes will then fail, causing further redistributions of load, and so on. As a consequence, a large fraction of the network can break down.

There are a few recent studies on cascading failures in complex networks [42; 43; 44; 91; 92; 93]. In Motter and Lai's work [42], a simple mechanism has been proposed to incorporate the dynamics of the network in both random and scale-free networks. The model generates results that are completely consistent with the above intuition on cascading failures. Zhao *et.al* [43; 44] further improve the work on modeling the cascading breakdown. They understand the cascading failures in complex networks in terms of a phase transition process. In free flow stage the network functions properly, while in the congestion stage the demands exceed the network's capacity in some parts and cause a chain reaction sometimes leading to a global cascade. Holme *et.al* [40; 41; 94] analyse the vulnerability of complex networks in node and edge overload breakdown by evolving the networks. It simply generates the complex network while examining the networks' functionality at each generation step. Measurement of the network operation condition can be done by calculating the average shortest path length (or inverse of average shortest path length) and the size of the largest connected subgraph after an attack.

2.5.3 Cascade to Catastrophe

Cascading failure can occur in many real world physical systems and sometimes leads to a catastrophe. In a power transmission grid, for instance, each node (a generator) deals with a load of power. Removal of nodes, in general, can

cause redistribution of loads over all the network, which can trigger a cascade of overloading failures. As an example, the recent massive power blackout was caused by a series of seemingly unrelated events on 5th Nov 2006 [95]. A German electric company had a high-voltage transmission line shut down over a river to let a ship to pass. This have caused a chain-reaction power outages that left about 10 million people in the dark across Europe. A similar incident occurred in the United States in 2003, when tree limbs touching a power line in Ohio triggered a blackout that cascaded across the eastern part of the country and into Canada, affecting 50 million people [95].

The example of the power grid shows that end-user nodes pull electric load from the power sources. But the situation is different for the Internet. On one hand, server nodes would push data to the end-user nodes according to the users' requests through the network. On the other hand, end-user might share data using peer-to-peer technology where they would pull and push data from the network. Despite the differences, similar failure examples can be found in the Internet, where the load represents data packets and a node (router) is requested to transmit and overloading corresponds to congestion. The redistribution of data packets from a congested router to another may spread the congestion to a large fraction of the network. From the "Network World" news in April 1998, the AT&T frame relay network suffered a catastrophic failure [96]. All of the company's 145 switches were brought down, leaving thousands of frame relay customers off-line for more than 24 hours. For the Internet, there is also a kind of cascade failure found in the early years called congestion collapse [97]. It is an overload condition that the network has reached the state that traffic demand is higher than throughput available, and there are high levels of packet delay and loss. Nevertheless, it was solved by the introduction of improved congestion control mechanisms [98; 99].

However with the increasing dependence on the Internet, we could face severe congestion. Firstly, there is a sharp increase in UDP (User Datagram Protocol) traffic on the Internet. UDP is defined to make available a datagram mode of packet-switched computer communication networks using the Internet Protocol (IP) as the underlying protocol. It provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism.

Further it is transaction oriented, therefore delivery and duplicate protection are not guaranteed. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or lost. Time-sensitive applications often use UDP rather than TCP (Transmission Control Protocol) [100] because dropping packets is preferable to using delayed packets. Network protocols that use UDP are Video streaming protocols (i.e., Real-time Transport Protocol, RTP) [101], Voice over IP (VoIP) (i.e. Session Initiation Protocol, SIP) [102], the Domain Name System (DNS), and Trivial File Transfer Protocol (TFTP). Moreover, there is an increasing amount of consumers that prefer to watch TV programs (i.e., BBC iPlayer), movies through the Internet and making International phone calls using VoIP applications (Skype). It has injected a lot more traffic into the Internet than expected and might cause congestion due to the increasing demand. But at the same time, Internet Service Providers (ISP) are keen to cut cost, save energy, increase the efficiency by pushing the existing infrastructure to its operation limit. In this case, the rapid growth in user number and user demands for network resources can not be met by the slow network system upgrade provided by ISPs. All these would cause congestion, and affect the whole user experience with very poor QoS. One example is the United States AT&T Mobile Network that provides 3G (Third Generation mobile service) and WiFi (WLAN) for all the Apple iPhone users with unlimited Internet usage. Users are getting full use of the advantage of IPhone and its applications for video streaming and VoIP calls, although AT&T terms of service prohibited certain kind of streaming. While AT&T is not prepared for the excessive amount of traffic, users find it very difficult to make phone calls and slow to surf the Internet. Other threats for the Internet are from the Botnets, which is a collection of software robots that run autonomously and automatically. It is a network of computers that have been infected with malicious software and controlled by hackers. The Botnets are very popular and used to initiate attacks in the Internet [103]. For example:

- Denial-of-service attacks (DoS) [104], where multiple systems autonomously access a single Internet system or service in a way that appears legitimate, but much more frequently than normal use and cause the system to become unavailable for normal users. The most devastating attack is DoS on Central

DNS servers, which could keep the DNS servers out of normal service and lots of computers could not explain the domain names.

- E-mail spam, where e-mail messages disguised as messages from people, but are either advertising, annoying, or malicious in nature.
- Adware exists to advertise some commercial entity actively and without the user's permission or awareness.
- Spyware is the software which sends information to its creators about a user's activities.

Above all, cascade or catastrophic failures do exist in real-world complex networks. These failures might be triggered by different events in the network system. But most of them propagate through networks in the form of overloading and load redistribution. Under overload conditions, the network components might fail to function properly or stop accepting extra load. Then the load will be redistributed to other components.

2.6 Summary

Complex networks are modeled as graphs with the vertices representing nodes and edges for links between the nodes. As a starting point, several important structural properties of networks (graphs) are reviewed with their definitions, notations and how they could be used to observe networks. They are node degree, degree distribution, path, shortest path, and most importantly the betweenness centrality. The network degree distribution can help us to identify which type the observing network is, either regular, random, or scale-free complex network. These three kinds of models are all used to study real world networks, where regular networks have the same degree for every node, random networks degree distributions follow Poisson distribution, and scale-free networks degree distributions follow the power law distribution.

Shortest path algorithms are used by most of the Internet routing protocols to calculate paths for routers based on the information obtained from routing table. Using betweenness centrality, which could give out the number of shortest paths

passing through a node, one could estimate the usage of a specific node in the computer network. The higher the betweenness centrality, the busier the node would be handling packets. When the load of a node in the network exceed its capacity of handling packets, this node might reject incoming packets and notice its neighbors using routing protocols it is too busy to accept packets. Then routing protocols would redirect packets to other router and avoid the congested node. In some cases, if the congested node has a high betweenness centrality that many packets would go through it, the redirected traffic may cause other nodes to fail. Because the volume of redirected traffic is too big for others to cope with. In the worst case, a cascade failure of network node might occur when the redirection of traffic cause load on a sequence of nodes exceed their limits.

In the following chapter, we are going to investigate congestion in the network where the focus point is the interaction between topology and traffic.

Chapter 3

Congestion – Interaction Between Topology and Traffic

3.1 Introduction

In an ideal world, information in the communication network should flow freely allowing people to communicate anytime anywhere. But in reality, network congestion happens from time to time due to different reasons. To ensure free flow of information on a network is of great interest to communication network engineers. And to achieve free flow of network traffic, they consider using control methods, such as routing protocols or congestion control mechanisms.

On the other hand, researchers interested in complex network theory mainly focus on the topological (static) properties of the concerning network, i.e. degree distribution, average shortest path length. From the late 1990's, researchers began to consider both topological and dynamical properties of communication networks where the binding elements between them are the routing and congestion control mechanisms [55].

The interest of this thesis is in the onset of congestion in a network and its propagation pattern throughout the network. The way discussed to study congestion is based on the information transmission and exchange of the Internet. In this chapter, we first introduce the concept of network congestion and its metrics. It is followed by the review of previous work in predicting congestion in

regular networks and general networks. Finally, we examine the invariants that relate to the congestion in networks.

3.2 Network Congestion

To the author's knowledge, there are many definitions of congestion. As from [105], the network is congested if the service quality noticed by the user decreases because of the increase of network load. From network engineers' point of view, network congestion occurs when demand of network resource exceed the network capacity.

Congestion can occur in the network due to different factors [53; 105]. For example, if excessive packets arrive at the same router, a queue will build up in it. And it will only drop packets if it is full. When packet loss occurs, the packet delivery is delayed for the whole network and the packets throughput will decrease. The underlying assumption about the queue design is that a subsequent traffic reduction would eventually allow the queue to empty, thus making it a buffer device to compensate for short traffic bursts. But storing packets in a queue can add significant delay to the packet transmission.

Furthermore, traffic in the network could follow the Poisson distribution or could be statistically self-similar [27]. The self-similar traffic is characterised by its large fluctuation which is also know as the "burstiness" property. Bursty traffic could easily fill up router queues in the network. Even deploying routers with infinite queue the congestion might get worse in some cases. By the time packets get to the front of the queue, they have already timed out and duplicates have been sent from the source. All these packets will be dutifully forwarded to the next hop router, further increasing the load on the network all the way from the source to destination. If a router has a slow processing ability, it can also cause congestion. If the router is slow at performing its tasks (routing calculation, routing table update), queues can build up, even though there is excess line bandwidth.

3.2.1 Congestion Metrics

There are several metrics that can be used to capture the congestion phase of a network. Classic metrics are End-to-End delay of packets, network throughput and packet loss rate [105; 106; 107]. When the network is congested, delay is caused by the overload of data packets. In this case, the data packets arriving at congested points (i.e. routers) will be queued. The waiting in the queue increases the delay and the number of packets that can reach their destination decreases. In the worst case, all packets arriving at the congestion nodes will be dropped when the network reaches its capacity. Other parameters to measure congestion are the average queue lengths at each node of the network, the total number of packets in the network and the standard deviation of packet delay.

Our measure of congestion focuses on the End-to-End delay of packets. The End-to-End delay of a packet can be attributed to many factors. It is important to be aware of such factors to better understand the reason of delay and congestion. There are several kinds of delays in the packet networks:

- Processing delay - the time routers take to process packets. A router takes a packet from an input interface, examines it and puts it into the queue of the output interface. It depends on the router's CPU speed and utilization, its architecture and configuration.

- Queuing delay - the time a packet resides in the queues of a router. It depends on the number and size of packets already in the queue, the speed of the interface and the type of queueing mechanism applied.

- Transmission (Serialization) delay - the time it takes to push the packet's bits onto the link.

- Propagation delay - the time it requires to transmit a packet from the source to the destination, which depends on the type of media used.

A typical client and server network setup is shown in Figure 3.1. The process delay would come from the process of packets on the user host, the server, two switches and three routers. But queueing delay normally happens on the three

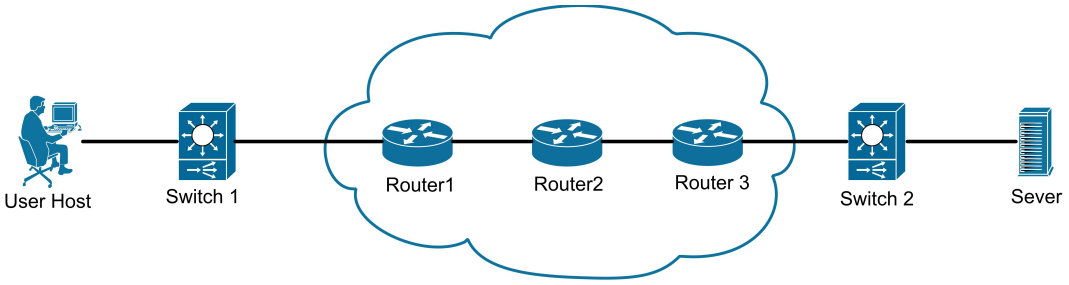


Figure 3.1: Network Delay

routers. Furthermore, the transmission and propagation delay depends on the connection type between all the devices in the network.

So the the delay time of a packet $\tau_{s,d}$ elapses from the creation at its source s to its arrival at the destination d. In a network, the average delay of all the packets up to time T is:

$$\bar{\tau}_T = \frac{1}{N(N-1)} \sum_{s,d;\tau_{s,d} \leq T} \tau_{s,d}, \tag{3.1}$$

where N is the number of nodes in the network. It is an important quantity to assess the performance of a network. To a first approximation, the average delay is proportional to the average number of nodes that the packets visit when traveling. If the traffic load λ (the packet creation rate at each node) presented to the network is low, then the queues on the nodes are empty. As the traffic load increases, the queues at the nodes start to build up, the average delay time increases accordingly. If the traffic load increases even further, then at the critical load λ^*, the queues of some nodes grow quickly and the average delay time increases dramatically or even diverge. At this critical load, it is considered that the network is congested.

This critical behavior is also noticed in the network throughput. The throughput is defined as the number of packets reaching their destination per simulation unit time per host [33]. Starting from a low load, the throughput increases proportionally to the increase of the load, until congestion is reached. At the congestion point the network has its maximum throughput. If the number of packets at node i at time t is denoted by $s_i(t)$, then the total number of packets in the network

is

$$S(t) = \sum_{i=1}^{N} s_i(t) = \sum_{i=1}^{N} Q_i(t), \tag{3.2}$$

where Q_i is the queue length of node i. If the network is not congested, and the average delay time is finite, then the average number of packets on the network:

$$\bar{S} = \frac{1}{t} \lim_{t\to\infty} S(t), \tag{3.3}$$

is also finite. At the congestion point, the queues of the congested nodes increase rapidly and this implies that the total number of packets in the network continues to increase.

3.3 Topology and Traffic

Ohira and Sawatari [29] were the first to study a simple model of computer network traffic which exhibits a phase transition from a low (free flow) to high (congestion) state. It was measured in terms of packet average travel time τ (End-to-End delay) as a function of the packet creation rate λ in the network, $\tau = f(\lambda)$. Through simulations on a two-dimensional lattice model network, they found that the transition to congestion phase depended on how each router chose a path for the packets in its queue (Routing Strategy). Later, Fukś and Lawniczak [30], Solé and Valverde [31; 32], and Woolf *et.al* [33], found that, in a regular network, it was possible to predict the traffic load where congestion occured (a dynamical characteristic) by only considering the average shortest path lengths from all sources to all destinations (a topological characteristic). Their result can be expressed as: [30; 31; 32; 33]

$$\lambda^* = \frac{1}{\rho\bar{\ell}}. \tag{3.4}$$

Where λ^* is the critical load that the network goes into congestion phase, ρ is the percentage of nodes that are hosts (a source and sink of traffic), $1-\rho$ is the percentage of routers and $\bar{\ell}$ is the average path length. Their prediction does

not depend on a specific network topology and has been tested in rectangular network, random network and small-world ring-shape networks.

3.3.1 Critical Point of Congestion

There is a simple way to estimate the critical load when the network changes from free-flow stage to congestion. It is to look for the total distance that all the packets at time t have to travel to reach their destinations. In the congested phase if there are queues at all the nodes, then the change in total distance is $D(S_{t+1}) - D(S_t)$, where $S(t)$ is the number of packets in the network at time t, and $D(S_t)$ is the aggregated distance of all packets from their destinations at time t. The increase in the number of packets per simulation unit time is $\rho\lambda N$. The overall added distance by the generation of new packets is $\rho\lambda N\bar{\ell}$. By contrast, the aggregated distance is reduced by N given that every packet at the head of the queue moves one step closer to its destination. Thus the change in total distance to destination between time t and $t+1$ is:

$$D(S_{t+1}) - D(S_t) = \rho\lambda\bar{\ell}N - N. \tag{3.5}$$

The critical load λ^* can be found when the total distance no longer decreases, $D(S_{t+1}) - D(S_t) = 0$, which implies Equation (3.4). But this simple way can only measure at which point the network system would begin to accumulate packets, while it can not tell how congested the system is.

Another possible way to determine the critical load is to use Little's law [108]: "The average number of customers in a stable system (over some time interval) is equal to their average arrival rate, multiplied by their average time in the system". Simply saying, in a steady state, the number of delivered packets is equal to the number of generated packets.

$$\frac{\mathrm{d}S(t)}{\mathrm{d}t} = \rho\lambda N - \frac{S(t)}{\bar{\tau}(t)}, \tag{3.6}$$

where $\rho\lambda N$ is the average arrival rate to the queues per unit of time, $S(t)/\bar{\tau}(t)$ is the number of packets delivered per simulation unit time with $\bar{\tau}(t)$ being the average End-to-End delay. If the load is low, the queues at each node tend to be

empty and the average delay time can be approximated by the average shortest-path:

$$\bar{\tau}(t) \approx \bar{\ell}. \tag{3.7}$$

However if the load is high, the delay time can be approximated by the average shortest-path length plus the time packets spend waiting in the queues.

$$\bar{\tau}(t) \approx \bar{\ell} + T(S(t), \bar{\ell}). \tag{3.8}$$

3.3.2 Model for Regular-Symmetric Network

If the traffic is evenly distributed in a network and the network is not congested, there exists a steady state solution S^* for the number of packets in the network. Each queue, on average, contains S^*/N packets and the delay can be approximated by

$$\bar{\tau}(t) \approx \bar{\ell}(1 + \bar{Q}) = \bar{\ell}(1 + \frac{S^*}{N}) \tag{3.9}$$

where on average, a packet visits $\bar{\ell}$ queues with average queue length of $\bar{Q}$. From steady state solution ($dS(t)/dt = 0$) the total number of packets in the system is

$$S^* = \frac{\rho\lambda\bar{\ell}N}{1 - \rho\lambda\bar{\ell}}. \tag{3.10}$$

So the average traffic load generated at node w is

$$\lambda_w = \frac{1}{\rho\bar{\ell}(1 + N/S^*)}. \tag{3.11}$$

As the traffic load increases, the number of packets in the network increases accordingly. At the congestion point the number of packets in the network diverges, $S^* \to \infty$, and the critical load is $\lambda^* = 1/\rho\bar{\ell}$, which is also the same as Equation (3.4).

3.3.3 General Solution

Notice from previous research that the transition from the free flow phase to the congested phase is well approximated by the equations above for the case of

regular-symmetric networks, but not for other network topologies. For regular-symmetric networks it gives a good approximation to the critical load because the average queue size for all nodes is very similar. But in a scale-free network, due to the heterogeneous structure, the importance of the nodes vary from each other. If the more important nodes get congested, most part of the network gets congested. An alternative approach is to use the betweenness centrality to measure the node usage.

For example, in social network analysis, graph-theoretic concepts are used to understand and explain social phenomena. A social network consists of a set of actors, who may be arbitrary entities like persons or organizations, and there exists one or more types of relations between them. A quantity of interest in many social network studies is the "betweenness" of an actor w, which is defined as the total number of shortest paths between pairs of actors that pass through w. This quantity is an indicator of who the most influential people in the network are, the ones who control the flow of information between most others. The same concept can also be applied to the computer network where nodes are connected through links. The load of a node or the whole network is affected by the flow of packets through the communication network from their sources to their destinations. The pattern of these flows can be measured by looking into the shortest-path that packets used while traveling. A common choice is the betweenness centrality C_B introduced in previous chapter. The nodes with high betweenness centrality are relatively more important in the network as they are visited by more routes. Consequently queues on them can be built up more easily. It is expected that the removal of these nodes will worsen the network performance. The nodes with high betweenness also result in the large increase in average distance between others when they are removed.

So it is possible to use the betweenness and the Little's law to obtain an approximation of the onset of congestion when the routing is done using the shortest–delay routes instead of the shortest paths. Firstly, the normalized betweenness centrailty of a node w is introduced below:

$$\hat{C}_B(w) = \frac{C_B(w)}{\sum_{v \in V} C_B(v)}, \tag{3.12}$$

where $\sum_{v\in\mathcal{V}} C_B(v)$ is the sum of all the nodes' betweenness centrality in the network. Then the average queue size at node w can be approximated by

$$\bar{Q}_w \approx \hat{C}_B(w)S^*, \tag{3.13}$$

where S^* is the steady state solution of the number of packets in the network. It is possible to approximate the average time a packet spends in all the queues along its path, that is

$$\tau^*(s,d) \approx \sum_{w\in\mathcal{R}(s,d)} \hat{C}_B(w)S^*, \tag{3.14}$$

where $\mathcal{R}(s,d)$ is the subset of nodes that the packet visits which depends on the route calculation of routers. The average delay time is the propagation delay plus the queueing delay:

$$\bar{\tau}(t) \approx \bar{\ell} + \frac{1}{N(N-1)} \sum_{s\in V} \sum_{d\neq s\in V} \sum_{w\in\mathcal{R}(s,d)} \hat{C}_B(w)S(t) = \bar{\ell} + DS(t), \tag{3.15}$$

where

$$D = \frac{1}{N(N-1)} \sum_{s\in V} \sum_{d\neq s\in V} \sum_{w\in\mathcal{R}(s,d)} \hat{C}_B(w). \tag{3.16}$$

Here the queueing delay is considered the average delay along the chosen paths for transmitting the data packets by the routers. The critical load in this general case is given by:

$$\lambda_C = \frac{1}{\rho N D}. \tag{3.17}$$

It gives a very good approximation of the phase transition from free flow phase to congested phase for the general network topologies. It is not difficult to see Equation (3.17) also works for regular-symmetric networks. From Chapter 2, in regular-symmetric networks C_B is a constant for everynode. In this case, the normalized betweenness centrality is

$$\hat{C}_B(w) = \frac{1}{N} \tag{3.18}$$

then $D = \bar{\ell}/N$ according to Equation (2.8). Substituting D in Equation (3.17),

Equation (3.4) is recovered.

3.4 Discussion – Network Invariants

The definition of an invariant in mathematics is: something that does not change under a set of transformations. In the case of a network simulation, the invariants are not very obvious as there are a few variables in the simulation, such as number of nodes and links of the concerning network. The way of considering the invariant of networks that catches most of the interests would be focusing on the scale of the network. Would the characteristic of a small scale network be the same as or similar to that of a large scale network? This is the question that is the focus point of the following section.

A first approach to search for invariants is to start with examining the topology of the network. The approach would be to simplify the topology in the way shown in Figure 3.2. The original network is split into different regions, shown as colored disks in the graph. In each region the network is a "tree" like subnet. This means for nodes located inside one of these subnetworks, there is only one path between them. By representing each tree with a super-node, the original network becomes a network where all its nodes can be reached by more than one path. This procedure is different from dividing the Internet into domains, as some domains will be split into different subnets, and at AS-level some domains form trees themselves.

But our interest lies not only in this kind of structural properties of the network, but also in considering the traffic on it. The real world networks, for example the Internet, are dynamic networks, either the topology or the traffic in the network is changing all the time. The starting point to study network topology and traffic invariant is to analyse the interaction between traffic and topology. This interaction can be seen through the phase transition property of networks, which indicates the network goes from free-flow phase to congestion phase as the network traffic load increases.

For the case of Regular-symmetric network [30; 31; 33], it is possible to predict the traffic load when congestion occurs, by only considering the average of the shortest path lengths from all sources to all destinations. This prediction of

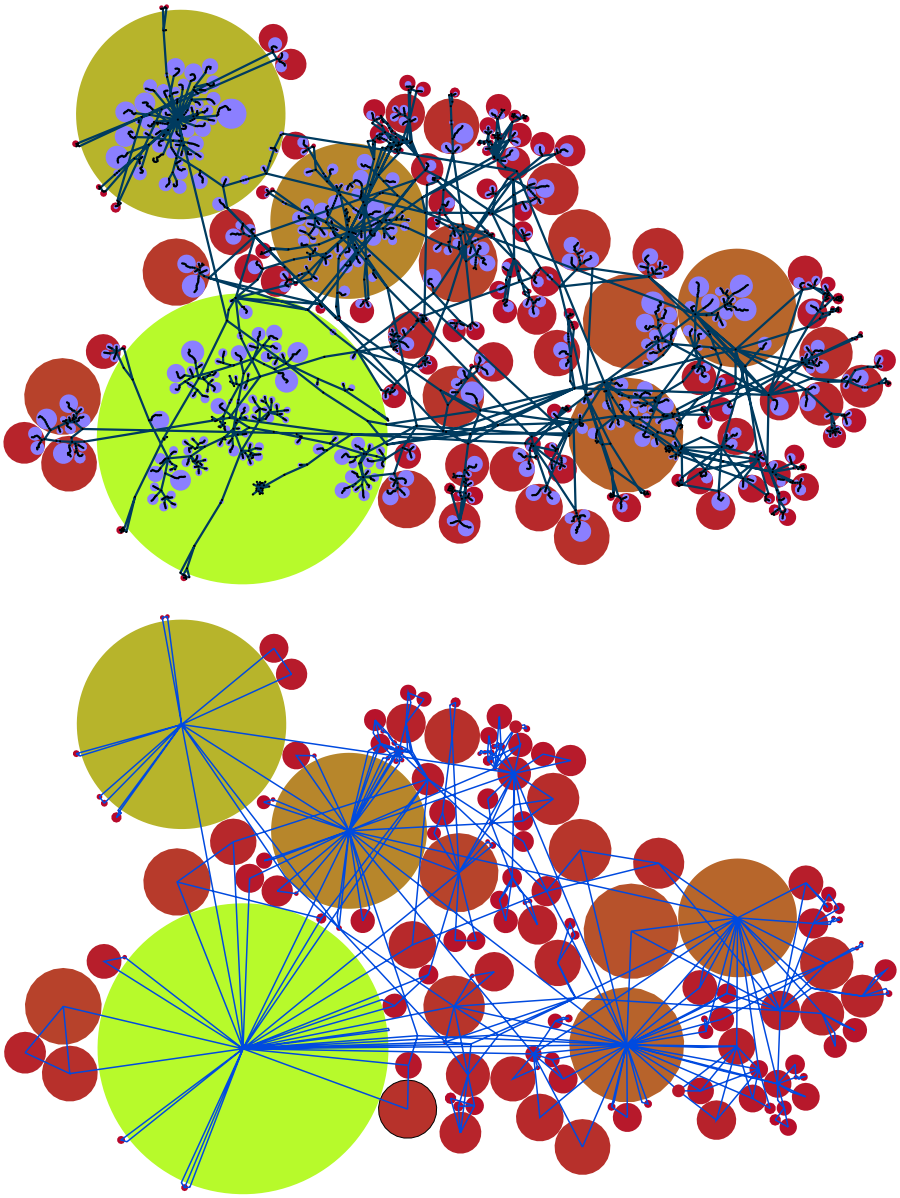


Figure 3.2: A small portion of the router-level QMUL network obtained using traceroute(upper) and the implication(lower)

critical load does not depend on the concerning network size. Furthermore, the general solution for predicting network critical load is also considered as an invariant for different topologies. It can be used to predict the critical point where networks go into congestion which depends on the size of the network.

The verification of the critical load as an invariant can be done by examining several metrics in network simulation. One of them is total number of packets in the network system $S(t)$. It is the summary of all the queue lengths in the network nodes $S(t) = \sum_{i=1}^{N} Q_i(t)$. In Figure 3.3, we verify the Little's Law for different networks. In all cases the network have $N = 100$ nodes. The coordinates have been normalized by dividing them by the critical load of the corresponding network [55].

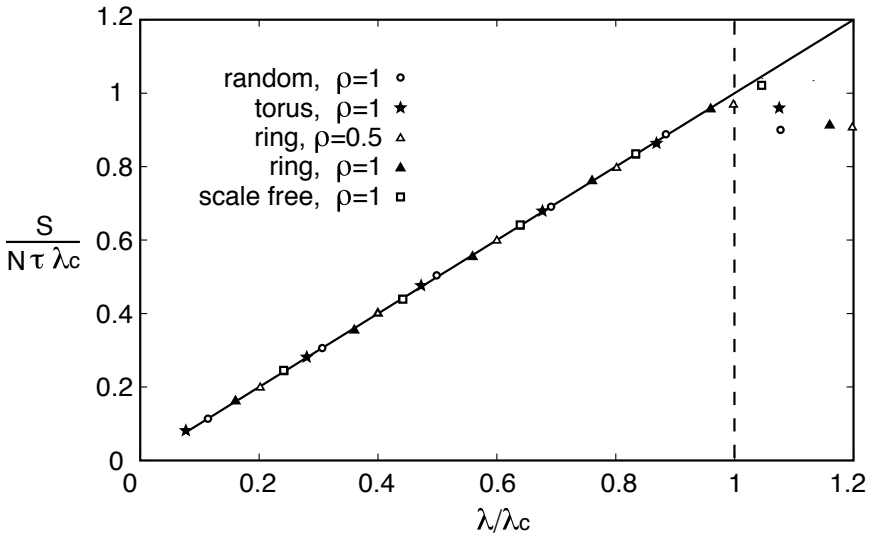


Figure 3.3: Verification of Little's Law for different network topologies.

3.5 Summary

In this chapter, a closer look has been taken at the interaction between network topology and network traffic. The consequence of the interaction is network

performance under different traffic loads. In free-flow stage, packets can be transferred to its destination with small delays and the network throughput is high. When the network traffic load passes the critical point, the network throughput decreases significantly and the packets arrival rate also decreases. At this point, the network is considered congested.

To further examine this phase transition process, a simple solution, Little's Law and a general solution using betweenness centrality are introduced. Finally, the concept of invariants for the networks are explained. All these serve as the basis for further investigation of the congestion and failures in the network.

Chapter 4

Building Catastrophic Networks

4.1 Introduction

Previous researches have shown complex networks are sometimes vulnerable to collapse [20; 24; 34; 35; 36]. Especially, Man-made complex networks such as the Internet, power transmission grids and telephone systems are susceptible to catastrophic failures in which the entire or most of the network ceases to function [37; 38; 39; 109; 110]. A common mechanism behind this kind of failure is an avalanche or cascade failure triggered by a wide variety of events, i.e. hardware failure, software failure, human error, or intentional attacks.

In a network that carries flows, such as data packets network, electricity power grid, road transport network [111], social acquaintance network [112], its individual nodes experience a load. In normal circumstances, the load is not expected to exceed the capacity that a node could handle. Most of the man-made networks are built to accommodate changes of network load and sometimes could even react to topology changes. But from time to time, networks experience a rare but destructive cascade failure. Cascade failures are initiated when a node in the network is lost because of malfunction and overload. Then the network's load on that node will be redistributed to other nodes, i.e. the routing protocols would recalculate the best route to redirect data packets. This redistribution of the flows may cause the load on other nodes to exceed their capacity. Even if these newly overloaded nodes do not fail, the network would again try to avoid

them by redistributing network traffic again. Hence the number of failed or heavily overloaded nodes could increase, triggering an avalanche-like event. In the worst case, the entire network may fail to function, or fragment into disconnected subnetworks.

In communication networks, there are different ways to eliminate network congestion. An obvious approach to route packets through a network is to send them through the shortest path. In the Internet, routing weights are placed on links according to different metrics. These weights are used to calculate shortest paths and generate routing tables [113; 114]. Much work has been done in finding better alternatives to shortest path routing [115; 116]. Although these routing methods have shown considerable improvements in carrying capacity, that is the load that can be carried by the network before jamming occurs. However, as discovered by Sreenivasan *et.al* [117], there is a limit to how much improvement may be made in this way. Besides all heavily loaded networks are at the end vulnerable to cascade failure.

It is claimed that networks with heterogeneous node degree distribution, such as scale-free networks, are much more likely to suffer from cascade failures [42; 87]. Because a small subset of core nodes are highly connected and manage much of the traffic in a scale-free network. If one of these heavily connected core nodes ceases to function, it will have a large impact on other nodes in the network. However, similar catastrophic failures are also possible in networks with more homogeneous degree distributions. If the network degree distribution has just a small amount of heterogeneity, then avalanche-like breakdowns are possible when all nodes are close to their failure load. Similar behavior has been seen in social networks [39]. In the theoretical case of a completely homogeneous network in which all nodes are close to their maximum load, failure of a single node could cause the entire network to collapse in a single step.

In this chapter, we firstly introduce the computer network congestion collapse and router failures. These simple failures in the network would sometimes propagate through the network. We then introduce a novel, simple and successful method to construct a "catastrophic network", based on the calculation of betweenness centrality to estimate the potential traffic load of network nodes. The

catastrophic network built is a type of network that is designed to fail catastrophically by avalanche-like (or cascade) breakdowns. We construct them by following the cascade in reverse: adding a new node that is designed to be congested first in the new network and then add a subnetwork of nodes connecting to it. This node serves as the bridge between the existing network and the subnetwork. The "random growth" method is used in search of the possible topology that meet all the requirements of the cascade. We find out that the catastrophic network generated using "random growth" method has similar degree distribution as the random network rather than power-law degree distribution. It indicates that a random network also has the possibility to experience cascade failures.

4.2 Background and Motivation

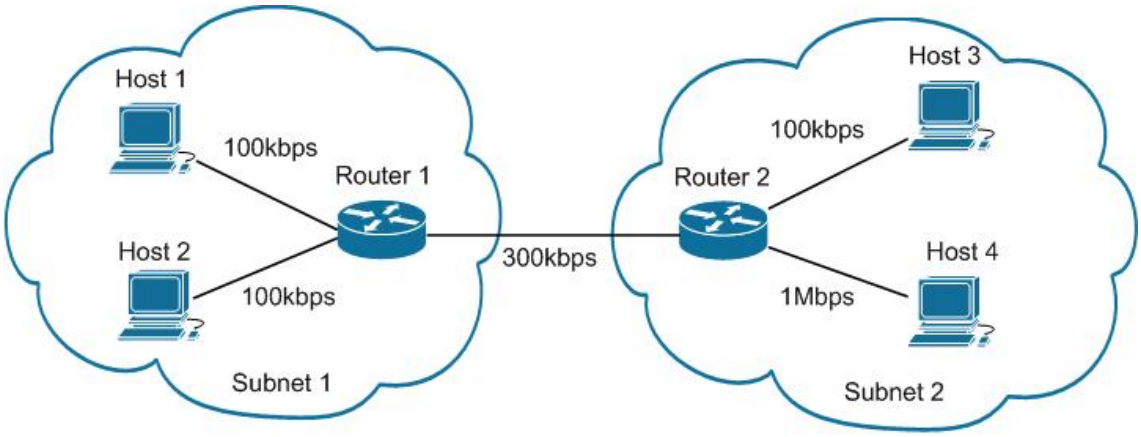
Compared to its ancestor the ARPANET, the Internet is rapidly expanding and becoming much more complex. With the growth of the Internet infrastructure, the traffic it carries almost double every year [118]. In particular, the recent introduction of video and voice applications has brought the biggest challenge yet to the communication industry. Home users of the Internet take it for granted to place VoIP calls using "Skype" or other instant messaging software, watch shared videos on "YouTube", or watch popular TV shows and movies using "BBC iPlayer" or their video game consoles. Cooperate users on the other hand embrace the Internet conference tools which intend to deliver high definition video and voice, for example the Cisco's "TelePresence" which provides high-definition video and spatial audio. What is even more unexpected is the introduction of the third generation mobile network, which enables users to stream multimedia content right to their mobile handset known as the smart phones. The trend of multimedia communication and entertainment will dominate the network traffic in the coming years. With this trend comes the same old problem congestion and even failure of the network.

4.2.1 Congestion Collapse

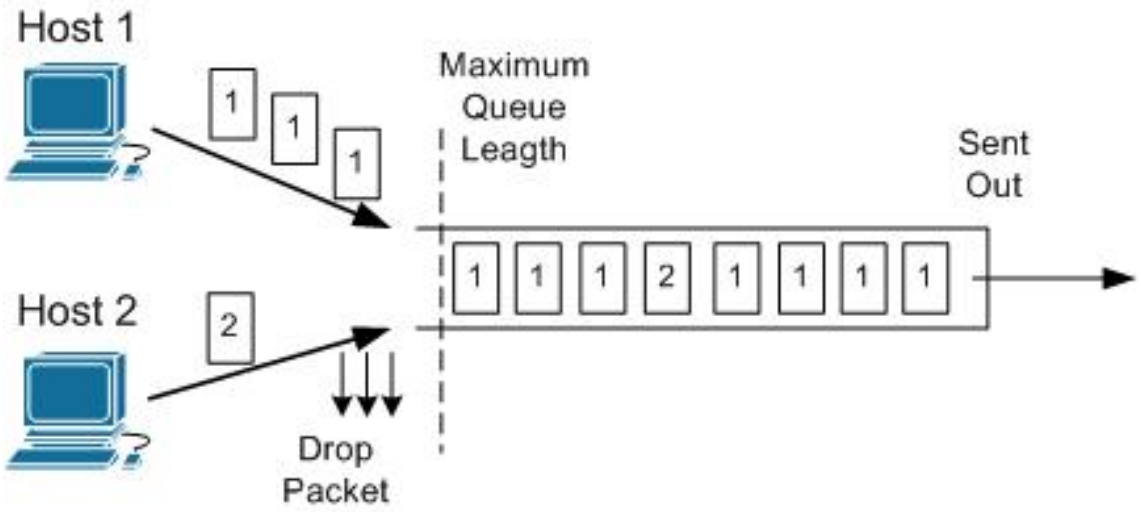
In the early years of the Internet, it was mainly deployed in universities where every September the network suffered congestion collapse due to the coming back of the students which increased network traffic dramatically. Four congestion control algorithms: slow start, congestion avoidance, fast retransmit and fast recovery were introduced by Van Jacobson to solve this problem [98; 119]. They have been implemented in the TCP to control the congestion. During the peak period, more packets would be sent to the network than it could handle. Thus routers will pile up packets in their buffers known as queues, which are designed to be cleared up later. In a more extreme situation when the queues are overloaded, the routers will begin to discard incoming packets. The control mechanism of TCP will kick start to tell the traffic sources to slow down the transmissions. This feedback control behavior of TCP is vital for the operation of the Internet that we rely on. In RFC 2914 [99], best current practice for congestion control in the Internet is described.

Consider a simple example of congestion collapse [105]. Figure 4.1 (a) shows two subnets (Subnet1 and Subnet2) with two hosts in each of them. Host1 sends data to Host4, while Host2 sends data to Host3, and both sources always send as much as possible (100 kbps). There is no congestion control in place. Administrator will notice that Subnet1's outgoing link is not fully utilized (2×100 kbps is only 2/3 of the link capacity). Thus, a decision is made to upgrade the link from Host1 to the Router1 to 1 Mbps. Since both sources keep increasing their rates without concerning the bandwidth, there will be congestion at Router1. Figure 4.1 (b) shows the queue builds up on Router1 which has more packets from Host1. Roughly, for every packet from Host2, there are 10 packets from Host1. This means that the packets from Host1 unnecessarily occupy bandwidth of the bottleneck inter-router link that could be used by the data flow from Host2. The more the Host1 sends, the greater this problem becomes.

With the rapid increase of streaming traffic, the best effort oriented Internet may not guarantee that video and voice will stream smoothly to the user. Firstly, routers discard packets randomly which may cause some of the streaming transmission to stall [120]. Secondly, the packets that are queued because of



(a)



(b)

Figure 4.1: Simple network scenario to analyse congestion collapse.

overloading experienced substantial and non-uniform delays, significantly reducing throughput and may worsen the congestion. This may be the result of the fundamental design of the Internet and its topology. The main idea of the original Internet design was to share the available network infrastructure by sending data as small independent packets which though they might arrive at different times, would still generally make it to their destinations. Consider a typical router that receives two packets with the same source and destination address and port, it would examine the two packets separately and enquire the routing table twice. This same CPU intensive process repeats again and again even with packets belong to the same stream.

Surprisingly, we are not experiencing unacceptable VoIP calls and online videos. This is because the Internet has been over-provisioned by the Telecoms. ISPs (Internet Service Provider) used to deploy extra equipment to cope with traffic spikes. But recently they cannot install new equipment fast enough, due to high cost of hardware, energy bills and environmental issues. ISPs tend to push the existing infrastructure to their operating limits. Although this is cost effective, but sometimes it is not a safe and easy way to operate the network. One of the potential risk is network failure due to congestion.

Besides, there are increasing amount of video and voice traffic in today's network because of our growing demand of multimedia entertainment. These traffic usually use UDP as their transport protocol. UDP provides a minimal, unreliable, best-effort, message-passing transport protocol. Different from TCP, it does not establish end-to-end connections between communicating end systems. That's why UDP can offer a very efficient communication transport for streaming applications. But on the other hand, UDP provides no inherent congestion control mechanisms. Many systems can send UDP packets at the line speed of the link interface, which is often much greater than the available path capacity, and doing so contributes to the congestion all along the path.

4.2.2 Router Failures

A router is specialized hardware/software equipment that routes packets to their desired destination [73; 76]. Because of their design, manufacturing and redun-

dancy deployment, a router is actually very reliable that it seldom fails by hardware faults. But on the other hand, because of its vital role in the Internet, it is always generating problems and targeted by attacks.

Causing a router to fail can be done by exhausting its available resources. The simplest problem with some routers are software bugs which cause the router operating system to crash or hacked. For example, there is a buffer-overflow attack that jams the buffer with spurious packets. These packets might be caused by "Ping" flow generated by a "Bot-net" with millions of controlled "Zombies" (hacked computers). Sometimes a router may be stalled by the sheer amount of traffic going through which cause the memory overflow. There are of course many other ways to exhaust a router's resources, but we are emphasizing congestion related ones.

An example of the recent router failure in the Internet is the widespread outage of Gmail's web interface in September 2009 [1]. It is known that many people rely on Gmail or other online mail system for personal and professional communications. The interesting point for us is not how bad everyone is affected by the lack of Email services, but rather the cause and the spread of the failure. From the announcement of the Gmail official blog, the outage happened as follows. Firstly, Google took a small fraction of Gmail's servers offline to perform routine upgrades. But this isn't itself a problem, web services providers do this all the time, because Gmail web service servers run in many locations with request routers. These routers simply send traffic to other locations when one is offline. However, Google had underestimated the load on the routers that was upgrading. They designed these routers to improve service availability by directing web queries to the appropriate Gmail server for response. But a few of the request routers became overloaded because of planned maintenance on a small fraction of servers. In effect they told the rest of the routers in the system to stop sending them traffic because they are too slow. This transferred the load onto the remaining routers, causing a few more of them to become overloaded, too. Within minutes, nearly all of the request routers were overloaded. As a result, people couldn't access Gmail via the web interface because their requests couldn't be routed to any Gmail servers.

The Gmail engineering team was alerted to the failures within seconds but without solving it immediately and efficiently. After establishing that the core problem was insufficient available capacity, they brought a large amount of additional routers online to distribute the traffic across their network. We notice that this simple but costly solution is to increase the network capacity well beyond the demand to provide more resource. But not every service provider or company can afford this, neither the cost of equipment nor the energy. It is claimed by Google that their routers do not have enough failure isolation (i.e. if there's a problem in one data center, it shouldn't affect services in another datacenter) and do not degrade gracefully (e.g. if many request routers are overloaded simultaneously, they all should just get slower instead of refusing traffic and shifting their load).

Although the Gmail server network is an overlay network, the event shown the possibility that might congestion spread throughout the whole network. In the following sections, we try to investigate the potential congestion failures and sometimes cascade failures by generating a special kind of network topology.

4.3 Building Catastrophic Networks

4.3.1 Previous Studies

The method used by researchers to study cascade failures is to overload one or more nodes in a pre-existing network and study the resulting cascade [39; 42; 43]. Holme and Kim [40] have a slightly different approach, evolving a scale-free network until cascade failure occurs due to the increasing load in the network. In [43; 44], breakdown is simulated by computer, whereas [39; 42] use mathematical models. One difficulty of the former approaches [40; 43; 44] is that this type of simulation is very computationally demanding and unpredictable. It may also imposes a limit on the size of network that can be modeled. This makes it difficult to find out how well the mathematical models scale with network size. In our recent paper [121], a new way to study cascading failure has been approached from another direction. The building of networks in such a way as to ensure their breakdowns in essence follow the cascading breakdown in reverse. By doing so, it is hoped that one can have a better understanding of the dynamics of the

process. This approach is also less computationally demanding and will therefore allow the simulation of larger networks. The model can also be easily extended to real-world networks.

4.3.2 Proposed Method

The model for building a new set of networks with cascading failure properties is introduced in this section [121]. The method is to simulate an avalanche-like failure in reverse, building a network that is designed to fail by cascading breakdown. This model, which is the simplest one, focuses primarily on networks with a single type of nodes and undirected unweighted edge. The generalization to more complicated network types are certainly possible. There are two central features of our method that distinguish it from other researchers' approaches. First, it approaches the network congestion and failure from a "growing" point of view, whereas most of other researches try to study the robustness of existing network topologies. From the "growing" of the cascade network topology, one can investigate the pattern and properties of network congestion more clearly, because cascade networks are designed to fail. It can also be used to examine exiting networks' robustness. Secondly, most of the network topology generation methods are node degree related, such as the famous ER-model and BA scale-free model. Instead of considering the degree, the method proposed takes the node betweenness centrality as the measure on step basis, while the method bases on edge betweenness could be considered in the future implementations. Generally, the model is based on two critical processes: "Growing" and "Subnetwork Attachment".

4.3.3 Betweenness Centrality and Network Load

To further study the congestion behaviour of networks, we correlate the topology of the network with the traffic that it carries via a routing mechanism. The simplest routing mechanism is to deliver the packets through the shortest paths connecting a source and destination node in the network. The load on a particular node w can be approximated by counting the fraction of shortest paths that pass through it. This approximation is given by the betweenness centrality

C_B. Suppose the network is working very efficiently and that the traffic in it is distributed evenly throughout all shortest paths by the routing protocols. Then we again use the normalized betweenness centrality of a node w in the network which gives the relative usage of this node against the rest of nodes.

$$\hat{C_B}(w) = C_B(w) / \sum_{v \in \mathcal{V}} C_B(v) \tag{4.1}$$

For simplicity we consider that each node w in the network produces packets at the same rate $\Lambda_w = \Lambda$, distributed evenly between the $N-1$ destination nodes available. The total flow in the network, $F(\Lambda, N)$, increases linearly with the network size $F(\Lambda, N) = N\Lambda$.

