Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D –70569 Stuttgart

Diplomarbeit Nr. 3082

Development of a Load Model for Distributed Systems

Kai Zhou

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer:	Dipl.-Inf. Andreas Grau
begonnen am:	21. 09. 2010
beendet am:	23. 03. 2011
CR-Nummer:	C2.0, C2.l, C2.2, C2.5

Abstract

NETplace is an efficient algorithm to assign virtual nodes to physical nodes on the network emulation testbed, while reducing the experiment runtime for network emulation up to 64%. As an assumption of this algorithm, a detailed defined cost model for communication cost has been provided. This cost model needs expected data rates of the links between each pair of virtual nodes as well as CPU load (in CPU cycles) on the virtual nodes, which are the experimental data produced by SoftwareunderTest (SuT). Therefore, the goal of the thesis is to define a generic load model to efficiently provide placement algorithm with a realistic estimation of experimental load data.

In order to reach this goal, several problems should be solved. First, there are thousands of network links and virtual nodes in the model, so it is impossible to manually inquire and input all experimental CPU load and data rates into the model, because it takes too much time. A possible approach to resolve this problem is to divide the nodes into several groups, in which all the nodes have nearly similar characteristics. That is to say, a node classification is made. Thus, we only need specify one node for each group, and the other nodes can be automatically assigned according to the node classification. Second, nowadays the network is already very large, so it still costs much time, if we classify the nodes in the whole network together. A better solution is to analyze the nodes in part networks with the help of network clustering. So generally speaking, through combining the network graph clustering with the nodes classification we can provide the load information, which NETplace needs.

Kurzfassung

NETplace ist ein effizienter Algorithmus, um mehre virtuelle Knoten zuzuweisen jedem physischen Knoten, der auf der NET liegt. Dabei wird die benötigte Laufzeit auf bis zu 64% reduziert. Dabei wird angenommen, dass ein Kostenmodell für die Komummunikationskosten bereitgestellt wird. Dieses Kostenmodell benötigt die Datenrate zwischen den verschieden virtuellen Knoten sowie die CPU-Auslastung der virtuellen Knoten. Diese sind die experimentelle Daten, die vom SuT produziert werden. Daher ist das Ziel dieser Diplomarbeit, ein allgemeines Modell zu definieren, das effizient den placement - Algorithmus mit einer realistischen Einschätzung der experimentellen Daten liefert.

Um dieses Ziel zu erreichen, müssen einige Probleme gelöst werden. Es gibt tausende Netzwerkverbindungen und virtuelle Knoten in diesem Model, weswegen es unmöglich ist, alle experimentelle Daten manuell abzufragen und aufzunehmen. Dies würde zu viel Zeit kosten. Ein Lösungsansatz wäre es, die Knoten, die alle ähnliche Eigenschaften haben, in Gruppen zu unterteilen. Das heißt wir brauchen eine Knotenklassifizierung. Daher müssen wir nur die Daten einem Knoten für jede Gruppe abfragen und aufnehmen. Die anderen werden automatisch über die Knotenklassifizierung bestimmt. Heutzutage sind die Netzwerke sehr groß geworden, weshalb es sehr viel Zeit in Anspruch nehmen würde, wenn wir alle Knoten im gesamten Netzwerk klassifizieren würden. Eine bessere Lösung ist es, nur einen Teil der Knoten im Netzwerk mithilfe des Network-Cluster zu analysieren. Kurz gesagt, durch das Kombinieren von Network- Graph – Clustering mit der Knotenklassifizierung können wir die Daten bereitstellen, die NETplace braucht.

Acknowledgments

I would like to sincerely thank my tutor Andreas Grau for his help, support and guidance during my diploma thesis. He put me on the road to doing good research and his easy accessibility to discuss various issues was invaluable during my research.

Chapter 1

Introduction

1.1 Motivation

Today more and more dynamic[1] large-scale distributed systems are used in world, so network evaluation becomes more and more important. In order to monitor the load information in whole system of such dynamic large-scale network, an appropriate emulation system should be set up. Now the NET (Network Emulation Testbed) project [1] of the Institute of Parallel and Distributed Systems (IPVS) at University of Stuttgart is a solution for such network emulation. The system is consisted of a 64-nodes PC cluster with flexible hardware and software tools. The nodes are connected with a high performance switch. Each node is able to emulate many virtual nodes, which represents a network component, (such as a terminal, router, gateway or switch etc.) in real world network. And those virtual nodes are running now in a network emulation environment, which means Time Virtualized Emulation Environment (TVEE) [2] here.

A basic concept of network emulation is node virtualization, in which multiple virtual nodes are put onto each physical node of the emulation bestbed. But just using node virtualization is not scalable for large network, because the number of physical node is not always scalable. On the other hand, we also cannot put too many virtual nodes on each physical node. Therefore, time virtualization must also be used in TVEE, which can reduce the load on physical nodes through using a virtual time running slower than the real time on them. If a load is higher than the capability of a physical node, it will cause bias in the results, which is named overload. Conversely, if the load is much lower than the capability on a physical node, many calculation resources will be wasted, which is named underload. In order to keep the load between overload and underload, an adaptive virtual time is needed, which can adjust the virtual clock rate on physical nodes according to the load of them.

[1] Here “dynamic” means that the load in the network such as CPU load and data rates is often changed, but not the network topology.

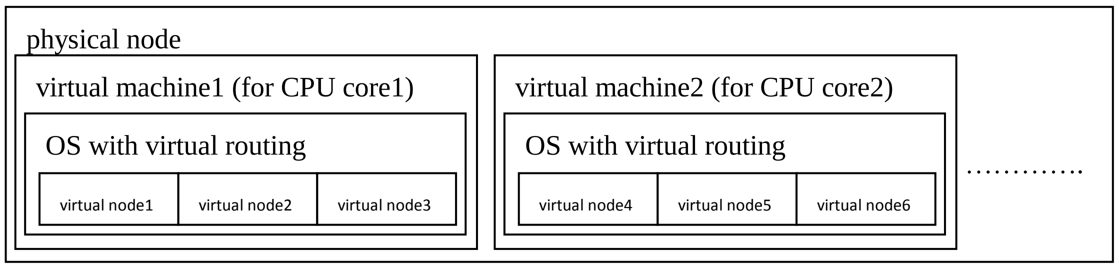


Fig.1.1 TVEE architecture

As showed in Figure 1.1, TVEE provides a nested virtualization. A virtual machine (VM) is running on each CPU core of each physical node, which offers virtual time to the operation system and SuT in VM. The virtualization inside VM is virtual routing (VR), which is more lightweight than VM. Using VR, the resources are partitioned into parts to create virtual nodes.