Congestion happens in the network when node w becomes overloaded. That is the packet arrival rate to node w, λ_w, is equal to or larger than its packet service rate, μ_w. The average number of packets that arrive to node w is [44]

$$\lambda_w = \Lambda N \bar{\ell} \hat{C_B}(w) = \frac{\Lambda C_B(w)}{N-1}, \tag{4.2}$$

where N is the number of nodes, ΛN is the number of packets generated in one unit time by the whole network, $\bar{\ell}$ is the average shortest path of the network to account for the average number of packets that were produced in the past and they are still in transit. And $\hat{C_B}(w)$ is the proportion of all the packets in transit that pass through the node w. Equation (4.2) is obtained by relating the topology of the networks via C_B with the traffic that it carries via Λ. Besides, one of the properties of betweenness is also used:

$$\sum_{w \in \mathcal{V}} C_B(w) = N(N-1)\bar{\ell} \tag{4.3}$$

This can be understood by performing the sum on the left-hand side of the equation in a different order: taking each node pair in the network and counting which node's betweennesses they contribute to [121].

As all the nodes produce the same amount of traffic Λ, from Equation (4.2), the condition of congestion is the critical network traffic generation rate (critical

load of the network) Λ^*:

$$\Lambda^* \geq \frac{\mu_w(N-1)}{C_B(w)}. \tag{4.4}$$

Previous work shows that this congestion model can be used to study the robustness of mission critical networks [122].

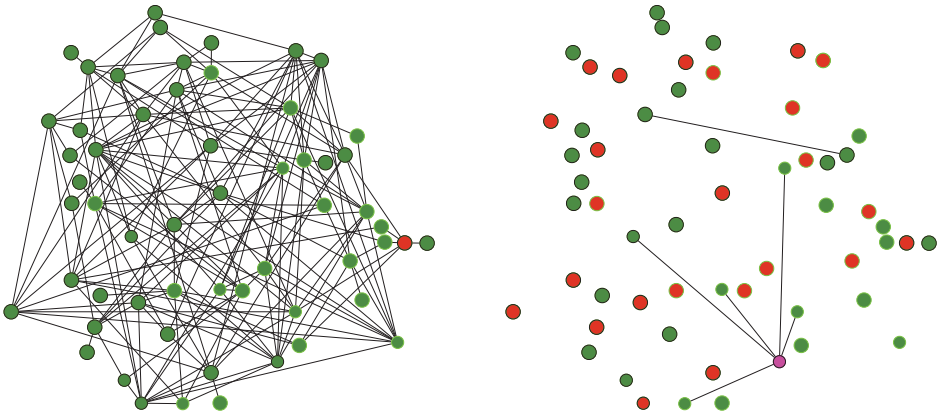
4.3.4 Cascading Failures

When nodes in the network are getting congested, an avalanche might happen in some cases. We consider overload as the cause of node failure. The model for the avalanche is: when a node becomes congested, all links connected to it and the node itself will be removed from the network. Also any traffic generated by the failing node is removed, see Figure 4.2. This means that the node and its links cannot serve the incoming packets anymore and send out no more packets to the network. After the removal, traffic in the network is redistributed by routing protocols by means of recalculating the routes. Thus loads on other nodes might increase and they might also suffer from overloading. These newly congested nodes would be removed from the network and the traffic is redistributed again. This process will go on until no nodes in the network is overloaded. In a large cascade, the network will break down into many disconnected subnetworks [41].

4.3.5 Network Segmentation

The idea of building a catastrophic network is to follow the avalanche process in reverse. Starting with one or more small "seed" networks, the catastrophic network is built step by step. Suppose during building step i, we add a node n_f^i and connect it to the existing network A. And we target to make n_f^i to congest first in this step. To achieve this, a subnetwork of nodes and links also need to be added. This subnetwork B should be connected to n_f^i only and has no connection to A. In this way n_f^i is designed to be the bridge between the existing network and the newly added sub–network. So that shortest paths between nodes in A and B must pass through node n_f^i which would have higher load.

Thus at each step, a new "congestion" node n_f^i is added at first (see Fig. 4.3). Then this node is connected to the existing network A that contains N_A nodes.



(a) (b)

Figure 4.2: Example of network cascade the original network (a) has 57 nodes. If node 56 shown in red is congested, its links would be removed from the network. Then the load is redistributed, making a sequence of node (also shown in red) to be congested and their links removed. In a extreme case, the network is broken down into disconnected subnetworks.

The sub–network B with N_B nodes is introduced and connected to n_f^i. In this way, three parts are examined at each step, A, B and n_f^i. The main difficulty then lies in how could we find out the set of connections between n_f^i and A and the topology of B with connections to n_f^i efficiently.

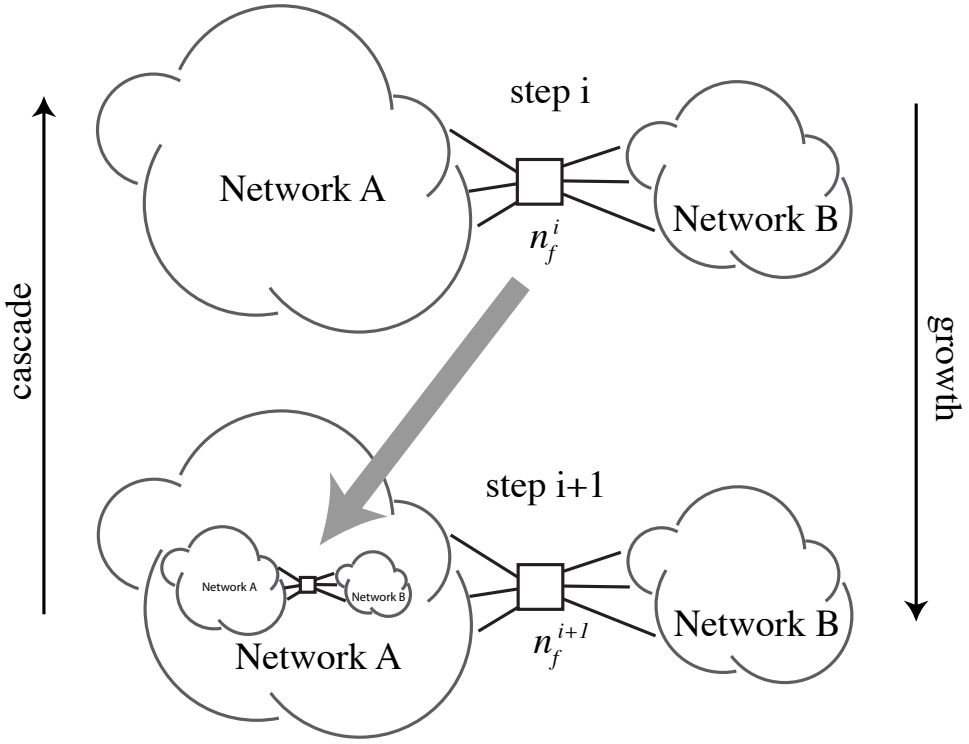


Figure 4.3: Schematic representation of the method to build the network cascade in reverse.

4.3.6 Basic Building Steps

The basic steps for building the catastrophic networks is to start with a small seed network. It could be different kinds of network, i.e. ER random network. Then at every growing step, we follow:

- For a new growing step $i+1$, the new network G_{i+1} can be divided into three parts. The subnetwork A which is the network G_i from the last growing

step i, new node n_f^{i+1} which is designed to congest first in the new network, and the new subnetwork B.

- Connect n_f^{i+1} to subnetwork A with number of links ranging from 1 to N_A.
- Generate a small size random network with network size starts from one as subnetwork B and connect it to node n_f^{i+1}. We must try different subnetwork B and make sure n_f^{i+1} is the first node to congest in G_{i+1} and the congested load of n_f^{i+1} is close to n_f^i.
- n_f^{i+1} acts like a bridge between subnetwork A and B. In this way, there is no direct connection between A and B and n_f^{i+1} has very high betweenness centrality.

The resulting network will have the property to congest at the designed load and a cascade failure would happen when it reaches this load.

Note that from the growing methodology, the whole network load will increase following the growth of the network. In the constructed catastrophic network, if node n_f^{i+1} gets congested when

$$C_B(n_f^{i+1}) = \mu_{n_f^{i+1}}(N^{i+1} - 1)/\Lambda^*, \tag{4.5}$$

node n_f^{i+1} and its edges will be removed. Node n_f^i becomes the next congested node in the cascade-down network. n_f^i has

$$C_B(n_f^i) = \mu_{n_f^i}(N^i - 1)/\Lambda^*, \tag{4.6}$$

where $C_B(n_f^i)$ is the betweenness of node n_f^i in the reduced network and N^i is the reduced network size. Hence $C_B(n_f^{i+1})$ and $C_B(n_f^i)$ must satisfy the following equation for the cascade to happen.

$$C_B(n_f^{i+1}) = C_B(n_f^i)\left[\frac{\mu_{n_f^{i+1}}(N^{i+1} - 1)}{\mu_{n_f^i}(N^i - 1)}\right], \tag{4.7}$$

To start with a simple case, we supposed that the service rate for all nodes in the network is the same, one packet in a unit time $\mu = 1$. A bound of the

betweenness centrality of node n_f^i in each growing step could be obtained using Equation (4.2). In this case the node with the highest critical load is also the node with the largest betweenness centrality. If the average betweenness centrality in the network is given by :

$$\frac{1}{N}\sum_{w\in V} C_B(w) = (N-1)\bar{\ell}, \tag{4.8}$$

and the maximum betweenness $C_B(max) = max\{C_B(w), w \in V\}$. A lower bound for the maximum betweenness centrality $C_B(max)$ is [94] :

$$C_B(max) \geq (N-1)\bar{\ell}. \tag{4.9}$$

From the growing methodology, we want node n_f^i to be the first to congest in each growing step. Hence, $C_B(n_f^i) \geq (N-1)\bar{\ell}$. During the growing step, if $C_B(n_f^i)$ reaches this lower bound, the growing would stop, because n_f^i would not be the first node to become congested.

Notice that the node with the largest betweenness centrality would be the first to become congested and the network critical load must be greater than or equal to that of the network of previous step

$$\Lambda^*(n_f^{i+1}) \geq \Lambda^*(n_f^i) \geq ... \geq \Lambda^*(seed), \tag{4.10}$$

where $\Lambda^*(seed)$ is the network critical load of the seed network. In this case, also using Equation (4.2), we have:

$$\frac{N^{i+1}-1}{C_B(n_f^{i+1})} \geq \frac{N^i-1}{C_B(n_f^i)} \geq \Lambda^*(seed). \tag{4.11}$$

An upper boundary for the maximum betweenness centrality in the network for every growing step can obtained:

$$C_B(n_f^i) = C_B(max) \leq (N-1)/\lambda^*(seed). \tag{4.12}$$

4.4 Random Growing Method

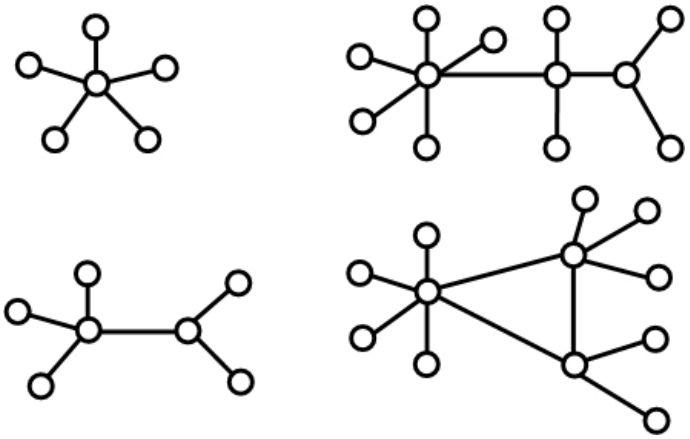
Following the two basic steps to create a catastrophic network: "Growing" and "failure node and subnetwork attachment", the straight forward way of finding the solution is to generate a random subnetwork B and also randomly connect n_f^i to the subnetwork A and subnetwork B (see Figure 4.3). The random connection is done until the condition $\Lambda^*(n_f^{i+1}) \geq \Lambda^*(n_f^i)$ and $C_B(n_f^{i+1}) \geq C_B(i)$ $(i = 1, ..., N^{i+1})$ are met. These are the two boundary conditions stated in the last section. To guarantee that the cascade process is able to be reversed, in the random growing method we need to make sure:

- New nodes in subnetwork B can only be linked to n_f^i and the newly added subnetwork B nodes.
- New links can be added between the subnetwork A and n_f^i; between n_f^i and nodes subnetwork B; between nodes in subnetwork B.

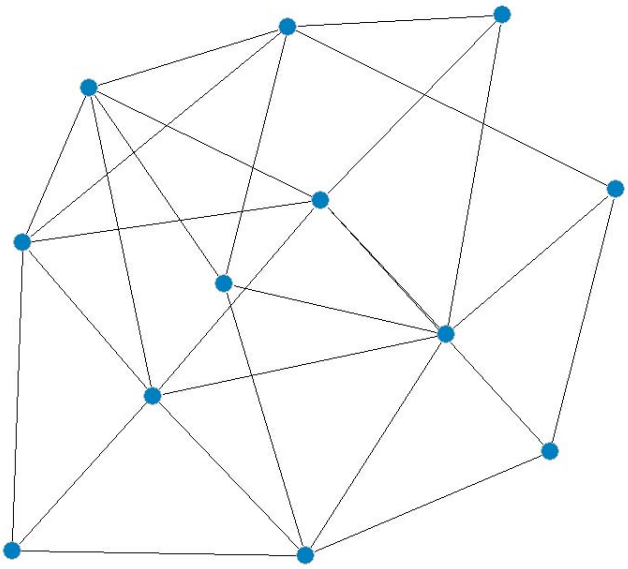
These rules make sure that there is no direct links between nodes in A and B. This results in a dramatic increase of the betweenness centrality of node n_f^i, where a large number of shortest-paths have to go through it.

4.4.1 Seed Networks

The seed networks from which the catastrophic networks are constructed have an impact on the resulting network topology, because the metric for growing the catastrophic network is the betweenness centrality. Although the betweenness centrality will change dramatically during the growing process, network generated at each step always depends on the betweenness of previous step, which would finally depend on those of the seed network. Also from the node degree point of view, seed network's structure could change the overall connectivity of the resulting catastrophic network by satisfying the betweenness centrality condition during the growing. Five different seed networks are used in the random growing method: a 6 node star-shape network, a 7 node two-stars shape network, a 12 node star-chain shape network, a 12 node three-stars shape network, and an ER random network (see Figure 4.4).



(a)



(b)

Figure 4.4: (a) The seed networks used by the random growth method to generate catastrophic networks. (b) small random seed network.

The star-shape networks are chosen because nodes in them have very extreme betweenness centrality values. The star-shape network has two kinds of nodes: the center node n_c and ray nodes n_r. The betweenness centrality of the rays of a star network is given by $C_B(n_r) = N-1$, while the well-connected center node of the star has the betweenness centrality $C_B(n_c) = (N-1)^2$. The center node will become congested at the critical packet production rate $\Lambda^*_{n_c} = (N-1)/C_B(n_c)$. Besides, many of the small size computer networks form a star shape topology with the center of the star the head office and the rays are the branches. We also use the two-stars, three-stars and chain network as more complicated seeds. In contrast, we also use small random network where betweenness centrality of nodes are relatively similar.

4.4.2 Random Growth Results

A 73-node catastrophic network grown from a 6 node star-shape seed network is shown in Figure 4.5. The growing steps of this network is briefly shown in Figure 4.6. Starting with the seed network (Figure 4.6(a)). Figure 4.6(b) in the second step of the growing the programme added one congest node (in red) and subnetwork (two bottom nodes in yellow) with their connections. At later steps, the catastrophic networks at the 24th step with 43 nodes (Figure 4.6(c)) and the 49th step with 73 nodes (Figure 4.6(d)) the networks are much larger with dense connections between their nodes.

To further investigate the property of this catastrophic network built using the random growing method, we first examine the betweenness centrality of n_f^i for each growing step. In Figure 4.7 the maximum betweenness (that is the betweenness of node n_f^i) is plotted against the network size. The line with short bars shows the maximum betweenness of n_f^i at each growing step; the line with stars represents the lower boundary; and the line with crosses shows the upper boundary. At each step in the building of the catastrophic network, the program searches for a network satisfying $\Lambda^*_{n_f^{i+1}} \approx \Lambda^*_{n_f^i}$ with the constraint that $\lambda^*_{n_f^{i+1}} \geq \lambda^*_{n_f^i}$, otherwise the avalanche will not occur.

Finding a network satisfying these conditions is not always possible. A network that becomes congested at the target load Λ^* is always harder to find as

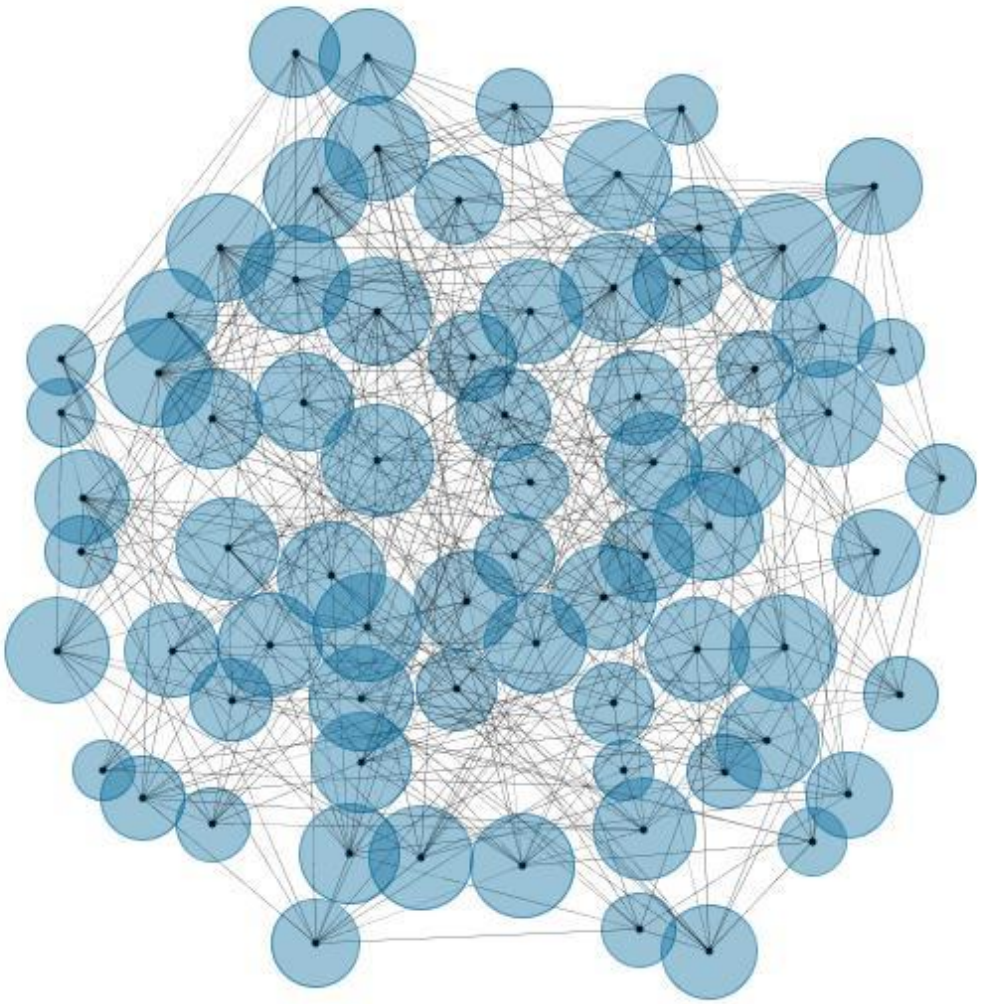


Figure 4.5: The 73-node catastrophic network generated using the random growth method

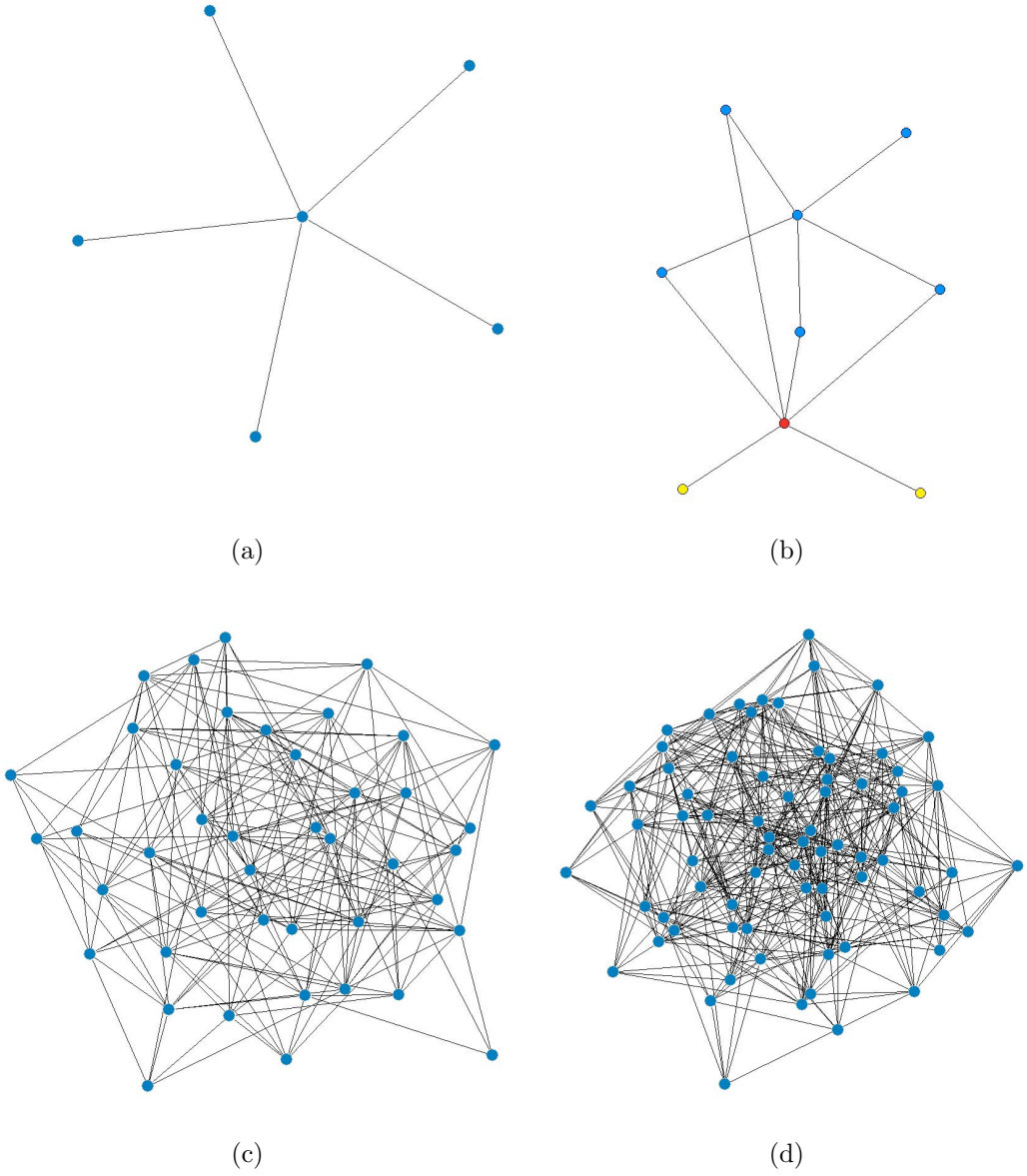


Figure 4.6: The process of building catastrophic network: (a) is the seed network to start the growth of catastrophic network. (b), (c) and (d) are the catastrophic networks at the second step of the growth with 9 nodes, 24th step with 43 nodes and 49th step with 73 nodes. They are all plotted using Pajek [4].

the network grows bigger. This explains the divergence of $C_B(n_f^i)$ from the upper limit. In Figure 4.7, the betweenness centrality of the congested node n_f^i for the first few steps follow the upper boundary closely. But the betweenness centrality value deviates from the upper boundary when the network size reaches 20 nodes and even further when the network grows bigger. The reduction in the betweenness centrality growth might be due to the random process introduced in choosing: subnetwork B, the connections between subnetwork B and n_f^i, and the connections between subnetwork A and n_f^i. One of the conditions set would allow $\Lambda^*(n_f^{i+1}) \in (\Lambda^*(n_f^i), 1)$, which means the load of the new network G_{i+1} could be much larger than that of G_i. When the load of the network reaches 1 we cannot grow it anymore, because the network is a hundred percent utilized. It is also noticed that the betweenness centrality grows with the network size. If the growth of betweenness centrality is not following the upper boundary which is the ideal solution, the maximum betweeness centrality of the network will sooner or later reach the lower bound. In this case, we cannot grow the catastrophic network anymore. Because if node n_f^i's betweenness is less than the lower boundary, it would not be the first node to congest in the network. This puts a limit on the growing of catastrophic network.

We take a closer look at the change of network load and the shortest path length which link to the boundary conditions, see Figure 4.8. The average shortest path length starts with a growth and stays between 2.2 and 2.5 when the network size is below 45 nodes. After that, the average shortest path in the network experiences a constant drop. It means the network is getting very dense that one node can reach the other one in just one or two hops. In the ideal situation, the critical load of the catastrophic network in each growing step should follow the target load, which is the congestion load of the seed network. But again due to the randomness introduced , it is hard to find the solution due to growing complexity to calculative network topology and the betweenness centrality. From Figure 4.8(b), the congestion load keeps increasing with the growing of the network and has the trend of reaching 1 when the network is large enough.

Degree distribution and cumulative degree distribution of two 73 node catastrophic networks are also shown in Figure 4.9. The degree distribution is approximately following a bell shape. It shows that these catastrophic networks are

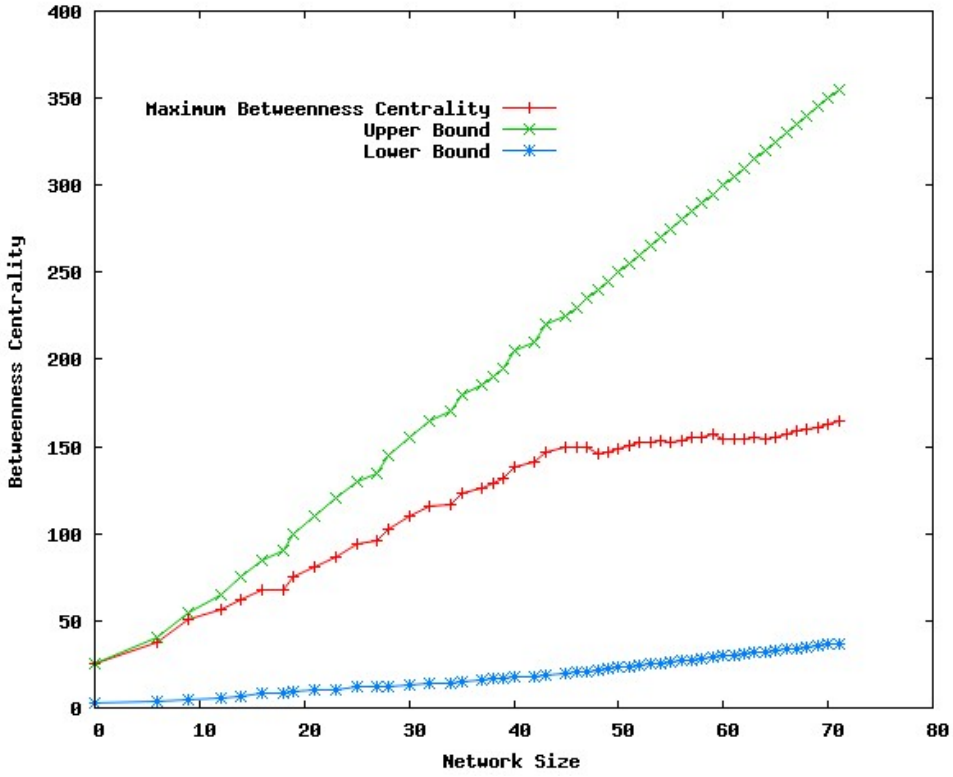
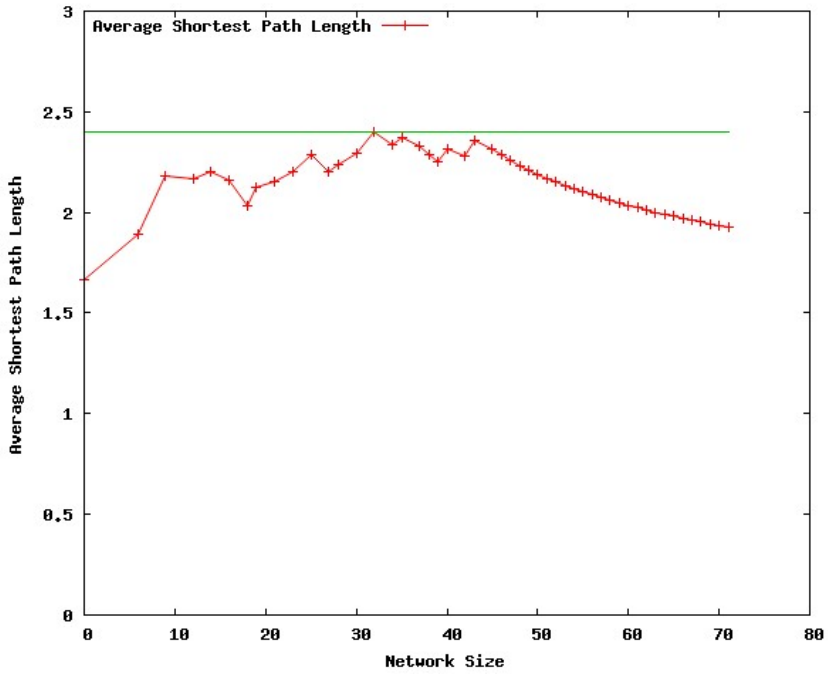
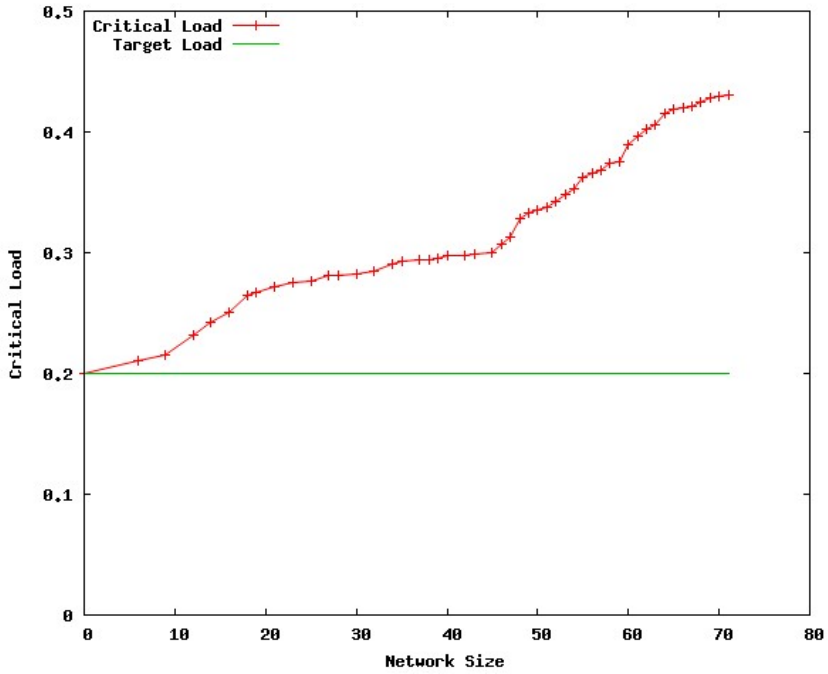


Figure 4.7: The maximum value of betweenness centrality as a function of the numbering of nodes



(a)



(b)

Figure 4.8: (a) the change of average shortest-path length during the growth. (b) the change of critical load during the growth. It is drifting away from the target load.

more like random networks with most of their node having 10 to 15 links each. The cumulative degree distribution also shows that they are closer to a random network structure.

4.4.3 Catastrophic Networks Growing from Other Seed Networks

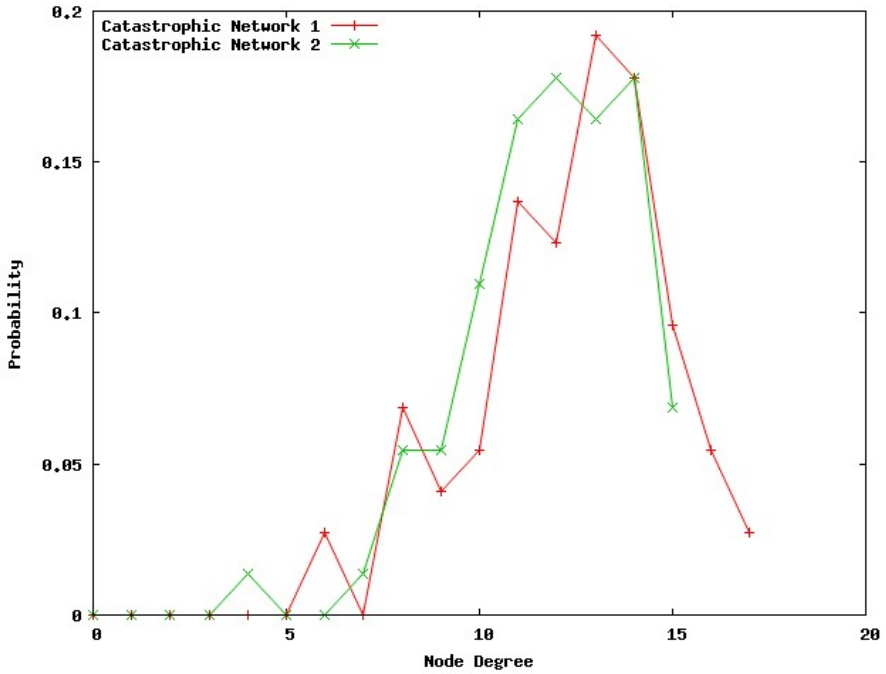
To further examine the growth process we have evaluated other seed networks: two-stars shape network, three-stars shape network, chain shape network and ER random network. These examples of the catastrophic network built using the random growing method are shown in Appendix A. The betweenness centrality growth and degree distribution of the resulting catastrophic networks built upon different seed networks are show in Figures (A.1,A.2,A.3,A.3) in Appendix A.

4.4.4 Discussion

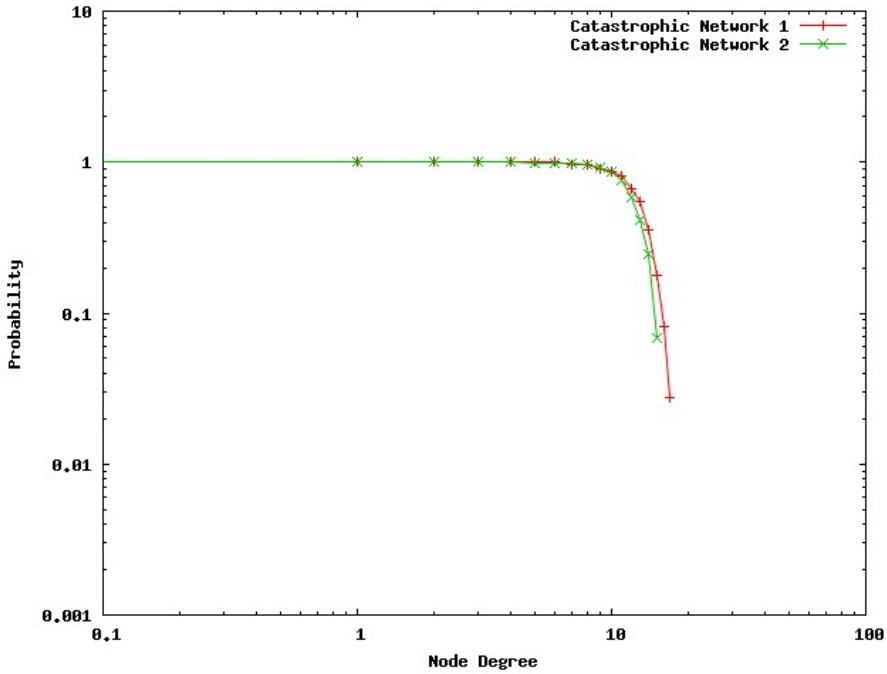
From all the results shown in this chapter, growing a catastrophic network is not a fast process. Firstly, it is time and resource consuming to search in the random space for all possible subnetwork B and connections between A and n_f^i, B and n_f^i that meet the growing conditions. Secondly, the random growing method is not very precise as more often it misses the ideal target condition for congestion. One of the indication is the reduction in the betweenness centrality growth, which can be found in most of the catastrophic networks built with different seed network type.

But it is already a big step forward from previous researches. In that, no previous research has grown a network that will fail catastrophically under certain traffic loads. Further, the size of networks is reaching 150, which can be seen as a mid-size router-level network inside an AS. Thirdly, the degree distribution of the catastrophic networks depends the seed network structure and the random process introduced.

We have also noticed that it is possible to construct networks in which the service rate μ is not the same for all nodes. This might make it possible to build networks in which avalanches begin at nodes having a heavy traffic load while the



(a)



(b)

Figure 4.9: The degree distribution (a) and the cumulative degree distribution (b) of the 73-node catastrophic networks.

remaining nodes are having a lighter traffic. Networks with nodes of different service rates could better simulate man-made networks. For example, the Internet or the power grids where a core router or an electrical substation has a large output traffic flow but not many links and consequently have a more "important" role in the network. Failure may begin with a node with high betweenness centrality, but the next node to fail in the avalanche may have a relatively small centrality, yet depending on the propagation of the avalanche through load redistribution. In this case the node betweenness might not reflect its importance in the cascade sequence. This behavior is illustrated in Figure 4.10, which is a specially constructed network. In this network normal unmark nodes handle one packet per unit time, Node B handle twice the rate at two packets per unit time. Node A, C and D can handle twice the flow B can. The radius of the nodes in the figure are proportional to the square root of their betweenness centrality. When the network traffic reaches the critical load, nodes fail in sequence: A, B, C, D, even though node C and D can handle twice the flow node B can. It is found out that the order of failure is unrelated to the betweenness centrality of a node. Due to the limitations of resource and time available, more in-depth research is needed to build the different service rate catastrophic network topology in the future.

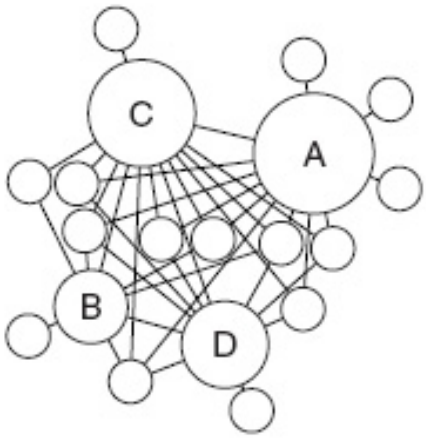
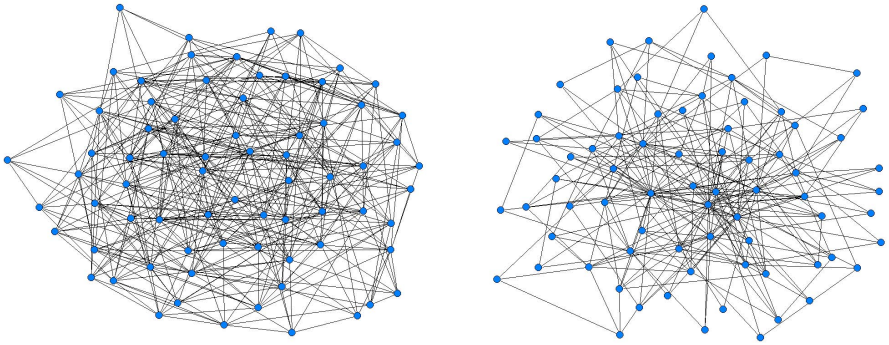


Figure 4.10: Catastrophic network with heterogeneous nodes

4.4.5 Comparison with Random and Scale-free Networks

The catastrophic networks generated using the "random growth" method used a random process. On the other hand, this method also introduces bias when adding in a node n_f^i at each step to meet betweenness centrality conditions. To find out the characteristic of this special family of catastrophic networks, we select one catastrophic network generated with 73 nodes (Figure 4.5). We try to compare it with a random network and a BA scale-free network of the same size, see Figure 4.11.



(a) (b)

Figure 4.11: Network topology of a 73 node (a) ER random network and (b) BA scale-free network.

Network Type	N	L	$\overline{k}$	k_{max}	$C_B(max)$	$\bar{\ell}$
Catastrophic Network	73	452	12.3	17	167.4	1.9
Random Network	73	422	11.6	20	244	2.0
Scale-free Network	73	257	7.0	29	989	2.2

Table 4.1: Comparison of topological properties of 73-node catastrophic, random and scale-free network.

By looking at the topology through plotting in the figures, it is not easy to tell the difference between these networks. Thus we analyse their topological properties in detail which are shown in Table 4.1. These three networks have the same number of nodes N. We calculate in our network analysis program: L the

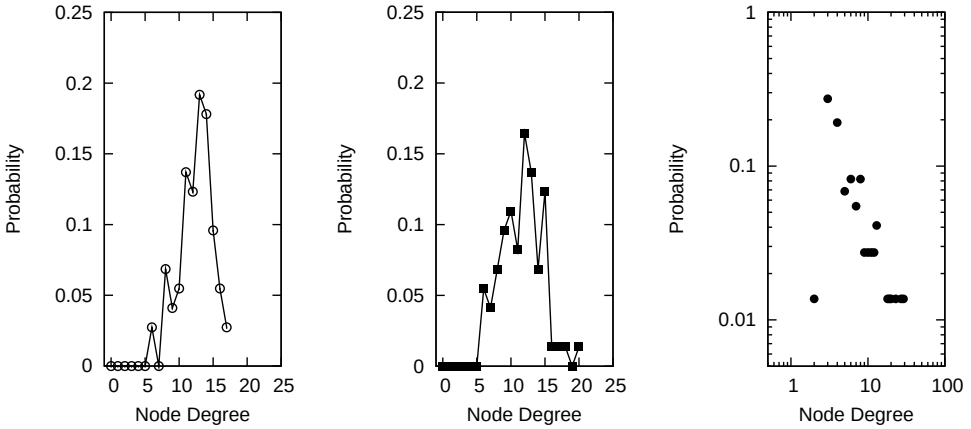
number of links in the network, $\bar{k}$ average degree of nodes, k_{max} maximum node degree, $C_B(max)$ maximum betweenness centrality and $\bar{\ell}$ average shortest path length.

This family of catastrophic networks has very similar topological properties as those of random networks. Structurally, the catastrophic network is very similar to a random network. But their difference lies in the maximum betweenness centrality value. The catastrophic network has even smaller maximum betweenness centrality than that of the random network, and five times smaller than that of a scale-free network. This is because of the random process introduced during the catastrophic network building process.

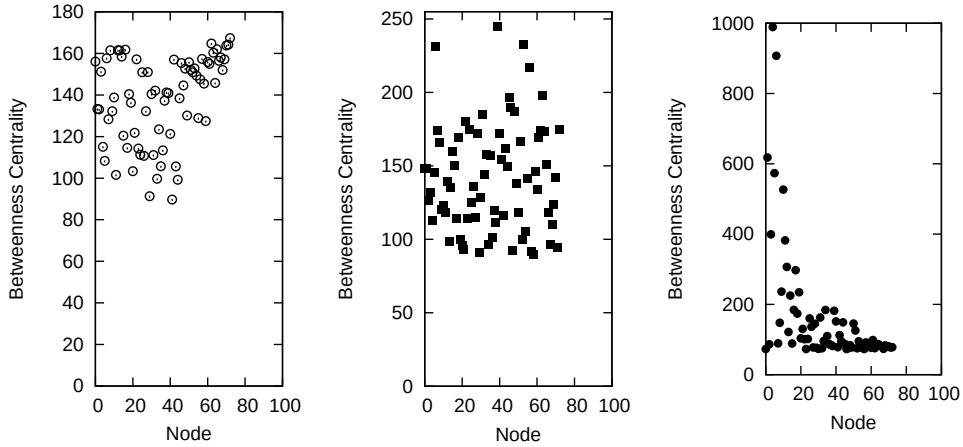
To further confirm this, Figure 4.12 shows the node degree distribution and the betweenness centrality of nodes against the node sequence number when they were added to network. We could see that degree distribution and the betweenness centrality of the catastrophic network and random network are quite similar, which significantly different from the ones of the scale-free network. Although this kind of catastrophic network looks like a random network, given a random network we still cannot tell whether it would fail catastrophically or not. And the cascade depends on the first node to congest in the concerning network and the way the load is redistribution afterwards.

4.5 Conclusion

A simple mechanism for building networks designed to fail catastrophically has been presented. The technique could be improved to better build such catastrophic networks. It has the potential to construct networks with nodes that have different service rates where they have low topological importance (or centrality) but are crucial in the catastrophe. In the case where total flow in the network increases linearly with the network size, it is found that the network has a small amount of heterogeneity in its node degree distribution. This shows that catastrophic failure does not only occur in highly heterogeneous networks like the Internet. If all nodes have similar loads and are close to their failure threshold, then cascade failure is also possible in almost homogeneous networks.



(a)



(b)

Figure 4.12: Comparison of the network degree distribution and betweenness centrality of nodes. (a) degree distribution of catastrophic network (left) random network (middle) and scale-free network (right). (b) betweenness centrality of nodes in catastrophic network (left) random network (middle) and scale-free network (right).

To further improve this method, modification can be made that the generated networks have more realistic topologies. There are many ways to extend the method. Other measures of centrality representing different flow mechanisms could be used, and more complicated routing mechanisms other than pure shortest path might be considered. Another possibility is to allow the creation of edges between nodes that do not get congested.