In order to assign the appropriate virtual nodes onto the physical nodes of the 64-nodes PC cluster, an automatic placement algorithm NETplace is used. As we said, the physical nodes must not be overloaded during the placement. We can achieve it by adjusting the dynamic virtual time. But if the virtual time is too slow, then the experiment runtime will be very long. If we deploy the virtual nodes onto the most suitable physical nodes, the running time of the experiment can be reduced substantially. So it is important to find out an efficient algorithm to minimize the experiment runtime. Fortunately NETplace is such an efficient algorithm to achieve this assignment. [3]

For the input for NETplace algorithm, two input parameters, expected CPUs' load on the virtual nodes and expected data rates of the data links between each pair of virtual nodes must be provided. Therefore, in this thesis an efficient load model will be established, in which the two parameters can be precisely offered as soon as possible.

1.2 Goal of the study

In order to place the virtual nodes onto the physical nodes of the PC cluster, a generic cost model for the communication of the systems is necessary. With two input parameters (CPU load and expected data rates), together with defining data links between virtual nodes in three different types (intra-vm, inter-vm and inter-pnode links), such a cost model can be set up. And the NETplace algorithm is based on it. But the two parameters, expected CPU load and expected data rates are just provided as an assumption now.

In order to get the expected CPU load and expected data rates, the load information in the system must be inquired. But the network may be very large. Maybe there are thousands of nodes and edges in the load model. Manually inquiring all the load information of all the virtual nodes and data links and offering them to NETplace are impossible. So it is important to find out a relative automatic assignment method. A possible approach is to define a generic load model[2] at first. According to the load model, a node classification can be carried out. Besides that, considering the large-scale network, a network clustering is also necessary. If network clustering is executed, the runtime of the node classification can be reduced. And the information of network topology could be used to achieve the network clustering and node classification.

Therefore, the goal of this diploma thesis is to set up an algorithm, in which the expected CPU load and expected data rates can be provided for NETplace as precisely as possible.

The algorithm is named "NETclassify", which is running as following: Firstly, we give a detailed and generic load model for real world networks, the characteristics of the elements in the network will be mapped into this model according to the given real world network topology. Then a network is clustered, and the node classification algorithm can be carried out in relative small part network. Finally, in each class a node will be elected. Only the load of those nodes and data links from them is manually inquired, calculated and inputted, the load of the other data can be automatically allocated according to the node classification.

1.3 Outline

The reminder of this thesis is structured as follows:

Chapter 2 represents related work and the differences to this diploma thesis.

Chapter 3 describes possible design ideas for the network classification and issues design approaches of network classification.

Chapter 4 shows the implementation of the network classification, which is based on the concept in chapter 3.

[2] "Generic load model" here means that this load model is suitable for all kinds of networks, not for a specific one.

In chapter 5 the procedures and results of evaluation of the implementation, which are defined in chapter 4, are described.

In chapter 6 a summary of the diploma thesis and the possible enhancements of the diploma thesis are given.

Chapter 2

Related work

In this chapter, the related work of this diploma thesis will be introduced. The purpose of the study is to efficiently provide the expected load on the virtual nodes and expected data rates on the data links to NETplace. In order to reduce the work for inquiry and input of these two parameters in a large-scale network, a suitable node classification algorithm is needed. After the successful classification, the nodes, which have similar CPU load and similar data rates on the outgoing paths, are in the same group. This is also the goal of node classification. In order to achieve the goal, the communication capacity from a node to other nodes is measured. A possible approach is to create the Node Classification Algorithm with the help of bandwidth estimation for communication, while bandwidth is the worst-case estimation of the expected data rates. The reason why we use the bandwidth instead of the data rates here is that, the data rates on the data links are changeable.

The concept of node here is the network component. The nodes classification, which we discuss here, is the network components classification.

Automatic nodes classification is a method, with which the nodes could be automatically classified in many different classes by one or some properties of them. In some cases, we may classify the nodes by some properties of them, while the nodes may be classified by some other properties of them in some other cases. The classification is based on the selection of the properties of the nodes, while the selection is decided by the demand. For example, the result of automatic bandwidth estimation is the property we should use here.

Unfortunately, as far as I know, there is no systematic approach for automatic nodes classification with the help of automatic bandwidth estimation. However, there exist some approaches for automatic nodes classification, if we do not care about the properties of the nodes for classification.

Therefore, we can divide the problem in two part problems and focus on the related work on them: One is automatic nodes classification, and the other is automatic bandwidth estimation.

2.1 Automatic Nodes Classification

For automatic nodes classification, there is an approach of automatic classification of node types in switch-level description. [18]

This approach is just the classification of nodes types in switch-level, which is a part of network nodes. The metric for classification for this approach is the memory quality of the switch: weather it is temporary or memory.

In this approach, according to some properties of the nodes, here is the property of the switch: temporary and memory. And the concrete method is:

If the memory of a node is lost and it cannot affect the circuit operation, then it is classified as a temporary node. On the contrary, if the memory of a node is maintained, then it is classified as a memory node.

However, the metric for classification is independent of bandwidth estimation.

2.2 Automatic Bandwidth Estimation

Here we want to find out an efficient method to measure the communication capacity in the whole network. The most accurate value of communication capacity is the real-time data rates on data links. With them we can know how much data exist on a link.

There are some approaches to estimate the bandwidth on a link. [19] [20]

For example, the packet pair mechanism is a reliable method to measure the bottleneck link capacity on a network path and the initial gap increasing (IGI) method and the packet transmission rate (PTR) method are two good measurements for available bandwidth. The themes in these papers are focused on how to measure the bottleneck link capacity or the available bandwidth capacity on a link.

However, in our approach, we assume that, the method of measurement of the bottleneck link capacity or the available bandwidth capacity on a link is known. We want analyze the transport capacities between nodes in a network with the influence of other communication in the same network. So it is not suitable for us.

Chapter 3

Design Issues

In this chapter, the design issues of the algorithm NETclassify will be represented.

For a better understanding, at first, a basic architecture of NETclassify is introduced.

After that, some approaches of network clustering are described. Through network clustering, a network is split into many small part networks.

Furthermore, the Node Classification Algorithm is discussed in detail. As a foundation of Node Classification Algorithm, a transmission cost model is set up. This model could help us to decide the routing paths between nodes in network, which is very important for the calculation of data transport capacities. As a result of successful division, the nodes with similar characteristics are in the same group.

Finally, a manual inquiry and assignment of the CPU load on a node and data rates on the outgoing data links of it in each group is given, and according to the manual input, the CPU load on other nodes and data rates on the outgoing data links of them in the same group are also automatically assigned.

3.1 Basic Architecture

Above all, the relationship between NETclassify and its background is introduced. In the background of NETclassify the input and output of the algorithm are described.

As shown in Figure 3.1, the Network Topology Generator can automatically map the network topology to a directed graph, in which nodes represent the network components and edges between the nodes represent the network data links. Furthermore, the characteristics of network components (i.e. nodes) and data links are also mapped. The characteristics of nodes here are node's ID and the software running on it, while the characteristics of data links are link source node's ID, link destination node's ID, packet loss rate, maximal delay, and bandwidth on the link. This is one important input for NETclassify. The other input is sample data rate, which must be manually inquired by human once for each group. Certainly, frequently manual inquiring takes much time, so the number of groups is kept as small as possible.

The output of NETclassify is two kinds of values for NETplace[3] algorithm: one is CPU load on the virtual nodes, while the other is the data rates of the edges between them.

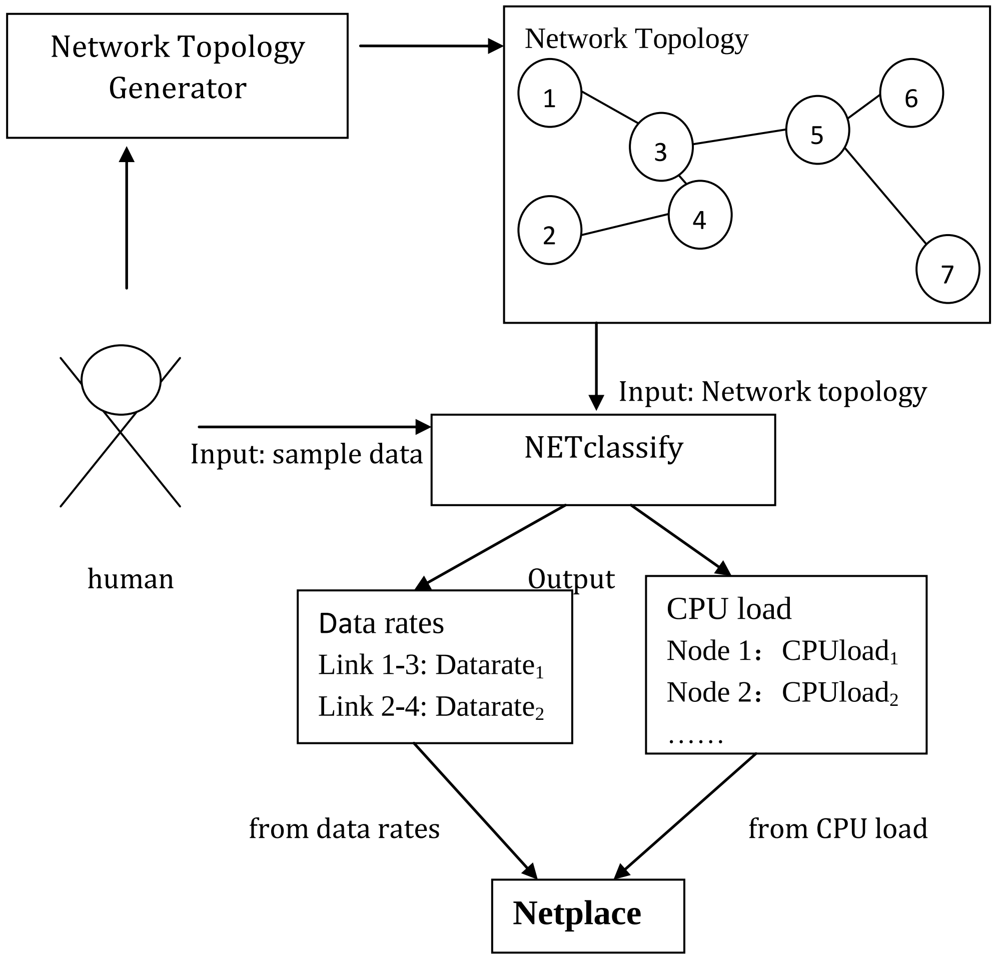


Fig 3.1 background of the diploma thesis

After the explanation of the task, now the rough process of the NETclassify will be introduced. It is running as following in Figure 3.2:

In the picture, the rough process of the NETclassify is shown. Firstly, as an input, a directed graph is given, in which all the parameters of the network topology is written. This graph can be very large. In order to reduce the runtime of the NETclassify, a network clustering is carried out. As a result, a graph is into many small part graphs split.

With the cost information (i.e. bandwidth, maximal delay and packet loss rate) on the links in each part graph, a transmission cost model is set up. Through this model we could know the shortest path between every pair of nodes, with which the routing information is known. The routing information is an important condition of the Node Classification Algorithm.

Furthermore, a preparation work is necessary, in which some basic concepts and definitions are given. After that, the nodes are divided with similar functions in identical groups through the Node Classification Algorithm in the end.

Besides the classified nodes, we still need sample data and suitable assignment method for an output. The sample data here is the CPU load of a random selected node and the data rates of the outgoing links from it in each group. We inquire the values of them in the network at first. Then with the assignment method the expected CPU load of other nodes and expected data rates of other data links in the same group can be automatically assigned.