Chapter 5

An Efficient Method to Build Catastrophe: A Family of Catastrophic Networks

5.1 Introduction

In the previous chapter we consider a type of catastrophic network built using the random growing method. It is relatively easy to implement and generate results. However as the catastrophic networks are constructed, there is a drift of the traffic production rate from its maximum value which implies a bound on the size of the network. The random growing method has its limitation for building catastrophic networks. The reliability and efficiency of this model make it difficult and time consuming to generate large-scale topologies. Alternative methods are needed to improve the random growing method. Therefore in this chapter, we present a new method based on the "branch and bound" approach, to speed up the construction of catastrophic networks. We introduced bounds to eliminate unsuitable solutions and introduced a better search algorithm for catastrophic networks. It is shown that using the optimized "branch and bound" method, we can generate a special family of catastrophic networks with two types of nodes and predictable degree property that can be seen as network invariant.

5.2 Network Segmentation

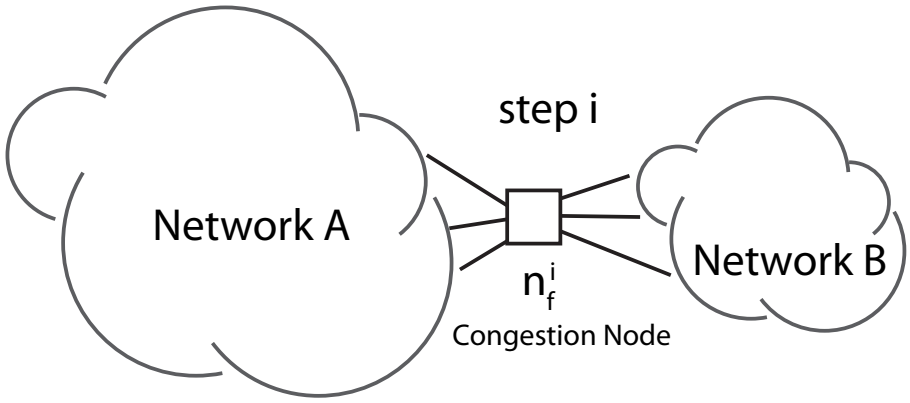


Figure 5.1: Segmentation of the network in each growing step.

Recall the network segmentation introduced in the last chapter that we divide the network into three parts, Figure 5.1. They are: the network A from last step, n_f^i the node that is designed to congest first at this step, and the subnetwork B that is added. At the ith–step of the reverse–avalanche process, when the node n_f^i congests (Equation (4.4)), we have

$$C_B(n_f^i) = \mu_{n_f^i}(N_A - 1)/\Lambda^*. \tag{5.1}$$

In this case, node n_f^i overloads if the average packet–production rate is $\Lambda_{n_f^i} = \Lambda^*$. However, for the given Λ^* we may not able to find a network with n_f^{i+1} at the next step. Nevertheless we may find that node n_f^{i+1} would fail when

$$\Lambda_{n_f^{i+1}} \geq \Lambda_{n_f^i}. \tag{5.2}$$

This is the situation we encountered when building the catastrophic network using the random growing method.

Assume that the average packet load for all the nodes is $\Lambda_{n_f^{i+1}}$, then it will guarantee that at the ith and $(i+1)th$ step both node n_f^i and n_f^{i+1} will overload. In other words, to construct the avalanche in reverse requires an increase on the

average packet load. This observation with Equation (5.1) and Equation (5.2) implies that

$$0 \leq C_B(n_f^{i+1}) - \frac{\mu_{n_f^{i+1}}(N_A + N_B)}{\mu_{n_f^i}(N_A - 1)} C_B(n_f^i) \leq \epsilon. \tag{5.3}$$

where $\epsilon \geq 0$ is a small number and the size of the network at $(i+1)th$ step is $(N_A + N_B + 1)$. In this chapter, we try to find the solution that minimizes ϵ at each step of the inverse–cascade process.

This leads us to search for possible boundaries and conditions that could eliminate unsuitable topology solutions which will reduce the searching space for catastrophic networks. Using a similar process as introduced in previous chapter, the catastrophic network is also built by following the avalanche in reverse. First by dividing the network into three subnetworks, the betweenness centrality of node n_f^i can be split into different contributions from these subnetworks because of the addictive property of the betweenness centrality.

- N_A is the number of shortest paths that start from n_f^{i+1} and end in A
- N_B is the number of shortest paths that start from n_f^{i+1} and end in B
- $2N_AN_B$ is the number of shortest paths that start in A and end in B which correspond to N_A sources going to N_B destinations (and vice versa).
- $C_B(A)$ is the number of paths that start and end in A and they go through node n_f^{i+1}
- Similarly, $C_B(B)$ is the number of paths that start and end in B and they go through node n_f^{i+1}.

Then the betweenness centrality of n_f^i can be written as

$$C_B(n_f^{i+1}) = C_B(A) + C_B(B) + N_A + N_B + 2N_AN_B, \tag{5.4}$$

To obtain the desired $C_B(n_f^{i+1})$, which satisfies Equation (5.3) and minimizes ϵ, one can either change the linkage between subnetwork A and n_f^{i+1} or change the topology of B and its linkage to n_f^{i+1}. Because N_A is the size of the network from previous growing step, it will not change in the corresponding growing step.

Therefore there are three variables left in Equation (5.4) associated with these changes: $C_B(A)$, $C_B(B)$, and N_B. To speed up the search for the new catastrophic network topology that meet the conditions set, it is more efficient to find out the bounds for the solution space. With our improvement to set the boundary and search conditions, at each growing step, the program does not have to examine every possible topology. In the next few sections, we are going to investigate the possible boundaries and conditions for the three variables in $C_B(n_f^{i+1})$.

5.3 Branch and Bound Method: Bound for Subnetwork B

When adding the subnetwork B during the random growing process, we start with a network topology with one node, then two nodes, and so on. If we want to reduce the time searching for possible topologies for subnetwork B, the obvious solution is to find an upper limit for the size of subnetwork B. In this section, we are going to set a bound for N_B, the number of nodes in subnetwork B.

Firstly suppose there is no subnetwork A connecting to n_f^{i+1}, see Figure 5.2. In this separated network, we try to investigate how the topology of subnetwork B would affect $C_B(n_f^{i+1})$. Because there is no subnetwork A, $C_B(A) = 0$. Subnetwork A does not contribute to $C_B(n_f^{i+1})$, therefore there are no shortest paths that start and end in A past through n_f^{i+1}. So if subnetwork B and n_f^{i+1} forms a star-shape network with n_f^{i+1} as its center, $C_B(n_f^{i+1})$ would reach its maximum value $C_B(B_{max}) = N_B(N_B - 1)$, because all nodes need to use the center node to get to each other. Using Equations (5.3) and (5.4), we have

$$N_B^2\alpha + N_B(2N_A\alpha - C_B(n_f)\mu_{n_f^{i+1}}) + N_A(\alpha - C_B(n_f)\mu_{n_f^{i+1}}) \leq \epsilon. \tag{5.5}$$

where $\alpha = (N_A - 1)\mu_{n_f^{i+1}} > 0$. Therefore we can calculate the maximum N_B. Furthermore, the minimum value of $C_B(B)$ can be obtained when subnetwork B with n_f^{i+1} is a fully connected network, $C_B(B_{min}) = N_B - 1$.

Although Equation (5.5) is not always easy to solve, from our numerical experiment, we noted that $\max(N_B) \leq 4$ when we are using small size seed networks $N \leq 15$. Hence, we evaluated $C_B(B)$ for all possible network topologies with

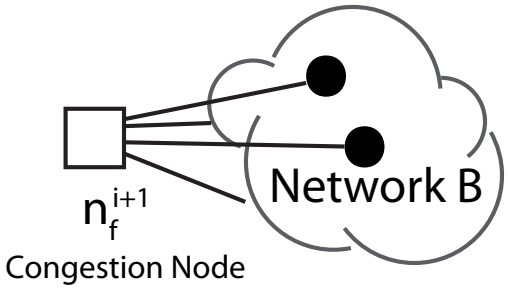


Figure 5.2: Subnetwork$B$ and congestion node n_f^{i+1}.

size $0 \leq N \leq \max(N_B)$ and created a database (lookup table) for all possible subnetworks topologies. The catastrophic network generator can query this table and efficiently evaluate the contribution of $C_B(B)$ to $C_B(n_f^{i+1})$. The detail of this lookup table is described below.

5.3.1 Graph Enumeration

To build the lookup table including all possible subnetworks B and sort them in an order easy for query, we need to consider the graph enumeration calculation. When constructing this table we focus on all possible subnetworks with 0 to 4 nodes. This is similar to graph enumeration in graph theory [7; 50]. Suppose a graph G consists of a finite non-empty set of nodes N, together with a specific set of unordered pairs of distinct links where there are no loops and multiple links. In this case, a network of four nodes will have eleven possible unlabeled topologies, as illustrated in Figure 5.3. As the number of nodes in the network grows, the possible unlabeled topologies increase rapidly, see table 5.1 below [7]. We find out that look up table for larger size subnetwork is really hard to generate and further optimization on graph numeration program is needed.

Furthermore, we also need to label the network topologies for subnetwork B. This is because in a labeled network, each node is distinctive. By placing the congestion node n_f^{i+1} at different position in the same network topology, the

Number of Nodes	4	5	6	7	8
Number of Possible Networks	11	34	156	1044	12346

Table 5.1: Possible number of unlabeled networks

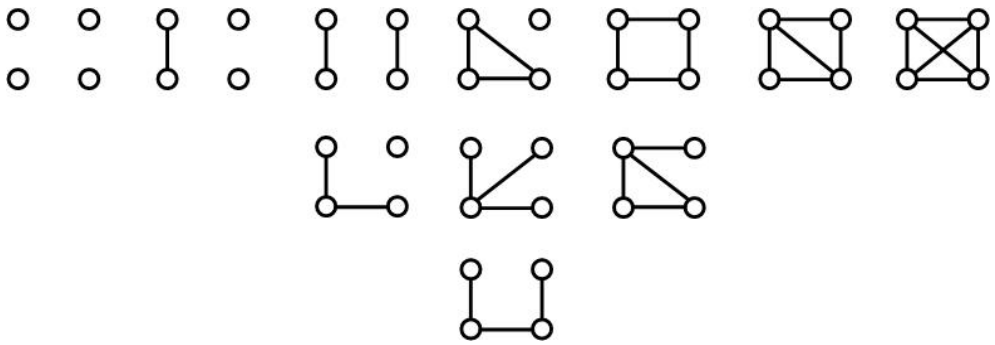


Figure 5.3: Eleven different topologies of network with four nodes.

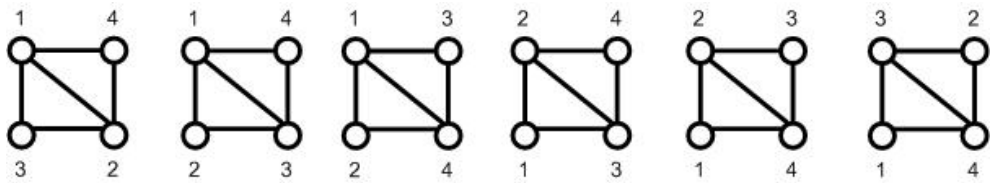


Figure 5.4: Six different labeling of a graph with four nodes and five links [7].

$C_B(n_f^{i+1})$ will be different. For example in Figure 5.4, there are six different ways to label a graph with four nodes and five links. In other words, for a combination of four nodes and five links one can build six different networks. At each catastrophic network growing step, we are more interested in the subnetwork B's betweenness centrality contribution to n_f^{i+1}. Network topologies with 4 nodes might have a higher contribution to the betweenness centrality of n_f^i, i.e. the star–shape topology, while some network topologies have little contribution to the $C_B(n_f^i)$, i.e. a fully connected graph. So we sorted network topologies first by their size (number of nodes) and then by their betweenness centrality contribution to n_f^i. When a certain betweenness centrality is needed during the catastrophic network growing process, potential subnetworks at the same size with the equal betweenness centrality contribution to n_f^i are selected randomly from the topology table. The random selection of equal betweenness contribution topologies makes sure that we do not end up with the same topology for different growing processes. In Figures 5.5, 5.6 and 5.7, we show all possible topologies of the subnetwork B with n_f^i by arranging them with different size and number of links in the network, which is the unsorted input data for the lookup table.

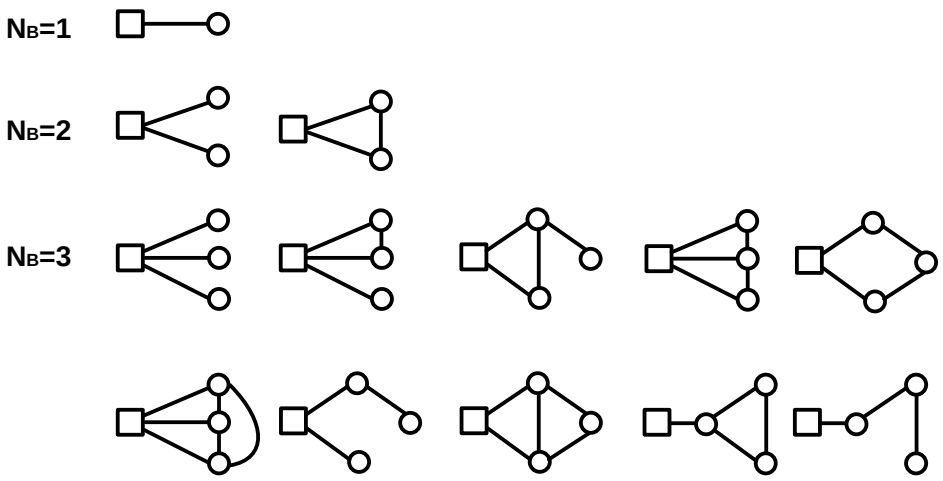


Figure 5.5: Network B with 1, 2 and 3 nodes.

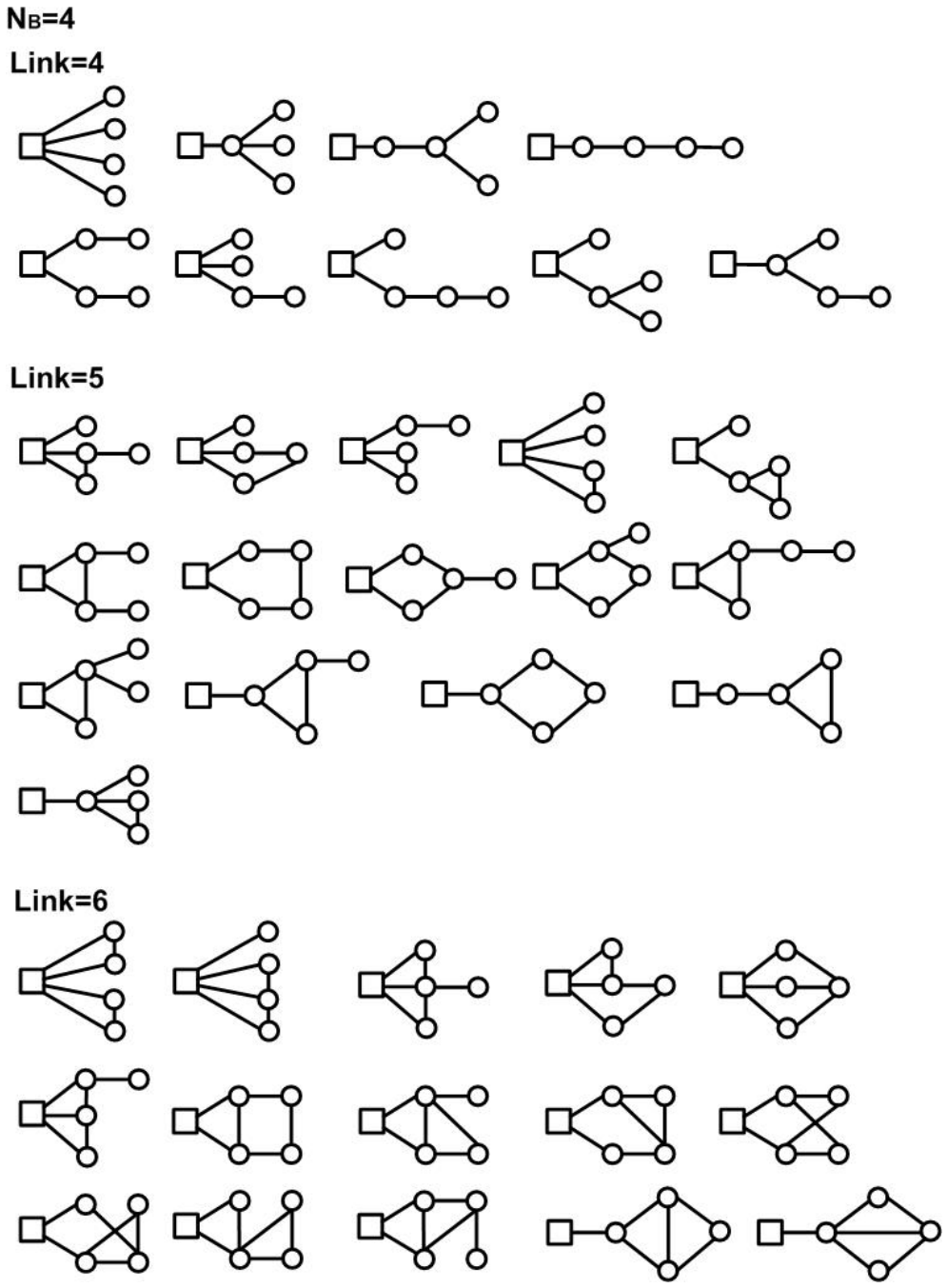


Figure 5.6: Network B with 4 nodes and 4 to 6 links.

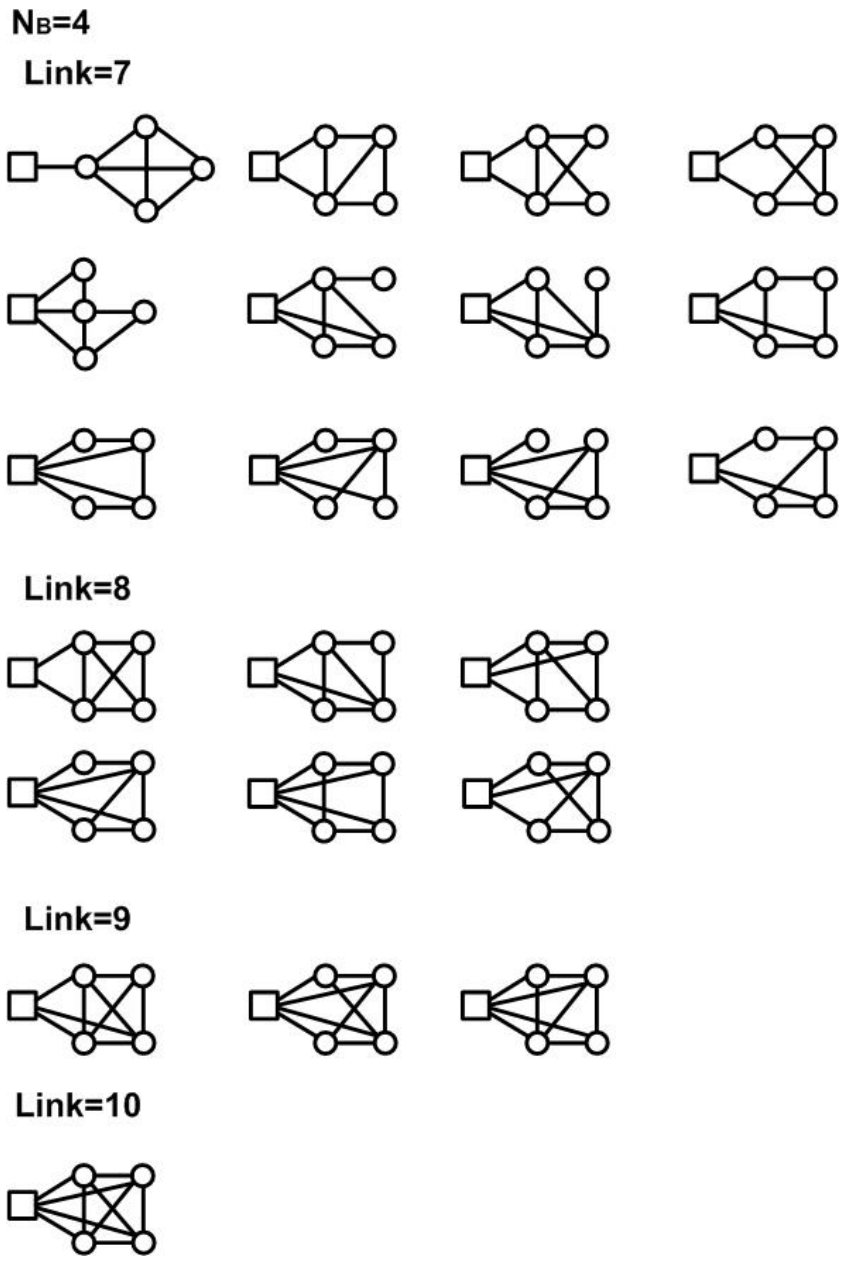


Figure 5.7: Network B with 4 nodes and 7 to 10 links.

5.4 Branch and Bound Method: Bound for Subnetwork A

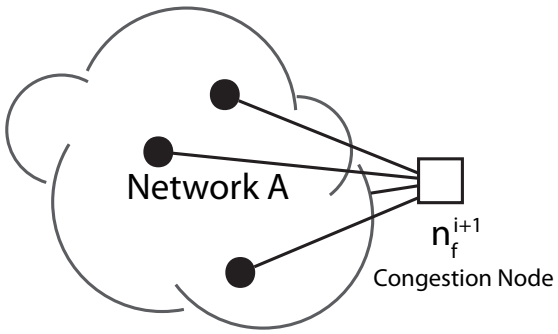


Figure 5.8: Subnetwork$A$ and congestion node n_f^{i+1}.

In this section we are going to examine how the connection between subnetwork A and n_f^{i+1} would affect $C_B(n_f^{i+1})$, see Figure 5.8. Suppose that sub-network B is not connected to n_f^{i+1}. This means that $C_B(B) = 0$ and $N_B = 0$ in this case. The betweenness centrality of n_f^{i+1} can be rewritten as

$$C_B(n_f^{i+1}) = C_B(A) + N_A, \tag{5.6}$$

leaving $C_B(A)$ the only variable. The only way to modify $C_B(A)$ is to add and delete links between congestion node n_f^i and subnetwork A. Suppose there are two nodes $n_A(s)$ and $n_A(d)$ in subnetwork A, there is no direct link between them, but they are both connected to n_f^i. Thus the shortest paths between them have to include the path "$n_A(s) - n_f^{i+1} - n_A(d)$". So that n_f^i is on one of the shortest-paths between these two nodes. Suppose to the extreme condition, all nodes in subnetwork A have a link connecting to congestion node n_f^{i+1}. Then for the node pairs in A that are not directly connected, one of their shortest paths should go pass n_f^{i+1}. Then we can obtain $C_B(A)_{max}$. Notice that in this extreme situation, removing connections between subnetwork A and n_f^{i+1} could only decrease the value of $C_B(A)$. When there is only one link between subnetwork A and n_f^{i+1}, we can have $C_B(A)_{min}$. Because we are targeting n_f^{i+1} as the node with highest

betweenness in the network, it is reasonable to initially connect all subnetwork A nodes to n_f^{i+1} then remove the links to find the solutions. In this way, we will get the right connection between A and n_f^{i+1} plus B that meets the growing requirement faster.

5.5 Branch and Bound Method: Conditions

Building catastrophic networks using the "branch and bound" method, we also need to meet two requirements set in the previous chapter.

1. The traffic production rate for new node n_f^{i+1} to congest should be higher than but close to that for previous step node n_f^i. It is $\Lambda_{i+1}^* - \Lambda_i^* \geq \epsilon$, where $\epsilon \to 0$.

2. Furthermore the congestion node n_f^{i+1} should be the first node to congest in the catastrophic network. It means n_f^{i+1} should have the small congestion (critical) load in the catastrophic network of this step. So n_f^{i+1}'s congestion load would at least be smaller than that of node with the highest congestion load λ_{max}: $\lambda_{i+1}^* \leq \lambda_{max}$.

From last section we can estimate the betweenness centrality of nodes in subnetwork B using the lookup table built. But the betweenness centrality of nodes in subnetwork A is hard to calculate and depends on the connectivities of the whole network. To meet the requirements, how can we compare the betweenness centrality value of n_f^{i+1} to all other nodes quickly without running the time and computer resource consuming betweenness centrality calculation program?

From observation and investigation, the topological structure of subnetwork B would not affect the shortest-paths from n_f^{i+1} to nodes in network A, because n_f^{i+1} acts as a bridge between subnetwork A and B. Here we try to work out fast way to calculate the betweenness centrality of nodes in A.

- In subnetwork A, the betweenness centralities of nodes are $C_B(A_i^0)$, where $i = 1..., N_A$.

- By connecting n_f^{i+1} to A with links, the betweenness centralities of nodes in A changes to $C_B(A_i^1)$.

- Suppose there is only one node B_0 in subnetwork B and it is connected to n_f^{i+1} through one link. The shortest path between B_0 and A_i must pass through n_f^{i+1}. If there is x shortest paths from n_f^{i+1} to A_0, then there will be x shortest paths from B_0 to A_0.

From numerical experiments we found out that the addition of nodes and connections in subnetwork B increases the betweenness centrality of nodes $C_B(A_i)$ in subnetwork A linearly as

$$C_B(A_i^1) = C_B(A_i^0) + \alpha_i N_B, \tag{5.7}$$

where α is the linear increase rate. This linear increase rate α could be calculated numerically and dynamically during each step of the catastrophic network growing process. Consider there is only one node in subnetwork B ($N_B = 1$), α can be calculated as:

$$\alpha_i = C_B(A_i^1) - C_B(A_i^0). \tag{5.8}$$

Hence α can be obtained at the very beginning of each growing step when n_f^{i+1} is added and linked to subnetwork A. Then $C_B(A_i)$ can be calculated efficiently when the topology of subnetwork B is changed.

5.6 Branch and Bound Method: Branch Search Algorithm

Recall Equation (5.3), we want to minimize the value of ϵ. So we need to evaluate how the different connections between subnetwork A, subnetwork B and n_f^{i+1} would contribute to $C_B(n_f^{i+1})$. If the nodes in A are labeled as $n_1, n_2, ...n_i$, the links of these nodes with n_f^{i+1} can be expressed as a binary sequence

$$L = \{l_1, \ldots, l_{N_A}\} = \{0, 1, \ldots, \}, \tag{5.9}$$

where $l_i = 1$ if node n_i in A is connected with n_f^{i+1}, otherwise $l_i = 0$. Note that the number of different binary sequence combinations grows exponentially at 2^{N_A}, which is another major challenge for building very large catastrophic network.

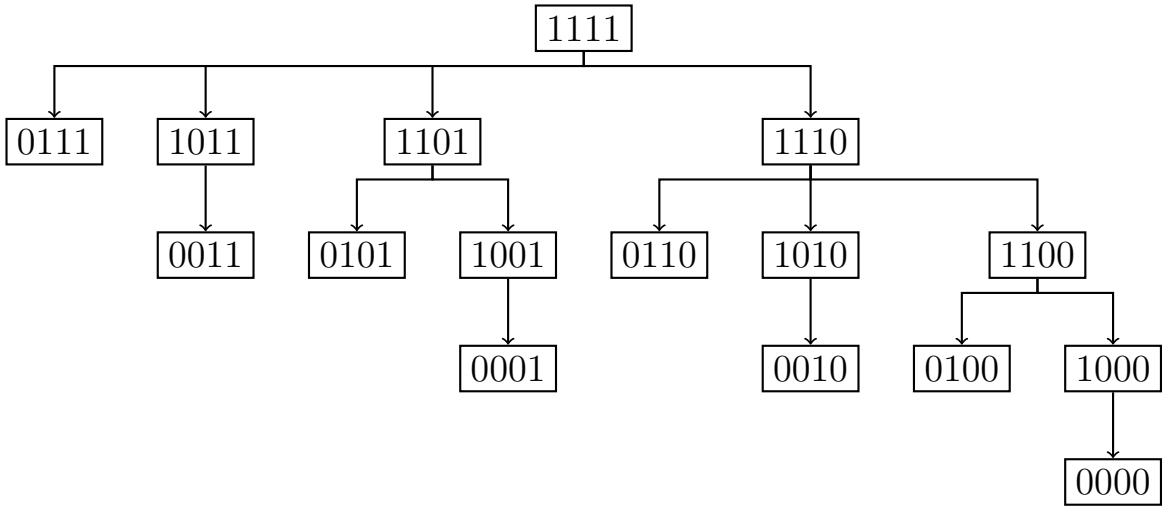


Figure 5.9: Solution space represented as a tree for the case when the network A has four nodes

To search which of the binary sequences with a subnetwork B can better minimize ϵ, we use a branch and bound algorithm based on the following observation. If all the nodes in A are directly connected to n_f^{i+1}, then for any pair of nodes in A which are not directly connected, there will be at least one shortest path between these two nodes that passes through n_f^{i+1}. The length of this path is 2 hops. From last section, this connectivity between A and n_f^{i+1} gives the maximum possible value of $C_B(A)$. Removing any link between A and n_f^{i+1} will reduce only $C_B(A)$. To use this property in the search for the minimal ϵ, the binary sequence L is employed to organise the space of solutions as a rooted tree. The root corresponds to the solution where all nodes of A are linked to n_f^{i+1}. The branches of the tree are explored by removing connections between A and n_f^{i+1}, where the corresponding l_i is set to 0. Figure 5.9 shows an example of the solution space tree for $N_A = 4$ with 16 possible connections. The branch and bound algorithm to solve one step of the inverse cascade is given below.

Algorithm: Inverse–Cascade

1: Start with the case that all the nodes in A are connected to n_f^{i+1} (i.e. $C_B(A)$ is maximum and $\{l_i, \ldots, l_{N_A}\} = \{1, \ldots, 1\}$.

2: Evaluate $C_B(A)$, initialize $k = 1$, $j = 1$
3: Disconnect n_k from n_f^{i+1}, i.e. $l_k = 0$
4: **for all** sub–networks B of size $\leq \max(N_B)$ **do**
5: evaluate $\min(\epsilon)$ using Equations (5.3) and (5.4)
6: **end for**
7: **if** $\min(\epsilon) \geq 0$ **then**
8: **if** $\min(\epsilon) \approx 0$ **then**
9: stop, solution found
10: **end if**
11: **Search Branches:** remove link l_j where $j < k$
12: $j \leftarrow j + 1$
13: goto step 3 unless there are no more branches to select
14: **else**
15: **Cut Branches:** Deleting links between network A and n_f^{i+1} will only decrease $C_B(n_f^{i+1})$. So sub-branches under this branch will not produce high enough values of $C_B(n_f^{i+1})$, which is not necessary to investigate.
16: Connect n_i back to node n_f^{i+1}.
17: $k \leftarrow k + 1$ return to step 2 unless there are no more links to select i.e. $k = N_A$ then stop
18: **end if**

The output of the algorithm is the sequence L and connectivity of the sub–network B. The algorithm is also shown as flow chart in Figure 5.10. This algorithm is similar with the Depth First Search in graph theory using the recursive function calls. But it has advantage that it will consider a escape condition during the search. We find out that by setting the last bit of the binary list to be '0' makes the binary number to be an even number and '1' makes it an odd number. Furthermore, within the set of numbers of N_A bits, half of them are even numbers while the other half are odd numbers. By looking at the binary solution tree again, half of the tree that is going to be searched is down in the branch "11...10" with all even numbers. And the odd numbers are in the rest branches with "xx...x1" and the root of the tree "11...1". We have made a further improvement on the branch and bound algorithm by diverting the search to begin with the odd numbers of the tree structure. Rather than go down the

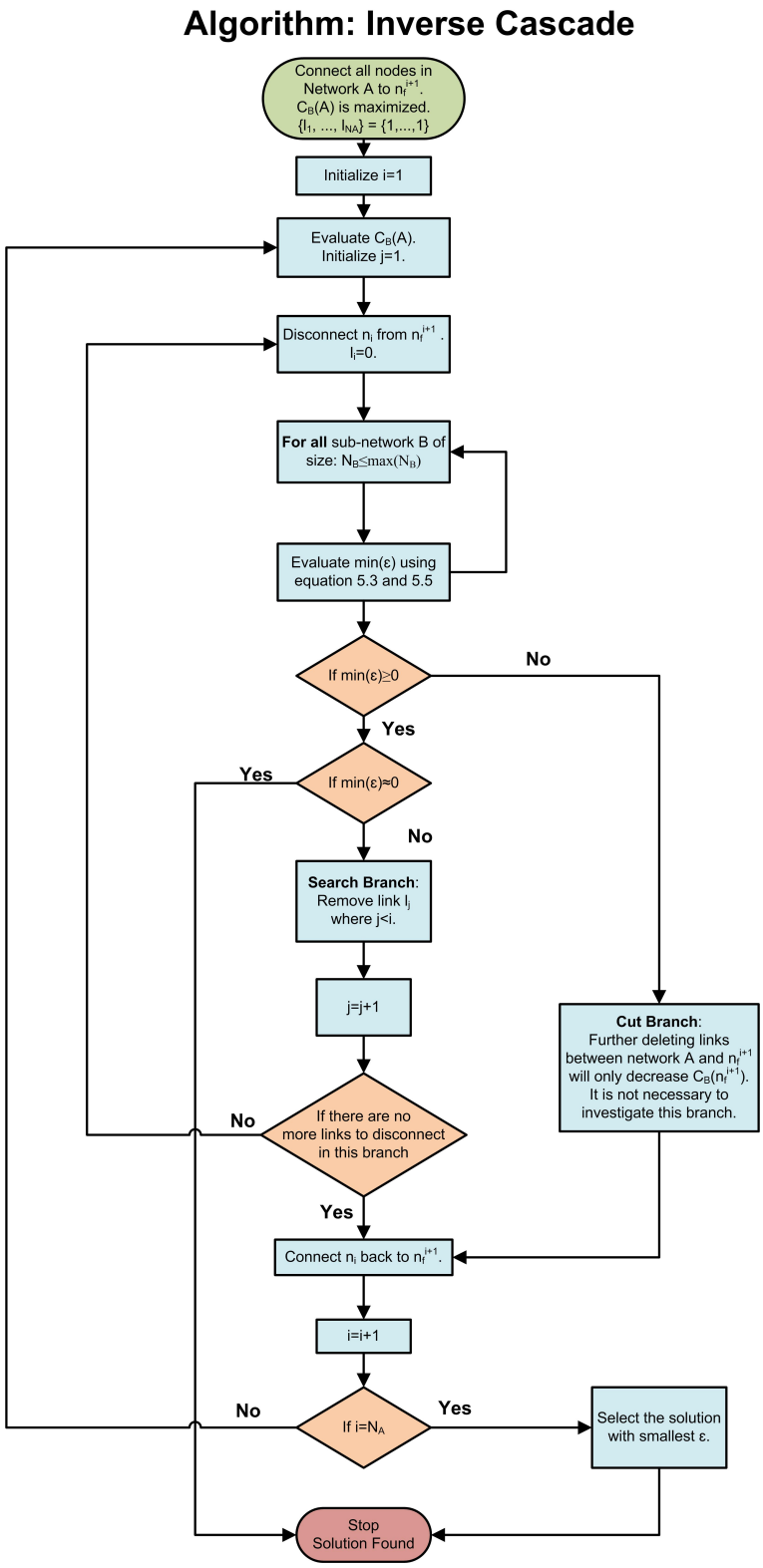


Figure 5.10: Flow chart for the algorithm of the branch and bound method.

even number branch, the searching on the odd branches are much faster to get topologies that meet the requirements set. But also notice that the searches begin with the odd branches will sometimes come up different topology solutions with those starting in the even branch. Finally, we also implement the code to search hopping between the branches, in order to shorten the time needed to come up with the right topology. Although it only shows slight improvement on search time.

5.7 Results and Analysis

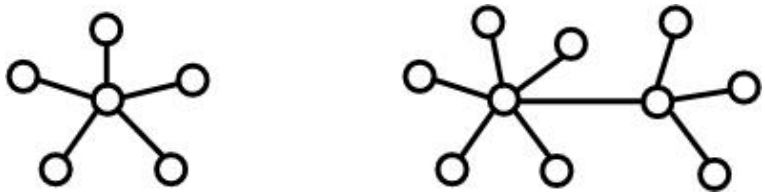
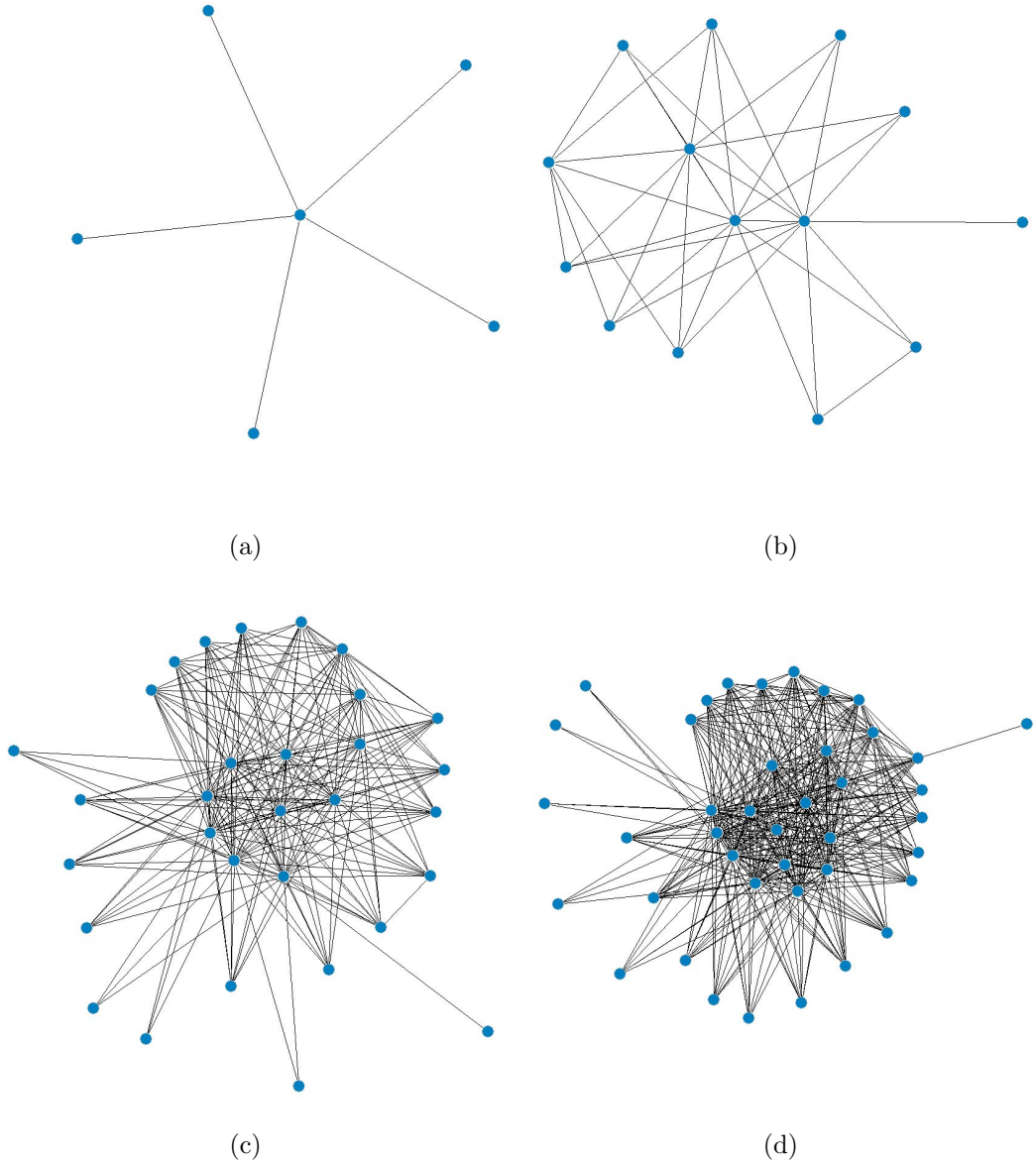


Figure 5.11: Seed network chosen are the one star and two stars shape network.

Two seed networks are used to build the catastrophic network with the improved "branch and bound" method proposed in this chapter, as illustrated in Figure 5.11. We focus on the catastrophic network generated using the six node star-shape network first. The growing steps of this network have been shown previously in Figure 5.12.

Starting with the seed network (Figure 5.12(a)), the "branch and bound" method added the congest node n_f^i and subnetwork B with their connections. Figure 5.12(b) is the third step of the growing with 14 nodes. But at the 11th step and 16th step, the catastrophic networks become very different from the ones generated using the random growing method in Chapter 4. The catastrophic networks generated using branch and bound method have more links. We can identify two layers of nodes when they are plotted in Figure 5.12(c) and (d). There seems to be a core subnetwork in the middle of the network with more connections than the rest of nodes in the network.



(a) (b)

(c) (d)

Figure 5.12: (a) is the seed network to start the growth of catastrophic network. (b), (c) and (d) are the catastrophic networks at the third step of the growth with 14 nodes, 11th step with 31 nodes and 16th step with 41 nodes.

To further investigate the property of this family of catastrophic networks, we again examined the betweenness centrality of n_f^i for each growing step for the 41 node catastrophic network. Strictly following the conditions set by the "branch and bound" method, the congestion traffic generation rates at each growing step are very close to each other, where we control $\epsilon \leq 0.001$. So the betweenness centralities of congested node n_f^i of each growing step follow the upper boundary closely, such that $\Lambda_{n_f^{i+1}} \geq \Lambda_{n_f^i}$ is satisfied. We do not plot the $C_B(n_f^i)$ of each step again here because it almost overlaps with the upper bound. This gives us a way to construct a network that the traffic generation rate to congest the network at every step of the cascade is almost the same.

As mentioned, by looking at the plotting of this type of catastrophic networks, one can see the special structural features. Figure 5.13 shows the node degree distribution and the cumulative degree distribution of two catastrophic networks built using branch and bound method. The degree distribution shows that there are a large portion of nodes with degree 15 and 16 and the number of nodes with other degrees are almost the same. It is different from the networks built using the random growing method whose degree distribution are similar to random networks. Considering the cumulative degree distribution, they are also different from those of catastrophic network built using random growing method. Hence we take a closer look at these properties.

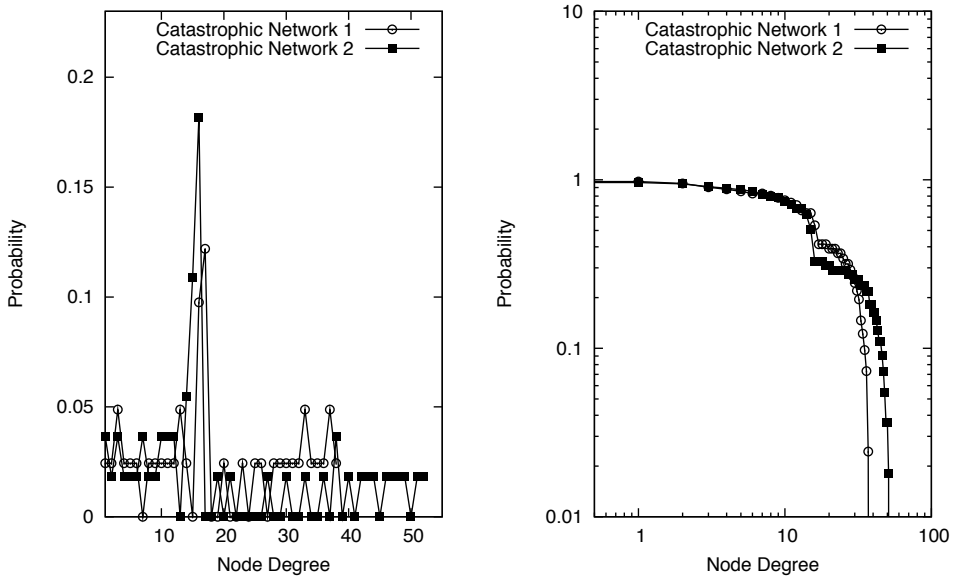
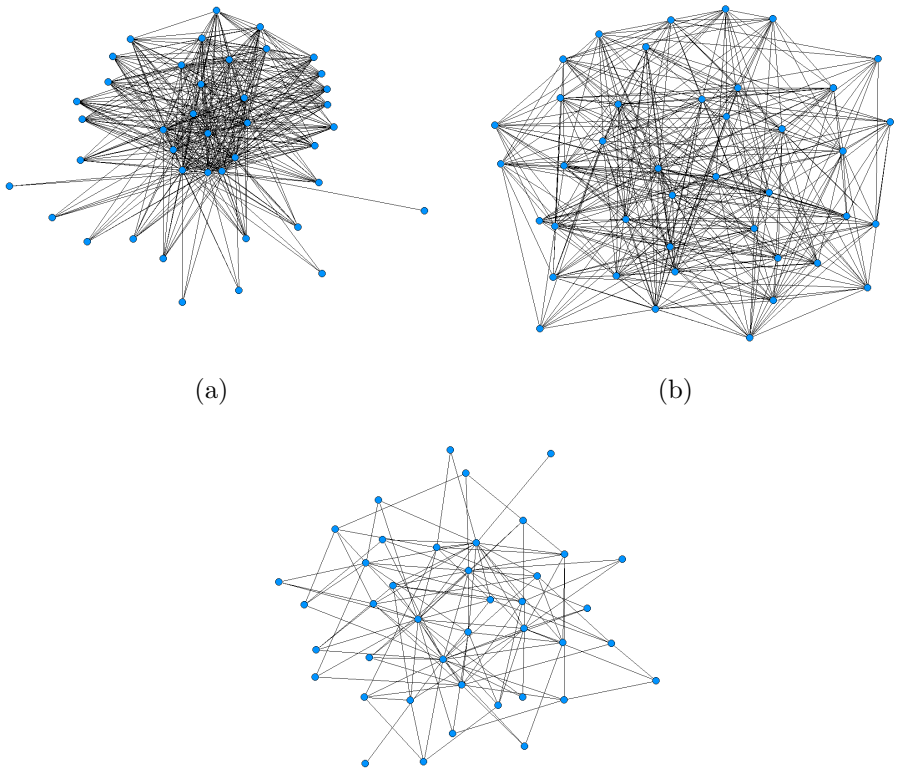


Figure 5.13: Degree distribution and cumulative degree distribution of 41-node catastrophic network (catastrophic network 1: generated using star shape seed network) and 55-node catastrophic network (catastrophic network 2: generated using two-stars shape seed network).