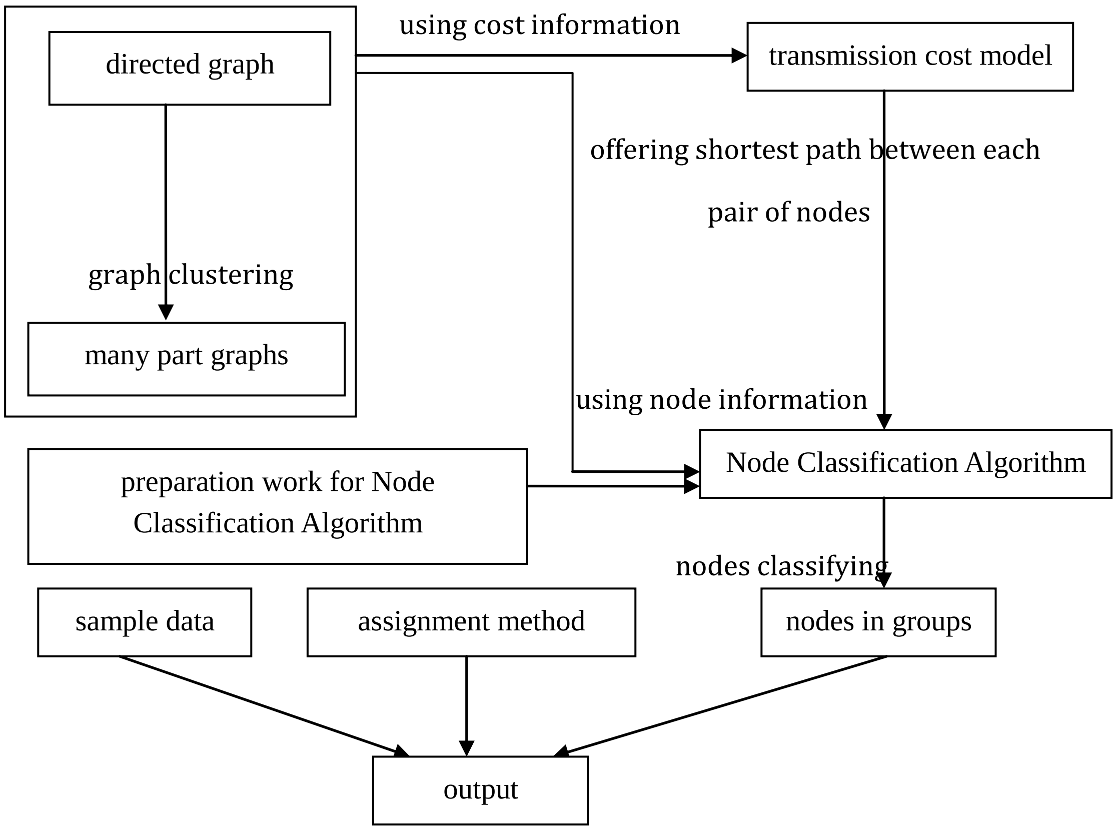


Fig 3.2 process of NETclassify

3.2 Network Clustering

In order to reduce the runtime of NETclassify, a network clustering algorithm is executed, which is also a graph clustering algorithm, because the network topology has been already mapped into a graph. Then the problem of network clustering becomes a problem of graph clustering.

The structure of the real world network must not be destroyed by the clustering algorithm. Otherwise, after Node Classification Algorithm we could not get a correct result.

For example, for a node classification we do need to analyze the flow of load information on the communication paths between each pair of nodes. The structure information of the edges in a graph is very important for the analysis of the communication between each pair of nodes. The communication between each pair of nodes within groups is much more than the communication between groups.

In some of the clustering literature, such a group in a graph is also named a cluster or community. [11]

3.2.1 Different Clustering Methods

Up to now, there are many network clustering algorithms [4] [5]. The global methods for graph clustering can be a flat structure clustering, which comprises single partition and cover, or defined as a hierarchical structure clustering, where each top-level cluster is always composed of sub-level clusters.

Almost all the structures of the networks today belong to hierarchical structure. For this reason, we will put more effort on the hierarchical structure and search a suitable clustering algorithm in it.

In a hierarchical structure, a single cluster in a level can be composed further of several sub-clusters in the lower level. Certainly, it can also merge with other clusters in the same level to a large cluster in the higher level. The number of clusters in each level is different. The higher the level is, the lower the number of clusters is. For different requirements, (for example, the graph must be split in more than 100 part graphs or the number of nodes in each part graph cannot be more than 80.) we can find out a suitable dividing possibility in one level.

In Figure 3.3 there is a dendrogram of hierarchical structure for a 23 nodes in a graph. In the highest level, the root cluster is an entire dataset, while the 23 elements are the leaf clusters in the lowest level. Between them there are four intermediate levels. Each level in the dendrogram, which is marked by dotted lines, can be regarded as a kind of clustering.

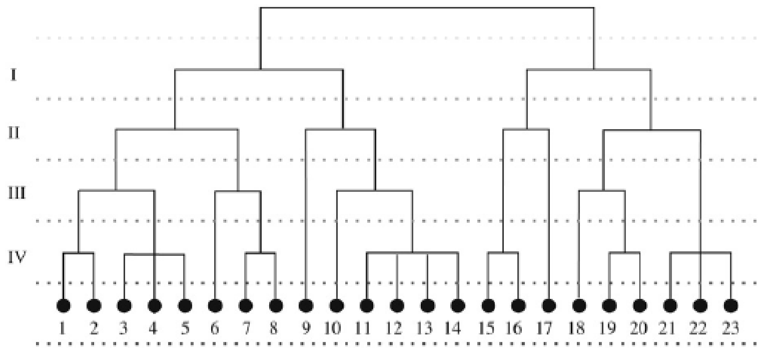


Fig 3.3 a dendrogram of hierarchical structure [4]

The clustering method of the hierarchical structure can be divided into two big classes: divisive global clustering (top-down, recursively partitioning) and agglomerative global clustering (bottom-up, merging).

Following algorithms belong to the divisive global clustering method:

Such as cuts, maximum-flow, betweenness, resistor networks and so forth.

3.2.1.1 Cuts and Maximum-Flow

In cuts method the graph is split in two part graphs by removing a cut[3]. Usually we are looking for a small cut, but there are various possibilities. The most famous one, minimum-cut can be considered with maximum-flow algorithm. [12, 13] With min cut/max flow method, we can find out the shortest path, max flow, min cost-flow in the graph. But it is not useful for our task, because the min cut/max flow is used for a one source one sink approach, and the graph is a directed (weighted) graph.

3.2.1.2 Betweenness

According to the idea of Newman and Girvan, the weights on the edges are determined by the structural properties of the graph. The weight on each arbitrary edge {n1, n2} is the number of the shortest paths connecting any pair of nodes that passes through the edge. [4] And this weight of the link is the betweenness of the link. Therefore, the edge, whose betweenness with the highest value can be easily removed. If an edge is the connector of two part

[3] A partition of all the nodes in a graph into two nonempty sets is called a cut.

networks, then each communication between each pair of nodes in different parts will go through it. The structure of the graph has a smallest influence on the removal of such an edge, because the structures of part networks are not destroyed. This method is suitable for my work. And I will introduce it further in detail later.

3.2.1.3 Resistor Networks

In this method the graph is mapped into an electric circuit, in which a unit resistor is placed on each edge and unit current flows (or random walks) into and out of the source and destination vertices. The random-walk and current-flow measures are proved precisely the same by M. E. J. Newman and M. Girvan in the Literature [7]. However, the time complexity for this method is very high. It takes O((n+m)*m*n^2) for the entire community structure algorithm, where m is the number of edges in a graph and n is the number of nodes.

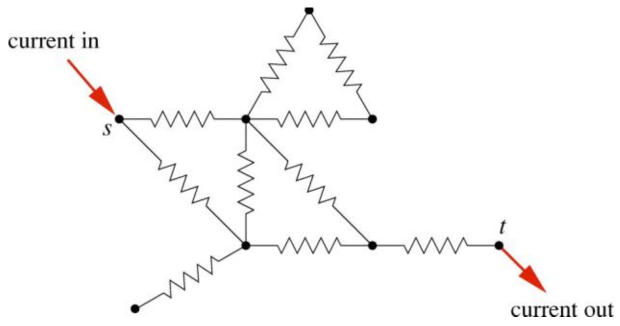


Fig 3.4 an example of type resistor networks from source s to destination t. All the black points are the nodes in the graph, and the resistors represent the data links between them. [7]

Now in NETclassify I will choose a clustering algorithm, which is based on betweenness of the edges. As we have already said, that the structure information of the edges in the graph is very important for the analysis of the communication between two nodes. And the structure information here is the edge-betweenness, which is the number of the shortest paths between any pair of nodes that pass through the edge. The higher the edge betweenness of an edge is, the more probable the edge is a boundary of two part network.

3.2.2 Girvan-Newman Algorithm

Girvan-Newman algorithm (Girvan & Newman, 2002) [6] [8] is such an algorithm, which can be used in Network clustering and is running as following steps:

1. The betweenness of all the edges in the graph is calculated.
2. The edge, whose betweenness is the highest, is removed.
3. The betweenness of the edges, which has an influence on the last removal of the edge, is recalculated.
4. Repeat step 2 and step 3 until there is no edge in the graph.

However, the Girvan-Newman algorithm has a big problem, that this algorithm is not scalable for a large network. As what is pointed in [7], the time complexity is very high, the algorithm is running in O (m^2n), where m is the number of edges and n is the number of nodes or O (n^3) for a sparse graph (because in a sparse graph, m is as big as n).

3.2.3 Clauset-Newman-Moore

There are some faster approaches. One of them is the Clauset-Newman-Moore Algorithm [9], which is based on a greedy optimization. A key definition here is the concept modularity.

3.2.3.1 Modularity

Modularity is a metric, which represents the result of division. It is showed, whether the division is good or not. The value of modularity is always between 0 and 1. If it is a good division, i.e. the value of modularity is relative high, it means, that the communication within each part is much more than the communication between parts.

In literature [7], the following detailed definition of Modularity is given:

For each particular division of a network into k communities, a k×k symmetric matrix e is set up. Each element e_{ij} in the matrix is the fraction of all edges in the network that link vertices in community i to vertices in community j. The sums of row (or column) in the matrix $a_i = \sum_j(e_{ij})$ represent the fraction of edges that connect to vertices in community i.

The formula of modularity measure is defined:

$Q = \sum_i(e_{ii}-a_i^2) = \text{Tr } e-||e^2||$

Where Tr e = $\sum_i(e_{ii})$ is the fraction of edges in the network that connect vertices in the same community and ||x|| indicates the sum of the elements of the matrix x. [7]

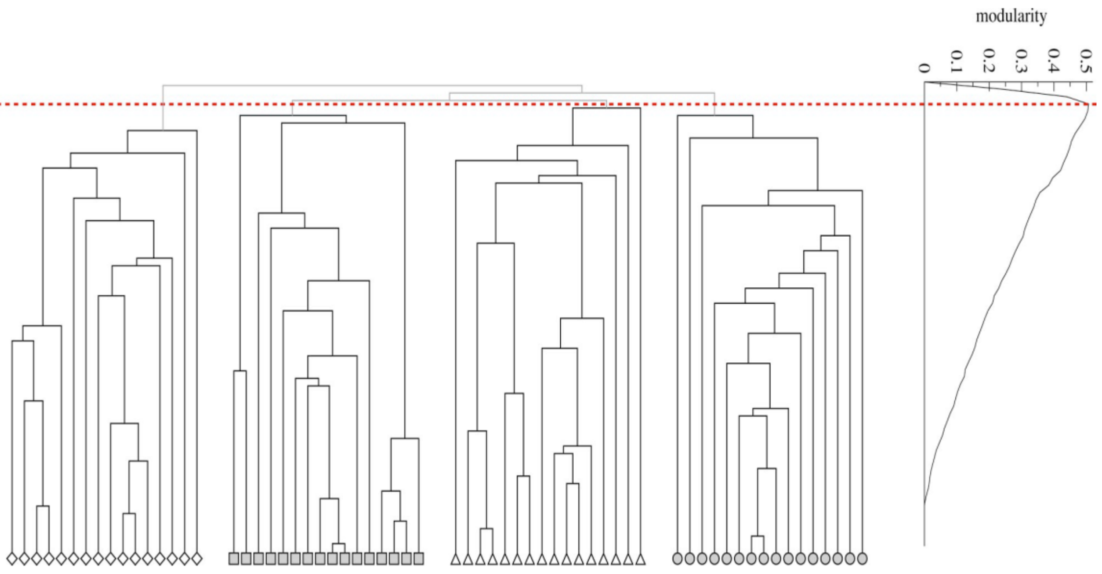


Fig 3.5 an example of network clustering with the value of modularity

Source: page 8 of paper [7]

In the Figure 3.5, under the best division, the graph is split into four part graphs. At that time, the value of modularity is 0.5. If the number of part graphs is smaller than four, the less the number of part graphs is, the smaller the value of modularity is and vice versa. Obviously, the value 0.5 is the highest value here. In this structure, division into four part graphs is the best result.

The time complexity of the Clauset-Newman-Moore Algorithm is O (m*d*log(n)), where m is the number of edges, n is number of nodes and d is the depth of the dendrogram, which describe the structure of the community division. In a sparse graph, the depth d equals to log (n) and m is also as big as n. So the time complexity becomes O ($n*\log^2 n$).

Clauset-Newman-Moore is an efficient algorithm to find community structures in large network. A community structure is a group of nodes, in which the density of edges is higher than density of edges between groups. It could be a real world department of a company. So using the Clauset-Newman-Moore algorithm does not destroy the structure of the real world network.

3.2.4 Weighted Graph

The most networks are studied in binary form, that is to say, either the edge between two nodes exists or not. A simple expression can be written in a matrix M: [10]

$$M_{ij} = \begin{cases} 1, \text{if i and j are connected,} \\ 0, \text{otherwise.} \end{cases}$$

Where i and j belongs to the nodes set N.

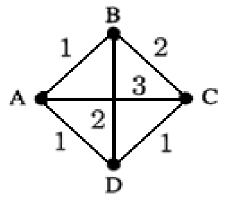
In the real world, some graphs are weighted graphs. The weight can be a property of the graph. Then the expression becomes:

$$M_{ij} = (\text{weight of the connection from i to j})$$

Where i, j $\in$ N.

Now in our approach, if a weighted graph is used, bandwidth can be a weight of the connection. The higher the bandwidth is, the higher the weight is.

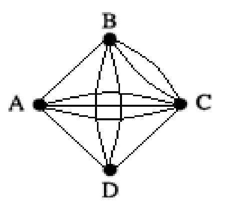
For example,



$$\equiv \begin{array}{cccc|c} A & B & C & D & \\ 0 & 1 & 3 & 1 & A \\ 1 & 0 & 2 & 2 & B \\ 3 & 2 & 0 & 1 & C \\ 1 & 2 & 1 & 0 & D \end{array}$$

Fig 3.6 matrix is set up with weight

A basic idea of the weighted graph is that the weight on a link represents the number of communication of links on this connection.



$$\equiv \begin{array}{cccc|c} A & B & C & D & \\ 0 & 1 & 3 & 1 & A \\ 1 & 0 & 2 & 2 & B \\ 3 & 2 & 0 & 1 & C \\ 1 & 2 & 1 & 0 & D \end{array}$$