5.7.1 Comparison with Random and Scale-free Networks

The catastrophic networks generated using the "branch and bound" method introduced a special structural feature during the growing process. To find out the characteristics of this special family of catastrophic networks, we choose two of these networks: a 41 node network built using a 6 node star-shape seed network and a 55 node network built using a 10 node two-stars shape seed network. Then we generate ER random networks and BA scale-free networks using our topology generator with the same number of nodes, see Figure 5.14 and Figure 5.15. By looking at the plot, this type of catastrophic network is neither similar to a random network nor a scale-free network. Instead, they show two types of nodes:

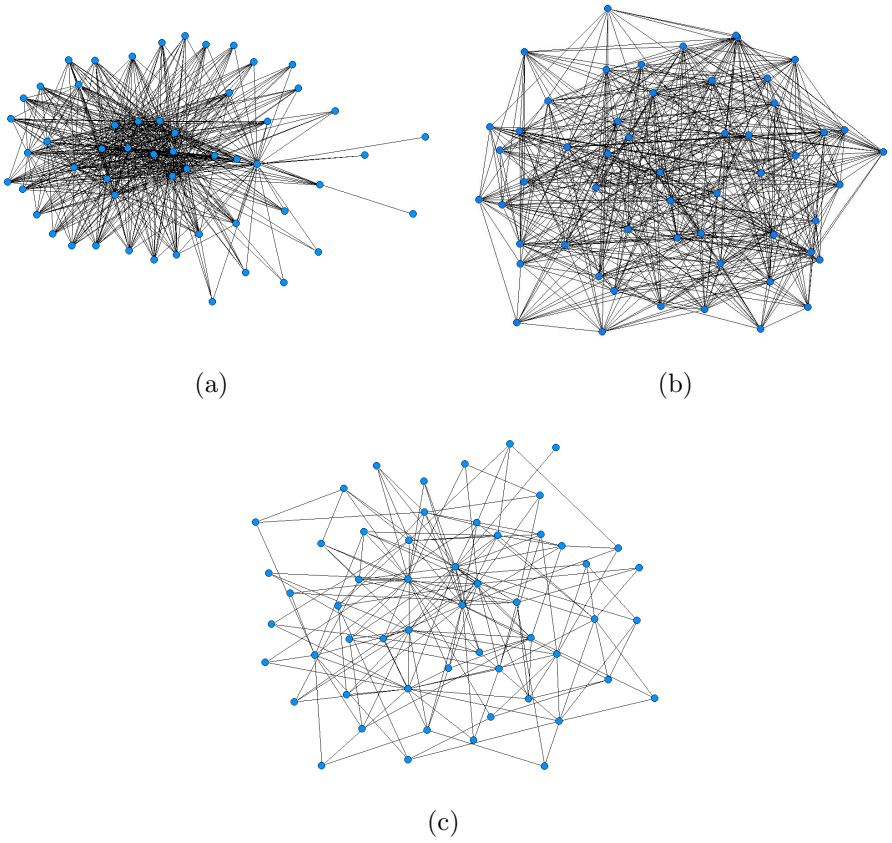
- Highly connected nodes that positioned in the center cluster of the network in Figure 5.14(a) and Figure 5.15(a).
- Relatively poorly connected nodes positioned in the outer layer in the figures.



(a) (b)

(c)

Figure 5.14: (a) 41-node catastrophic network generated using star shape seed network, (b) 41-node ER random network and (c) 41-node BA scale-free network.



(a) (b)

(c)

Figure 5.15: (a) 55-node catastrophic network generated using two-star shape seed network, (b) 55-node ER random network and (c) 55-node BA scale-free network.

We then analyse the topological properties of these networks and compare them in Table 5.2 and Table 5.3. These networks have 41 nodes in Table 5.2 and 55 nodes in Table 5.3. This type of catastrophic networks still has similar number of links, average degree, and average shortest path length with random networks. Besides their difference in maximum betweenness centrality value, they also have very different maximum node degree. This type of catastrophic network even has higher maximum node degree than scale-free network, but still don't have the same level of maximum betweenness centrality value. Furthermore, the catastrophic networks have almost three times the number of links, average degree and two times the maximum degree to those of the scale-free networks.

Network Type	N	L	k	k_{max}	$C_B(max)$	ℓ
Catastrophic Network	41	395	9.6	38	200	1.5
Random Network	41	362	8.8	25	90	1.6
Scale-free Network	41	130	3.1	22	370	2.0

Table 5.2: Compare 41-node Networks' Properties

Network Type	N	L	$\overline{k}$	k_{max}	$C_B(max)$	$\bar{\ell}$
Catastrophic Network	55	559	10.2	52	412	1.6
Random Network	55	542	9.9	28	136	1.6
Scale-free Network	55	183	3.3	22	497	2.2

Table 5.3: Compare 55-node Networks' Properties

Figures 5.16 and 5.17 show the node degree distribution and the betweenness centrality of nodes against their sequence number when they were added to network. Again we could see the degree distributions of catastrophic networks have a significant peak which is different from both random and scale-free networks. The betweenness centrality of the catastrophic networks show a special feature: there is a group of nodes with the same betweenness centrality level and another group of nodes with betweenness centrality increase rapidly with the node sequence number. It matches what we see from the plots of this type of network with two types of nodes. We investigate this special feature in the next section.

But it is important to point out here that we are investigating this specific family of catastrophic networks and compare its topological properties with random and scale-free networks. Although these networks have different properties as random and scale-free networks, there may be random and scale-free networks that would fail catastrophically and further work is needed.

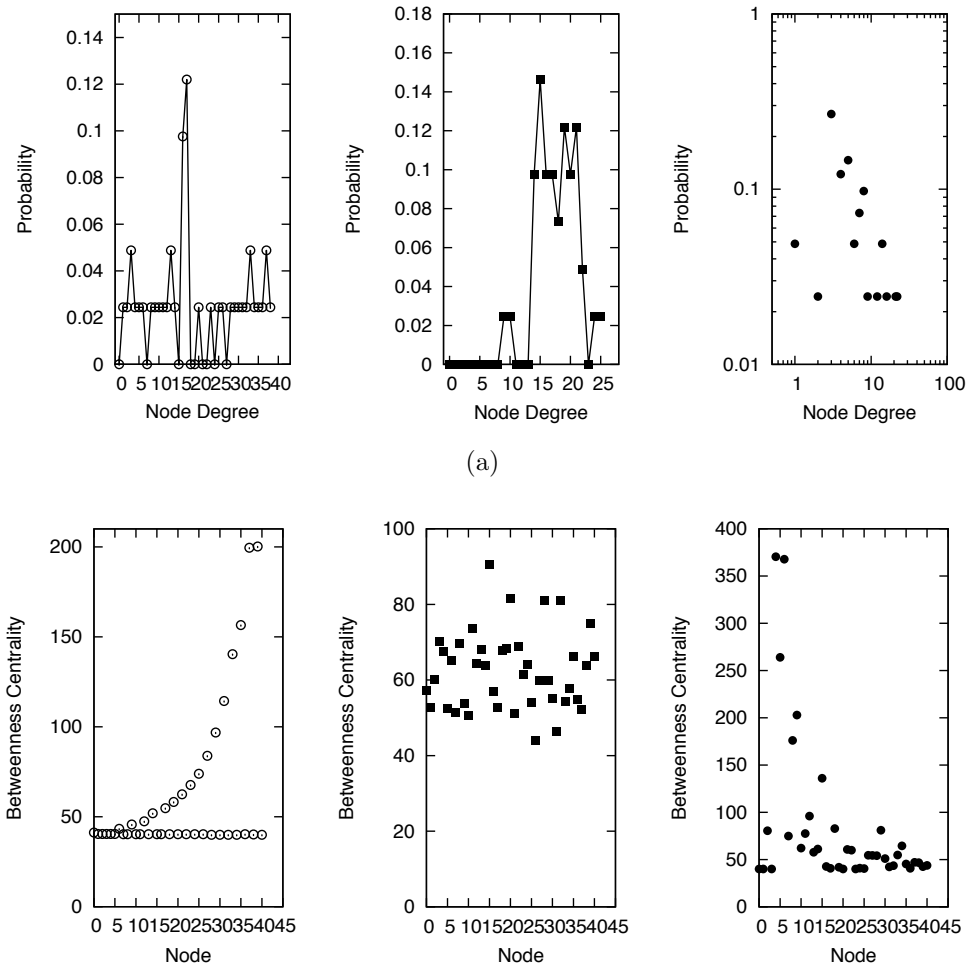
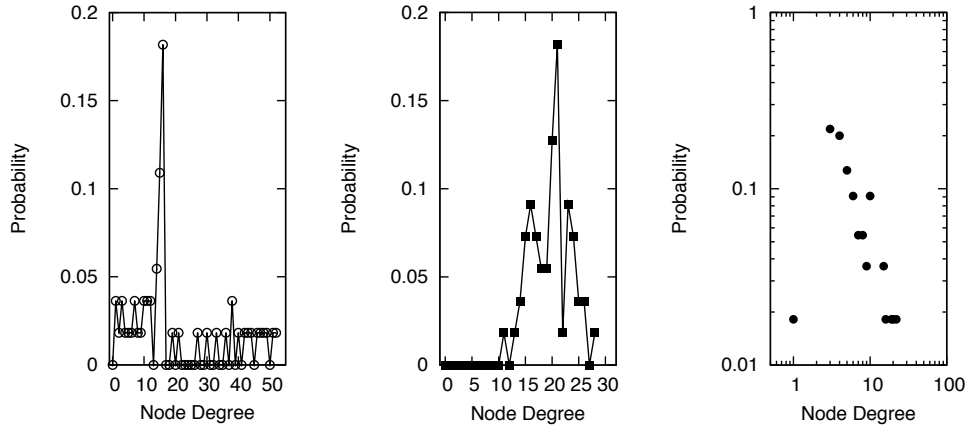
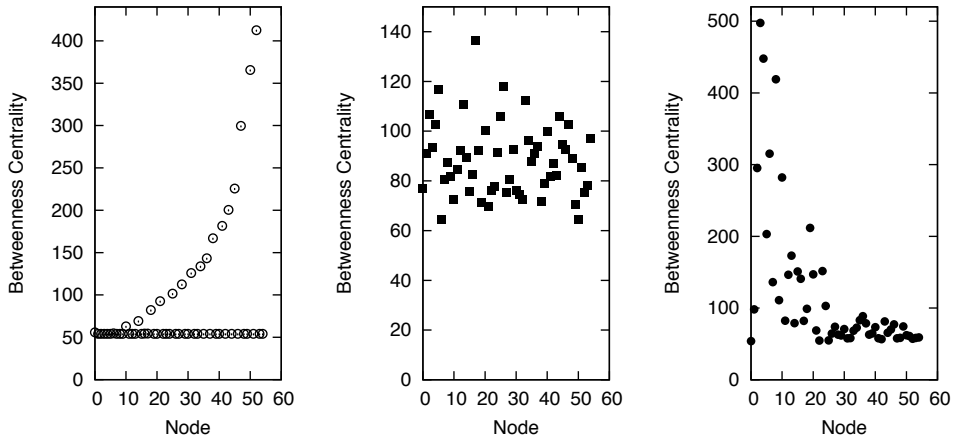


Figure 5.16: (a) Compare the degree distribution of 41-node catastrophic network (left) generated from star shape seed network with same size random network (middle) and BA scale-free network (right). (b) Compare the networks' betweenness centrality of 41-node catastrophic network (left) with same size random network (middle) and BA scale-free network (right).



(a)



(b)

Figure 5.17: (a) Compare the degree distribution of 55-node catastrophic network (left) generated from two-star seed network with same size random network (middle) and BA scale-free network (right). (b) Compare the networks' betweenness centrality of 55-node catastrophic network (left) with same size random network (middle) and BA scale-free network (right).

5.8 Discussion

5.8.1 Seed Network Constraint

From Chapter 4, we have $\Lambda^* \geq \frac{\mu_w(N-1)}{C_B(w)}$, the connectivity of the seed network defines the traffic production rate Λ_0^* when congestion happens in it. When building the catastrophic networks, the target is to construct a network that at each step of the avalanche, the new congestion load changes as little as possible. That is $\Lambda_{n_f^{i+1}} \geq \Lambda_{n_f^i} \geq ...\Lambda_0^*$. However there is not always a solution. When building the catastrophic networks, it is possible that the congestion only occurs at loads larger than Λ_0^*. This opens the question if it is possible to build a catastrophic network at any given Λ^*. Figure 5.18 shows the change of Λ^* as the catastrophic network is built from a four-node and a five-node star-shape seed network. For the four-node star-shape network, the initial congestion load is $\Lambda_0^* = 0.25$. The figure shows that to sustain the avalanche as the network grows, the congestion load has to be increased. However for the five-node star-shape seed network with $\Lambda_0^* = 0.2$, a catastrophic network can be constructed that the catastrophic network at each step fails at the same congestion load.

In general, we noticed that if the congestion load of the seed network is relatively "high", i.e. $\lambda^* \geq 0.25$, then the congestion load has to be increased to build a catastrophic network.

5.8.1.1 Invariant

For those networks that fail at each step at the same load Λ^*, we observe that there is a special family of catastrophic networks whose connections follow a pattern. Figure 5.19 shows (a) the betweenness centrality and (b) the degree distribution of a network at two different steps of its growth. By construction the nodes are split into two groups, the nodes that will congest and trigger the avalanche and the rest of the nodes in the network. We noticed that this distinction between the nodes is also reflected in the betweenness centrality and the degree distribution of the nodes.

Figure (5.19) (b) shows that the degree k of congestion (failure) and non-failure nodes increase linearly with network size. But the former increases with

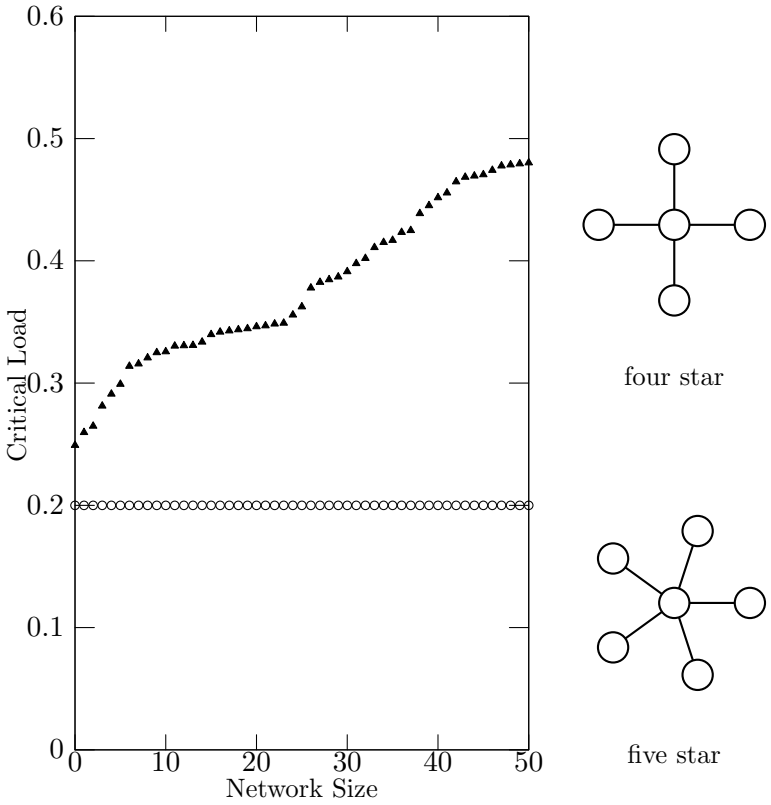


Figure 5.18: The critical load as the network size increases for two different seed networks.

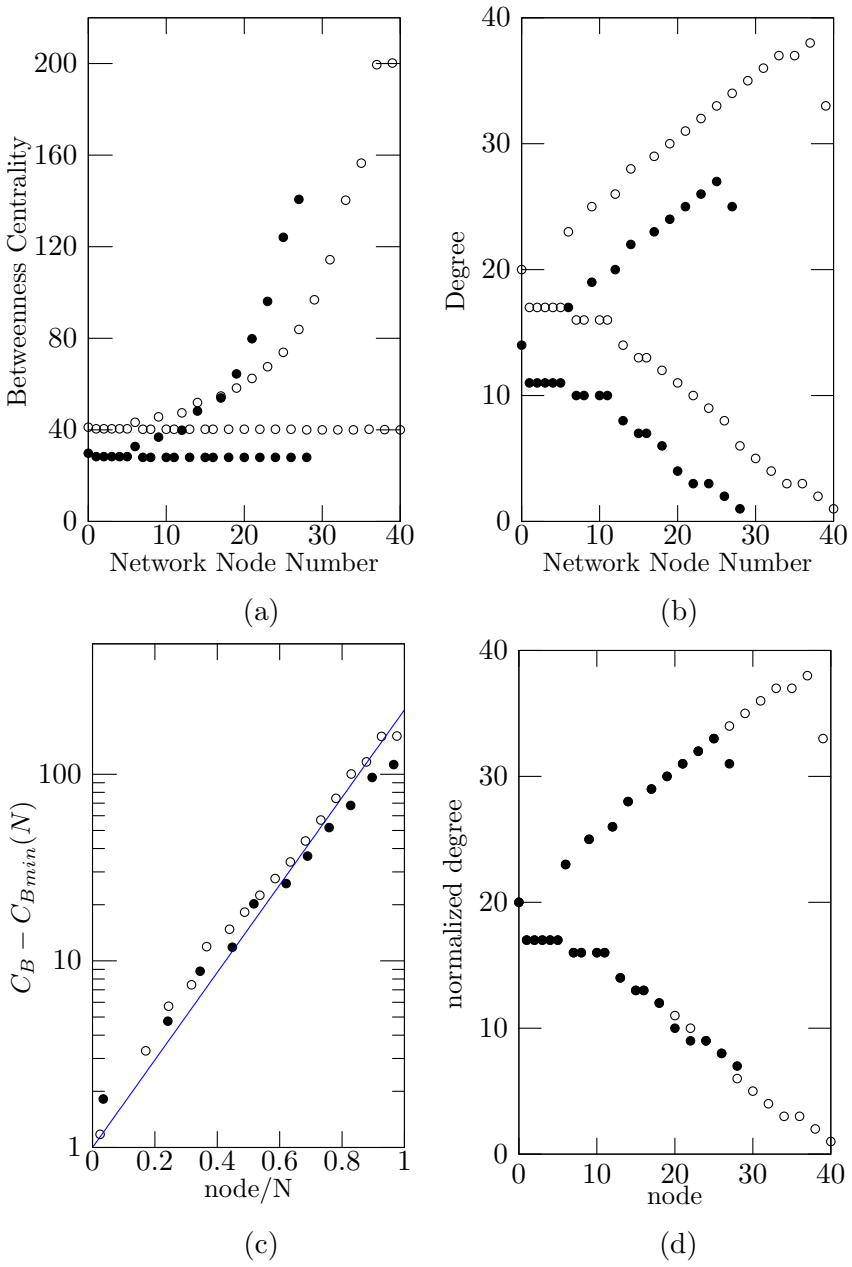


Figure 5.19: The betweenness centrality (a) and the degree (b) of the nodes of the catastrophic network as the size of the network increases. The black dot represent a smaller size catastrophic network with 29 nodes and the circle represent a bigger size network with 41 nodes. And the normalisation of betweenness centralities (c) and degree (d) independent of the network size.

the node sequence number (the sequence the node is added to the network) and the latter decreases. These two trends follow the lines

$$k = \alpha \pm \beta n, \tag{5.10}$$

where α and β are constants here, n is the number that labels the nodes (the sequence number). Using different seed networks and checking this linear behaviour at different stages of the avalanche we obtained that

$$\alpha \approx \gamma n \tag{5.11}$$

and

$$\beta \approx 0.5, \tag{5.12}$$

where γ is a constant. In Figure 5.19(c) and (d) the data sets with catastrophic networks at different growing steps are re-scaled, in order to show how the betweenness centrality and degree of the nodes follow an invariant as the network grows. This observation hints that it is possible to construct very large catastrophic network by only specifying the degree and betweenness centrality. Notice that this result covers only one family of potential catastrophic networks. It is possible to create a catastrophic network that it is not evident how the failure/non-failure nodes are related to each other.

5.8.1.2 Asymptotic Problem

In Figure 5.19 (a)-(b), the nodes with the highest value of degree or centrality correspond to the congestion nodes of each growing steps n_f^i. We have also found that the centrality of these nodes tends to increase with the network size as:

$$C_B(n_f^i) \approx \exp(n_f^i/A(N)) + C_B(min), \tag{5.13}$$

where N is the number of nodes in the network, $A(N)$ is an increasing function of N and $C_B(min) = N - 1$ which is the minimum betweenness that a node can have (see Figure 5.19(c)). From these constructed catastrophic networks , we are not able to determine if $A(N)$ tends to be a constant or grows linearly as the size

of the network N increases. From Equation $\lambda(n_f^i) = \Lambda N \bar{\ell} \hat{C}_B(n_f^i) = \frac{\Lambda C_B(n_f^i)}{N-1}$, and if the betweenness grows exponentially, then

$$\lambda(n_f^i) = \Lambda \frac{\exp(n_f^i / A(N)) + N - 1}{N-1} \geq \mu(n_f^i). \tag{5.14}$$

If $A(N)$ tends to a constant for large N, then the arrival rate at a node is

$$\lambda(n_f^i) \to \infty. \tag{5.15}$$

It implies that after a certain size the networks will fail catastrophically. If $A(N)$ grows linearly with N,

$$A(N) \approx a + bN \tag{5.16}$$

then

$$\lambda(n_f^i) \to \Lambda \geq \mu(n_f^i), \tag{5.17}$$

as $N \to \infty$ If $\mu(n_f^i) = 1$, then the catastrophe will occur if $\Lambda = 1$ and the network is fully connected. If $\mu(n_f^i) \geq 1$ it is still an open question if catastrophic networks exist for very large N.

5.9 Conclusion

In this chapter we introduce a new approach to study the problem of cascade failure in networks. Instead of taking a given network and test if and how it will fail catastrophically, the catastrophic networks are constructed.

We first re-consider the network segmentation and study the betweenness centrality contributions to the congestion node n_f^i. This gives us the opportunity to find bounds for the solution space and eliminate unwanted network topologies. The first bound is the maximum number of node of subnetwork B, where a lookup table of all possible networks with different betweenness centrality contribution to n_f^i is sorted and built. The other bound for connection between subnetwork A and congestion node n_f^i and the conditions introduced inspire the creation of the "branch and bound" method.

Analysis of the catastrophic networks generated using this method shows that they are neither random nor scale-free, but with their own properties. The congestion nodes that are designed to fail at each growing step have different degree and betweenness centrality value from other nodes in the network. The results also show that we have control over the critical load that produces the cascade as well as the number of steps that the cascade has. The invariant discussed also indicates these networks can be very large and have a predetermined degree distribution. We believed that extending this approach will provide further important insights into how topology and traffic flow are linked to catastrophic failure.

Chapter 6

Network Simulation

While we are celebrating the fortieth anniversary of the Internet, the computer network has evolved into a complex system. This makes its analysis more challenging. Analytical techniques (e.g., the queueing theory) are adequate for small scale networks. But the increasing deployment of network hardware devices and different protocols have made network simulation one of the tools to analyse realistic scenarios of modern network operations [123].

Simulation is the technique of using computers to imitate the operations of various kinds of real-world facilities or processes [124]. Therefore, network simulation is the technique where the software models the behavior of a network either by calculating the interaction between the different network entities using mathematical formulas, or actually capturing and playing back observations from a operating network.

There are more reasons why we use network simulation. Real networks are difficult to instantiate (purchase, install, and configure) in order to experiment with scenarios, especially for large network configurations. Moreover, it is hard to create desired network conditions in real networks for controlled experimentation. For example, network with specific traffic loads and congestion patterns. Besides, network simulation is also used to test early prototypes of new network technologies or duplicate existing networks for analysis. Because of this, in this chapter we use network simulations to study catastrophic networks in a controllable and repeatable environment. We want to verify that the catastrophic network topologies obtained based on the assumption that traffic flows are approximated using

betweenness centrality.

To begin with, we briefly introduce the network simulation model and simulation tools available. The open-source network simulator OMNet++ is used as the network simulation platform with basic network scenarios. Finally, we analyse the results from network simulations using OMNet++ to see if a catastrophic network built using the assumptions of previous chapters do reflect the failures on a simulated network.

6.1 Network Simulation Model

Network simulation can be model as a Discrete-event simulation which is the basis of many existing network simulators [125; 126]. Discrete-event simulation concerns the state variables change instantaneously at separate points in time. These points in time are the ones at which an event occurs which would change the state or variables of the network system. Because of the dynamic nature of discrete-event simulation, one have to keep track of the simulation time as the simulation proceeds.

A mechanism to advance the simulation clock is also needed and it is the timing system of the network simulation. The next-event time advance method is utilized by most of the major simulation software. The flow chart of the next-event advance method is shown in Figure 6.1. Firstly, the start of the simulation would go into the main program to invoke the initialization routine to set up the simulation environment. The main program would invoke the timing routine to determine the next event and transfer control to the corresponding event routine to update system state appropriately. It also checks the termination of the simulation and invokes the report generator to build up the needed information as results. The event routine updates the system state when a particular type of event occurs. It would invoke the random number generator if it needs one.

Most network simulators provide graphical user interface where users can configure the simulation inputs easily. They can provide tools to visualize simulated network and simulated events [1]. But there are still some simulators that require

[1]Network visualization is introduced in Appendix B

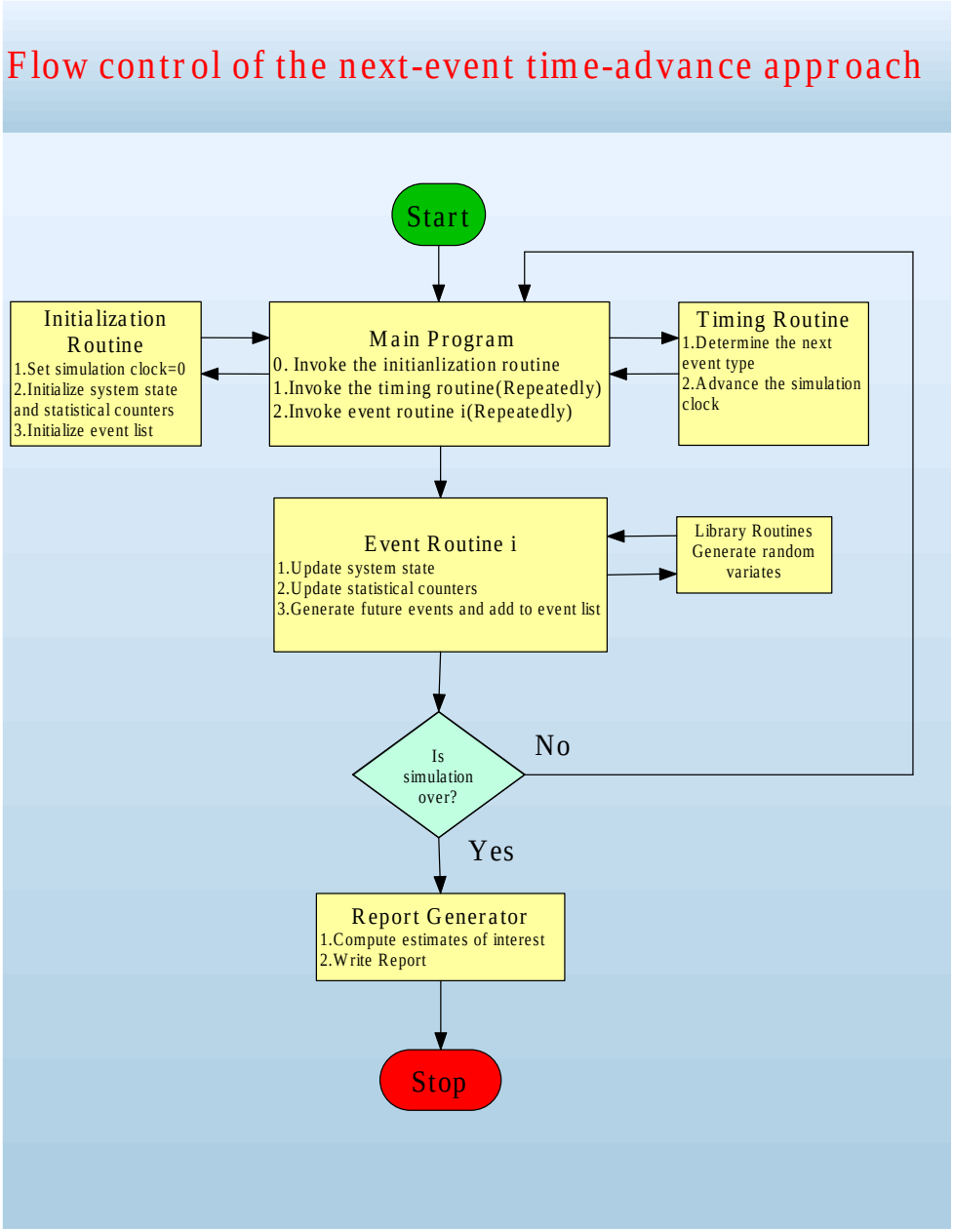


Figure 6.1: Next Event Flow Chart

their users to provide formatted input scripts or commands. These are the network parameters employed to show the state of the network (node and links) and the events (data traffic generation, node or link failures, etc). On the other hand, the important output of network simulations are the trace files. Trace files can document every event that occurred in the simulation and are used for analysis. Users could configure the simulator to output specific data from the simulation according to their interest. Certain simulators have added functionality of capturing this type of data directly and processing the raw data. But most of the time, extra software is needed to sort and analyse trace files.

For further details about network simulation modeling and analysis, please refer to book [124] and [123].

6.1.1 Network Topology Generation

The network topologies used for network simulations are either copy of the real network or generated using a topology generator. The existing network topology generators can be used to construct small size network of less than a hundred nodes to large networks with a million nodes. It is vital to set up the precise network topology to model different scenarios according to the purpose of the simulation. There are many topology generators available freely for academic research. A brief introduction and comparison of these topology generators can be found in Appendix B. However in our simulations, to verify the catastrophic networks generated, we use the small scale catastrophic network topology obtained in previous chapters.

6.1.2 Network Traffic Generation

The discovery of Self-similar Long Range Dependence traffic [27] in the Internet changed people's view on the traditional "Poisson-like" traffic to simulate packet network. The Internet traffic show more burstiness across wide time scale than what is obtains using Poisson models. Most network simulators have both traditional Poisson-like traffic generator and bursty traffic generator for different simulation requirements.

6.1.3 Routing Support

In computer networking the term routing generally refers to selecting the best paths in a the network along which to send packets and the exchange of routing information between individual routers. The algorithms to choose the routes, the data structures used, and the way routers exchange routing information are a major concern in network simulation. The forwarding process handles each packet as it arrives, looking up the outgoing link to use for it in the routing tables. The other process is responsible for filling in and updating the routing tables due to time or topology change. Simulators normally include very basic routing support such as pass on the packets according to the global shortest path calculation. This is not implementing a existing real routing protocol. It simply passes on packets through shortest paths and ignoring the exchange of routing information (global routing). Some sophisticated simulator would include open standard routing protocols, such as Routing Information Protocol (RIP) and Open Shortest Path First (OSPF) routing protocol both available free from the "Internet Engineering Task Force" (IETF) [74; 77; 78].

6.2 Simulation Tools

A number of network simulators, both commercial and open source have been developed and widely used. Well known commercial simulators includes OPNET, Shunra, QualNet, and NetSim. On the other hand in public domain, simulators includes NS2, NS3, and OMNeT++ are utilized for academic researches and educational studies where they support the simulation of IP, TCP, routing, and multicast protocols over wired and wireless networks.

The NS2, NS3 and PDNS [125] are popular tools for simulating networks. While the update, documentation, and support of NS2 is limited due to the on-going development of NS3 which is intended as an eventual replacement for the NS2. They are now not suitable for simulations for our research. The other popular and free network simulation platform is OMNeT++ [126]. It is a public-source, component-based, modular and open-architecture simulation environment with strong Graphical User Interface support and an embeddable simulation kernel. Its

primary application area is the simulation of communication networks. Although OMNeT++ is not a network simulator itself, it is currently gaining widespread popularity as a network simulation platform in the scientific community as well as in industrial settings, and building up a large user community. Because of its complicated and detailed support for different network protocols, the size of the simulated network is constrained by the amount of computer resources available.

We decided to use OMNeT++ as our network simulation platform with its communication network simulation packages. It is because it supports a simple simulation kernel library written in C++ which is easy to modify to suit different simulations. OMNeT++'s IDE is based on the Eclipse platform with Graphical User Interface for simulation execution and result analysis. It also has a large amount of manual, tutorials, publications, and sample simulations that are very useful for setting up our own simulations. Furthermore, it could be installed and run on different platforms including Linux, Mac OS X, other Unix-like systems and on Windows (XP, Win2K, Vista, 7). This makes it very convenient to migrate the simulations from development workstation to servers which could run the simulations faster.

6.2.1 Validation and verification

One of the most difficult problems facing a simulation analyst is trying to determine whether a simulation model is an accurate representation of the actual system being studied. The main factors that must be considered in order to ensure the credibility of simulation results is verification and validation of simulation model and the setup of valid simulation experiment [124]. Verification is concerned with determining whether the conceptual simulation model has been correctly translated into a computer "program". While validation is the process of determining whether a simulation model is an accurate representation of the system for the particular objectives of the study.

The following verification and validation techniques were applied to the simulation models used in this research [124]:

- Applying C/C++ Debugging tools to codes in the simulation models.

- Structured walk-through of the model and compare with real network devices and protocols.
- Simplified test cases: using official verification TicToc tutorial from OMNeT++ [126], running the ARPTest simulation from INET framework [127].
- Examine the event-log output which is a time-ordered list of events, i.e. a packet entering and leaving a router.
- Continuity test: running the simulation with slightly different values of input parameters which should produce only slight changes in the output. For example, increase the UDP traffic generation rate.
- Random number generator check: see if similar results are produced for different seed values in the random number generator. Note that OMNet++ has the Mersenne Twister RNG as default RNG.
- Output data analysis: the most definitive test of a simulation model's validity is to establish that its output data closely resemble the output data that would be expected from the actual system (see Figure 6.2). Again we use both TicToc tutorial for OMNeT++ platform and ARPTest for INET framework.

As a infrastructure to support network simulation, OMNet++ and its network simulation package INET is one of the favorite simulation tools for academic research and educational studies. More than four hundred related publications could be obtained from its web site [128]. So OMNet++ and INET have also been verified and validated by other users and proofed to be a good open source network simulation platform.

6.3 Network Scenario

6.3.1 Basic Scenario

Firstly, we build a simple network scenario to test congestion on networks when it is under heavy traffic load. It is expected that over certain traffic load queues

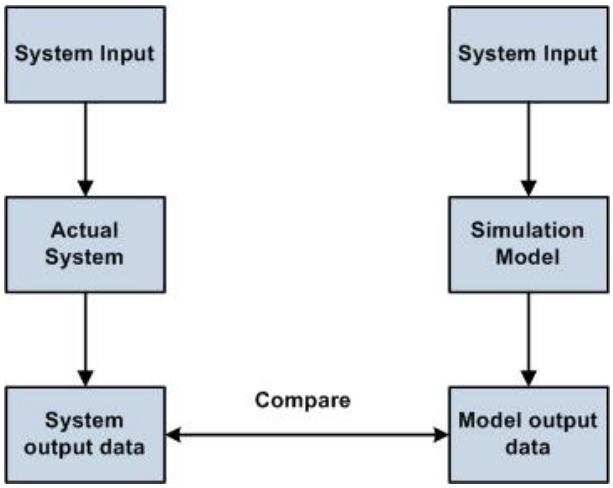


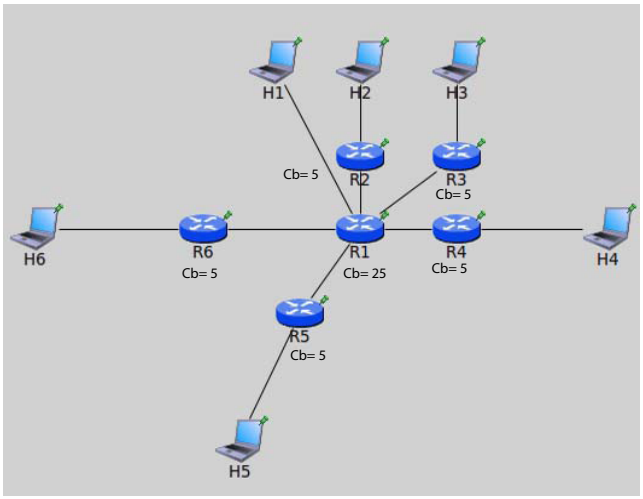
Figure 6.2: The validation of simulation model output data.

would begin to build up in the congestion routers. Increasing the traffic load will make the queues bigger and eventually reaching the limit queue length when we consider the router too busy to handle anymore incoming traffic.

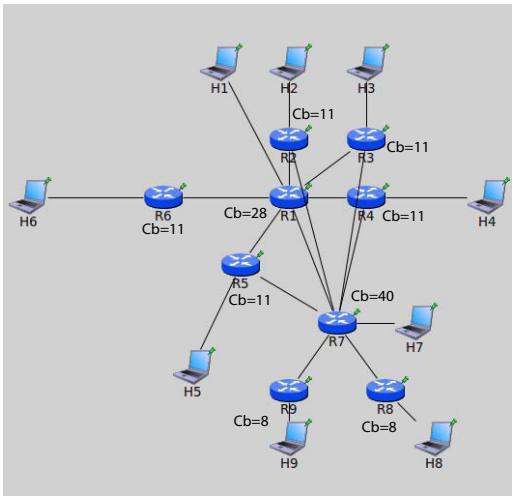
6.3.1.1 Topology

The topology for the basic scenario is the star shape six node topology. It is one of the seed network to grow the catastrophic network topology. So the network in this basic scenario contains 6 routers with one router at the center of the star connecting to other 5 routers.

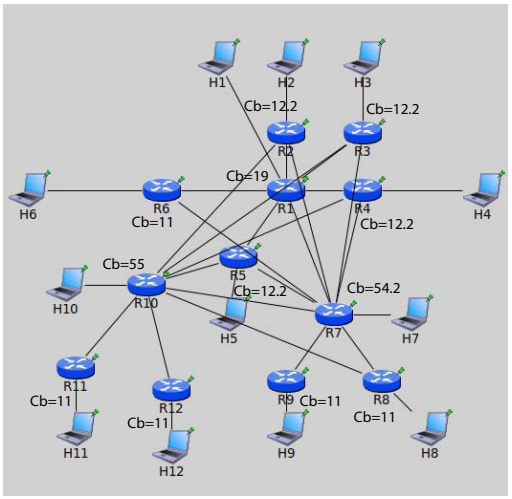
The router model chosen from the INET framework has the ability to store and forward data packets based on its building blocks: PPP (Point to Point Protocol) interface, Ethernet interface, OSPF routing, routing table, Interface table and network layer module [127]. Besides, we also have one network host connecting to each router representing the source and sink of packets. Queues in the routers are interface queues (buffers) with limited length like any real router interfaces. The queue could be a drop-tail queue, drop-tail QoS queue (classify packets based on their type of service (ToS) field), or RED (Random Early Discard) queue. In this scenario we use a simple drop-tail queue model in both PPP and Ethernet interface. By default the PPP interface has a transmit



(a)



(b)



(c)

Figure 6.3: The star shape six-node, nine-node and twelve-node small catastrophic network topology used for building simulation scenarios.

queue limit of 1000 packets and MTU (Maximum transmission unit) of 4470 bytes. The Ethernet interface is used to connect the host to the router. And they have a default data rate of 100Mps and MTU of 1500 bytes. In this case the router can be seen as a hybrid router-switch, which provides layer 3 routing and layer 2 Ethernet connections to hosts. We use only Ethernet interface on the routers assuming that they are controlled by one ISP and close to each other that Ethernet link is enough to support inter-router connection.

Hosts in the network are standard hosts supporting: TCP, UDP, TCP application, UPP application, PPP interface, Ethernet Interface, interface table, routing table, etc. But only the Ethernet interface is used to connect to the router.

6.3.1.2 Traffic

So packets are generated at each host and sent randomly to all other hosts in the network. INET framework provided: basic UDP application (with constant bit rate), basic TCP application, RTP application (streaming traffic), and also IP layer traffic generator directly send and receive IP packets. TCP is a protocol that forces competing users to share bandwidth and generally behave in ways that are good for the productivity of the overall network. But a large amount of TCP traffic, such as Bit-Torrent or sometimes web browsing, would congest network routers. While UDP blasts packets out as fast as it can (i.e., online games, music, voice and video streaming). So we made use of the TCP, UDP and IP layer traffic generator. The examples shown later in this chapter have UDP applications as traffic source.

6.3.1.3 Routing

The routing protocol chosen is OSPF, it is the only supported and sophisticated interior gateway protocol in INET framework. We are not going to discuss the detail implementation of OSPF in INET, for more information please refer to INET documentation [127]. Packets are forwarded from the source host to the routers and finally to the destination host. The router would use the OSPF's Dijkstra shortest-path algorithm to calculate its forwarding path to create the routing table. Links between routers have the same weight.

Each host would be assigned an IP address in separate Class C subnets (i.e., IP address: 192.168.1.2, subnet mask 255.255.255.0). The routing metric for the link between the host and the router is '1'. And the default gateway for the host is obviously its connected router. Each interface on the router would be assigned with different IP addresses (i.e., IP address: 192.168.60.64, subnet mask: 255.255.255.254). But all these routers would be configured to be within one OSPF area.

6.3.2 Catastrophic Network Scenario

The basic model is to test if congestion in the network would really build up queues in the network under real TCP and UDP application. And whether the center router of the star shape topology would fail to cope with the excessive amount of load. Then we would like to test a very small catastrophic network with nine nodes. This scenario is intended to test the cascade failure, where the router designed to be heavily congested and becomes the first to fail to cope with the amount of traffic. As the topology changes, the routers and hosts in the network would be the same as in the basic scenario. The hosts would use the same application and traffic generator as the traffic source. The OSPF is also used as the routing protocol with links having the same weight.

6.4 Catastrophic Network Simulation Results

In this section, we are going to look at the simulation results using the three catastrophic topologies: 6 node star-shape seed network, 9 node catastrophic network built using branch and bound method, and 12 node catastrophic network using branch and bound method. Due to the limitation of the computer system and simulation platform used for the simulations, there are several constraints to the final outcomes. First, the computer system used for simulation is Dell Workstation with Intel Duo Core Xeon 5500 CPU and 6G DDR RAM. It limits the size of the network we could simulate, because of the level of simulation carried out. The OMNET++ and INET framework could simulate network from Physical Layer up to the Application Layer level. Secondly, OMNET++ has a

limitation on the size of the logging file of 2 Gbyte. This limits the simulation time for each simulation depend on the traffic load used. The heavier the traffic the more packets logged, hence the bigger the log file.

In the following subsections, we try to monitor the queue size of the routers to examine how congested they are. And these queues are First In First Out Drop Tail queues. Although there are other metrics for network congestion, such as delay and throughput, we focus on the queue lengths. Because packets are dropped when the router memory is full, the delay of the dropped packets tend to be infinity. The simulator only logs the delivered packets, making the delay measurement not reflecting the real time congestion. Network throughput is calculated by delivered packets in the period of time, but it would significantly slow down the simulation and further increase the log file.

6.4.1 Six-node Star-shape Topology

This scenario uses the six-node star-shape topology as the seed network to grow the catastrophic networks, see Figure 6.3(a). Because of its unique shape, the center node of the star topology carries the traffic between all the other nodes. Hence, we expect to see longer queues in the center node although it is under light and medium traffic load. It would also be the first node in the topology to be overloaded under heavy traffic condition.

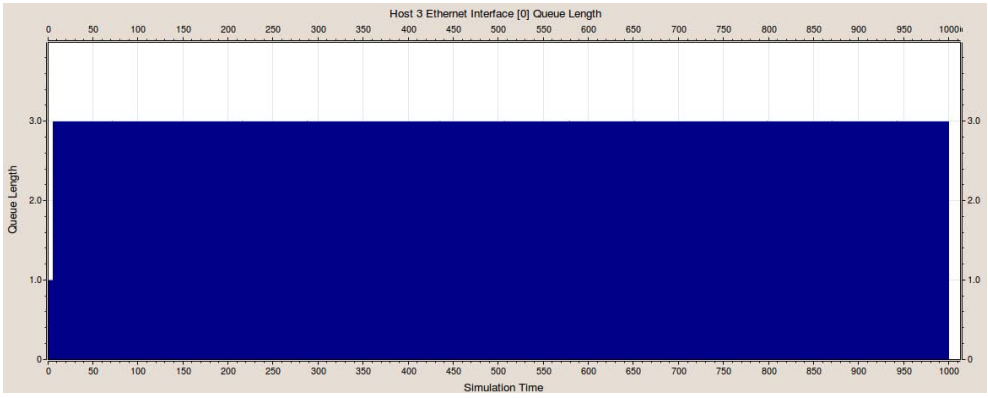
The PCs (hosts) in this network all run four instances of the UDP application which send out basic UDP traffic to all other PCs in the network by configuration. The UDP ports used are 1234, 1235, 1236 and 1237. The UDP applications send out message every 0.1 second (low rate), 0.01 second (medium rate) and 0.001 second (hight rate) with message size of 6500 bytes. Every PC has its own routing configuration file. This file defines the PC's IP address (i.e. 192.168.1.2), subnet mask, MTU size, routing metric for OSPF (in this case it is 1), default gateway and enables broadcast and multicast. On the other hand, the router has its own routing configuration file. It defines similar properties for each of the router's interfaces while the metric for the link between routers is manually set to 2. Meanwhile, OMNET++ INET framework holds a global routing configuration file for OSPF to define the OSPF Area (just one Area in this case), address

ranges, and OSPF router setup parameters. For more detail please see Appendix A of the example for six-node star-shape topology.

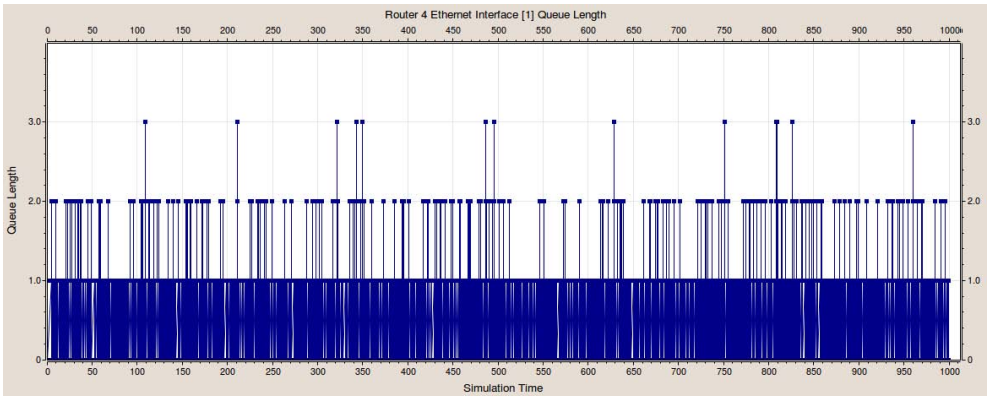
Firstly, we test light traffic on the six-node star-shape catastrophic network. The UDP application message length is 6500 bytes and sent every 0.1 unit simulation time, generating a data flow of $(6500 \times 8)/0.1 = 520kbit/s$. With four UDP instances of applications running on a single host, the total amount of traffic from the host to the router is $2080kbit/s$ or $2.08Mbit/s$. Secondly, medium traffic is applied, with UDP application message length of 6500 bytes and sent every 0.003 unit simulation time. The data rate of one UDP application is $6500x8/0.003 = 17.3Mbit/s$, making the total data flow from host to the route $69.3Mbit/s$. Finally, very heavy traffic is applied with UDP application message length with 6500 bytes and sent every 0.001 unit simulation time. The data rate of one UDP application is $6500x8/0.001 = 52Mbit/s$, making the total data flow from host to the route $208Mbit/s$. Finally, we are not trying to simulate the situation that the network goes from free-flow state to congestion. Because it would take the network very long time to converge and finally congested.

When applying a low data traffic rate from the host's UDP application and sending packets evenly to all other hosts in the network, a free-flow state is expected for the whole network. Figure 6.4(a) shows the queue in the Ethernet Interface(0) of Host 3 in the network. Host 3 is directly connected to Router 3 and uses it as its default gateway to reach other networks. The queue size stays constant at three. It indicates that most of the packets are sent with a minor delay. On the other hand, there is no queue on Router 4 Ethernet Interface(0) which is connecting to Host 4. It indicates that the router can handle the incoming packets quickly and pass on to the outgoing interface. Figure 6.4(b) shows the queue in Ethernet Interface(1) of Router 4 which is connected to Router 1. This interface has a queue that fluctuates through out the the 1000 unit simulation time. But we could see that the queue size just occasionally grows to three and two, while most of the time it stays at one. In short, Router 4 is shown to handle the packets from the host and transfer them to the next hop router Router 1. The queue length in Ethernet Interface(1) depends more on the condition of Router 1.

Because of the special structure of the star-shape topology, Router 1 has six Ethernet interfaces connecting to all other routers and Host 1. So it carries all the

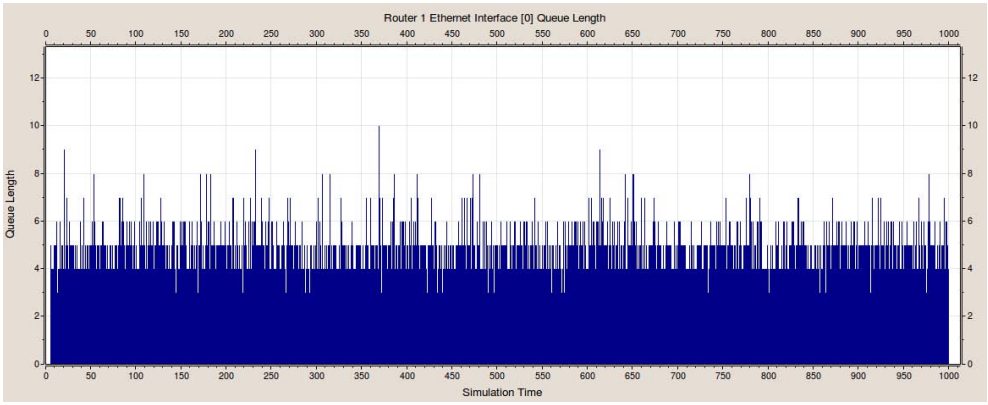


(a)

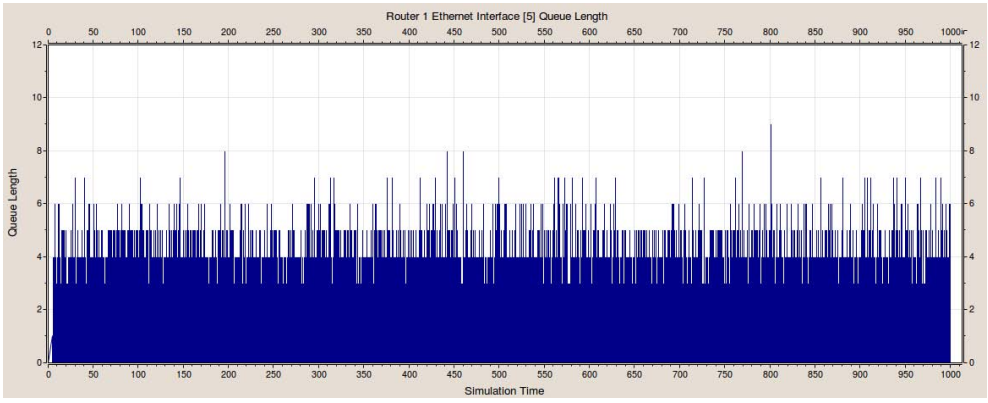


(b)

Figure 6.4: 6-node star-shape catastrophic network with constant UDP traffic at low load. (a) Ethernet Interface(0) Host 3, (b) Ethernet Interface(1) Router 4.



(a)



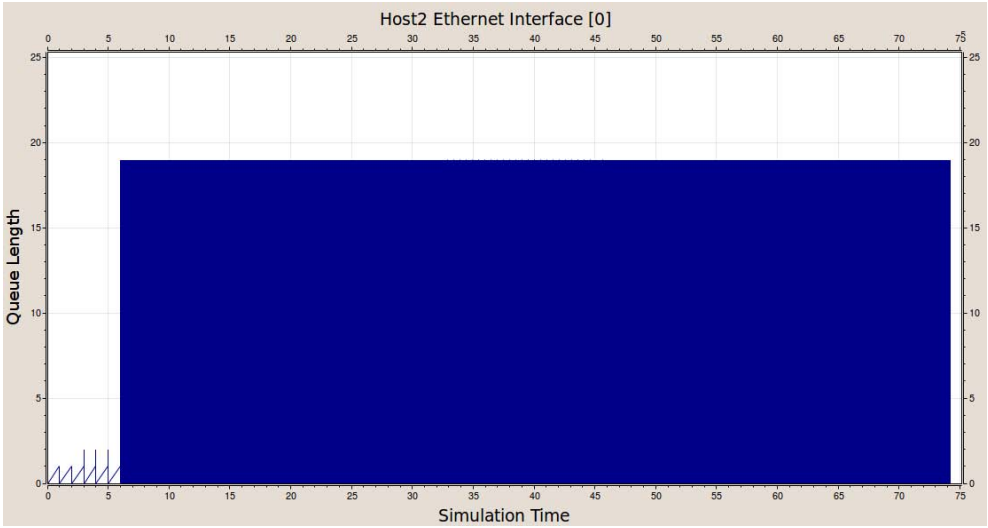
(b)

Figure 6.5: 6-node star-shape catastrophic network with constant UDP traffic at low load. (a) Ethernet Interface(0) Router 1, (b) Ethernet Interface(5) Router 1.

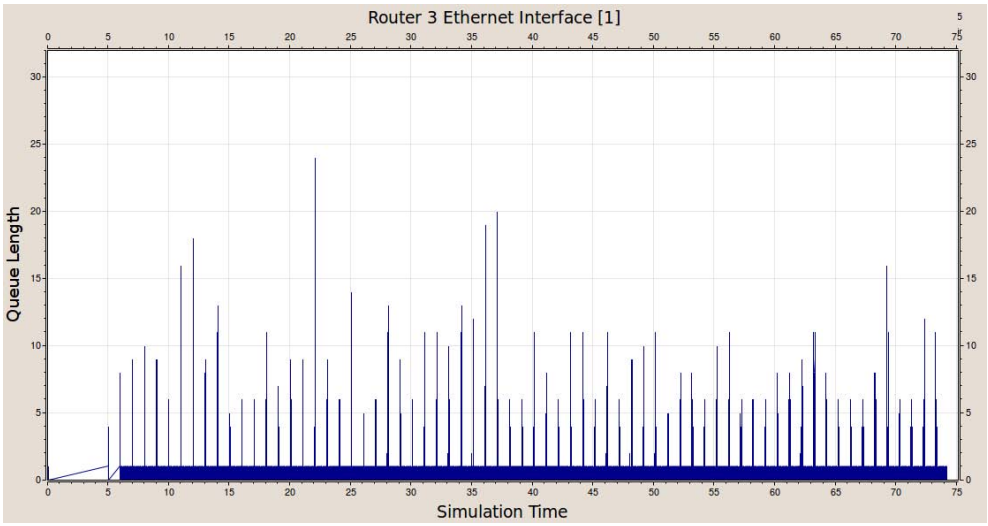
traffic flow from host to host and experiences a higher load than other routers. Figure 6.5(a) shows that the change of queue length on Ethernet Interface(0) Router 1 is very different, which is connecting to Host 1. While other routers have no queues on this interface, Router 1's queue on Interface(0) has an average of four packets and peaks at 10 packets. In Figure 6.5(b), queue on another Router 1 interface has a similar pattern. It is Ethernet Interface(5) of Router 1 connecting to Router 6. The reason for these patterns is the combination of all the packets queued in Router 1 which take time to be processed. This indicates that even at relatively low traffic generation rate, Router 1 carries more traffic than other routers in the network. Thus Router 1 has the opportunity to get congestion first and even fail to route packets in the six-node star-shape network topology.

As the network traffic generation rate is increase to a medium level, queues on the hosts increase as expected. Figure 6.6(a) illustrated the queue in Ethernet Interface(0) of Host 2 with a average length of 19 packets. The same increase of the queue lengths could also be found on the queues of Router 3 Interface(1), as shown in Figure 6.6(b). But more significant increase of queue size could be found on all Router 1 interfaces, Figure 6.7(a) and (b). The queue sizes fluctuate between 15 and 30 packets and once shoot up to 70 packets. Adding up all the packets queued in Router 1 of the six interfaces at this traffic rate, Router 1 is seen as heading towards congestion.

Finally, very high traffic generation rate is applied on the UDP application, which is generating data stream is very large. It brings excessive packets to all routers' Ethernet interfaces pushing all the routers to their capacity limits. Queue on Host 1 shows that host queues are almost always hitting the maximum queue length of 1000 packets with occasional drops, Figure 6.8. Queues are built up for both Interfaces(0) and Interface(1) of Router 3. It is different from the ones seen at low and medium traffic rate with no queues on the interface connecting to the hosts. In Figure 6.9(a), queue in Ethernet Interface(0) at first grows and experience fluctuation in first 20 simulation time, then drop down to zero as illustrated. But queue in the interface(1), shown in Figure 6.9(b), the queue builds up and reaches the buffer limit and enters huge fluctuation period after simulation time 15. It is interesting to note that the queues on Router 1

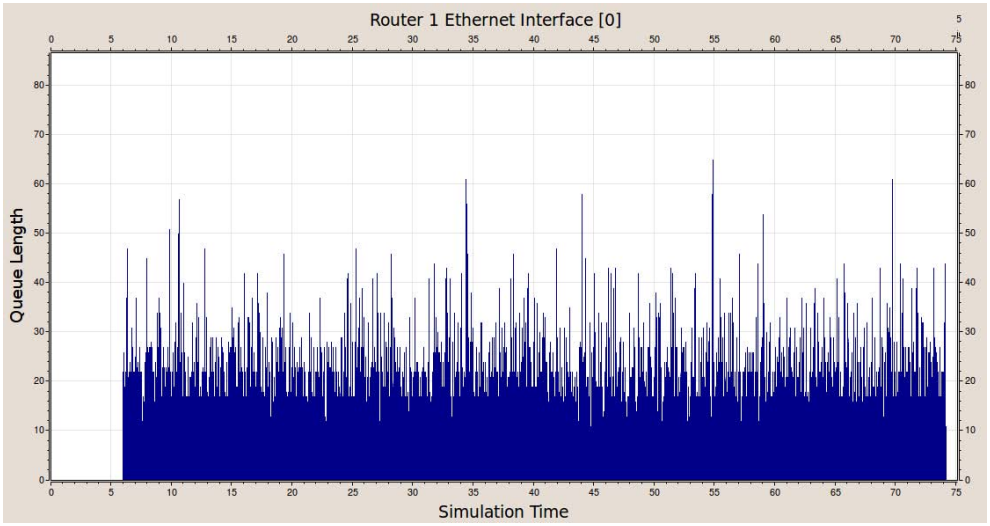


(a)

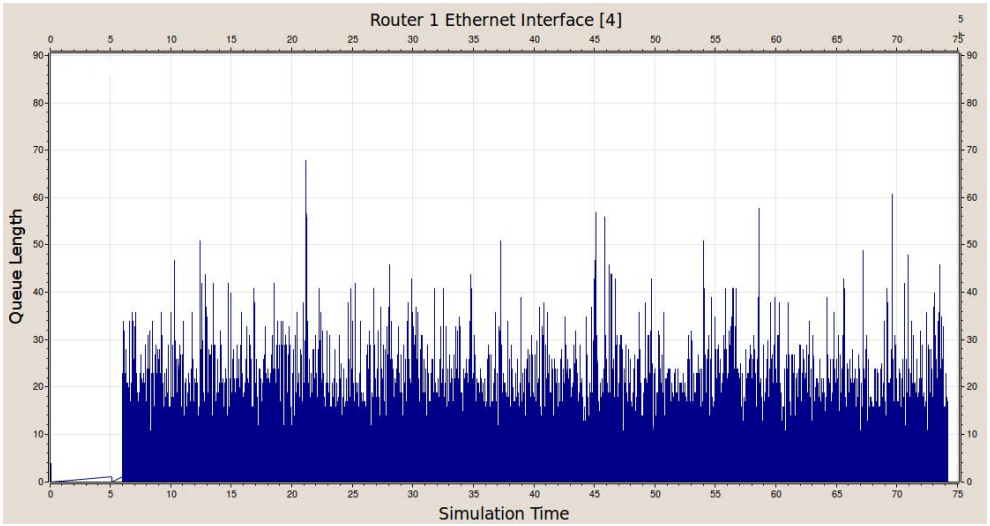


(b)

Figure 6.6: 6-node star-shape catastrophic network with constant UDP traffic at medium load. (a) Ethernet Interface(0) Host 2, (b) Ethernet Interface(1) Router 3.



(a)



(b)

Figure 6.7: 6-node star-shape catastrophic network with constant UDP traffic at medium load. (a) Ethernet Interface(0) Router 1, (b) Ethernet Interface(4) Router 1.

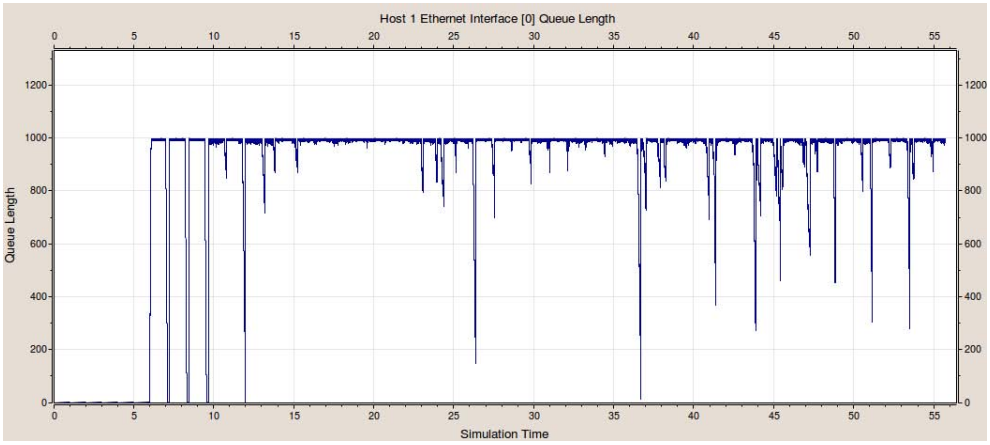
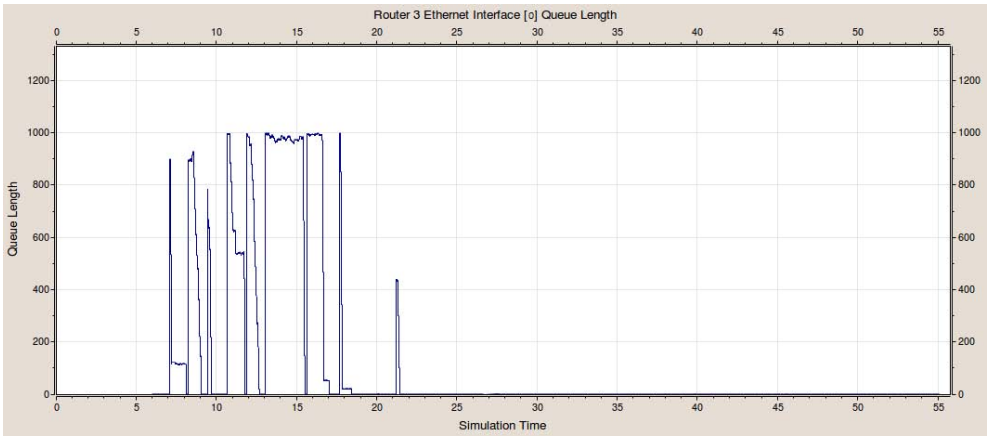


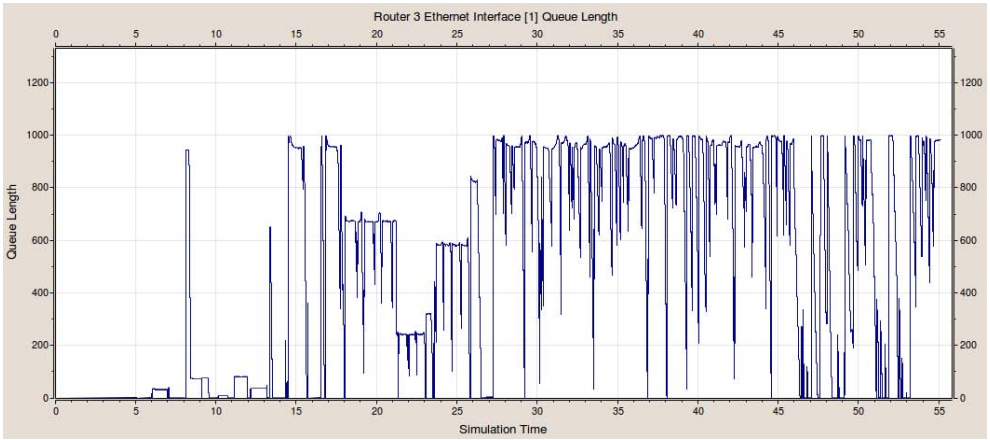
Figure 6.8: 6-node star-shape catastrophic network with constant UDP traffic at high load. Ethernet Interface(0) Host 1.

shown Figure 6.10(a) and (b), shoot up from simulation time 7 right after the queues built up on interface(0) of other routers. It shows that Router 1 enters the extreme congested period before other routers do. This proves our assumption of using betweenness centrality as an approximation of the network load.

In Figure 6.10(a) and (b), queues on Router 1 experience a dramatic growth up to the buffer limit and down to zero. Under very high traffic load, all the routers are forced to reach their maximum performance capacity. But rather than seeing the queue lengths staying at their limit, huge increases and drops are experienced. There are several reasons behind this. Traffic generated is larger than the link could carry and routers just drop packets that could not be sent. Secondly, when queues are full, these FIFO drop-tail queues also start to drop packets that could not be stored. The busy link and full queues will drop data packets as well as the routing information packets (OSPF hellos). This would affect the routing update which would make routers declare their neighbors to be unavailable or dead. The route used to sent packets might be dropped from the routing table. Hence, packets in the queues are discard because of unreachable destination. But when the link is free and queue is cleared, the neighbor relationship establishes again with the route added back to routing table. Thus all these cause the significant fluctuations in queue length.

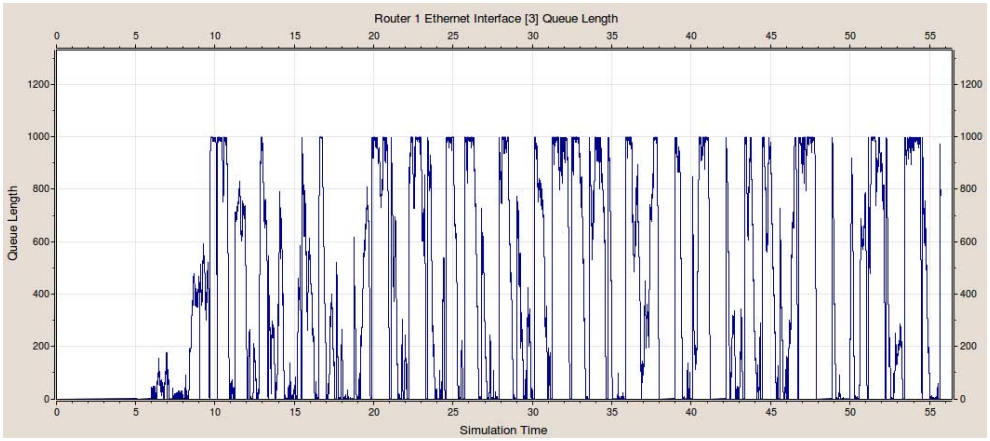


(a)

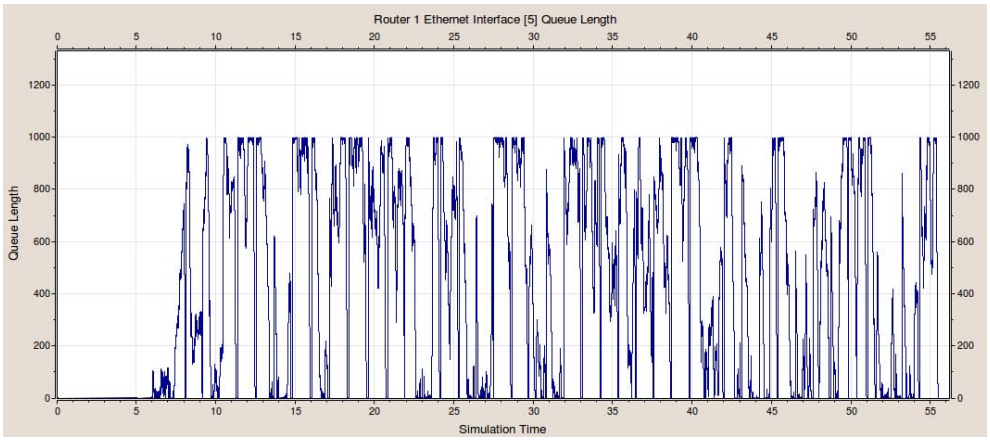


(b)

Figure 6.9: 6-node star-shape catastrophic network with constant UDP traffic at high load. (a) Ethernet Interface(0) Router 3, (b) Ethernet Interface(1) Router 3.



(a)



(b)

Figure 6.10: 6-node star-shape catastrophic network with constant UDP traffic at high load. (a) Ethernet Interface(3) Router 1, (b) Ethernet Interface(5) Router 1.

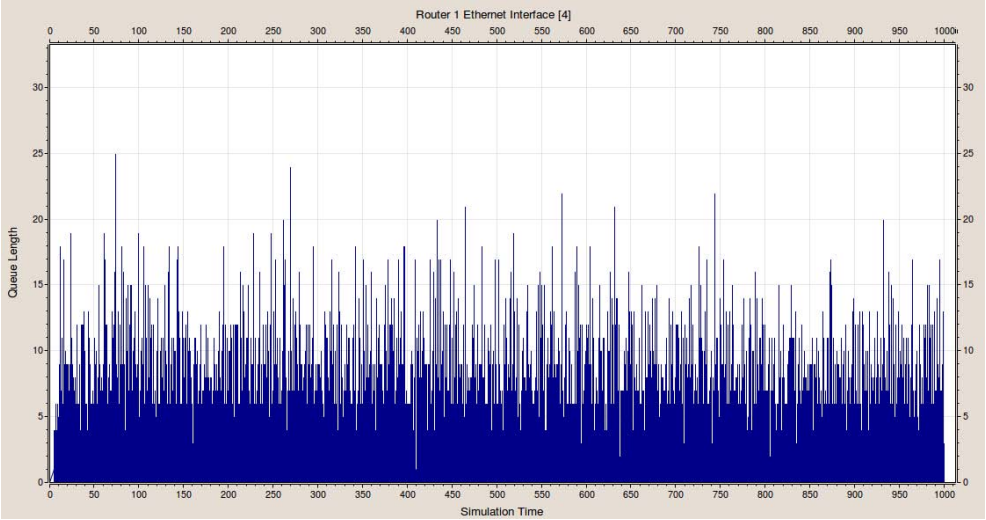
6.4.2 Nine-node Catastrophic Network Topology

We examined the six-node star-shape topology in the last subsection, which is also the seed network to grow the catastrophic network. It is important to further analyse the catastrophic networks built based on it. In this subsection, we investigate the behavior of the queues in the nine-node catastrophic network topology under different traffic levels. Similar to the one done in the previously, low, medium and high traffic is applied to the hosts' UDP application traffic generator. The UDP message size is still 6500Byte and sent every 0.1 second (low), 0.01 second (medium), and 0.001 second (high). The corresponding data flow sizes are 2.08Mbps, 20.8Mbps, and 208Mbps.

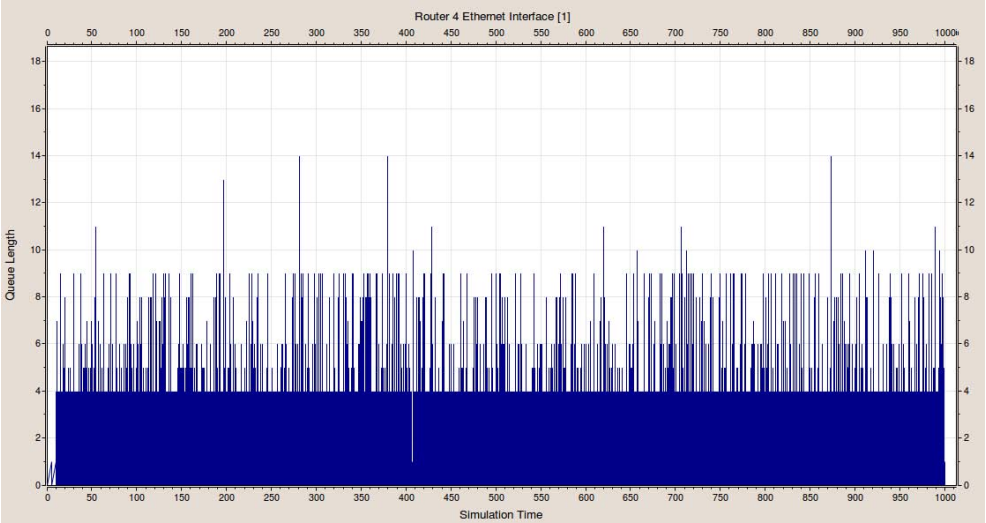
At the low traffic level, queues behave similarly as seen in the six-node topology and we are not showing the graphs again. But there are more traffic sources in the network, because more routers and hosts are added to the topology. The total amount of the traffic in the network has increased accordingly. It is expected that the queues on the routers will be longer than the ones of the six-node network, as can be seen from Figure 6.11(a) and (b). Also notice that Router 7 is the new router that is designed to congest first in this topology. Queues on it are built up faster and longer than those in the other routers, shown in Figure 6.12(a) and (b). Similar behaviour to the six node star-shape topology could also be found when this network experiences medium traffic load, as illustrated in Figure 6.13(a), (b), Figure 6.14(a) and (b).

At very high traffic generation level, the queues behave differently. The queue on Ethernet interface(5) of Router 1 connecting to Router 6 is not as busy as it is in the six-node star-shape network. Because it is no longer the first node to congest in this new network. Although huge fluctuations occur, as illustrated in Figure 6.15(a), they are still below the queue buffer limit for most of the simulation period. On the other hand in Figure 6.15(b), queue in Interface(1) of Router 8 is much more congested and keep reaching the queue's limit after simulation time 20. This is the interface connecting to Router 7 which is the router designed to congested first. By looking at the queues on two Router 7's interfaces, in Figure 6.16(a) and (b), the queue length jump up to its capacity

limit at 10 and 12 simulation time way before other routers. This indicates that Router 7 would be the first router to congest or fail to process data packets.

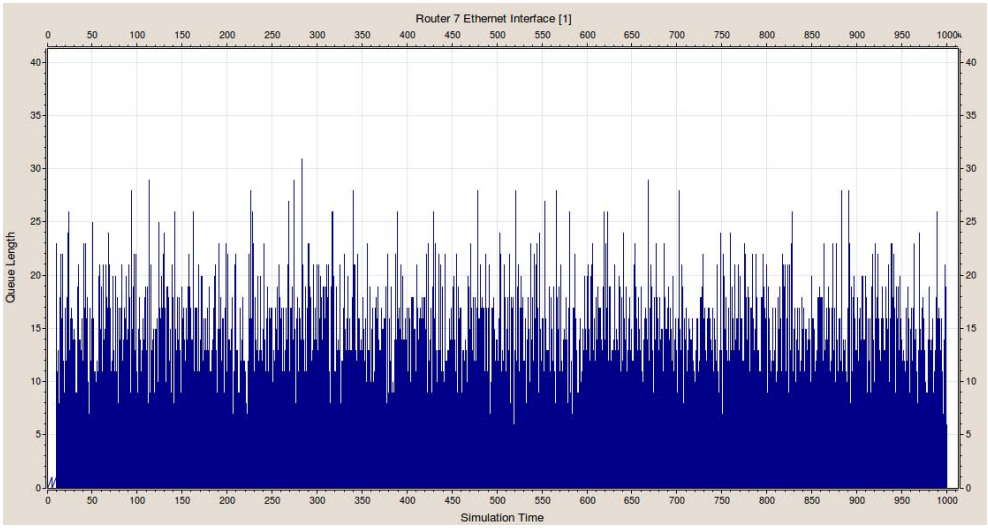


(a)

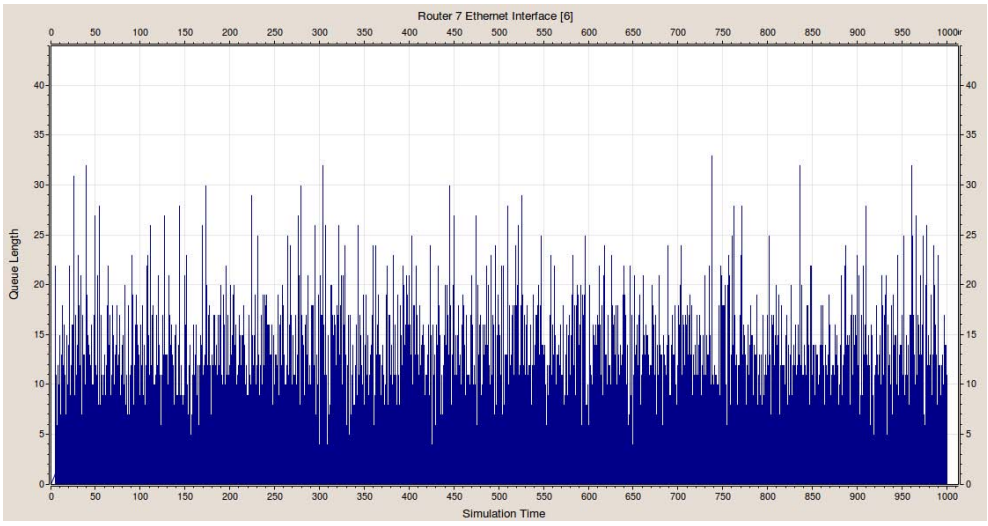


(b)

Figure 6.11: 9-node catastrophic network with constant UDP traffic at low load. (a) Ethernet Interface(4) Router 1, (b) Ethernet Interface(1) Router 4.

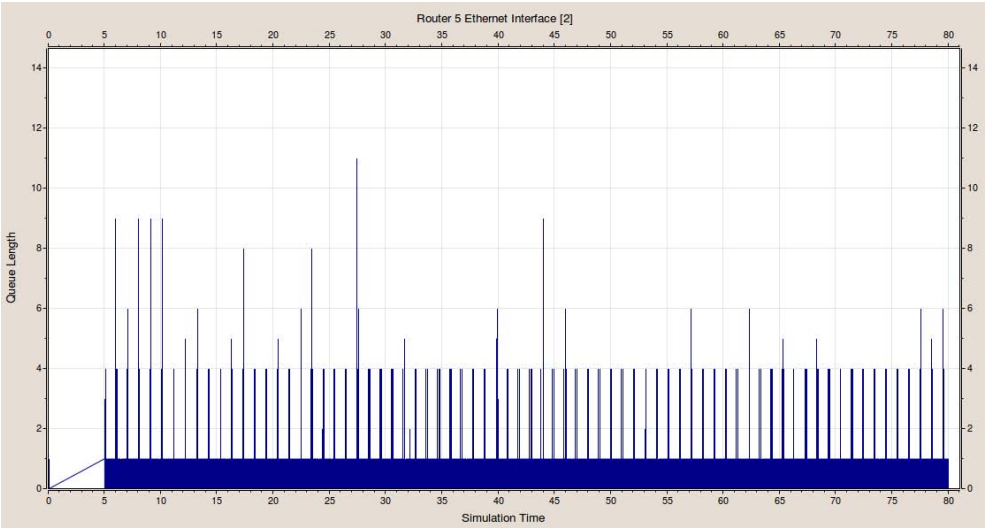


(a)

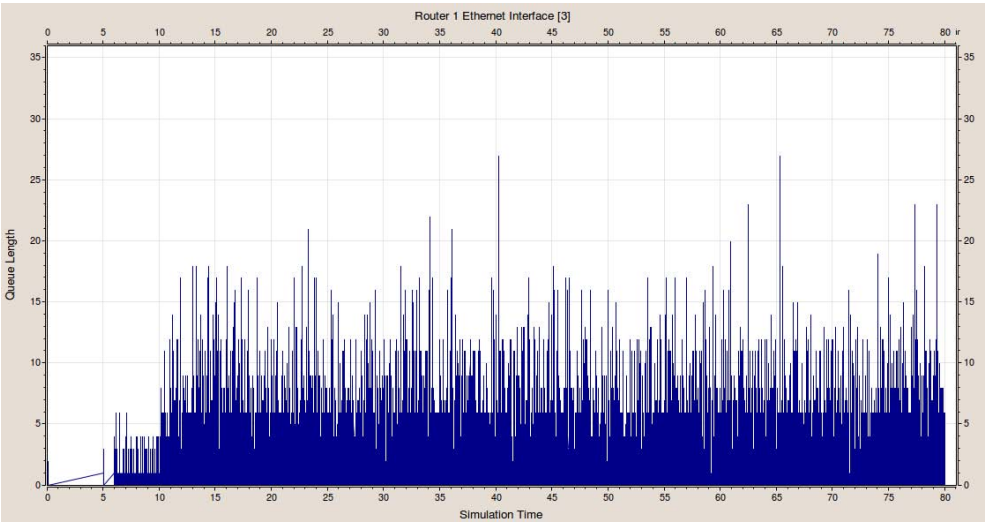


(b)

Figure 6.12: 9-node catastrophic network with constant UDP traffic at low load. (a) Ethernet Interface(1) Router 7, (b) Ethernet Interface(6) Router 7.

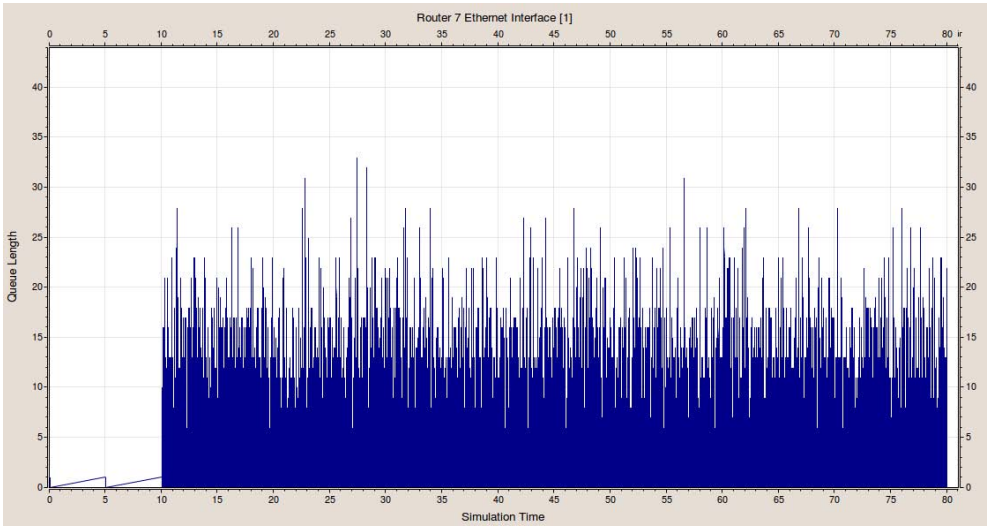


(a)

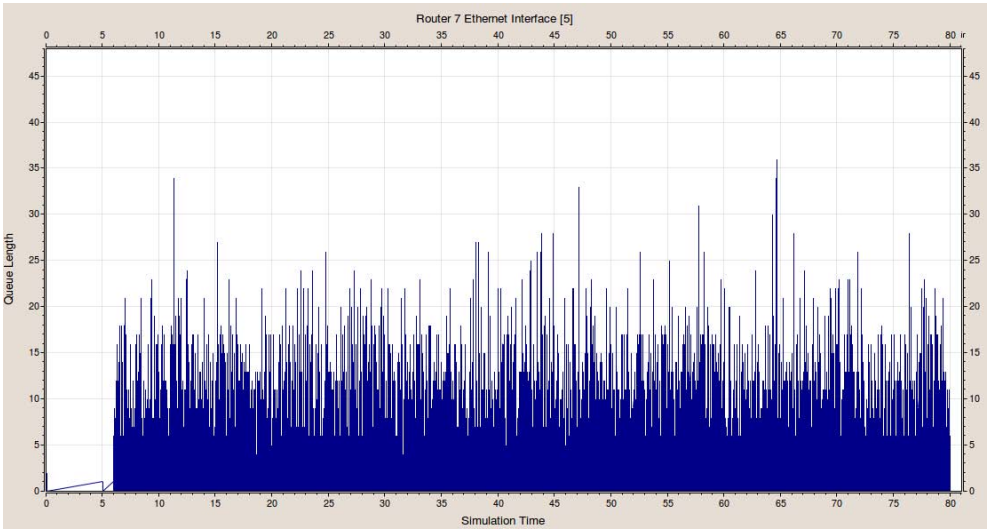


(b)

Figure 6.13: 9-node catastrophic network with constant UDP traffic at medium load. (a) Ethernet Interface(2) Router 5, (b) Ethernet Interface(3) Router 1.

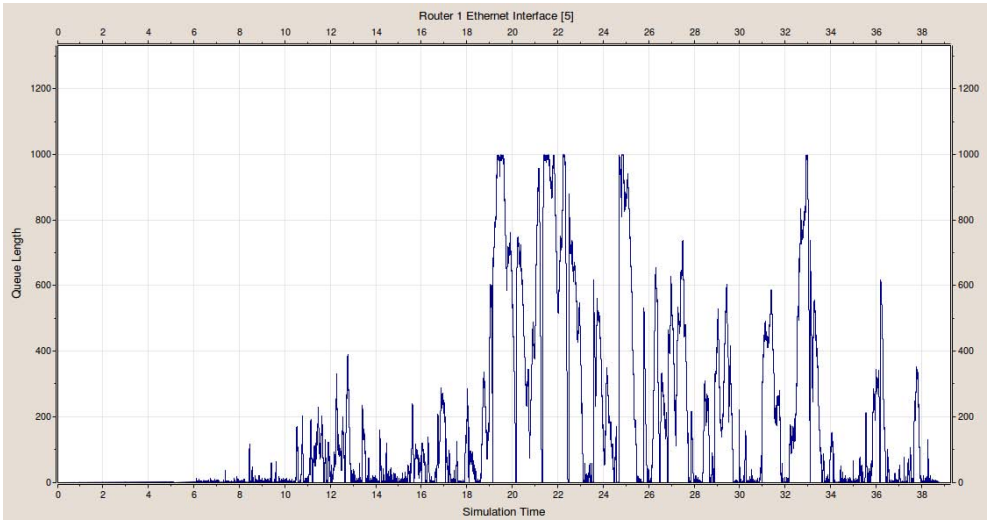


(a)

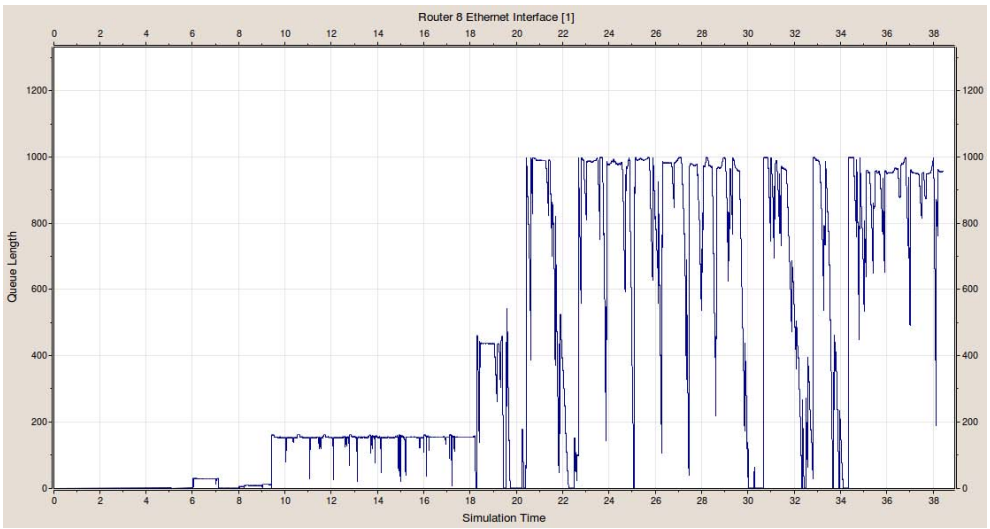


(b)

Figure 6.14: 9-node catastrophic network with constant UDP traffic at medium load. (a) Ethernet Interface(1) Router 7, (b) Ethernet Interface(5) Router 7.

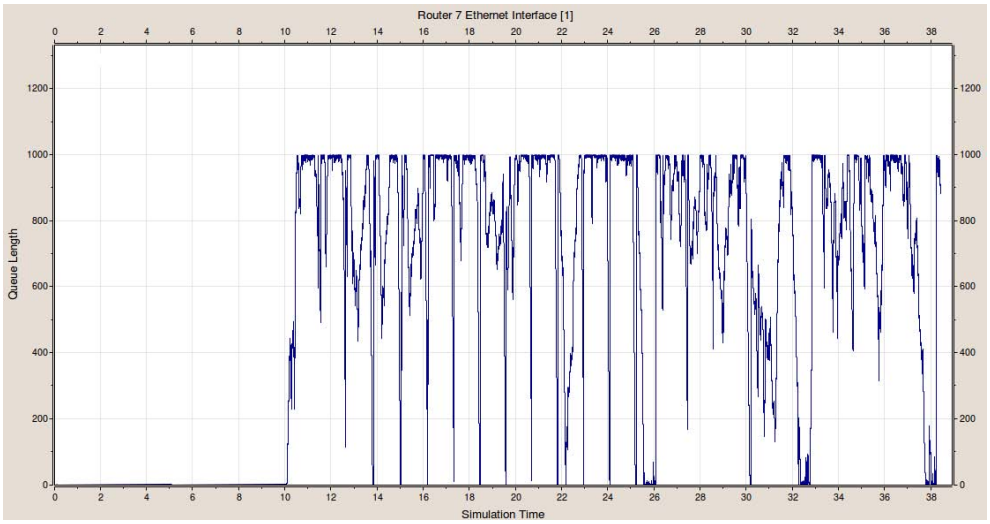


(a)

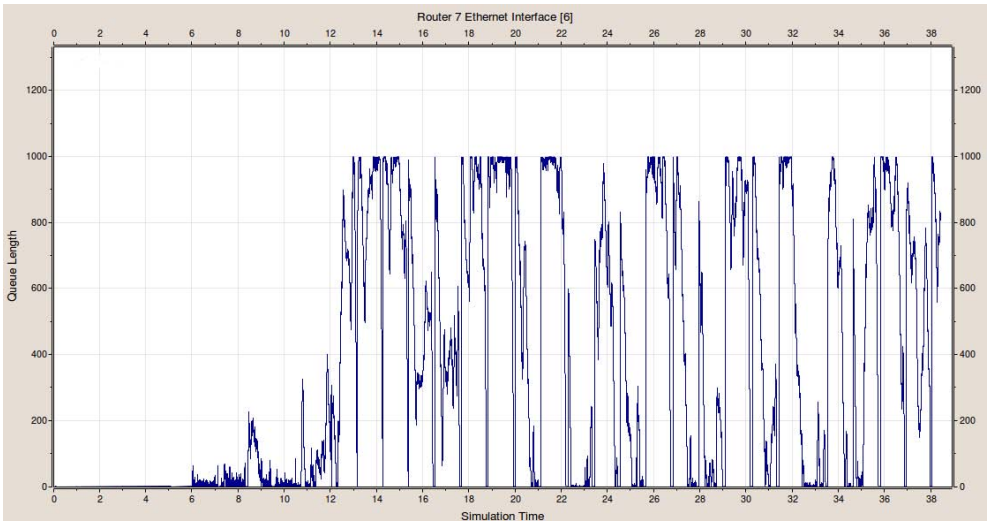


(b)

Figure 6.15: 9-node catastrophic network with constant UDP traffic at high load. (a) Ethernet Interface(5) Router 1, (b) Ethernet Interface(1) Router 8.