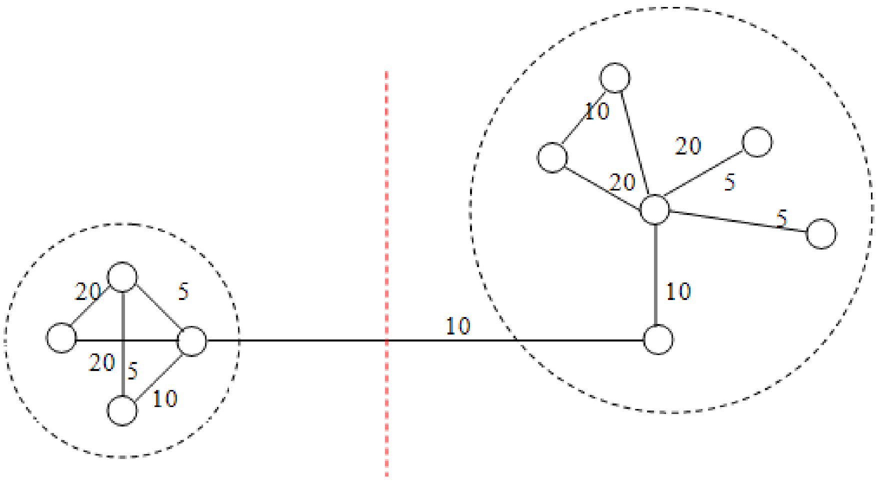
Fig 3.7 weight is represented by several links

Under this condition, that the bandwidth is the weight of the link:

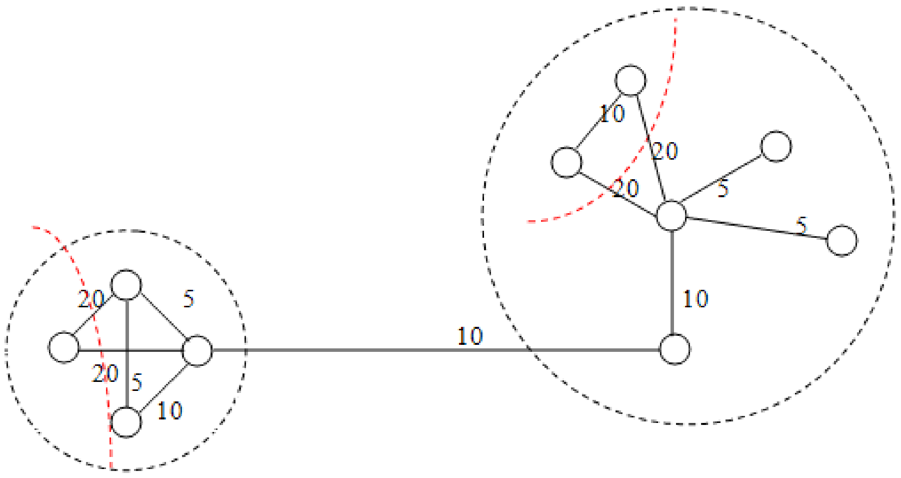
If the bandwidth of a link is more than 1, we will assume that, there are multiple edges on the link, which is shown in Figure 3.7. Then the betweenness of such a link also becomes higher. According to the idea of Network clustering algorithm, which uses betweenness method, if the betweenness of two nodes are very high, then the link between the two nodes is assumed to be a link between clusters.

Thus, the nodes, which are connected with a low-value bandwidth, have a better chance in the same cluster. The link, which has a high bandwidth, is considered as a boundary between two clusters, and will probably be removed.

But in the real world, the bandwidth of a link, which is used to connect two communities, cannot be high. If we use this weighted approach, perhaps the link cannot be recognized as the boundary of a division. And a link in a community, whose bandwidth is high, is recognized as the boundary of division.



(a)



(b)

Fig. 3.8 two results of network clustering (the digit on each link is the bandwidth of the link)

In Fig 3.8, the picture (a) shows us the division of the network without considering weight on each link.

With the weight of the link, the result of division is shown in picture (b), in which the structure of network is destroyed.

Therefore, the weighted graph model is not used here, unless a suitable property of a link can be found as the weight of the link instead of bandwidth.

As showed in Figure 3.2, we need a Transmission Cost Model for each part directed graph, which represents the transmission cost in each part network.

3.3 Transmission Cost Model

The Transmission Cost Model consists of a set N of virtual nodes and a set E of edges between nodes. Once data is transmitted through an edge $e_i \in E$, it takes Cost c_i. The set of Cost for all the edges is C. Cost c_i is related with upload/download bandwidth, maximal delay and packet loss rate, which are already provided by the network topology. Obviously, the higher the bandwidth is, the lower the Cost is. On the contrary, the higher the maximal delay and packet loss rate are, the higher the Cost is.

Therefore, the formula is defined as following:

$$C = \frac{\mathrm{D} \times \mathrm{L}}{\mathrm{B}}$$

Where D is the maximal delay, L is the packet loss rate and B is the bandwidth.

But for a directed graph, there are two directions for each link. Therefore, for arbitrary edge $e_i \in E$, the maximal delay d_{ui} and d_{di} belong to D, where d_{ui} is the upload maximal delay of edge e_i and d_{di} is the download maximal delay of edge e_i; the packet loss rate l_{ui} and l_{di} belong to L, where l_{ui} is the upload packet loss rate of edge e_i and l_{di} is the download packet loss rate of edge e_i; the bandwidth b_{ui} and b_{di} belong to B, where b_{ui} is the upload bandwidth of edge e_i and b_{di} is the download bandwidth of edge e_i.

Thus, for each edge e_i there are two kinds of Cost, c_{ui} and c_{di}:

$$\mathrm{c_{ui}} = \frac{\mathrm{d_{ui}} \times \mathrm{l_{ui}}}{\mathrm{b_{ui}}}$$

And

$$c_{di} = \frac{d_{di} \times l_{di}}{b_{di}}$$

Where c_{ui} is the upload Cost of edge e_{i}, and c_{di} is the download Cost of it. They are the weights of edge in two directions.

Now with a suitable shortest path algorithm, we can get the shortest paths between each pair of nodes in this directed weighted graph, where the upload Cost and download Cost defined before are the weights on the edges.

Generally speaking, in a routing algorithm, the path, which has the lowest Cost between two nodes, is selected as the routing path between them. So with the shortest Cost paths between each pair of nodes, we will get the routing information in the network.

3.4 Preparation Work of Node Classification Algorithm

In this section, some preparation work for Node Classification Algorithm is described.

The main task of the Node Classification Algorithm is to classify the nodes in groups by comparing one or some characteristics of the nodes.

In NETclassifty, the reason why we need a Node Classification Algorithm is the necessity to reduce the effort of assignment of expected CPU load on the nodes and expected data rates on the data links between nodes.

Therefore, the characteristics of the nodes we need for node classification algorithm are the software running on the nodes, which has a great influence on the expected CPU load and data transport capacity of the node, which determines the outgoing expected data rate.

The critical value, real time transport capacity is a dynamic value. A possible substitute is to use the speed of the network links of the virtual network topology, i.e. bandwidth is used as a worst-case estimation of the transport capacity. So the bandwidth on each link becomes also important.

As an input, the network topology is given, i.e. all parameters of the real world network are known. Therefore, after the successful mapping, we know which software is running on which node as well as the upload/download bandwidth on each link.

3.4.1 Preparation Work for Nodes

Firstly, we focus on the characteristics of nodes.

Not all the nodes will communicate with other nodes in a network. Generally speaking, a communication exists just between p2p nodes or client and server. We can also say that, communication exists between terminals. This is due to the different functions of nodes.

Concerning the software running on the nodes, we can divide the nodes into two big classes by the nodes' functions at first. One is called terminal, whose function is just to send and receive data while the other is named router[4], whose function is to forward data, i.e. get data in incoming paths and put them in suitable outgoing paths, no message is produced in the transmission. Obviously a node in terminals is not similar as a node in routers. Concerning the completely different roles of nodes, nodes can be discriminated in class terminal and class router. And through checking the software on the nodes, we know the different roles of different nodes.

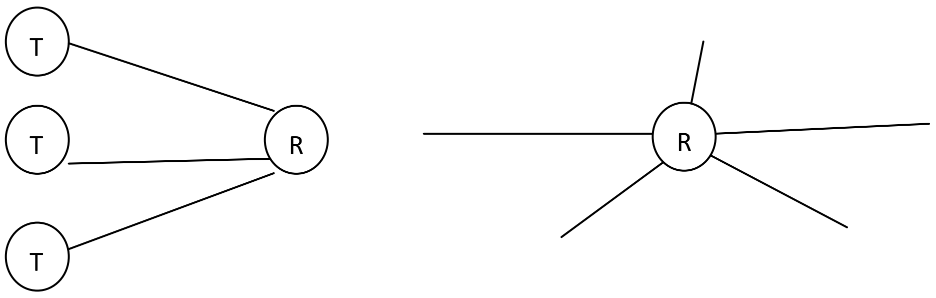


Fig 3.9 normal connection structures for terminal and router

As what is in Figure3.9 shown, our assumption is that, many terminals may connect to a router, but each terminal connects just to one router. So the connection grad of a terminal is one, and the connection grad of a router is bigger than two.

Certainly, there are also some extreme cases: for example, maybe a router has only one connecter, that is to say, the data, which is transported to the router cannot be forwarded. In that case, we can analyze the network without regard for this router.

[4] The "router" here is not only a hardware router, it can also be a gateway, switch and so on.

3.4.2 Preparation Work for Edges

Then the characteristics of the edges will be introduced.

In this directed graph, for each edge, there are two values for the bandwidth. One is for upload bandwidth, and the other is for download bandwidth. These two values could be used as two directions of data transformation. Bandwidth is the maximal transport capacity of a data link, which is fixed for a data link, while the data rate is the actual transport flow of a data link, which can be often changed.

In Figure 3.10, there is a link between node 1 and node 2.

As showed in the picture a, the upload bandwidth and download bandwidth on a link is fixed. However, the data rates on the upload and download channels of link can be independently changed. In picture b and c, the data rates are different.

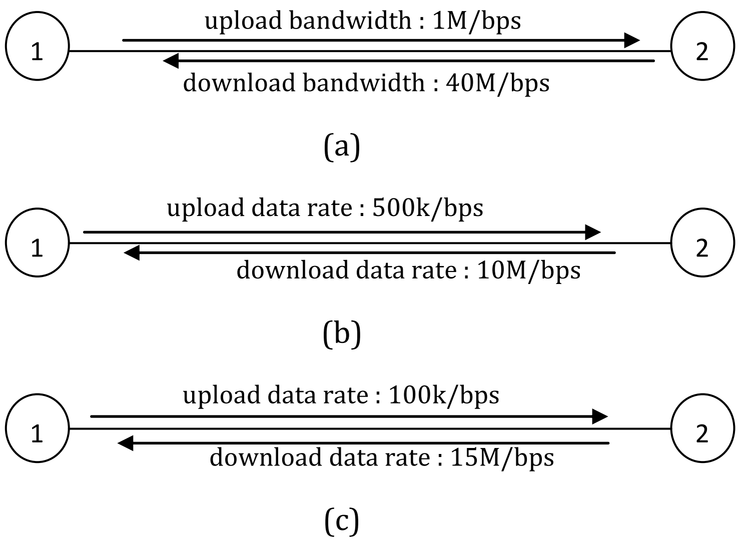

upload bandwidth : 1M/bps
1
2
download bandwidth : 40M/bps
(a)
upload data rate : 500k/bps
1
2
download data rate : 10M/bps
(b)
upload data rate : 100k/bps
1
2
download data rate : 15M/bps
(c)

Fig 3.10 difference of bandwidth and data rate of data link

The relationship between data rate and bandwidth on a link in the same direction is:

$$0 \leq \text{data rate} \leq \text{bandwidth}$$

3.5 Node Classification Algorithm

We want to classify the nodes in some groups, where the nodes have similar characteristics. Above all, we should make clear that, why we need nodes classification. As what we have already talked about, we want to reduce the runtime of data inquiry and assignment. And the data, which needs to be inquired and assigned in the model for NETplace, is CPU load on the nodes and data rates between nodes. Therefore, the nodes in the same group should have similar CPU load and data transport capacity.

3.5.1 Basic Concepts

Before the introduction of the Node Classification Algorithm, we do need explain some basic concepts.

3.5.1.1 CPU Load and Data Transport Capacity

At first, the two concepts CPU load and data transport capacity are introduced. CPU load here is the load of CPU on a node.
And the data transport capacity of a node here is divided in two cases:

1. data transport capacity of a terminal:
   The data transport capacity from a terminal to all other corresponding terminals. (Communication exists just between terminals, which has been in chapter 3.1 discussed.) The reason, why only the outgoing transport capacity is considered, is that, every outgoing link of a node is also an incoming link of another node at the same time. The set of outgoing link of all the nodes is the set of links in the network. If we consider all the outgoing link of nodes, all the links in the network have been already considered. After we use the outgoing transport capacity to calculate the data transport capacity of a terminal, the nodes in the same group have same outgoing data transport capacity, i.e. they have similar expected data rates on the corresponding outgoing links.
2. data transport capacity of a router
   The data forward capacity of a router, which is determined by the communication between terminals through this router.

3.5.1.2 Analysis for CPU Load

We can make a compare among all software running on each node, because the conclusion, in which the CPU load on some nodes is similar, only works on the premise that the software is the same. If the premise is met, and the data transport capacities of the nodes, which will be discussed later, are also alike, then we could draw the conclusion that they possess the similar CPU load. I.e. the nodes, on which the same software is running and of which the data transport capacities are similar, have the similar CPU load.