(a)



(b)

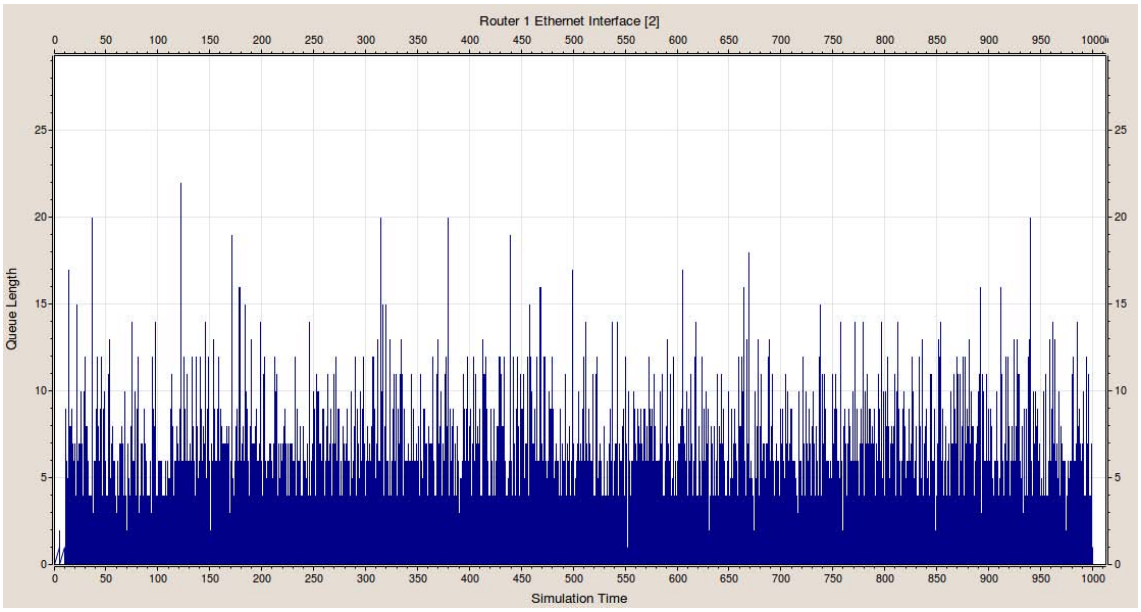
Figure 6.16: 9-node catastrophic network with constant UDP traffic at high load. (a) Ethernet Interface(1) Router 7, (b) Ethernet Interface(6) Router 7.

6.4.3 Twelve-node Catastrophic Network Topology

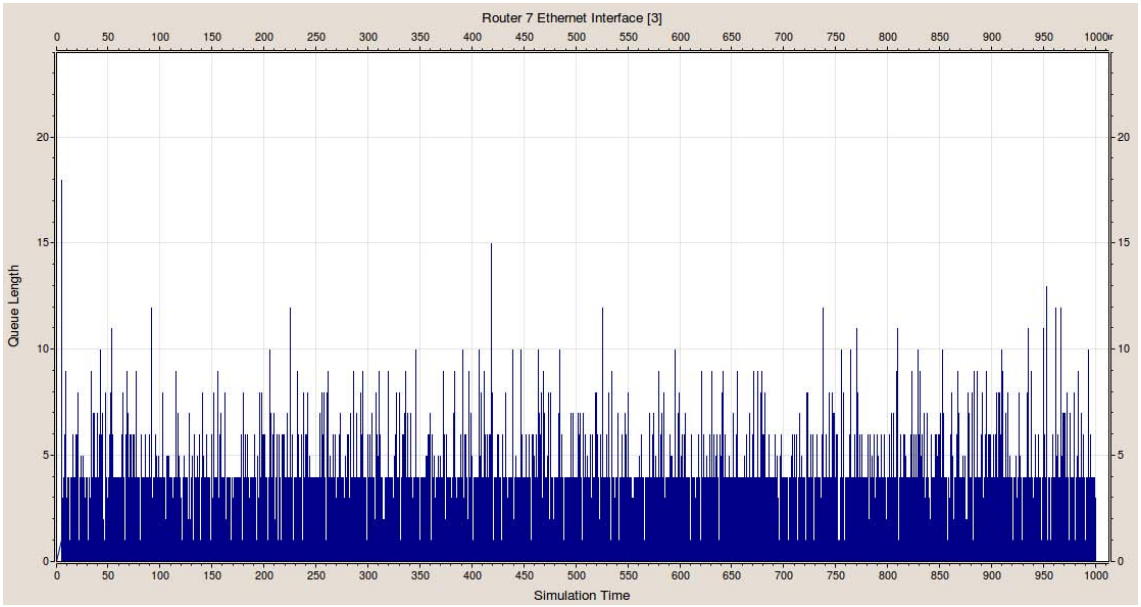
In this subsection we continue to examine the catastrophic network that is one step larger than the nine-node catastrophic network. At low and medium traffic level, very similar behavior could be found for the queues on routers in the network, as illustrated in Figure 6.17(a) and (b), Figure 6.18(a) and (b), Figure 6.19(a) and (b), and Figure 6.20(a) and (b).

But a different queue performance is shown at the high traffic generation rate for the UDP applications. The queues on Router 1 and Router 7 are not so long compared to nine node catastrophic network, shown in Figure 6.21(a) and (b). Although they grow up to almost 400, it is still within the limit of 1000 packets. Because more traffic is concentrated on Router 10. Router 10 is the new router in this twelve-node network that is designed to fail first. Being the new center of data flows, queues on it built up quickly and fluctuate up and down between the boundaries of 1000 and 0 packets. Notice that the pattern of the queue length fluctuation on Router 10 is different from the ones on Router 7 in the nine-node network. Although the absolute traffic data rate remains the same from every host, it is sent evenly distributed among the 11 remaining hosts. And it reflects differently on the routers' queues for different topology structures.

By observing the simulations on the twelve, nine and six node catastrophic networks, we can see the congestion nodes at each network congest first at high traffic load. So when router R10 in the twelve node network becomes highly congested and its links are disconnected, router R11 and R12 will be isolated from the rest of the network. The topology will then be the nine node catastrophic network. According to the result from the last subsection, router R7 will then become highly congested. Then we will remove its links where router R8 and R9 are isolated. Thus, the connected network topology is the seed network – a six node star-shape network.

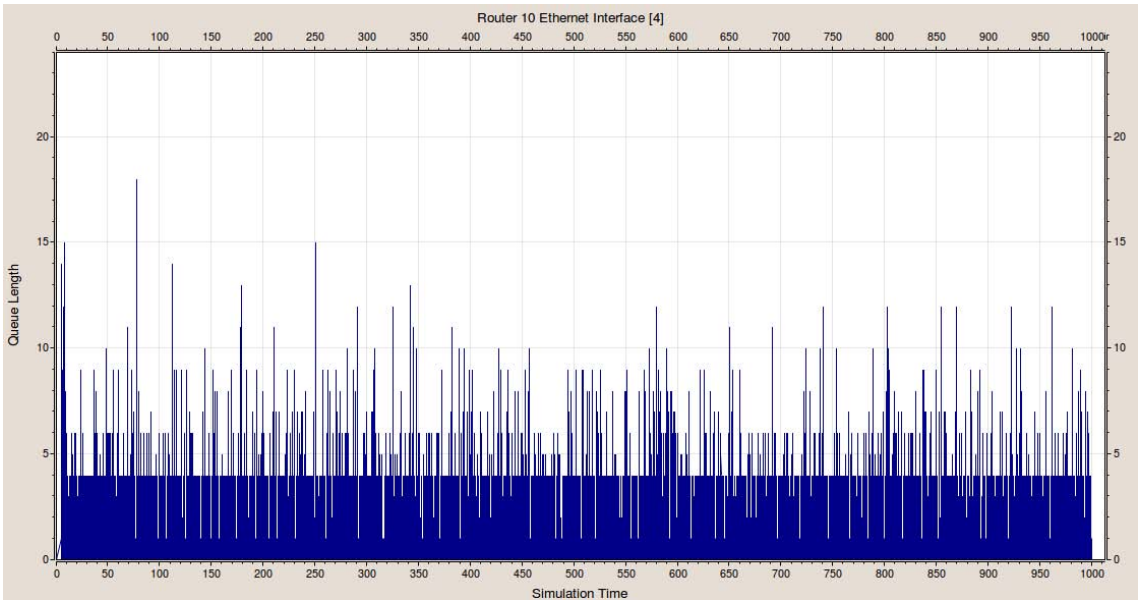


(a)

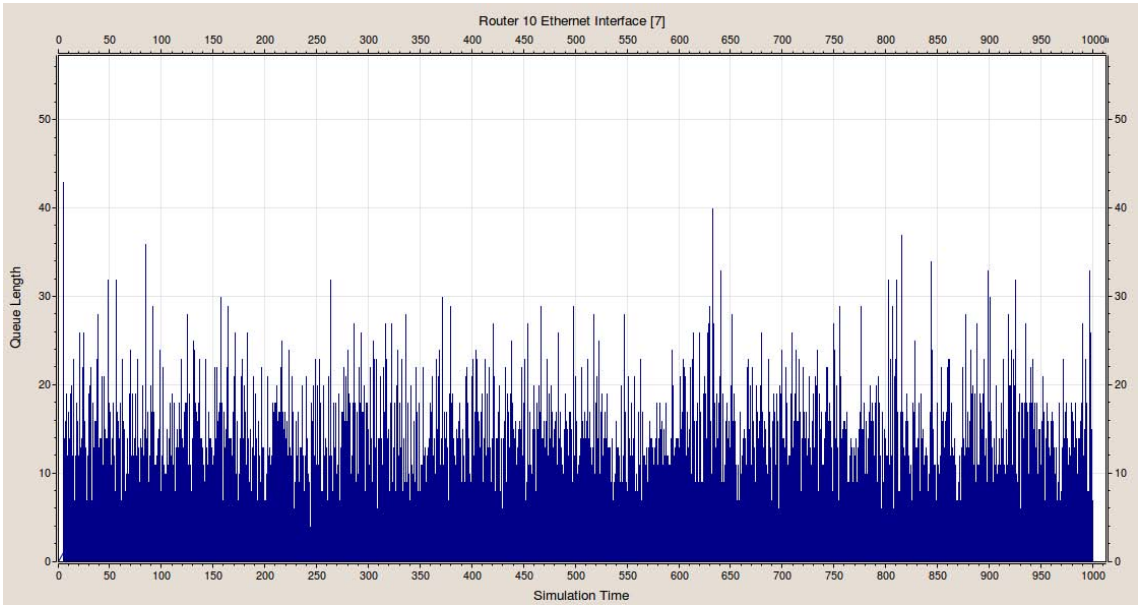


(b)

Figure 6.17: 12-node catastrophic network with constant UDP traffic at low load. (a) Ethernet Interface(2) Router 1, (b) Ethernet Interface(3) Router 7.

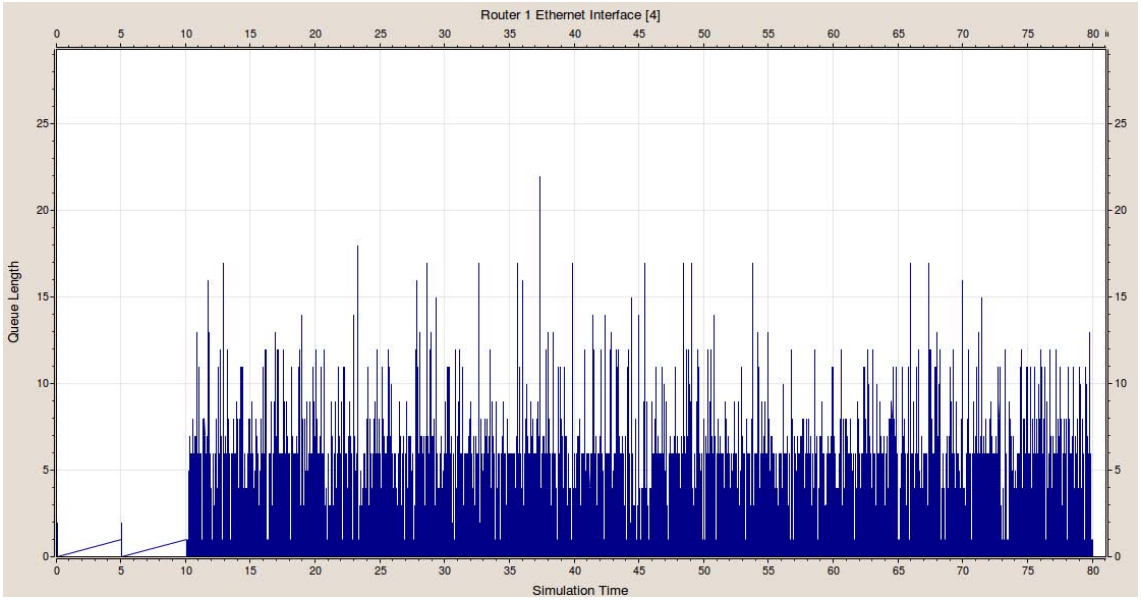


(a)

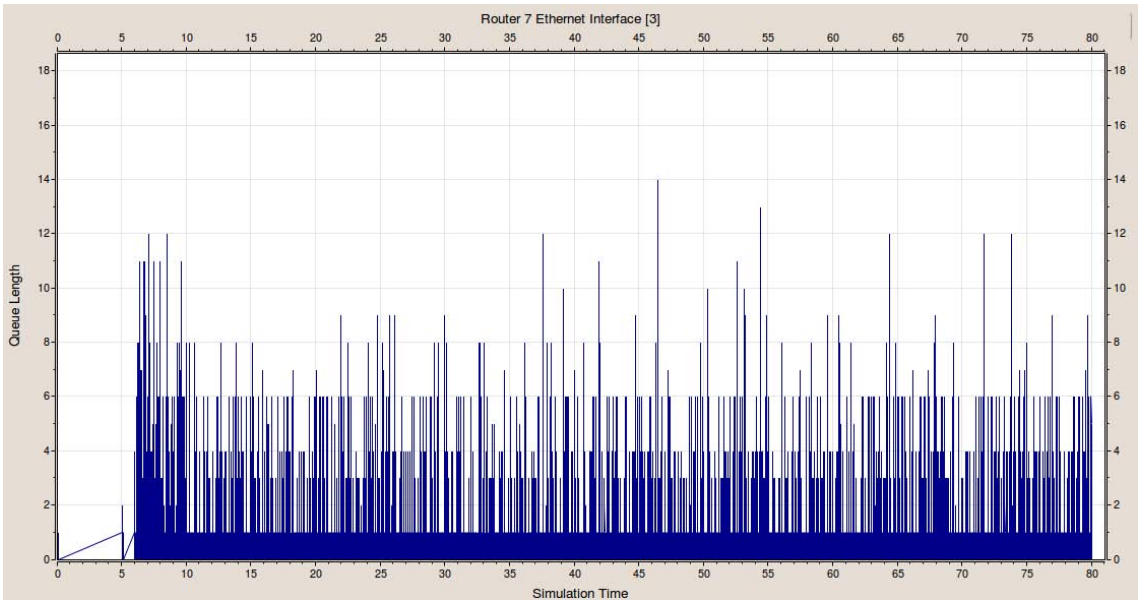


(b)

Figure 6.18: 12-node catastrophic network with constant UDP traffic at low load. (a) Ethernet Interface(4) Router 10, (b) Ethernet Interface(7) Router 10.

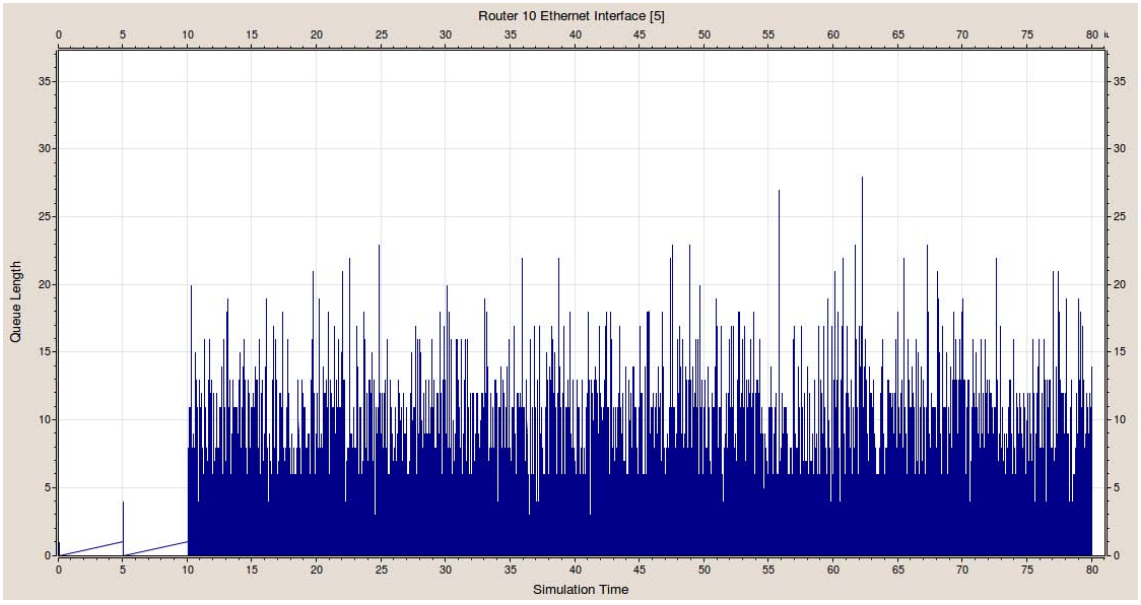


(a)

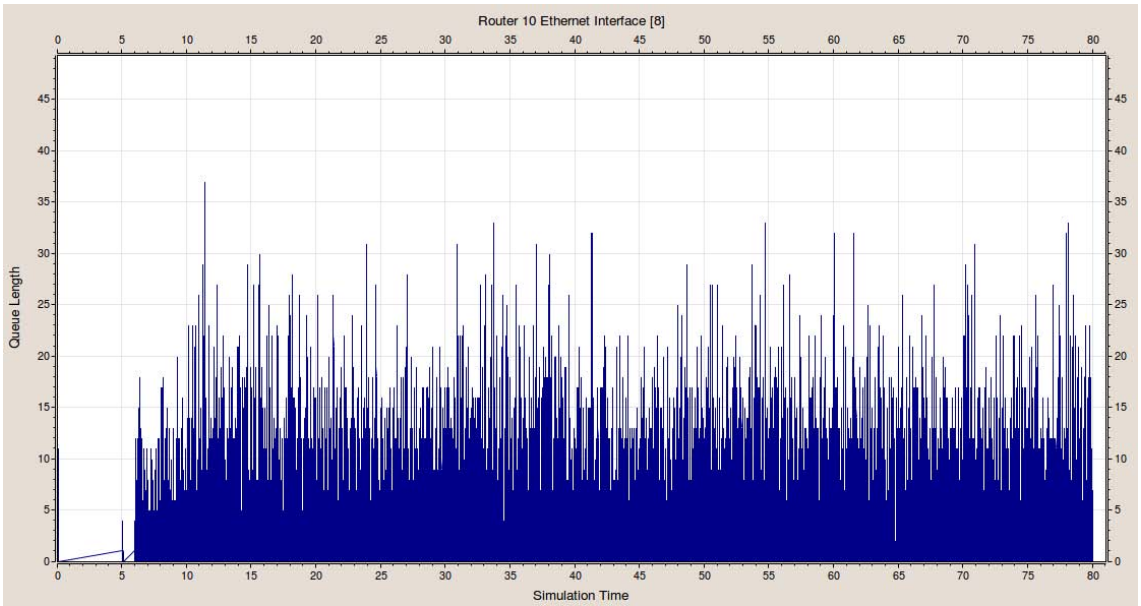


(b)

Figure 6.19: 12-node catastrophic network with constant UDP traffic at medium load. (a) Ethernet Interface(4) Router 1, (b) Ethernet Interface(3) Router 7.

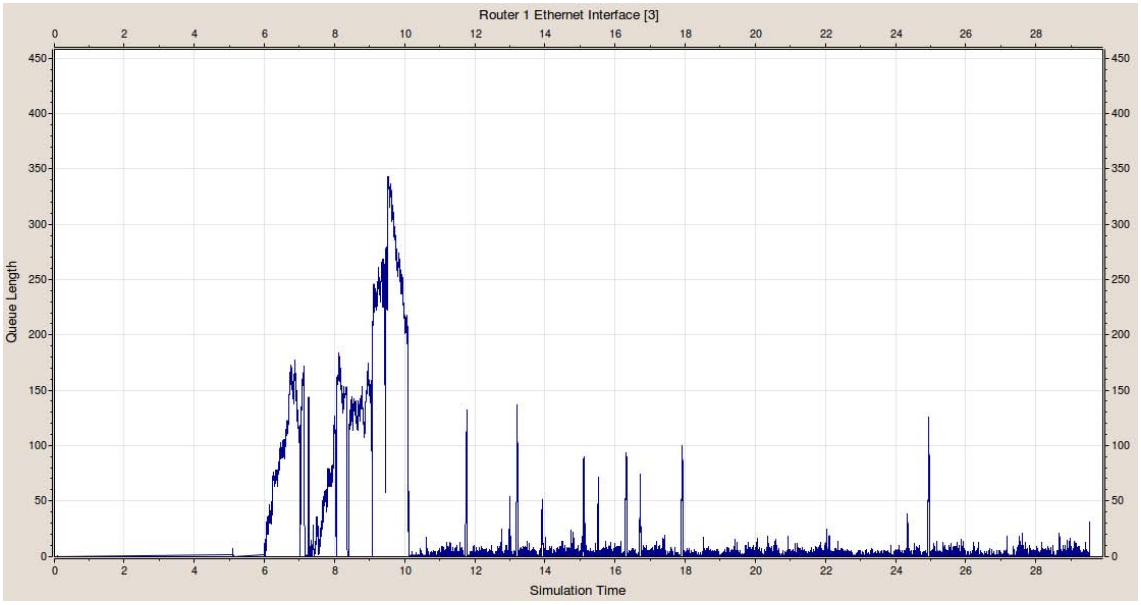


(a)

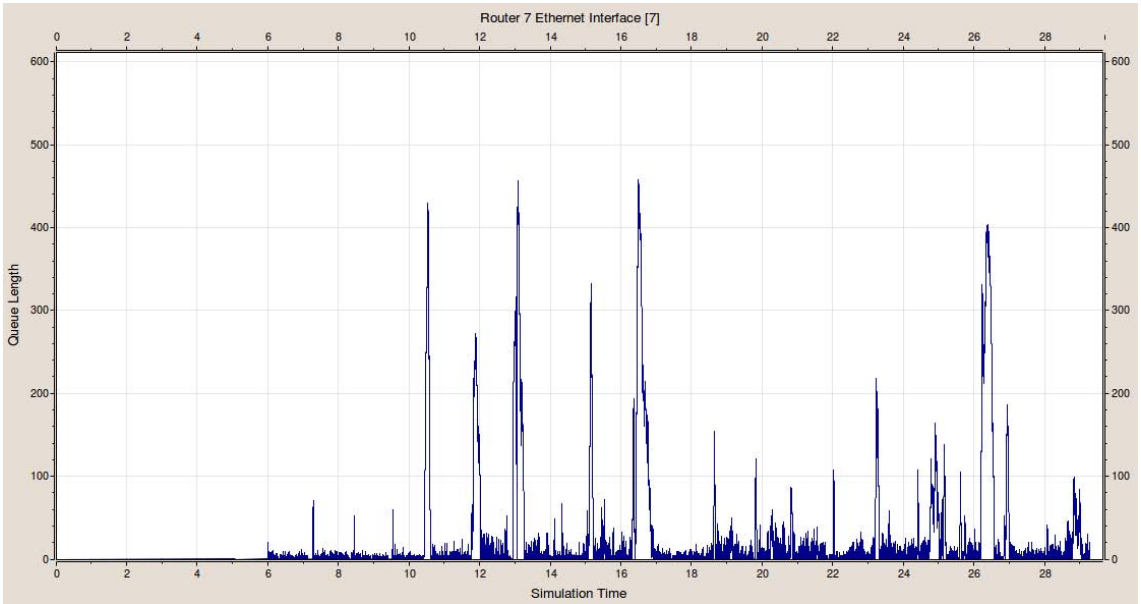


(b)

Figure 6.20: 12-node catastrophic network with constant UDP traffic at medium load. (a) Ethernet Interface(5) Router 10, (b) Ethernet Interface(8) Router 10.

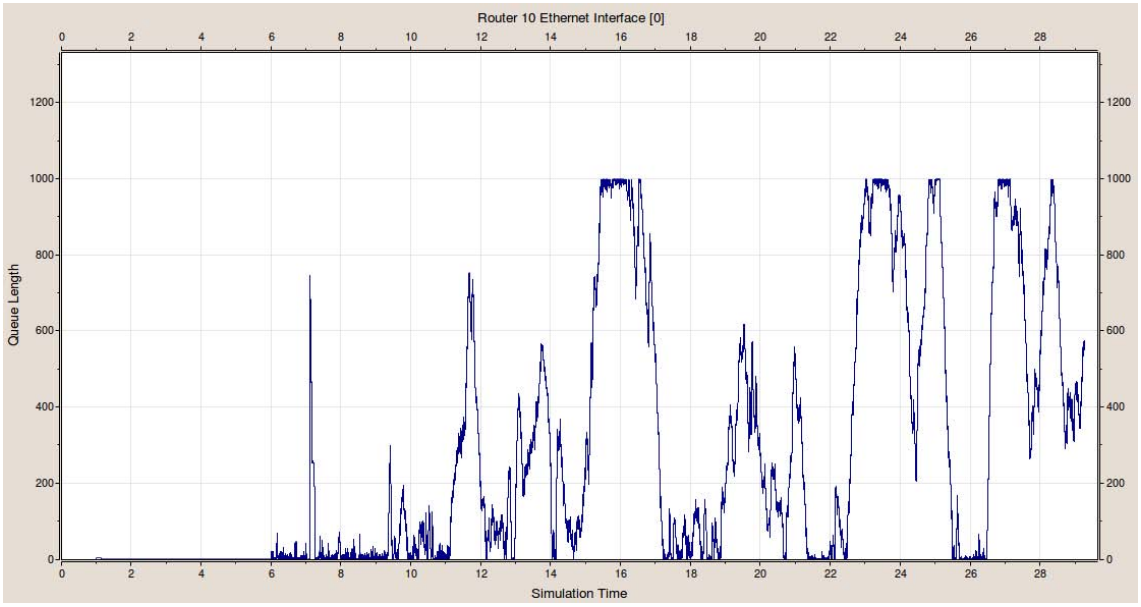


(a)

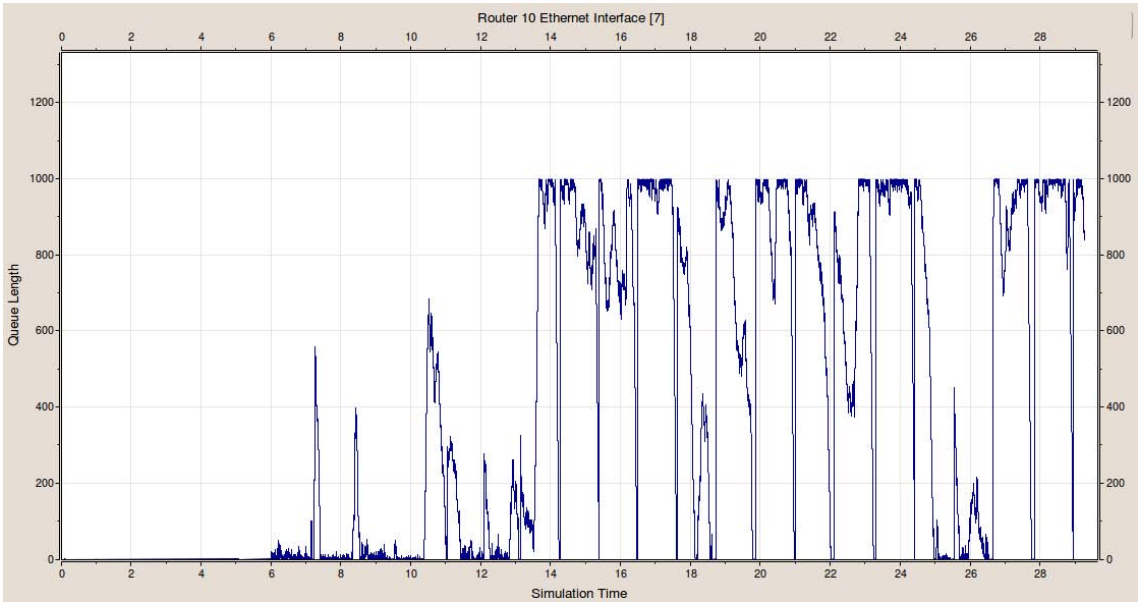


(b)

Figure 6.21: 12-node catastrophic network with constant UDP traffic at high load. (a) Ethernet Interface(3) Router 1, (b) Ethernet Interface(7) Router 7.



(a)



(b)

Figure 6.22: 12-node catastrophic network with constant UDP traffic at high load. (a) Ethernet Interface(0) Router 10, (b) Ethernet Interface(7) Router 10.

6.5 Conclusion

By introducing the process and building blocks of simulation, we first explain the network simulation models. Then different simulation tools are compared and we choose to use OMNet++ the open source C++ based network simulation platform. Afterwards, the network scenarios and simulation setup are explained in detail. Finally, we analyze the queue status of routers and hosts in three different networks: six-node star-shape network, nine-node catastrophic network and twelve-node catastrophic network. Although we connect only one host to a router as the source and destination of data flows, it has four UDP applications running at the same time simulating the aggregation of traffic from LAN hosts. The routers simulated are also not fully functioned real routers, where many detail protocols and router features are omitted. For example, there is no Access Control List configured on the routers, making their process of routers much faster. But the traffic flow put on the routers are close to the capacity of real routers. From Cisco System's router performance and capacity document, small branch router 1800 series and 2600 series have the performance capacity of 35.84Mbps. Medium branch router 3800 and edge router 7200 could handle up to 256Mbps and 1Gbps. And only the very expensive ISP carrier routers 10000, 12000, CRS-1 have the capacity of 10 to 40Gbps. But only small and medium branch routers would be actually used to run Interior Gateway Protocols (OSPF, EIGRP, and IS-IS). So the maximum capacity of these routers are similar to the ones simulated with traffic from 2Mbps upto almost 200Mbps.

Last but not least, the simulation proves that the catastrophic network do fail in the sequence as it is designed. The routers designed to congest first do experience much heavier traffic load than other routers. It would be ideal for us to build a test-bed using physical routers to test the topology. Due to the limitation of time, funding and other resources, we strongly suggest this as the future work of this thesis. But the relationship between the queues of the routers, routers and traffic is not so simple. A further detail analysis of the correlations queues of different routers in the network could be a potential topic for further future study on the catastrophic networks.

Chapter 7

Conclusions and Future Work

7.1 Aim of the research

The research in this thesis concentrates on the generation of catastrophic network topologies concerning both static and dynamic network properties. Examples in the thesis show that various mission critical networks, for example communication systems and electricity transmission grids, are susceptible to catastrophic failures. In many cases, this kind of failure follows an avalanche-like breakdown of the network's components or cascade failures. In these networks the traffic and the topology are coupled by the routing mechanisms either using software or hardware. And all network components have their functioning capacity over which they will fail to operate properly or fail to operate at all. If a network component failed due to various reasons, its load would be redistributed by the routing mechanism. Other components might consequently be overloaded and fail. This failure can sometimes propagates throughout the entire network, causing a collapse of the whole critical infrastructure.

From studies of catastrophic failures in different technological networks, the consensus is that the occurrence of a catastrophe is due to the interaction between the connectivity and the dynamical properties of the network's elements. Currently we still cannot predict whether a network will fail catastrophically or not by simply examining its topology. Although previous studies introduced in this thesis have shown various way to study cascade behaviour of networks, some

questions have not been answered: what causes cascade failures; can we replicate these failures and study their properties?

7.2 Main Results

In our research, we begin with an investigation into the relationship between network dynamic and structural properties. We find out that congestion is related to the traffic and topology properties of the concerning network. The critical load when the network congests could be predicted based on several topological properties. The critical load is also an invariant which does not depend on the network size and topology type.

Betweenness centrality is introduced and used to indicate the importance of a network node, considering the traffic load and using the shortest-path going through the node. Only analyzing existing networks is not enough to study the breakdown of critical man-made networks. A simple mechanism for building networks that are designed to fail catastrophically has been presented. The network is built by simulating the cascade in reverse. Starting with a small "seed" networks, the catastrophic network is built step by step. At each building step the network is segmented into three subnetworks. It is to make sure the congestion node is the first to congest in the network at the current step and the congestion load needs to be larger than that of the previous step. A random search method has been introduced at each step in search of topology that meets the conditions. But the random search is not very efficient and effective in finding the target catastrophic networks.

To optimize the method to grow catastrophic networks, we then presented a new method based on the "branch and bound" approach, to speed up the construction of catastrophic networks. It searches for network topology solutions systematically and keep the network load close to the original congestion load of the seed network. To limit the possible topology solution and remove the unwanted solutions, the upper limit of the size of the subnetworks connecting to the congestion node is obtained. A lookup table was built including all possible subnetwork topologies. We also introduced a tree structure to analyse the links between network from previous step and congestion node. Thus we have precise

control over the critical load that produces the cascade and the number of steps the cascade has. Analysis of the catastrophic networks generated also shows that networks built using random method are close to random networks. But networks built using branch and bound method have their own properties. The congestion nodes that are designed to fail at each growing step have different degree and betweenness centrality value from other nodes in the network. The invariant is also found and indicates that this type of catastrophic networks can be very large and have a predetermined degree distribution.

To verify if our algorithmic approach to build catastrophic networks can reflect real situations, we evaluate the performance of a small catastrophic network. The simulation scenarios proved that catastrophic networks do fail in the sequence as designed. The router designed to congest first does experience much heavier traffic load than other routers in the network. Although the simulation has many limitations, it gives strong evidence that some real-world networks might have fail in the same way, such as the Gmail server network.

7.3 Conclusion

- We have developed a growth algorithm to build catastrophic network topologies based on theoretical analysis. This family of networks would suffer cascade failures over the critical network load. We have successfully constructed catastrophic networks using our proposed methods. Simple catastrophic networks have also been simulated using router network scenario built with OSPF routing protocol. Simulations prove that they do fail according our estimation using betweenness centrality for traffic load.

- The growth algorithm can be used to build catastrophic networks successfully. Two methods are proposed: random method and branch and bound method. From the results, there are at least two kinds of catastrophic networks. There is a family of catastrophic networks that have a scale invariant. Hence it is possible to predict the behaviour of large networks by studying a much smaller network. This also gives the opportunities to construct a larger size catastrophic networks.

- We still don't know how to fully characterize the catastrophic networks. But no previous studies have shown a systematic study of the catastrophic properties of networks, we believe with the work in this thesis we could do more to understand these networks better.

7.4 Future Work

- Currently the catastrophic networks generated cascade down to the original seed network, typically a small star-shape network. This is not a normal cascade failure in real-world networks, where the tendency is for the network to break down into disconnected subnetworks. To account for this future research could try to use a number of cores simultaneously in generating networks.

- The Internet Router-level network is not such kind of homogenous network, but network with different types of routers and link capacities and also a tier structure (briefly discussed in Subsection 4.4.4). A heterogeneous cascading network model could be built to create networks with routers of different service capacities and links with different bandwidth. This will also involve using more complicated routing mechanisms such as weighted shortest path. For example OSPF uses interface bandwidth parameter as weight, EIGRP uses the combination of bandwidth, delay and reliability interface parameters, and BGP uses more policy based complex route calculation methods.

- It is beneficial to further investigate that if the catastrophic breakdown of the Internet or other complex networks are with constant propagating speed ($acceleration = 0$) or with increasing propagating speed (constant acceleration but positive). For the catastrophic model developed, the propagating speed is almost constant, because it is designed as a multi-stage cascading failure with limited nodes in each stage.

- The real-world networks' optimization should be studied to see how we can change the self-propagation of failures to self-healing. Or is there any

optimization method could be carried out so that a cascading failure can be prevented? Then we can produce guideline for network building.

- In most of the cases, catastrophic failures are unwanted and efforts are made to prevent their occurrence. However, there are circumstances in which this property is desirable. In vehicle and shop windows, for example, tempered glass is used, partly because it is stronger, but also because it has the property of shattering into much safer small pieces when broken. In cases like this catastrophic failures might be seen as being engineered into the material. A cascade of disconnection of some subnetwork could then prevent virus from spreading out to the whole network or prevent an intruder from hacking the whole network system. We can sometimes make use of the catastrophic properties of networks.

- The simulations of catastrophic networks suggested that under very high packet generation rate traffic on the routers would experience significant fluctuation. As we use betweenness centrality to approximation the traffic load of a router, it only considers the average traffic. We could further improve the estimation model by also considering the traffic fluctuations.

- To predict the catastrophic failure, we can try to understand the correlation between different network traffic flows. One can always use real-time data from network analysis softwares, i.e. Cisco Netflow and NetScout network probes. It is quite useful if network congestion and failures could be investigated through a dynamical way where we can sent out real time alarms. From previous research, traffic flow cross-correlations are calculated to examine the relationship between different flows from node to node [129; 130; 131]. The correlation shows that most of the traffic flows are related to each other through the queues in the network. A network of queues model are studied but yet to find out the solution to predict and analyse the relationship between traffic correlation, traffic density, and topology.

Appendix A

Catastrophic Network Topologies and OMNET++ Simulation Setup

A.1 Catastrophic Network Topology

In Chapter 4 we proposed a random growth method to generate catastrophic networks using a small seed network. In this section we are going to show more results of the random growth method using other seed networks rather than star-shape networks. It is to check if the change of seed network would affect the catastrophic network generated. For example, would the network built has a larger size or not. Four different seed networks introduced in Chapter 4 are used: two-stars shape network, three-stars shape network, chain shape network and random network.

From Figure A.1, the catastrophic networks grown with two-stars shape seed could reach the size of 45 nodes. They are not large size networks and the degree distributions are similar to those of random networks. The resulting catastrophic networks grown from three-stars seed network could be as large as 110 nodes, Figure A.2. It might be due to the larger size and more complex connectivities of the seed network that make it easier to grow catastrophic networks. But their maximum betweenness centrality of nodes in the network do not follow the upper boundaries closely. With a chain shape seed network, the catastrophic

networks can be even larger with 150 nodes as can be seen in Figure A.3. The degree distributions show us that there are two types of nodes in the network: nodes with degree 4 to 5 and nodes with degree 10 to 11. We believe that the nodes with higher degrees are the congestion nodes while the nodes with lower degrees are nodes added in each growing steps. Finally, we can see from Figure A.4, networks grow from the random seed network have their congestion node betweenness centralities approaching the lower boundary very quickly. This means the growth run out of possible solutions and have to increase the traffic generation rate a lot to match the conditions.

Regarding the results from using different seed networks, no matter how we could build the catastrophic network larger or smaller, there is one thing in common. The betweenness centralities deviate from the upper boundary following the growth of the network. This is why we try to introduce a better method in Chapter 5, the branch and bound method.

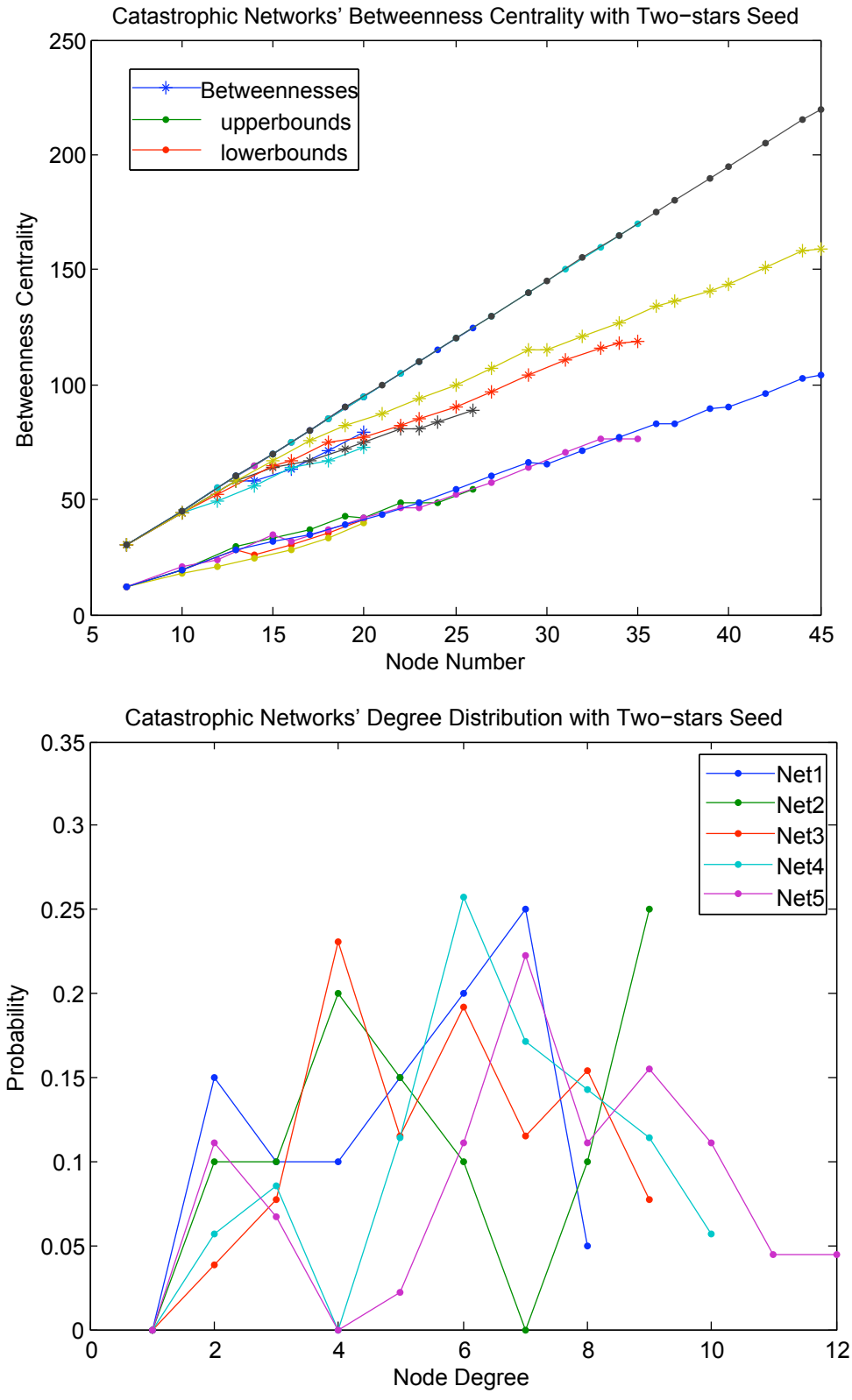


Figure A.1: Catastrophic Networks Built using Two-stars Shape Seed Network

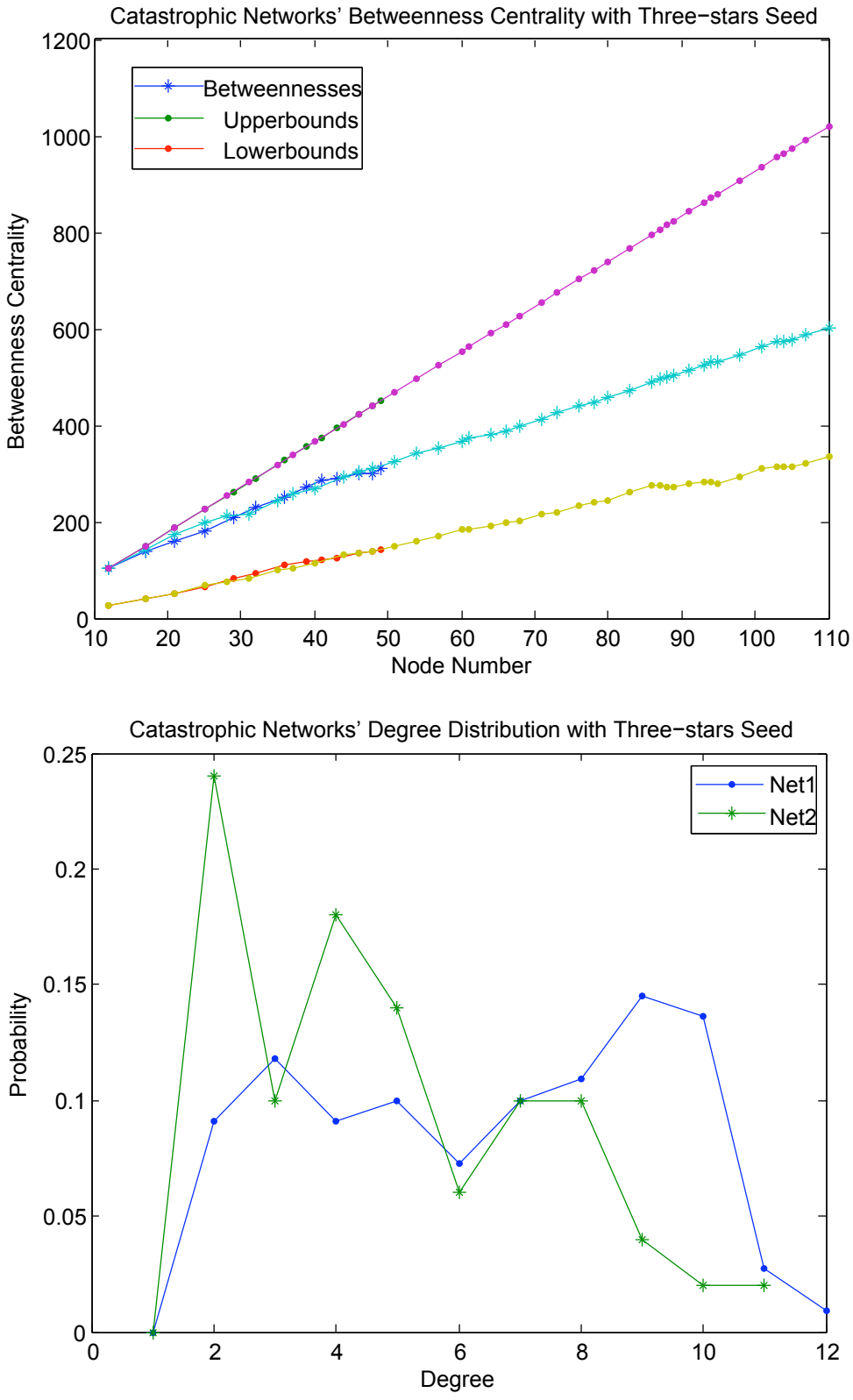


Figure A.2: Catastrophic Networks Built using Three-stars Shape Seed Network

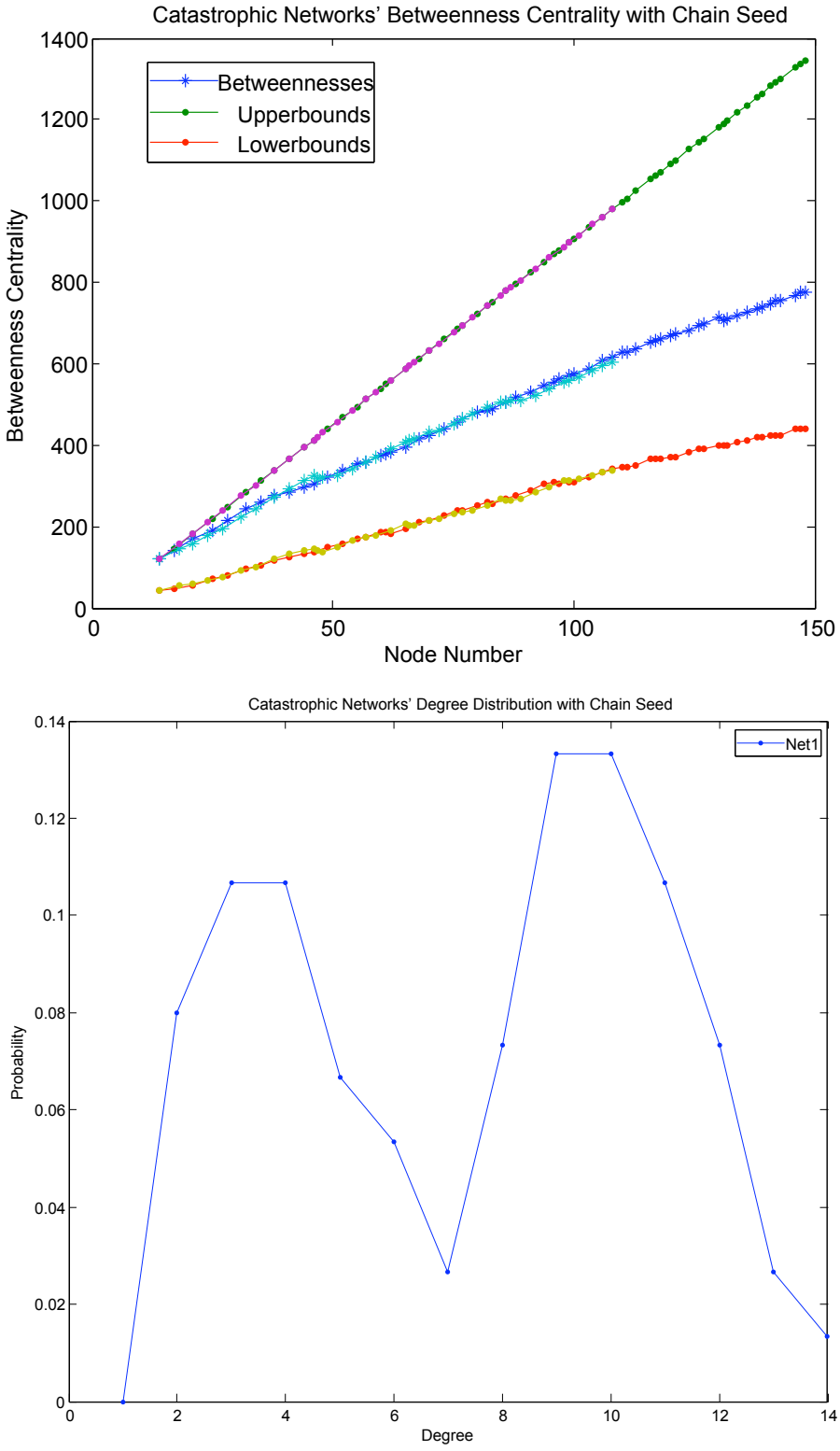


Figure A.3: Catastrophic Networks Built using Chain Shape Seed Network

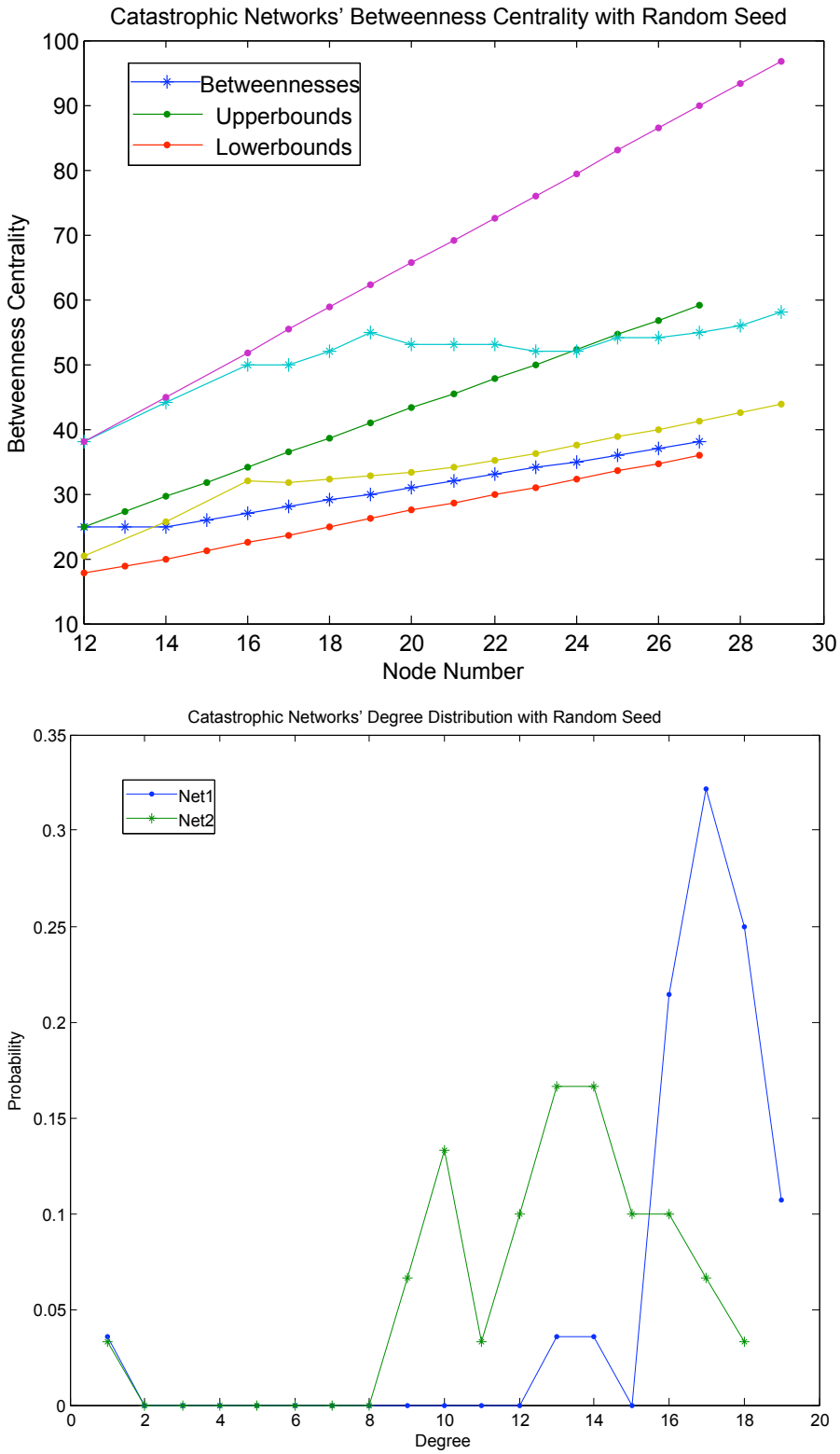


Figure A.4: Catastrophic Networks Built using Random Seed Network

A.2 OMNET++ Simulation Setup

This section includes the OMNET++ Simulation scenario setup using INET framework. Here we show the setup of the six-node star-shape network. Firstly the host IP configuration is shown with Class C private address, where it uses a default route to get to the directly connected router. Router IP configuration is more complicated involving the setup of each router interfaces. The router has an Ethernet interface with the Local Area Network (LAN) where the host belongs to. Router also has point-to-point connections to other routers using Ethernet interfaces. The OSPF routing information update Link State Advertisement (LSA) multicast address is also setup on the router point-to-point interfaces.

The network configuration example is introduced. It first loads the host and router configuration files. Then the application running on each host is set as the UDP basic application. We create four instance of the UDP basic applications with UDP local and destination port number, message length and message frequency. Finally the ARP (address resolution protocol) time out is set as 1 second by default.

In the final subsection, the OSPF configuration is introduced. It is similar to the real router OSPF process setup but we just need one file for the whole network. In that, it includes the OSPF area statement [1] and network declaration in it. It also contains he OSPF configuration for each router in the network which is vital for OSPF router neighbor discovery, shortest path calculation and routing information update. For example, Area ID, Interface Cost, Retransmission Interval, Hello Interval, Router Dead Interval, Authentication Type and key.

A.2.1 Host IP Configuration Example

ifconfig:
ethernet card 0 of host - connected to N1
name: eth0
inetaddr: 192.168.1.2
Mask: 255.255.255.0

[1] We just use a single area, area 0 in the simulation scenario.

MTU: 1500
Metric: 1
BROADCAST MULTICAST
ifconfigend.

route:
default: * 0.0.0.0 H 0 eth0
routeend.

A.2.2 Router IP Configuration Example

ifconfig:
ethernet card 0 of router - connected to N1
name: eth0
inetaddr: 192.168.1.1
Mask: 255.255.255.0
Groups: 224.0.0.5
MTU: 1500
Metric: 1
BROADCAST MULTICAST

ethernet card 1 of router - connected to R2
name: eth1
inetaddr: 192.168.60.61
Mask: 255.255.255.252
Groups: 224.0.0.5
MTU: 1500
Metric: 2
POINTTOPOINT MULTICAST

ethernet card 2 of router - connected to R3
name: eth2
inetaddr: 192.168.60.62

Mask: 255.255.255.252
Groups: 224.0.0.5
MTU: 1500
Metric: 2
POINTTOPOINT MULTICAST

ethernet card 3 of router - connected to R4
name: eth3
inetaddr: 192.168.60.63
Mask: 255.255.255.252
Groups: 224.0.0.5
MTU: 1500
Metric: 2
POINTTOPOINT MULTICAST

ethernet card 4 of router - connected to R5
name: eth4
inetaddr: 192.168.60.64
Mask: 255.255.255.252
Groups: 224.0.0.5
MTU: 1500
Metric: 2
POINTTOPOINT MULTICAST

ethernet card 5 of router - connected to R6
name: eth5
inetaddr: 192.168.60.65
Mask: 255.255.255.252
Groups: 224.0.0.5
MTU: 1500
Metric: 2
POINTTOPOINT MULTICAST
ifconfigend.

route:
224.0.0.0 * 240.0.0.0 H 0 eth0
224.0.0.0 * 240.0.0.0 H 0 eth1
224.0.0.0 * 240.0.0.0 H 0 eth2
224.0.0.0 * 240.0.0.0 H 0 eth3
224.0.0.0 * 240.0.0.0 H 0 eth4
224.0.0.0 * 240.0.0.0 H 0 eth5
routeend.

A.2.3 Network Configuration Example

OSPFv2 test network.
General
description = "Simple test"
network = SimpleTest
tkenv-plugin-path = ../../../etc/plugins
**.ospf.ospfConfigFile = "ASConfig.xml"
**.R1.routingFile = "R1.irt"
**.R2.routingFile = "R2.irt"
**.R3.routingFile = "R3.irt"
**.R4.routingFile = "R4.irt"
**.R5.routingFile = "R5.irt"
**.R6.routingFile = "R6.irt"
**.H1.routingFile = "H1.irt"
**.H2.routingFile = "H2.irt"
**.H3.routingFile = "H3.irt"
**.H4.routingFile = "H4.irt"
**.H5.routingFile = "H5.irt"
**.H6.routingFile = "H6.irt"

**.numUdpApps = 1
**.udpAppType = "UDPBasicApp"

**.udpApp[0].localPort = 1234
**.udpApp[0].destPort = 1234
**.udpApp[0].messageLength = 1024 bytes
**.udpApp[0].messageFreq = 0.01s

**.udpApp[1].localPort = 1235
**.udpApp[1].destPort = 1235
**.udpApp[1].messageLength = 1024 bytes
**.udpApp[1].messageFreq = 0.1s

**.udpApp[2].localPort = 1236
**.udpApp[2].destPort = 1236
**.udpApp[2].messageLength = 1024 bytes
**.udpApp[2].messageFreq = 0.1s

**.udpApp[3].localPort = 1237
**.udpApp[3].destPort = 1237
**.udpApp[3].messageLength = 1024 bytes
**.udpApp[3].messageFreq = 0.1s

**.H1.udpApp[0].destAddresses="192.168.6.2 192.168.2.2 192.168.3.2 192.168.4.2 192.168.5.2"
**.H2.udpApp[0].destAddresses="192.168.6.2 192.168.1.2 192.168.3.2 192.168.4.2 192.168.5.2"
**.H3.udpApp[0].destAddresses="192.168.6.2 192.168.1.2 192.168.2.2 192.168.4.2 192.168.5.2"
**.H4.udpApp[0].destAddresses="192.168.6.2 192.168.1.2 192.168.2.2 192.168.3.2 192.168.5.2"
**.H5.udpApp[0].destAddresses="192.168.6.2 192.168.1.2 192.168.2.2 192.168.3.2 192.168.4.2"
**.H6.udpApp[0].destAddresses="192.168.1.2 192.168.2.2 192.168.3.2 192.168.4.2 192.168.5.2"

**.H1.udpApp[1].destAddresses="192.168.6.2 192.168.2.2 192.168.3.2 192.168.4.2 192.168.5.2"
**.H2.udpApp[1].destAddresses="192.168.6.2 192.168.1.2 192.168.3.2 192.168.4.2 192.168.5.2"
**.H3.udpApp[1].destAddresses="192.168.6.2 192.168.1.2 192.168.2.2 192.168.4.2 192.168.5.2"
**.H4.udpApp[1].destAddresses="192.168.6.2 192.168.1.2 192.168.2.2 192.168.3.2 192.168.5.2"
**.H5.udpApp[1].destAddresses="192.168.6.2 192.168.1.2 192.168.2.2 192.168.3.2 192.168.4.2"
**.H6.udpApp[1].destAddresses="192.168.1.2 192.168.2.2 192.168.3.2 192.168.4.2 192.168.5.2"

**.H1.udpApp[2].destAddresses="192.168.6.2 192.168.2.2 192.168.3.2 192.168.4.2 192.168.5.2"
**.H2.udpApp[2].destAddresses="192.168.6.2 192.168.1.2 192.168.3.2 192.168.4.2 192.168.5.2"
**.H3.udpApp[2].destAddresses="192.168.6.2 192.168.1.2 192.168.2.2 192.168.4.2 192.168.5.2"
**.H4.udpApp[2].destAddresses="192.168.6.2 192.168.1.2 192.168.2.2 192.168.3.2 192.168.5.2"
**.H5.udpApp[2].destAddresses="192.168.6.2 192.168.1.2 192.168.2.2 192.168.3.2 192.168.4.2"
**.H6.udpApp[2].destAddresses="192.168.1.2 192.168.2.2 192.168.3.2 192.168.4.2 192.168.5.2"

**.H1.udpApp[3].destAddresses="192.168.6.2 192.168.2.2 192.168.3.2 192.168.4.2 192.168.5.2"
**.H2.udpApp[3].destAddresses="192.168.6.2 192.168.1.2 192.168.3.2 192.168.4.2 192.168.5.2"
**.H3.udpApp[3].destAddresses="192.168.6.2 192.168.1.2 192.168.2.2 192.168.4.2 192.168.5.2"

**.H4.udpApp[3].destAddresses=”192.168.6.2 192.168.1.2 192.168.2.2 192.168.3.2 192.168.5.2”
**.H5.udpApp[3].destAddresses=”192.168.6.2 192.168.1.2 192.168.2.2 192.168.3.2 192.168.4.2”
**.H6.udpApp[3].destAddresses=”192.168.1.2 192.168.2.2 192.168.3.2 192.168.4.2 192.168.5.2”

**.arp.cacheTimeout = 1s

A.2.4 OSPF Configuration Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<OSPFASConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="OSPF.xsd">
  <!-- Areas -->
  <Area id="0.0.0.0">
    <AddressRange>
      <Address>192.168.1.0</Address>
      <Mask>255.255.255.0</Mask>
      <Status>Advertise</Status>
    </AddressRange>
    <AddressRange>
      <Address>192.168.2.0</Address>
      <Mask>255.255.255.0</Mask>
      <Status>Advertise</Status>
    </AddressRange>
    <AddressRange>
      <Address>192.168.3.0</Address>
      <Mask>255.255.255.0</Mask>
      <Status>Advertise</Status>
    </AddressRange>
    <AddressRange>
      <Address>192.168.4.0</Address>
      <Mask>255.255.255.0</Mask>
```

```
    <Status>Advertise</Status>
  </AddressRange>
  <AddressRange>
    <Address>192.168.5.0</Address>
    <Mask>255.255.255.0</Mask>
    <Status>Advertise</Status>
  </AddressRange>
  <AddressRange>
    <Address>192.168.6.0</Address>
    <Mask>255.255.255.0</Mask>
    <Status>Advertise</Status>
  </AddressRange>
  <AddressRange>
    <Address>192.168.60.61</Address>
    <Mask>255.255.255.255</Mask>
    <Status>Advertise</Status>
  </AddressRange>
  <AddressRange>
    <Address>192.168.60.62</Address>
    <Mask>255.255.255.255</Mask>
    <Status>Advertise</Status>
  </AddressRange>
  <AddressRange>
    <Address>192.168.60.63</Address>
    <Mask>255.255.255.255</Mask>
    <Status>Advertise</Status>
  </AddressRange>
  <AddressRange>
    <Address>192.168.60.64</Address>
    <Mask>255.255.255.255</Mask>
    <Status>Advertise</Status>
  </AddressRange>
  <AddressRange>
    <Address>192.168.60.65</Address>
```

```
      <Mask>255.255.255.255</Mask>
      <Status>Advertise</Status>
    </AddressRange>
    <AddressRange>
      <Address>192.168.60.66</Address>
      <Mask>255.255.255.255</Mask>
      <Status>Advertise</Status>
    </AddressRange>
    <AddressRange>
      <Address>192.168.60.67</Address>
      <Mask>255.255.255.255</Mask>
      <Status>Advertise</Status>
    </AddressRange>
    <AddressRange>
      <Address>192.168.60.68</Address>
      <Mask>255.255.255.255</Mask>
      <Status>Advertise</Status>
    </AddressRange>
    <AddressRange>
      <Address>192.168.60.69</Address>
      <Mask>255.255.255.255</Mask>
      <Status>Advertise</Status>
    </AddressRange>
    <AddressRange>
      <Address>192.168.60.70</Address>
      <Mask>255.255.255.255</Mask>
      <Status>Advertise</Status>
    </AddressRange>
 </Area>
 <!-- Routers -->
  <Router id="192.168.60.65"> <!-- R1 -->
    <RFC1583Compatible />
    <BroadcastInterface ifName="eth0">
      <AreaID>0.0.0.0</AreaID>
```

```
    <InterfaceOutputCost>1</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <RouterPriority>1</RouterPriority>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </BroadcastInterface>
  <PointToPointInterface ifName="eth1">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>2</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </PointToPointInterface>
  <PointToPointInterface ifName="eth2">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>2</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </PointToPointInterface>
  <PointToPointInterface ifName="eth3">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>2</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
```

```
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </PointToPointInterface>
  <PointToPointInterface ifName="eth4">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>2</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </PointToPointInterface>
  <PointToPointInterface ifName="eth5">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>2</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </PointToPointInterface>
</Router>
<Router id="192.168.60.66"> <!-- R2 -->
  <RFC1583Compatible />
  <BroadcastInterface ifName="eth0">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>1</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <RouterPriority>2</RouterPriority>
```

```
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </BroadcastInterface>
  <PointToPointInterface ifName="eth1">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>2</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </PointToPointInterface>
</Router>
<Router id="192.168.60.67"> <!-- R3 -->
  <RFC1583Compatible />
  <BroadcastInterface ifName="eth0">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>1</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <RouterPriority>2</RouterPriority>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </BroadcastInterface>
  <PointToPointInterface ifName="eth1">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>2</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
```

```
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </PointToPointInterface>
</Router>
<Router id="192.168.60.68"> <!-- R4 -->
  <RFC1583Compatible />
  <BroadcastInterface ifName="eth0">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>1</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <RouterPriority>2</RouterPriority>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </BroadcastInterface>
  <PointToPointInterface ifName="eth1">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>2</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </PointToPointInterface>
</Router>
<Router id="192.168.60.69"> <!-- R5 -->
  <RFC1583Compatible />
  <BroadcastInterface ifName="eth0">
    <AreaID>0.0.0.0</AreaID>
```

```
    <InterfaceOutputCost>1</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <RouterPriority>2</RouterPriority>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </BroadcastInterface>
  <PointToPointInterface ifName="eth1">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>2</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </PointToPointInterface>
</Router>
<Router id="192.168.60.70"> <!-- R6 -->
  <RFC1583Compatible />
  <PointToPointInterface ifName="eth1">
    <AreaID>0.0.0.0</AreaID>
    <InterfaceOutputCost>2</InterfaceOutputCost>
    <RetransmissionInterval>5</RetransmissionInterval>
    <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
    <HelloInterval>10</HelloInterval>
    <RouterDeadInterval>40</RouterDeadInterval>
    <AuthenticationType>NullType</AuthenticationType>
    <AuthenticationKey>0x00</AuthenticationKey>
  </PointToPointInterface>
  <BroadcastInterface ifName="eth0">
    <AreaID>0.0.0.0</AreaID>
```

```
      <InterfaceOutputCost>1</InterfaceOutputCost>
      <RetransmissionInterval>5</RetransmissionInterval>
      <InterfaceTransmissionDelay>1</InterfaceTransmissionDelay>
      <RouterPriority>2</RouterPriority>
      <HelloInterval>10</HelloInterval>
      <RouterDeadInterval>40</RouterDeadInterval>
      <AuthenticationType>NullType</AuthenticationType>
      <AuthenticationKey>0x00</AuthenticationKey>
    </BroadcastInterface>
  </Router>
</OSPFASConfig>
```

A.3 Source Code of Random Growing and Branch & Bound Method

A.3.1 Network.h

```
#ifndef NETWORK
#define NETWORK
#include "RandomNumber.h"
#include <vector>
#include <stack>
#include <list>
#include <queue>
#include <algorithm>
#include <iostream>
#include <fstream>
#include <stdlib.h>
using namespace std;

class BC{
vector<double> Cb;
vector<int> d;
vector<double> sigma;
```

```
vector<double> delta;
stack<int> S;
vector<queue<int>> P;
queue<int> Q;
vector<vector<int>> adjmatrix;
queue<int> a;
int V,w,activenode,sum,max,maxnode;
public:
BC(int v,vector<vector<int>> adj):Cb(v),d(v),sigma(v),P(v),delta(v){
adjmatrix = adj;
activenode = V = (int)adjmatrix.size();}
void InitialBC(int v,vector<vector<int>> adj){
adjmatrix.clear();
adjmatrix = adj;
activenode = V = (int)adjmatrix.size();
Cb.assign(V,0);
d.assign(V,0);
sigma.assign(V,0);
delta.assign(V,0);
P.clear();
P.assign(V,a);
while(!S.empty()){
S.pop();}}
void searchS(){
for(int i=0;i<V;i++){
while(!S.empty()){S.pop();}
while(!Q.empty()){Q.pop();}
for(int j=0;j<V;j++){
sigma[j]=0;
d[j]=-1;
while(!P[j].empty()){P[j].pop();}}
sigma[i]=1;
d[i]=0;
Q.push(i);
```

```
while(!Q.empty()){
int v = Q.front();
Q.pop();
S.push(v);
for(int k=0;k<V;k++){
if(adjmatrix[v][k]!=0){
//cout<<endl<<v<<" "<<k;
if(d[k] < 0){
Q.push(k);
d[k]=d[v]+1;
}//if found for the first time
if(d[k] == d[v]+1){
sigma[k] = sigma[k]+sigma[v];
P[k].push(v);
}//end if the distance meets
}
}
}
for(int l=0;l<V;l++){
delta[l]=0;}
while(!S.empty()){
w = S.top();
S.pop();
while(!P[w].empty()){
int d = P[w].front();
delta[d] = delta[d]+(((sigma[d]+0.0)/(sigma[w]+0.0))*(1.0+delta[w]));
P[w].pop();}
if(w!=i){
Cb[w]=Cb[w]+delta[w];}
}
}
for(int i = 0;i<activenode;i++){
Cb[i]+=(activenode-1);}
sum = 0;
```

```
max = 0;
maxnode = 0;
for( int i = 0;i<V;i++){
if(Cb[i]>max){max =(int) Cb[i]; maxnode = i;}
sum=sum + (int)Cb[i];
}}

   void printS(){
ofstream fout("betweenness.dat",ios::ate);
if(!fout)
{   cerr<<"Error Opening File!\n";
exit(0);//then exit}
fout<<"Betweenness Centrality: ";
for(unsigned int i = 0;i<Cb.size();i++){
fout<<endl<<" Node "<<i<<" "<<Cb[i];}}
int Getmax(){return max;}
int Getmaxnode(){return maxnode;}
int GetCB(int number){return (int)Cb[number];}};

class nodew{
  public:
int num,w;
nodew(int node,int weight):num(node),w(weight){}
int Nw(){return num;}
int W(){return w;}
bool bigger( nodew& n1, nodew& n2){
        return( (n1.w<n2.w) && (n1.num<n2.num) );}
bool equal( nodew& n1, nodew& n2){
        return ((n1.w==n2.w)&&(n1.num<n2.num)) ;}
};

class myDijkstra{
vector<int> d;
vector<int> p;
```

```
vector<int> color;
priority_queue<nodew*> Q;
vector<vector<int>> adjmatrix;
int V,activenode;
public:
myDijkstra(int v, int ActiveNode,
    vector<vector<int>> adj):V(v),activenode(ActiveNode){
  for(int i=0;i<v;i++){
color.push_back(0);
d.push_back(1000);
p.push_back(i);}
  cout<<endl<<"myDijkstra object created";
}
 void Initial(int v,int ActiveNode,vector<vector<int>> adj){
V = v;
activenode = ActiveNode;
adjmatrix = adj;
for(int i=0;i<v;i++){
color.push_back(0);
d.push_back(1000);
p.push_back(i);}
while(!Q.empty()){Q.pop();}}

void search(int s){
color[s]=1;
d[s]=0;
Q.push(new nodew(s,d[s]));
while(!Q.empty()){
int u = Q.top()->Nw();
int uw = Q.top()->W();
nodew* temp = Q.top();
Q.pop();
delete temp;
for(int j=0;j<V;j++){
```

```
if((adjmatrix[u][j]+d[u])<d[j]&&adjmatrix[u][j]!=0 ){
d[j]=adjmatrix[u][j]+d[u];
p[j]=u;
if(color[j]==0){
color[j]=1;// color change to gray, discover vertex j
Q.push(new nodew(j,d[j]));}
else if(color[j]==1){ }}
}
color[u]=2;}
}

void printS(){
  vector<int>::iterator i;
  cout<<endl<<"the parent tree"<<endl;
  for(i=p.begin();i!=p.end();i++){
cout<< *i<<" "; }
  cout<<endl<<"the d vector "<<endl;
  for(i=d.begin();i!=d.end();i++){
cout<< *i<<" ";}
}

bool connectivity(int sour){
search(sour);
for(int i = 0;i<activenode;i++){
if(d[i]==1000){
cerr<<endl<<"the two node "<<sour<<" and "<<i<< " is not connected";
return false;
break;}
else {}}
return true;}
}

class myFloyd
{  vector <vector<int>> p;
```

```
vector <vector<int>> d;
int V, activenode;
vector <vector<int>> adjmatrix;
public:
myFloyd(int v,int ActiveNode,vector <vector<int>> adj):p(v),
    d(v),activenode(ActiveNode){
V = v;
adjmatrix = adj;
for (int i = 0; i < V; i++){
p[i].assign(V, -1); d[i].assign(V, 65535); }
for (int s = 0; s < V; s++){
d[s][s] = 0;
for(int t = 0; t < V; t++){
if (adjmatrix[s][t]!=0){
p[s][t] =t;
d[s][t] = adjmatrix[s][t]; }
}}}

void Initial(int v,int ActiveNode,vector <vector<int>> adj){
V = v;
adjmatrix = adj;
activenode = ActiveNode;
for (int i = 0; i < V; i++){
p[i].assign(V, -1); d[i].assign(V, 65535); }
for (int s = 0; s < V; s++){
d[s][s] = 0;
for (int t = 0; t < V; t++){
if (adjmatrix[s][t]!=0){
p[s][t] =t; //G.edge(s, t);
d[s][t] = adjmatrix[s][t]; }}
}}

void search(){
for (int i = 0; i < V; i++){
```

```
for (int s = 0; s < V; s++){
if (p[s][i]!=-1){
for (int t = 0; t < V; t++){
if (s != t){
if (d[s][t] > d[s][i] + d[i][t]){
p[s][t] = p[s][i];
d[s][t] = d[s][i] + d[i][t];}
}
}
}
}
}
}

double Ell(){
double sum = 0.0;
int i, j;
for(i = 0; i<activenode; i++){
for(j= 0;j<activenode;j++){
sum += d[i][j];}}
double ellbar = sum/(activenode*(activenode-1));
return ellbar;}
int path(int s, int t) const
    { return p[s][t]; }
int dist(int s, int t) const
    { return d[s][t]; }
};

class Network{
int Vcnt, Ecnt,ActiveNode;
bool digraph;
char inputFilename[10];
ifstream inFile;
vector <vector <int> > adj;
```

```
vector <int> nodes, a;
BC* bc1;
myDijkstra* dijkstra1;
myFloyd* floyd1;
bool disconnected;

public:
Network();
Network(int V, bool digraph);
~Network( );
void Initial(int V, bool digraph1);
int V() const { return Vcnt; }
int Activenode() const {return ActiveNode;}
int E() const { return Ecnt; }
bool directed() const {return digraph; }
void insert(int s, int d);
void insert(int s, int d,int w);
bool deletelink(int s, int d);
void deletel(int s, int d);
void addNode( );
bool deleteNode(int n);
int edge(int v, int w) const { return adj[v][w]; }
void writefrom(vector<vector<int>> newadj,
        vector<int> newnodes,int newActiveNode,int newEcnt){
adj = newadj;
nodes = newnodes;
ActiveNode = newActiveNode;
Ecnt = newEcnt;}
int getNoNodes(){ return (int) nodes.size();}
vector<vector<int>> getAdj(){ return adj; }
vector<int> getNodes(){ return nodes; }
void IncActiveNode(){ActiveNode+=1;}
void printG();
int  getNetworkSize();
```

```
int  getadjsize();
void createGfromfile();
void RunBC();
int GetBCMax();
int GetBC(int number);
int GetBCMaxNode();
void PrintBC();
void RunFloyd();
double GetEll();
void PrintEll();};
#endif
```

A.3.2 Network.cpp

```
#include <iostream>
#include <fstream>
#include <time.h>
#include "Network.h"
Network::Network( ){ }
Network::Network(int V, bool digraph = false):
    adj(V),nodes(V),Vcnt(V),Ecnt(0),ActiveNode(0),digraph(digraph){
disconnected = false;
for (int i = 0; i < V; i++){
adj[i].assign(V, 0);
nodes.assign(V,0);}
cout<<endl<<"Simulation Network Object Created"<<endl;
}

Network::~Network(){}
void Network::Initial(int V, bool digraph1 ){
adj.assign(V,vector<int> (V,0));
nodes.assign(V,0);
Vcnt = V; Ecnt = 0;
ActiveNode = 0;
```

```
digraph = digraph1;
disconnected = false;
if(bc1!=NULL){bc1 = new BC(ActiveNode,adj);}
if(floyd1!=NULL){floyd1 = new myFloyd(V,ActiveNode,adj);}
cout<<endl<<"Simulation Network Initialized"<<endl;}

void Network::insert(int s,int d){
  int v = s, w = d;
      if (adj[v][w] == 0) Ecnt++;
  if (!digraph){adj[w][v] =adj[v][w] =1;}
  else{adj[v][w] =1;  }}

void Network::insert(int s,int d,int w){
  int v = s, x = d;
      if (adj[v][x] == 0) Ecnt++;
  if (!digraph){adj[x][v] =adj[v][x] =w;}
  else{adj[v][x] =w;}}

bool Network::deletelink(int s, int d)
{ if(dijkstra1!=NULL){
dijkstra1 = new myDijkstra(Vcnt,ActiveNode,adj);}
else{
dijkstra1->Initial(Vcnt,ActiveNode,adj);}
if(dijkstra1->connectivity(s)==false||
      dijkstra1->connectivity(d)==false){
return false;}
else{disconnected = false;
if (adj[s][d] != 0) Ecnt--;
adj[s][d] = 0;
if (!digraph) adj[d][s] = 0;
return true;}}

void Network::deletel(int s,int d){
if (adj[s][d] != 0){
```

```
Ecnt--;
adj[s][d] = 0;}
if (!digraph&&adj[d][s]!=0){
adj[d][s] = 0;}}

void Network::addNode(){
  ActiveNode += 1;
  for(int i=0;i<(int)adj.size();i++){
  adj[i].push_back(0);}
  a.clear();
  a.assign(ActiveNode,0);
      adj.push_back(a);
  nodes.push_back((ActiveNode-1));}

bool Network::deleteNode(int n){
if(floyd1!=NULL){
floyd1 = new myFloyd(Vcnt,ActiveNode,adj);}
else{floyd1->Initial(Vcnt,ActiveNode,adj);
floyd1->search();
for(int i=0;i<ActiveNode;i++)
{for(int j=i+1;j<ActiveNode;j++)
if(floyd1->dist(i,j)==65535){
disconnected = true;
break;}}
}
if(disconnected == true){
disconnected = false;
return false;}
else{
for(int i=0;i<(int)nodes.size();i++){
adj[i].erase(adj[i].begin()+3);}
adj.erase(adj.begin()+3);
nodes.erase(remove(nodes.begin(), nodes.end(), n), nodes.end());
cout<<endl<<"renumbering: "<<endl;
```

```
for(int i =0; i<(int)nodes.size();i++){
nodes[i]=i;}
return true;}}

void Network::printG(){
ofstream fout("matrix.dat",ios::ate);
if(!fout){
cerr<<"Error Opening File!\n";
exit(0);//then exit }
fout<<"********* the graph adjacent list***"<<endl;
for(int i =0;i<ActiveNode;i++){ //Vcnt
fout<<endl;
for(int j=0;j<ActiveNode;j++)//Vcnt
{fout<<adj[i][j]<<" ";}//end for}
fout<<endl<<endl<<"*******link list************"<<endl;
for(int i1 =0;i1<ActiveNode;i1++){//Vcnt
for(int j1=i1;j1<ActiveNode;j1++)//Vcnt
{ if(adj[i1][j1]!=0)
fout<<endl<<i1<<"    "<<j1<<"    "<<adj[i1][j1];}
}}

int Network::getNetworkSize(){
  char inputfile[] = "star6.dat";
  inFile.open(inputfile,ios::in);
  if (!inFile) {
    cerr<<"Can't open input file "<<inputfile<<endl;
    exit(1);}
  int num1,num2,weight;
  int min = 0;   int max = 0;
  while (!inFile.eof()) {
    inFile >> num1 >> num2>> weight;
if(num1<min||num2<min){
cerr<<endl<<"The numbers in the link list are less than zero!"<<endl;
exit(1);}
```

```
if(num1>max){max=num1;}
else if(num2>max){max=num2;}}
  inFile.close();
  if(max<=0){
cout<<endl<<"Network Size is zero!!!"<<endl;
return 0;}
  else{return (max+1);}}

void Network::createGfromfile(){
ifstream inputlink;
char inputfile[] = "star6.dat";
inputlink.open(inputfile,ios::in);
if (!inputlink) {
cerr <<"Can't open input file "<<inputfile<<endl;
exit(1);}
int num1,num2,weight;
    while (!inputlink.eof()) {
inputlink >> num1 >> num2>> weight;
insert(num1,num2,weight);}
inputlink.close();
for(int x=0;x<Vcnt;x++){
int y = 0;
while(y<Vcnt){
if(edge(x,y)!=0)
{ActiveNode+=1;
 break;}
y+=1;
}}
for(int x=0;x<ActiveNode;x++){
nodes.push_back(x);}
}

int Network::getadjsize(){
return (int)adj.size();}
```

```
void Network::RunBC(){
bc1->InitialBC(ActiveNode,adj);
bc1->searchS();
}
int Network::GetBCMax(){
return bc1->Getmax();}
int Network::GetBC(int number){
bc1->InitialBC(ActiveNode,adj);
bc1->searchS();
return bc1->GetCB(number);}
int Network::GetBCMaxNode(){
return bc1->Getmaxnode();}
void Network::PrintBC(){
bc1->printS();}
void Network::RunFloyd(){
floyd1->Initial(Vcnt,ActiveNode,adj);
floyd1->search();}
double Network::GetEll(){
return floyd1->Ell();}
void Network::PrintEll(){ }
```

A.3.3 RandomNumber.h

```
#ifndef RANDOMNUMBER_H
#define RANDOMNUMBER_H
#define N 624
#define M 397
#define MATRIX_A 0x9908b0dfUL   /* constant vector a */
#define UPPER_MASK 0x80000000UL /* most significant w-r bits */
#define LOWER_MASK 0x7fffffffUL /* least significant r bits */
class RandomNumber {
public:
RandomNumber();//constructor
~RandomNumber();//destructor
```

```
/* initializes mt[N] with a seed */
void init_genrand(unsigned long s);
/* initialize by an array with array-length */
/* init_key is the array for initializing keys */
/* key_length is its length */
void init_by_array(unsigned long init_key[], int key_length);
/* generates a random number on [0,0x7fffffff]-interval */
long genrand_int31(void);
/* generates a random number on [0,1]-real-interval */
double genrand_real1(void);
/* generates a random number on [0,1)-real-interval */
double genrand_real2(void);
/* generates a random number on (0,1)-real-interval */
double genrand_real3(void);
/* generates a random number on [0,1) with 53-bit resolution*/
double genrand_res53(void);

private:
/* the array for the state vector  */
static unsigned long mt[N];
/* mti==N+1 means mt[N] is not initialized */
static int mti;
/* generates a random number on [0,0xffffffff]-interval */
unsigned long genrand_int32(void);};
#endif
```

A.3.4 RandomNumber.cpp

```
#include <stdio.h>
#include <iostream>
#include <string.h>
#include <fstream>
#include <math.h>
#include "RandomNumber.h"
```

```
using namespace std;
int RandomNumber::mti=N+1;
unsigned long RandomNumber::mt[N];
RandomNumber::RandomNumber(){}
RandomNumber::~RandomNumber(){}

/* initializes mt[N] with a seed */
void RandomNumber::init_genrand(unsigned long s){
    mt[0]= s & 0xffffffffUL;
    for (mti=1; mti<N; mti++) {
        mt[mti] = (1812433253UL*(mt[mti-1]^(mt[mti-1]>>30))+mti);
        mt[mti] &= 0xffffffffUL;
}}

void RandomNumber::init_by_array(unsigned long init_key[],int key_length){
    int i, j, k;
    init_genrand(19650218UL);
    i=1; j=0;
    k = (N>key_length?N:key_length);
    for (; k; k--) {
        mt[i] = (mt[i]^((mt[i-1]^(mt[i-1]>>30))*1664525UL))
          + init_key[j] + j; /* non linear */
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++; j++;
        if(i>=N){ mt[0] = mt[N-1]; i=1; }
        if(j>=key_length) j=0;
    }
    for (k=N-1; k; k--) {
        mt[i] = (mt[i]^((mt[i-1]^(mt[i-1]>>30))*1566083941UL))-i;
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++;
        if(i>=N){ mt[0] = mt[N-1]; i=1; }
    }
 mt[0] = 0x80000000UL; /*MSB is 1; assuring non-zero initial array*/
```

```
}

/* generates a random number on [0,0xffffffff]-interval */
unsigned long RandomNumber::genrand_int32(void){
    unsigned long y;
    static unsigned long mag01[2]={0x0UL, MATRIX_A};
    /* mag01[x] = x*MATRIX_A  for x=0,1 */
    if (mti >= N) { /* generate N words at one time */
        int kk;
        if (mti == N+1)/*if init_genrand() has not been called,*/
            init_genrand(5489UL);/*a default initial seed is used*/
        for (kk=0;kk<N-M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+M]^(y>>1)^mag01[y&0x1UL];
        }
        for (;kk<N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(M-N)]^(y>>1)^mag01[y&0x1UL];
        }
        y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[N-1] = mt[M-1]^(y>>1)^mag01[y&0x1UL];
        mti = 0;
    }

    y = mt[mti++];
    /* Tempering */
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);
    return y;
}

/* generates a random number on [0,0x7fffffff]-interval */
```

```
long RandomNumber::genrand_int31(void){
    return (long)(genrand_int32()>>1);}

/* generates a random number on [0,1]-real-interval */
double RandomNumber::genrand_real1(void){
    return genrand_int32()*(1.0/4294967295.0);
    /* divided by 2^32-1 */}

/* generates a random number on [0,1)-real-interval */
double RandomNumber::genrand_real2(void){
    return genrand_int32()*(1.0/4294967296.0);
    /* divided by 2^32 */}

/* generates a random number on (0,1)-real-interval */
double RandomNumber::genrand_real3(void){
    return(((double)genrand_int32())+0.5)*(1.0/4294967296.0);
    /* divided by 2^32 */}

/* generates a random number on [0,1) with 53-bit resolution*/
double RandomNumber::genrand_res53(void){
    unsigned long a=genrand_int32()>>5, b=genrand_int32()>>6;
    return(a*67108864.0+b)*(1.0/9007199254740992.0);}
```

A.3.5 Binarygrowth.h

```
#ifndef BINARYGROW
#define BINARYGROW
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include "Network.h"
```

```
#include "RandomNumber.h"
using namespace std;
class BinaryGrow {
int Networksize,k;
int DuVcnt, DuEcnt,DuActiveNode,NewCenter;
bool Dudigraph;
int OldCbMax;
double OldLamda,CriticalLamda,Targetload, Bestload,delta;
    int Cstar;
int Num,Na,Nb,maxNB;
double lB,randpick;
vector <int> Cb_B;
vector <int> connection;
vector <int> bestconnection;
vector <int> finalcentrality;
vector <int> tempcentrality;
vector <int> alpha;
vector <int> conB;
vector <int> centrB;
vector <int>::iterator iter1;
vector <int> visitbit;
int Nf,currentstep;
bool stepflag;
vector<double> bestload;
vector <vector <int> > Duadj;
vector<int> Dunodes;
vector<vector <int> > Cbhistory;
vector<int> Cbmax;
vector<int> CenterNodes;
vector<double> Ellbar;
vector<double> Criticals;
vector<int> degrees;
RandomNumber rand1;
Network dg;
```

```
public:
BinaryGrow( );
~BinaryGrow();
void WritetoDu();
void ReadfromDu();
void printR();
void elapsed_time();
    void Startnet(int step);
bool isCbmax();
int  isCbmaxraul();
void fixBsubnet(int center, int subnet);
void rsearch(int currentbit);
void rrsearch(int currentbit);
void SearchT(int currentbit);
void printConnect();
void printadj();
void initialB();
void Stepoutput(int step);};
#endif
```

A.3.6 Binarygrowth.cpp

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cmath>
#include <cstdlib>
#include "BinaryGrow.h"

BinaryGrow::BinaryGrow( ){
rand1.init_genrand((unsigned)time( NULL ));
Networksize = dg.getNetworkSize();
dg.Initial(Networksize,false);
dg.createGfromfile();
```

```
initialB();
CenterNodes.push_back(0);
OldLamda = 0.0;
CriticalLamda =0.0;
DuVcnt = dg.V();
DuEcnt = dg.E();
DuActiveNode = dg.Activenode();
Dudigraph = dg.directed();
Duadj =dg.getAdj();
Dunodes = dg.getNodes();
}

BinaryGrow::~BinaryGrow( ){ }

void BinaryGrow::WritetoDu(){
Duadj = dg.getAdj();
Dunodes = dg.getNodes();
DuActiveNode = dg.Activenode();
DuEcnt = dg.E();}

void BinaryGrow::ReadfromDu(){
dg.writefrom(Duadj,Dunodes,DuActiveNode,DuEcnt);}

void BinaryGrow::Startnet(int step){
    dg.RunBC();
for(int i=0;i<(dg.Activenode());i++){
finalcentrality.push_back(dg.GetBC(i));
Cstar = dg.GetBCMax();
Cbmax.push_back(dg.GetBCMax());
bestload.assign(step,10);
OldLamda = (double)(dg.Activenode()-1)/(double)dg.GetBCMax();
Targetload = OldLamda;
Criticals.push_back(OldLamda);
    cout<<endl<<"Target Load: "<<Targetload;
```

```
Na = dg.Activenode();
cout<<endl<<"Na: "<<Na<<", C*: "<<Cstar;
delta = Cstar*Cstar-4*Na+12*pow((double)Na,(double)2)
            -12*pow((double)Na,(double)3)+4*pow((double)Na,(double)4);
cout<<endl<<"delta: "<<delta;
delta = sqrt(delta);
cout<<endl<<"squre root of delta: "<<delta;
if(delta>=0){
maxNB = (int)((Cstar+2*Na-2*pow((double)Na,(double)2)+delta)
                /(2*(Na-1)));
cout<<endl<<"maxN_B: "<<maxNB;}
else{cout<<endl<<"delta is negative, no solution!!!";}
for(int z=0;z<step;z++){
cout<<endl<<"**************step: "<<z<<" ***********";
stepflag=false;
currentstep=z;
dg.addNode();
CenterNodes.push_back(dg.Activenode());
for(int i=0;i<(dg.Activenode()-1);i++){
dg.insert(i,(dg.Activenode()-1));}
WritetoDu();
Na=dg.Activenode()-1;
Nf = dg.Activenode()-1;
connection.clear();
bestconnection.clear();
connection.assign(dg.Activenode()-1,1);
bestconnection.assign(dg.Activenode()-1,1);
if(isCbmax()){
cout<<endl<<
        "It is the maximum for full connection of A-F, but no solution yet"
        <<endl;}
else{cout<<endl<<"Not maximum for full connection of A-F"<<endl;}
for(int i=0;i<(int)connection.size();i++){
visitbit.push_back(i);
```

```
while(!visitbit.empty() && stepflag==false){
    int ccbit = (int)rand1.genrand_real3()*(int)visitbit.size();
visitbit.erase(visitbit.begin()+ccbit);
if(stepflag==true){
cout<<endl<<"already have the best load";
return;}
else{
if( connection[ccbit]!=0){
connection[ccbit]=0;
dg.deletel(ccbit,Nf);
WritetoDu();}
    else{cout<<endl<<"this bit is already zero?"<<endl;}
if(isCbmax()){
rsearch((int)connection.size()-1);}
        connection[ccbit]=1;
dg.insert(ccbit,Nf);
WritetoDu();}
}
visitbit.clear();
    Stepoutput(z);}
}

bool BinaryGrow::isCbmax(){
dg.RunBC();
finalcentrality.clear();
for(int i=0;i<(dg.Activenode());i++){
finalcentrality.push_back(dg.GetBC(i));
dg.addNode();
dg.insert(Nf,Nf+1);
dg.RunBC();//dg.PrintBC();dg.printG();
alpha.clear();
for(int i=0;i<Nf;i++){
alpha.push_back(dg.GetBC(i)-finalcentrality[i]);
}
```

```
ReadfromDu();
int subnet = 0; int transit = 0;  int j=0;
bool flag2; bool flag=false;
while(conB[subnet]<=maxNB && subnet<(int)conB.size() && stepflag!=true){
j=conB[subnet];
flag2 = false;// reset flag2 for the subnet fixed in
for(int i=0;i<Nf;i++){
transit = 2*j*Na+j+centrB[subnet];//2N_A*N_B+N_B+C_B
if(((double)(Na+1+j-1)/(finalcentrality[i]*1.0+alpha[i]*j*1.0))
               <=(((double)Na+1+j-1)/(finalcentrality[Nf]*1.0+transit*1.0)))
               {flag2 = true;}
}
if(flag2==false){
flag=true;
if(((float)(Na+j))/(finalcentrality[Nf]*1.0+transit*1.0)>=Targetload
               &&((float)(Na+j))/(finalcentrality[Nf]*1.0+transit*1.0)
               <bestload[currentstep]){
bestload[currentstep] =((float)(Na+j))/(finalcentrality[Nf]*1.0+transit*1.0);
bestconnection.clear();
for(int k=0;k<(int)connection.size();k++){
                bestconnection.push_back(connection[k]);}
if(bestload[currentstep] - Targetload < 0.001 ){
if(currentstep!=0){
if( bestload[currentstep]>=bestload[currentstep-1]){
cout<<endl<<"Got the best load for the step:"<<bestload[currentstep]<<endl;
stepflag = true;
fixBsubnet(Nf,subnet);
WritetoDu();
for(int k=0;k<(int)bestconnection.size();k++){
cout<<"-"<<bestconnection[k];}
dg.printG();
flag = true;
return flag;}}
else{
```

```
cout<<endl<<"Got the best load for the step:"<<bestload[currentstep]<<endl;
stepflag = true;
fixBsubnet(Nf,subnet);
WritetoDu();
for(int k=0;k<(int)bestconnection.size();k++){
cout<<"-"<<bestconnection[k];}
dg.printG();
flag = true;
return flag;}
}}}
subnet++;}
return flag;
}

void BinaryGrow::rsearch(int currentbit){
if(stepflag==true){
cout<<endl<<"already have the best load";
return;}
else{
int cbit = currentbit;
while(cbit>=0){
connection[cbit]=0;
dg.deletel(cbit,Nf);
WritetoDu();
if(isCbmax()){rrsearch(cbit);}
connection[cbit]=1;
dg.insert(cbit,Nf);
WritetoDu();
cbit--;}}
}

void BinaryGrow::rrsearch(int currentbit){
int cbit = currentbit; int zbit;
if( cbit<((int)connection.size()-1) ){
```

```
zbit = (int)connection.size()-1;
while(zbit>cbit){
connection[zbit]=0;
dg.deletel(zbit,Nf);
WritetoDu();
if(isCbmax()){
rrsearch(zbit);}
connection[zbit]=1;
dg.insert(zbit,Nf);
WritetoDu();
zbit--;}}
}

void BinaryGrow::SearchT(int currentbit){
if(stepflag==true){
cout<<endl<<"already have the best load";
return;}
else{
int cbit = currentbit;
for(int zbit = cbit;zbit<(int)connection.size();zbit++){
if(connection[zbit]!=0){
connection[zbit]=0;
dg.deletel(zbit,Nf);
WritetoDu();}
else{cout<<endl<<"this bit is already zero???"<<endl;}
if(cbit<(int)connection.size()&& zbit<(int)connection.size()){
if(isCbmax()){
cout<<endl<<"*******Right branch to move deeper";
SearchT(zbit+1);
}}
connection[zbit]=1;
dg.insert(zbit,Nf);
WritetoDu();
}}
```

```
void BinaryGrow::printConnect(){
cout<<endl;
for(int i=0;i<(int)connection.size();i++){
cout<<" "<<connection[i];}

void BinaryGrow::printadj(){
cout<<endl;
for(int i=0;i<dg.getadjsize();i++){
cout<<endl;
for(int j=0;j<dg.getadjsize();j++){
cout<<" "<<dg.edge(i,j);}
}
cout<<endl;}

void BinaryGrow::fixBsubnet(int center,int subnet){
switch(subnet){
case 0:
break;
case 1:
dg.addNode();//place 1 node in B
dg.insert(center,center+1);
break;
case 2:
dg.addNode();//place 2 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
break;
case 3:
dg.addNode();//place 2 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
```

```
dg.insert(center+1,center+2);
break;
case 4:
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
dg.addNode();//place the third node in B
dg.insert(center,center+3);
break;
case 5:
randpick=rand1.genrand_real2();
cout<<endl<<"randpcik"<<randpick;
if(randpick<=0.25 && randpick>0){
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
dg.addNode();//place the third node in B
dg.insert(center,center+3);
dg.insert(center+1,center+2);}
if(randpick>0.25 && randpick<=0.5){
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
dg.addNode();//place the third node in B
dg.insert(center,center+3);
dg.insert(center+2,center+3);}
if(randpick>0.5 && randpick<=0.75){
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
```

```
dg.addNode();//place the third node in B
dg.insert(center+1,center+3);}
if(randpick>0.75){
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
dg.addNode();//place the third node in B
dg.insert(center+2,center+3);}
break;
case 6:
randpick=rand1.genrand_real2();
cout<<endl<<"randpcik"<<randpick;
if(randpick>0 && randpick<=0.5){
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
dg.addNode();//place the third node in B
dg.insert(center,center+3);
dg.insert(center+1,center+2);
dg.insert(center+2,center+3);}
if(randpick>0.5){
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
dg.addNode();//place the third node in B
dg.insert(center+1,center+3);
dg.insert(center+2,center+3);}
break;
case 7:
randpick=rand1.genrand_real2();
cout<<endl<<"randpcik"<<randpick;
```

```
if(randpick>0 && randpick<=0.2){
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
dg.addNode();//place the third node in B
dg.insert(center,center+3);
dg.insert(center+1,center+2);
dg.insert(center+2,center+3);
dg.insert(center+1,center+3);}
if(randpick>0.2 && randpick<=0.4){
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
dg.addNode();//place the third node in B
dg.insert(center+2,center+3);
dg.insert(center+1,center+2);}
if(randpick>0.4 && randpick<=0.6){
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
dg.addNode();//place the third node in B
dg.insert(center+1,center+3);
dg.insert(center+1,center+2);}
if(randpick>0.6 && randpick<=0.8){
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center,center+2);
dg.addNode();//place the third node in B
dg.insert(center+1,center+3);
dg.insert(center+1,center+2);
```

```
dg.insert(center+2,center+3);}
if(randpick>0.8){
dg.addNode();//place 3 nodes in B
dg.insert(center,center+1);
dg.addNode();//place the other node in B
dg.insert(center+1,center+2);
dg.addNode();//place the third node in B
dg.insert(center+2,center+3);}
break;
default:
break;
}}

void BinaryGrow::initialB(){
centrB.push_back(0);
centrB.push_back(0);
centrB.push_back(2);
centrB.push_back(0);
centrB.push_back(6);
centrB.push_back(4);
centrB.push_back(1);
centrB.push_back(0);
conB.push_back(0);
conB.push_back(1);
conB.push_back(2);
conB.push_back(2);
conB.push_back(3);
conB.push_back(3);
conB.push_back(3);
conB.push_back(3);
}

void BinaryGrow::Stepoutput(int s){
ofstream cbout("cbs.dat",ios::app);
```

```
if(!cbout){
cerr<<"Error Opening File!\n";
exit(0);
ofstream degreeout("degrees.dat",ios::app);
if(!degreeout){
cerr<<"Error Opening File!\n";
exit(0);
    int n=dg.Activenode();
degrees.clear();
degrees.assign(n,0);
for(int i=0;i<n;i++){
for(int j=i;j<n;j++){
if(dg.edge(i,j)==1){
degrees[i]=degrees[i]+1;
degrees[j]=degrees[j]+1;}
}
}
  degreeout<<endl<<endl<<"step--"<<s+1<<"--is: ";
  for(int i=0;i<(int)degrees.size();i++){
    degreeout<<endl<<"Node "<<i<<"    "<<degrees[i];}
    cbout<<endl<<endl<<"step--"<<s+1<<"--is: ";
  for(int i=0;i<(int)degrees.size();i++){
    cbout<<endl<<"Node "<<i<<"    "<<dg.GetBC(i);}
}

void BinaryGrow::printR(){
for(int i =0;i<(int)bestload.size();i++){
cout<<endl<<"step:"<<i<<" load:"<<bestload[i]
           <<" Centernode:"<<CenterNodes[i];}
}

void BinaryGrow::elapsed_time( ){
      unsigned tim,run_time_hrs, run_time_mins, run_time_secs;
      tim=clock()/CLOCKS_PER_SEC;
```

```
      run_time_hrs=tim/3600;
      run_time_mins=(tim%3600)/60;
      run_time_secs=(tim%3600)%60;
  cout<<endl<<"The program has been running for: "<<run_time_hrs
         <<" Hours, "<<run_time_mins<<" Minutes, "
         <<run_time_secs<<" Seconds"<<endl;
}

int main(){
BinaryGrow test;
test.Startnet(100);
test.printR();
test.elapsed_time();
return 0;}
```

Appendix B

Topology Generation, Random Number Generation and Graph Visualization

B.1 Catastrophic Network Topology Generator

Based on the methods proposed in Chapter 4 and Chapter 5, we implement them using ANSI C and C++ standard programming language. The codes are compiled using GCC (GNU Compiler Collection) [132] and ICC (Intel C++ Compliler) [133]. The source code is not available in this thesis due to intellectual property restrictions. The hardware platform used is a four node cluster built with Viglen 1U server with two Intel Duo Core Xeron 5500 serial CPU and 6 Gigabyte DDR memory. The catastrophic topology generator not only include the code to generate the catastrophic networks introduced but also other basic topologies. For example, ER random network topology, Scale-free network topology, regular network topology. All these kinds of networks could be generated and used to compare with catastrophic networks. We also include shortest path calculation algorithms: Bellman-Ford, Dijkstra's, and Floyd's. They will be introduced in detail in the following section. Finally, the betweenness centrality calculation is implemented using Brandes' method [57]. It is vital component for the generation of catastrophic networks.

B.2 Graph Related Libraries

By considering the commercial and open-source simulators available and the recent research in network simulation [134] and the available graph libraries, C++ (or C) is a preferred programming language. It is decided to use the C++ (Object Oriented Language) as the programming language including its STL (Standard Template Library) with other graph libraries available.

B.2.0.1 BGL

Graphs are mathematical abstractions that are useful for solving many types of problems in computer science. Consequently, these abstractions must also be represented in computer programs. A standardized generic interface for traversing graphs is of utmost importance to encourage reuse of graph algorithms and data structures. Part of the Boost Graph Library(BGL) [84] is a generic interface that allows access to a graph's structure, but hides the details of the implementation. This is an "open" interface in the sense that any graph library that implements this interface will be interoperable with the BGL generic algorithms and with other algorithms that also use this interface. The BGL provides some general purpose graph classes that conform to this interface, but they are not meant to be the "only" graph classes; there certainly will be other graph classes that are better for certain situations. The main contribution of the BGL is the formulation of the interface.

The BGL algorithms consist of a core set of algorithm patterns (implemented as generic algorithms) and a larger set of graph algorithms. The core algorithm patterns are: Breadth First Search, Depth First Search, Uniform Cost Search. By themselves, the algorithm patterns do not compute any meaningful quantities over graphs; they are merely building blocks for constructing graph algorithms. The graph algorithms in the BGL currently include:

- Dijkstra's Shortest Paths
- Bellman-Ford Shortest Paths
- Johnson's All-Pairs Shortest Paths

- Kruskal's Minimum Spanning Tree
- Prim's Minimum Spanning Tree
- Connected Components
- Strongly Connected Components
- Dynamic Connected Components (using Disjoint Sets)
- Topological Sort

B.2.0.2 Graph Algorithms

The "Algorithms in C++, Graph Algorithms" is a whole set of graph related algorithms analogized and written by Robert Sedgewick [86]. His work was on describing the most important known methods for solving the graph-processing problems that arise in practice.

B.2.0.3 iGraph

It is another library for creating and manipulating graphs. The authors developed this library aiming to handle very large size graphs confronted more efficiently. Igraph started as an additional package to the GNU R statistical environment, and still some functions available only in R language. Like all other graph libraries, it provides a set of graph related algorithms: functions for generating regular and random graphs according to known algorithms and models in the network theory literature; routines for manipulating graphs, adding and removing edges and vertices; a set of structural property calculation functions like degree, betweenness; force based layout generators; simple and efficient way of walking through graphs.

B.3 Random Number Generation

Most of the simulations of complex networks which have random components in them, require a reliable method of generating or obtaining numbers that are

random in some sense. For example, when generating a random graph using the ER-model [61; 62], or when simulating the M/M/1 queue [25; 135]. Further, the specific distribution may also required which could be generated from the pseudo or quasi random variables. Most of the random variables are generated from the uniform distribution on the interval $[0, 1]$. This distribution was denoted by $U(0, 1)$. Although it is the simplest continuous distribution of all, it is extremely important that it able to obtain such independent random numbers. The important role of the $U(0, 1)$ distribution stems from the face that random variables from all other distributions (normal, binomial,exponential, etc.) and realization of various random processes (e.g., Poisson Process) can be obtained by transforming $U(0, 1)$ in a way determined by the desired distribution or process.

The main purpose of this section is not to analysis the random number generation in detail. While it is to show the mostly used and trusted random number generators which are also used in this thesis. Because the catastrophic network topology generation code and the simulation codes are mainly written in C/C++, several C++ library are used so that the codes are more reliable and efficient. The distribution "rand" function in the C/C++ library is a pseudo-random number generation function but not good enough for scientific research. The introduction of Intel Math Kernel Library [8] to the coding has provide a set of sophisticated random number generators and distribution generator.

The specific random number generation algorithm used in this thesis is the Mersenne-Tweister random number generation method. It is a pseudo random number generator developed in 1997 by Makoto Matsumoto and Takuji Nishimura [136]. It is one of the most commonly used and reliable random number generation algorithm. The version included in the Intel Math Kernel Library is the Mersenne-Twister MT19937 with 32-bit word length.

B.4 Analysis of Modern Topology Generators

B.4.1 Topology Generation

For generation a range of network topologies, from the regular-symmetric network to different Scale-free network, the general "topology" generation model is built

Basic Random-Number Generators (BRNGs)
Pseudo-random
MCG59: Multiplicative Congruential Generator 59-bit
MCG31m1: Multiplicative Congruential Generator 31-bit
MRG32k3a: Multiple Recursive Generator 32-bit
R250: Generalized feedback shift register
Wichman-Hill: A set of 273 basic generators
MT19937: Mersenne Twister
MT2203: A set of 1024 Mersenne Twister basic generators
Quasi-random
Sobol: A 32-bit Gray code-based generator
Niederreiter: A 32-bit Gray code-based generator

Table B.1: The table generated from Intel Math Kernel Library Manual. Detail of different random number generation methods can be found in the [8], [9], and reference there in.

using a few parameters and output the needed network topology. The input parameters of the topology generator is shown in Table. The output is printed

Parameter Name	Description
N	Total number of nodes in the network
Type of network	1,Regular Symmetric; 2,Pure Random; 3,Scale-free; 4, Tier Random
k	degree distribution for random network
w	Weighted graph with random weight(latency)
r	Router density

Table B.2: Topology Generation Input Parameter

on a file which would contain the topology information. It would be in the form of showing the source node, destination node of a link, and the weight of the link if it is a weighted graph.

B.4.2 Waxman

The Waxman model been widely used to generate random topologies for network simulations. It starts by placing n nodes uniformly on an n by n plane. Once

all nodes have been placed on the plane, the model computes the probability of creating an edge between two nodes u and v with the following probability function:

$$P_{u,v} = \alpha e^{-d(u,v)/\beta L}, \tag{B.1}$$

where $d(u, v)$ is the Euclidean distance between u and v, α is the average out-degree, L the maximum distance between any two nodes, and β determines the average edge length. Then a random number is generated between 0 and 1. An edge is created between u and v only if the random number is smaller than $P(u, v)$.It is the basic and reference model for many other random topology generators.

B.4.3 Inet

The Inet3.0 topology generator [137] is an Autonomous System (AS) level Internet topology generator. It is important to note that Inet only provides the connectivity information; the generated topologies do not have any information pertaining to latency, bandwidth etc. The parameters provided for the code to generate a AS-level topology includes: N,the total number of nodes; k,the fraction of degree-one nodes; n,the size of the plane used for node placement; sd,the seed to initialize the random number generator. It is a tool to generate the Internet like AS-level topology, but it is a little bit out-of-date and not considering some important properties in the Internet (clustering).

B.4.4 GT-ITM

Georgia Tech Internetwork Topology [137], generates topologies based on several different models. But the transit-stub model attracts much attentions in that it closely resembles the Internet topology. Similar to Tier model following, the transit-stub model has a well-defined hierarchical structure. It generates topologies with two levels of hierarchy: one consisting of transit ASs, and the other consisting of stub ASs. To generate a topology, GT-ITM first generates a connected random graph of T nodes; each node represents a transit AS. Each transit AS is then instantiated as, and replaced by, a connected random graph with an

average of N_t number of nodes. Next, each node in the transit AS are connected to, on the average, K number of stub ASs. Each stub AS consists of a connected graph with an average of N_s number of nodes. The connectivity used to generate each connected graph can be selected from one of the six methods: PureRandom, Waxman1, Waxman2, Doar-Leslie, Exponential, or Locality. But the deficiency is it do not bring in the Scale-free network, which newly discovered model for the complex network topology.

B.4.5 Tiers Network Topology

The Tiers generator [125] is based on a three level hierarchy that represents WAN, MAN, and LAN. To generate a random topology using Tiers, one specifies a target number of LANs and MANs. Currently Tiers cannot generate more than one WAN per random topology. For each level of hierarchy, one also has to specify a fixed number of nodes per network. For all these constraints, it is not easy to examine its properties and model the Internet. But still it is an good idea to based on while modeling the real-world Internet.

B.4.6 BRITE

BRITE [138] is another generator based on the power-law degree distribution which is used by the NS2 and OMNeT++ network simulator. Furthermore, BRITE also incorporates recent findings on the origin of power-laws and observations of skewed network placement and locality in network connections on the Internet. BRITE supports multiple generation models including models for flat AS, flat Router and hierarchical topologies. Models can be enhanced by assigning links attributes such as bandwidth and delay. The BRITE can be considered as the a sophisticated generation package.

B.5 Network Visualization

Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. Especially, when simulating the congestion and network failure, the graph's evolving and breaking down are much easier to be

shown in a visual way. For example, a cut off of a link, a break-down of a node and it's attached links. Automatic graph drawing has an important application in networking design and graph analysis.

B.5.1 Positioning Algorithm

The most important parameter for graph (network) visualization is the position of the vertices (nodes). It is the difference in the positioning pattern that shows the reader the specific network properties in a more eye-friendly way where the pattern of the network could be easily seen.

The most commonly used algorithm for graph positioning is the spring embedding algorithms also called FDP (Force Directed Placement). It can be used to sort randomly placed nodes into a desirable layout hat satisfies the aesthetics for visual presentation (symmetry, non-overlapping etc.) FDP [139] views nodes as physical bodies and edges as springs connected to the nodes providing forces between them. Nodes move according to the forces on them until a local energy minimum is achieved. Based on this serveral commonly used algorithm is available in most of the graph related programming libraries and some forming their own libraries.

- The Kamada-Kawai Algorithm [140]
- The Fruchterman-Reingold Algorithm [141]
- The k-core decomposition [72], which is used to visualize large scale complex networks in two dimensions.

B.5.2 Graph Visualization Tools

For most of the available algorithms, one do not have to code to get them work. Instead, there are a few very good existing graph visualization tools available either as a software application or as a library.

- **Pajek** is a program, for Windows, for analysis and visualization of large networks having some thousands or even millions of vertices. With Pajek one can: find clusters (components, neighborhoods of important vertices,

cores, etc.) in a network, extract vertices that belong to the same clusters and show them separately, possibly with the parts of the context (detailed local view), shrink vertices in clusters and show relations among clusters (global view) [4].

- **Processing** is an open source programming language and environment for people who want to program images, animation, and interactions. It is a very good visualization software package where one an explore different positioning algorithms by embedded them into the processing codes.
- **Graphviz** is open source graph visualization software. It has several main graph layout programs. It also has web and interactive graphical interfaces, and auxiliary tools, libraries, and language bindings.
- **LaNet-vi** provides images of large scale networks on a two-dimensional layout. The algorithm is based on the k-core decomposition.

B.5.3 Visualization Summary

Visualization is the best way of revealing the structural property of an network. By positioning the nodes and links in different patterns, one can easily tell the hidden nature of the networks. All of the graphs produced throughout this report are produced by the algorithms and software packages mention above.

Appendix C

Author's Publications

1. Z. Huang and R.J. Mondragón, "Topology and Congestion Invariant in Global Internet-Scale Networks", 7th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNet06), 2006.

2. M. Woolf, Z. Huang, and R.J. Mondragón, "Building catastrophes: networks designed to fail by avalanche-like breakdown", Focus on Complex Networked Systems: Theory and Application, New Journal of Physics, Volume 9, June 2007.

3. Z. Huang, M. Woolf and R.J. Mondragón, "Building Catastrophic Networks", Proceedings of NetSci 2008 workshop, Norwich, England, 2008.

4. K. Jacobson, Z. Huang, B. Fields, and M. Sandler, "An analysis of on-line music artist networks", Proceedings of NetSci 2008 workshop, Norwich, England, 2008.

5. Z. Huang, M. Woolf and R.J. Mondragón, "Growing Networks Designed to Fail Catastrophically", NAEC2008, Riva del Garda, Italy, 2008.

References

[1] "More on today's gmail issue." [Online]. Available: http://gmailblog.blogspot.com/2009/09/more-on-todays-gmail-issue.html 6, 75

[2] "Processing." [Online]. Available: http://processing.org/ 13, 31

[3] A.-L. Barabsi and R. Albert, "Emergence of scaling in random networks," Science, vol. 286, pp. 509–512, 15 October 1999. 14, 40

[4] V. Batagelj and A. Mrvar, Graph Drawing Software. Springer, Berlin, 2003, ch. Pajek - Analysis and Visualization of Large Networks, pp. 77–103. 14, 15, 43, 88, 233

[5] "Cooperative association for internet data analysis." [Online]. Available: http://www.caida.org/home/ 14, 23, 44, 45

[6] "Large network visualization tool." [Online]. Available: http://xavier.informatics.indiana.edu/lanet-vi/ 14, 46

[7] F. Harary and E. Palmer, Graphical Enumeration. Academic Press, 1973. 15, 103, 104

[8] Intel, Intel Math Kernel Library 10.0, Intel, 2007. 20, 228, 229

[9] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, Numerical Recipes in C++, the art of scientific computing third edition. Cambridge University Press, 2007. 20, 229

[10] H. Chang, S. Jamin, and W. Willinger, "Inferring as-level internet topology from router-level path traces," in SPIE ITCom 2001, Denver, CO, August 2001. 23

* The numbers at the end of each reference item are the page numbers where the reference are cited.

[11] Z. M. Mao, J. Rexford, J. Wang, and R. Katz, "Towards an accurate as-level traceroute tool," in *ACM SIGCOMM 2003*, 2003. 23

[12] R. Oliveira, B. Zhang, and L. Zhang, "Observing the evolution of internet as topology," in *ACM SIGCOMM, Kyoto, Japan*, August 2007. 23

[13] J.-J. Pansiot and D. Grad, "On routes and multicast trees in the internet," *ACM SIGCOMM Computer Communication Review*, vol. 28, pp. 41–50, 1998. 23

[14] W. Cheswick and H. Burch, "Mapping the internet," *IEEE Computer*, vol. 32, 1999. 23

[15] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," in *SIGCOMM 2002*, 2002. 23

[16] Y. Shavitt, X. Sun, A. Wool, and B. Yener, "Computing the unmeasured: An algebraic approach to internet mapping," *IEEE Journal on Selected Areas in Communications*, vol. 22 (1), pp. 67–78, January 2004. 23

[17] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," IETF, Tech. Rep., 2001. 23

[18] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 1999. 23, 41, 46

[19] A. Vazquez, R. Pastor-Satorras, and A. Vespignani, "Large-scale topological and dynamical properties of internet," *Physical Review E*, vol. 65, p. 066130, 2002. 23

[20] R. Albert and A.-L. Barabasi, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, pp. 47–97, Jan 2002. 23, 24, 29, 37, 39, 41, 42, 69

[21] S. H. Yook, H. Jeong, and A.-L. Barabsi, "Modeling the internets large-scale topology," in *Proceedings of the National Academy of Sciences 99*, 2002. 23

[22] A.-L. Barabasi and E. Bonabeau, "Scale free networks," *Scientific American*, pp. 60–69, May 2003. 23

[23] D. Alderson, L. Li, W. Willinger, and J. C. Doyle, "Understanding internet topology: Principles, models, and validation," *IEEE/ACM Transactions on Networking,*, vol. 13, No. 6, pp. 1205–1218, Dec 2005. 23, 47

[24] J. Doyle, D. Alderson, L. Li, S. Low, M. Roughan, S. Shalunov, R. Tanaka, and W. Willinger, "The "robust yet fragile" nature of the internet," *Proceedings of the National Academy of Sciences*, vol. 102, pp. 14 497–14 502, 2005. 23, 69

[25] L. Kleinrock, *Queueing System Volume I: Theory.* John Wiley & Sons, 1975. 23, 228

[26] J. Pitts and J. Schormans, *Introduction to IP and ATM Design and Performance with Application Analysis Software.* John Wiley & Sons, Ltd, 2001. 23

[27] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic," *IEEE/ACM Transactions on Networking*, vol. 2, p. 1, 1994. 23, 57, 133

[28] K. Fukuda, L. A. N. Amaral, and H. E. Stanley, "Dynamics of temporal correlation in daily internet traffic," in *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, vol. Volume 7, 2003, pp. 4069 – 4073. 23

[29] T. Ohira and R. Sawatari, "Phase transition in a computer network traffic model," *Physical Review E*, vol. 58, No. 1, pp. 193–195, 1998. 23, 60

[30] H. Fuks and A. Lawniczak, "Performance of data networks with random links," *Math and Computers in Simulation*, vol. 51, pp. 101–117, 1999. 23, 60, 65

[31] R. Sole and S. Valverde, "Information transfer and phase transition in a model of internet traffic," *Physica A*, vol. 289, pp. 595–605, 2001. 23, 60, 65

[32] S. Valverde and R. Sole, "Self-organized critical traffic in parallel computer networks," *Physica A*, pp. 636–648, 2002. 23, 60

[33] M. Woolf, D. Arrowsmith, R. J. Mondragn, and J. Pitts, "Optimization and phase transition in a chaotic model of data traffic," *Physics Review E*, vol. 66, p. 056106, 2002. 23, 59, 60, 65

[34] M. E. J. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, p. 167, 2003. 24, 29, 69

[35] S. Dorogovtsev and J. Mendes, "Evolution of networks," *arXiv:cond-mat/0106144*, Sep 2001. 24, 29, 32, 69

[36] L. da F. Costa, F. A. Rodrigues, G. Travieso, and P. R. V. Boas, "Characterization of complex networks: A survey of measurements," *Advances In Physics*, vol. 56, p. 167, 07. 24, 29, 32, 69

[37] C. DeMarco, "A phase transition model for cascading network failure," *IEEE Control System Magazine*, vol. Dec, pp. 40–51, 2001. 24, 50, 69

[38] L. Pereira, "Cascade to black," *IEEE Power Energy Magazine*, vol. 2, p. 54, 2004. 24, 69

[39] D. J. Watts, "A simple model of global cascades on random networks," *Proceeding of the National Academy of Sciences USA*, vol. 99, p. 5766, 2006. 24, 50, 69, 70, 76

[40] P. Holme and B. Kim, "Attack vulnerability of complex networks," *Physical Review E*, vol. 65, p. 056109, 2002. 24, 49, 51, 76

[41] P. Holme, "Edge overload breakdown in evolving networks," *Physical Review E*, vol. 66, p. 036119, 2002. 24, 51, 79

[42] A. Motter and Y. Lai, "Cascade-based attacks on complex networks," *Physical Review E*, vol. 66, p. 065102, 2002. 24, 51, 70, 76

[43] L. Zhao, K. Park, and Y. Lai, "Attack vulnerability of scalefree networks due to cascading breakdown," *Physical Review E*, vol. 70, p. 035101, 2004. 24, 50, 51, 76

[44] L. Zhao, Y. Lai, K. Park, and N. Ye, "Onset of traffic congestion in complex networks," *Physical Review E*, vol. 71, p. 026125, 2005. 24, 51, 76, 78

[45] S. Bornholdt and H. G. Schuster, Eds., *Handbook of Graphs and Networks: From the Genome to the Internet*. Wiley VCH, 2002. 29

[46] S. Dorogovtsev and J. Mendes, *Evolution of Networks–From Biological Nets to the Internet and WWW*. Oxford University Press, 2003. 29

[47] R. Pastor-Satorras and A. Vespignani, *Evolution and Structure of the Internet: A Statistical Physics Approach*. Cambridge University Press, 2004. 29

[48] M. Newman, A.-L. Barabsi, and D. Watts, *The Structure and Dynamics of Networks*. Princeton University Press, 2006. 29

[49] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. Hwang, "Complex networks: Structure and dynamics," *Physics Reports*, vol. 424, pp. 175–308, 2006. 29

[50] F. Harary, *Graph Theory*. Addison-Wesley, 1969. 29, 30, 103

[51] B. Bollobas, *Modern graph theory*. Springer, 1998. 29, 32

[52] ——, *Random graphs*. Cambridge University Press, 2001. 29

[53] A. Tanenbaum, *Computer Networks (4th Edition)*. Pearson Education International, 2003. 29, 57

[54] A. Gibbons, *Algorithmic Graph Theory*. Cambridge University Press, 1985. 30

[55] D. Arrowsmith, R. Mondragon, and M. Woolf, *Complex Dynamics in Communication Networks*. Springer, Aug 2005, ch. Data Traffic, Topology and Congestion, pp. 127–158. 32, 34, 37, 56, 67

[56] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, pp. 35–41, 1977. 33, 34

[57] U. Brandes, "A faster algorithm for betweenness centrality," 2001. 33, 34, 36, 225

[58] M. E. J. Newman, "Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality," *Physical Review E*, vol. 64, p. 016132, 2001. 34, 36

[59] ——, "A measure of betweenness centrality based on random walks," *Social Networks*, vol. 27, No. 1, pp. 39–54., Jan 2005. 36

[60] R. Solomonoff and A. Rapoport, "Connectivity of random nets," *Bulletin of Mathematical Biophycis*, vol. 13, pp. 107–117, 1951. 37, 38

[61] P. Erdos and A. Renyi, "On random graphs," *Publications Mathematicae*, vol. 6, p. 290, 1959. 37, 38, 228

[62] ——, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, p. 17, 1960. 37, 38, 228

[63] G. Chartrand and P. Zhang, *Introduction to Graph Theory*. McGraw Hill, 2005. 37

[64] M. E. J. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Physics*, vol. 46, p. 323C351, 2005. 41

[65] B. Bollobas and O. Riordan, "The diameter of a scale free random graph," *Combinatorica*, vol. 24, 2001. 42

[66] R. Cohen and S. Havlin, "Scale-free networks are ultrasmall," *Phys. Rev. Lett.*, vol. 90, p. 058701, 2003. 42

[67] B. F. K. Jacobson, Z. Huang and M. Sandler, "An analysis of on-line music artist networks," in *NetSci 08 – International Workshop and Conference on Network Science*, 2008. 42

[68] K. Jacobson, B. Fields, and M. Sandler, "Using audio analysis and network structure to identify communities in on-line social networks of artists," in *Ninth International Conference on Music Information Retrieval*, 2008. 42

[69] J. T. Sally Wehmeier, Colin McIntosh and M. Ashby, Eds., *Oxford Advanced Learner's Dictionary 7th Edition.* Oxford University Press, 2005. 42

[70] *X.200 : Information technology - Open Systems Interconnection - Basic Reference Model: The basic model*, International Standard Organization Std. 44

[71] R. Braden, "Rfc 1122 requirements for internet hosts - communication layers," *IETF Request for Comments*, 1989. 44

[72] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, "k-core decomposition: a tool for the visualization of large scale networks," *Advances in Neural Information Processing Systems 18, Canada (2006) 41.* 45, 232

[73] W. Odom, *CCNA ICND2.* Cisco Press, 2008. 45, 74

[74] C. Hedrick, "Rfc 1058 routing information protocol," *IETF Request for Comments*, 1988. 46, 47, 134

[75] J. Doyle, *Routing TCP/IP Volume I, 2nd edition.* Cisco Press, 2005. 46

[76] J. Doyle and J. D. Carroll, *Routing TCP/IP, Volume II (CCIE Professional Development).* Cisco Press, 2001. 46, 74

[77] J. Moy, "Rfc 2328 ospf version 2," *IETF Request for Comments*, 1998. 46, 48, 134

[78] R. Coltun, D. Ferguson, J. Moy, and A. Lindem, "Rfc 5340 ospf for ipv6," *IETF Request for Comments*, 2008. 46, 134

[79] Y. Rekhter, T. Li, and S. Hares, "Rfc 4271 a border gateway protocol 4 (bgp-4)," *IETF Request for Comments*, 2006. 46

[80] M. Luckie, Y. Hyun, and B. Huffaker, "Traceroute probe method and forward ip path inference," CAIDA: The Cooperative Association for Internet Data Analysis, Tech. Rep., 2008. 47

[81] K. Keys, "Ip alias resolution techniques," CAIDA: The Cooperative Association for Internet Data Analysis, Tech. Rep., 2009. 47

[82] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16 (1), pp. 87–90, 1958. 47

[83] L. Ford and D. Fulkerson, *Flows in Networks*. Princeton University Press, 1962. 47

[84] J. Siek, L. Lee, and A. Lumsdaine, *The Boost Graph Library, The: User Guide and Reference Manual*. Addison Wesley Professional, 2002. 48, 226

[85] T. H. Cormen, *Introduction to algorithms*. The MIT Press, 2001. 48

[86] R. Sedgewick, *Algorithms In C++, Part 5 Graph Algorithms (Third Edition)*. Pearson Education, 2002. 48, 227

[87] H. J. R. Albert and A.-L. Barabsi, "Error and attack tolerance of complex networks," *Nature(London)*, vol. 406, pp. 378–382, 2000. 49, 50, 70

[88] "Who needs hackers." [Online]. Available: http://www.nytimes.com/2007/09/12/technology/techspecial/12threat.html?scp=1&sq=Los+Angeles+airport+computer&st=nyt 49

[89] R. Cohen, K. Erez, D. ben Avraham, and S. Havlin, "Resilience of the internet to random breakdowns," *Physical Review Letters*, vol. 85, p. 4626, 2000. 50

[90] ——, "Breakdown of the internet under intentional attack," *Physical Review Letters*, vol. 86, p. 3682, 2001. 50

[91] A. Motter, "Cascade control and defense in complex networks," *Physical Reveiw Letter*, vol. 93, p. 098701, 2004. 51

[92] P. Crucitti, V. Latora, and M. Marchiori, "Model for cascading failures in complex networks," *Physical Review E*, vol. 69, p. 045104, 2004. 51

[93] J. Gleeson, "Cascades on correlated and modular random networks," *Phys. Rev. E*, vol. 77, p. 046117, 2008. 51

[94] P. Holme and B. J. Kim, "Vertex overload breakdown in evolving networks," *Physical Review E*, vol. 65, p. 066109, 2002. 51, 83

[95] S. Graham, "Europe blackout leaves millions in dark." [Online]. Available: http://biz.yahoo.com/ap/061105/europe_blackouts.html?.v=6 52

[96] D. Rohde and S. Gittlen, "Att frame relay network goes down for the count," *Network World www.networkworld.com*, 1998. 52

[97] J. Nagle, "Rfc 896 congestion control in ip/tcp," *IETF Request for Comments*, 1984. 52

[98] M. Allman, V. Paxson, and W. Stevens, "Rfc 2581 tcp congestion control," *IETF Request for Comments*, 1999. 52, 72

[99] S. Floyd, "Rfc 2914 congestion control principles," *IETF Request for Comments*, 2000. 52, 72

[100] J. Postel, "Rfc 793 transmission control protocol," *IETF Request for Comments*, 1981. 53

[101] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "Rfc 3550 rtp: A transport protocol for real-time applications," *IETF Request for Comments*, 2003. 53

[102] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "Rfc 3261 sip: Session initiation protocol," *IETF Request for Comments*, 2002. 53

[103] "A robot network seeks to enlist your computer." [Online]. Available: http://www.nytimes.com/2008/10/21/technology/internet/21botnet.html 53

[104] M. Handley and E. Rescorla, "Rfc 4732 internet denial-of-service considerations," *IETF Request for Comments*, 2006. 53

[105] M. Welzl, *Network Congestion Control Managing Internet Traffic.* John Wiley and Sons, 2005. 57, 58, 72

[106] S. H. Low, F. Paganini, and J. C. Doyle, "Internet congestion control," *IEEE Control Systems Magazine*, vol. 22, pp. 28 – 43, 2002. 58

[107] D. E. Comer, *Internetworking with TCP/IP, Vol 1 (5th Edition).* Prentice Hall, 2005. 58

[108] J. D. C. Little, "A proof for the queuing formula," *Operations Research*, vol. 9, No. 3, pp. 383–387, 1961. 61

[109] D. R. Kuhn, "Sources of failure in the public switched telephone network," *IEEE Computer*, vol. 30, No. 4, pp. 31–36, 1997. 69

[110] J. D. L. Ree, Y. Liu, L. Mili, A. Phadke, and L. DaSilva, "Catastrophic failures in power systems: Causes, analyses, and countermeasures," *Proceedings of the IEEE*, vol. 93, No. 5, MAY 2005. 69

[111] K. Nagel and M. Schreckenberg, "A cellular automaton model for freeway traffic," *J. Phys. I France*, vol. 2, p. 2221, 1992. 69

[112] M. E. J. Newman, "A measure of betweenness centrality based on random walks," *Social Networks*, vol. 27, pp. 39–54, 2005. 69

[113] M. Ericsson, M. Resende, and P. Pardalos, "A genetic algorithm for the weight setting problem in ospf routing," *routing J. Comb. Optim.*, vol. 6, p. 299, 2002. 70

[114] B. Fortz and M. Thorup, "Optimizing ospf/isis weights in a changing world," *IEEE J. Sel. Areas Commun.*, vol. 20, p. 756, 2002. 70

[115] G. Yan, B. H. T. Zhou, Z.-Q. Fu, and B.-H. Wang, "Efficient routing on complex networks," *Physical Review E*, vol. 73, p. 046108, 2006. 70

[116] P. Echenique, J. Gomez-Gardenes, and Y. Moreno, "Dynamics of jamming transitions in complex networks," *Europe Physics Letter*, vol. 71, p. 325, 2005. 70

[117] S. Sreenivasan, R. Cohen, Z. T. E. Lopez, and H. Stanley, "Communication bottlenecks in scale-free networks," *Physical Review E*, vol. 75, p. 036105, 2007. 70

[118] "The internet traffic report." [Online]. Available: http://www.internettrafficreport.com/ 71

[119] V. Jacobson, "Congestion avoidance and control," *Computer Communication Review*, vol. 18, no. 4, pp. 314–329, 1988. 72

[120] L. G. Roberts, "The internet is broken," *IEEE Spectrum*, vol. July, pp. 32–35, 2009. 72

[121] M. Woolf, Z. Huang, and R. Mondragon, "Building catastrophes: networks designed to fail by avalanche-like breakdown," *New Journal of Physics*, vol. 9, p. 174, 2007. 76, 77, 78

[122] A. Tizghadam and A. Leon-Garcia, "On congestion in mission critical networks," in *Computer Communications Workshops, 2008. INFOCOM*, 2008. 79

[123] R. Fujimoto, K. Perumalla, and G. Riley, *Network simulation*. Morgan & Claypool, 2007. 130, 133

[124] A. Law and W. Kelton, *Simulation Modeling and Analysis*. McGraw Hill, 2000. 130, 133, 135

[125] K. Fall and K. Varadhan, "The ns manual," online, a Collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC. 131, 134, 231

[126] "Omnet++, discrete event simulation system." [Online]. Available: http://www.omnetpp.org/ 131, 134, 136

[127] "Inet framework for omnet++ 4.0." [Online]. Available: http://inet.omnetpp.org/ 136, 137, 139

[128] "Omnet++ publications list." [Online]. Available: http://www.omnetpp.org/publications 136

[129] M. Barthelemy, B. Gondran, and E. Guichard, "Large scale cross-correlations in internet traffic," *Physical Review E*, vol. 66, p. 056110, 2002. 170

[130] ——, "Spatial structure of the internet traffic," *Physica A*, vol. 319, pp. 633–642(10), 2003. 170

[131] M. Barthelemy, "Betweenness centrality in large complex networks," *The European Physical Journal B*, vol. 38, pp. 163–168, 2004. 170

[132] "The gnu compiler collection." [Online]. Available: http://gcc.gnu.org/ 225

[133] "Intel c++ compiler." [Online]. Available: http://software.intel.com/en-us/intel-compilers/ 225

[134] A. T. Lawniczak, A. Gerisch, K. P. Maxie, and B. N. D. Stefano, "Netzwerk: migration of a packet-switching network simulation environment from ms windows pc to linux pc and to hpc," in *Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications*, 2002. 226

[135] E. Gelenbe and G. Pujolle, *Introduction to Queueing Networks*. John Wiley & Sons, 1998. 228

[136] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8 , Issue 1, pp. 3 – 30, 1998. 228

[137] J. Winick and S. Jamin, "Inet-3.0: Internet topology generator," online. 230

[138] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: Universal topology generation from a users perspective," April 12 2001. 231

[139] G. Battista, P. Eades, R. Tamassia, and I. Tollis, "Algorithms for drawing graphs: An annotated bibliography," *Computational Geometry: Theory and Applications*, vol. 4 (5), pp. 235–282, 1994. 232

[140] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Information Processing Letters*, vol. 31, pp. 7–15, 1989. 232

[141] T. Fruchterman and E. Reingold, "Graph drawing by force-directed placement," *Software–Practice & Experience*, vol. 21 (11), pp. 1129–1164, 1991. 232