3.5.1.3 Analysis for Data Transport Capacity

Then we turn to the analysis for data transport capacity. At first, the definitions of the maximal data transport capacity between nodes and the data transport capacity of a node are given:

3.5.1.3.1 Maximal Data Transport Capacity between Nodes

The maximal data transport capacity between two nodes is determined by the minimal bandwidth of a link, which is on the shortest cost path between them.

The formula of the maximal transport capacity from nodes m to node n is defined as following:

$$\omega_{mn} = \min(b_{mn_1}, b_{n_1n_2}, \dots, b_{n_xn})$$

The nodes n_1, n_2 … n_x are the intermediate nodes on the connecting path between node m and node n.

For example:

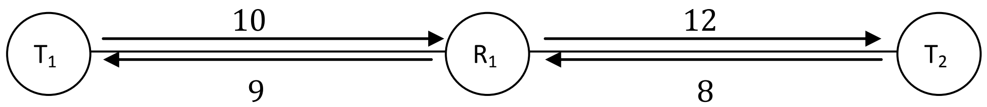


Fig 3.11 a simple network topology, where the bandwidth from T1 to R1 is 10, from R_1 to T_2 is 12, from T_2 to R_1 is 8, and from R_1 to T_1 is 9.

As showed in Figure 3.11, the maximal data transport capacity from T_1 to T_2 and that from T_2 to T_1 can be calculated with the formula above:

$\omega_{T_1T_2} = \min\left(b_{T_1R_1}, b_{R_1T_2}\right) = \min\ (10, 12) = 10;$

And $\omega_{T_2T_1} = \min\left(b_{T_2R_1}, b_{R_1T_1}\right) = \min\ (8, 9) = 8.$

3.5.1.3.2 Data Transport Capacity of a Node

The definition of the data transport capacity of a node is different from the one of maximal data transport capacity between two nodes.

Transport Capacity of a Terminal

The data transport capacity of a terminal is described as a vector.

For each terminal, there is a vector, which represents the data transport capacity of itself. Every item of the vector represents the data transport capacity from this terminal to a corresponding communication terminal. Therefore, the number of the items in a vector is the number of corresponding communication terminals. The data transport capacity of a terminal m can be defined in a vector as the following form:

$$\Omega_m = \begin{pmatrix} \mu_{mn_1} \\ \mu_{mn_2} \\ \dots \\ \mu_{mn_y} \end{pmatrix}$$

Where the nodes n_1, n_2 ... n_y are the corresponding communication terminals of the node m, and $\mu_{mn_1}, \mu_{mn_2}, \mu_{mn_3}$ are the transport capacities from m to n_1, n_2, n_3.

A transport capacity $\mu_{n_1 n_2}$ on a path from n_1 to n_2 is different in different transmission cases, which is smaller or equal to the maximal transport capacity $\omega_{n_1 n_2}$ on it, because in a part network, maybe more than one terminal will transmit data though the same path at the same time. For example:

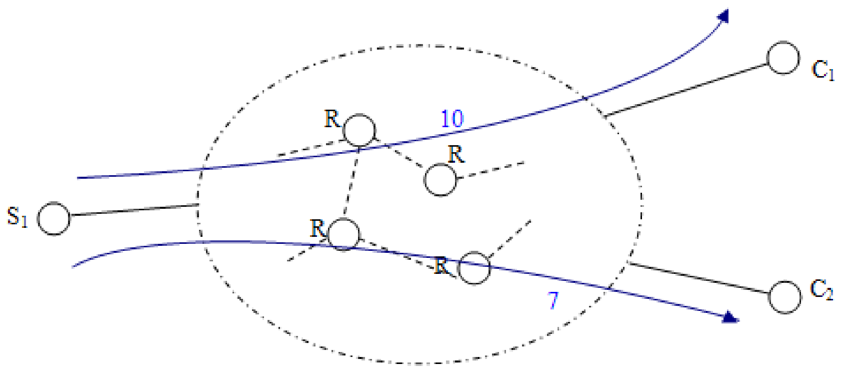


Fig 3.12 a server S_1 and two clients C_1, C_2; among which there are many routers. The transport capacity from S_1 to C_1 is 10; while the transport capacity from S_1 to C_2 is 7.

According to the data transport capacity in the Figure 3.12:

The vector, which means the transport capacity of node S_1 is $\Omega_{S_1} = \begin{pmatrix} \mu_{S_1C_1} \\ \mu_{S_1C_2} \end{pmatrix} = \begin{pmatrix} 10 \\ 7 \end{pmatrix}$, where the first item 10 represents the data transport capacity from S_1 to C_1 and the second item 7 represents the data transport capacity from S_1 to C_2.

Transport Capacity of a Router

Transport capacity of a router is described as two vectors. One is used for outgoing transport capacities on each link, and the other is used for incoming transport capacities on each link. The number of items of each vector is the number of connected links of the router.

As what we have already said, the calculation of the transport capacity of a router relays on the transport capacity between terminals, whose communication goes through it.

From the transport capacity of each terminal, we can know the communication from each terminal to all other corresponding communication terminals. Then we know, in this communication, how much data is transmitted through each link. This is the transport size on each link.

Then we can get the outgoing and incoming transport sizes on the connected links of a router, and the formula of transport capacity on a router r can be defined in two vectors:

The outgoing transport capacity of a router is written as:

$$\Omega_{r_{out}} = \begin{pmatrix} \lambda_{rn_1} \\ \lambda_{rn_2} \\ \cdots \\ \lambda_{rn_z} \end{pmatrix}$$

The incoming transport capacity of a router is written as:

$$\Omega_{r_{in}} = \begin{pmatrix} \lambda_{n_1r} \\ \lambda_{n_2r} \\ \cdots \\ \lambda_{n_zr} \end{pmatrix}$$

Where the nodes n_1, n_2 ... n_z are the neighbors of the router r.

For example, the transport capacity on the router R_1 in the Figure 4.3 is:

Outgoing transport capacity $\Omega_{r_{out}}$ equals $\begin{pmatrix}\lambda_{R_1T_1}\\ \lambda_{R_1T_2}\end{pmatrix} = \begin{pmatrix}8\\ 10\end{pmatrix}$.

And the incoming transport capacity $\Omega_{r_{in}}$ equals $\begin{pmatrix}\lambda_{T_1R_1}\\ \lambda_{T_{2R_1}}\end{pmatrix}$ =$\begin{pmatrix}10\\ 8\end{pmatrix}$.

3.5.1.3.3 Different Routing Schemes and Transmission Situations

As what is in the previous section written, data transport capacity of a router relays on the results of data transport capacities of terminals. Before calculating the data transport capacity of a terminal, the corresponding terminals of this terminal are known. However, in a network, the transmission situation is very complex. For each terminal, the routing schema can be anycast, unicast, multicast or broadcast. And in a part network, maybe there is just one terminal in the transport mode, maybe all the terminals simultaneous transport data.

Different Routing Schemes

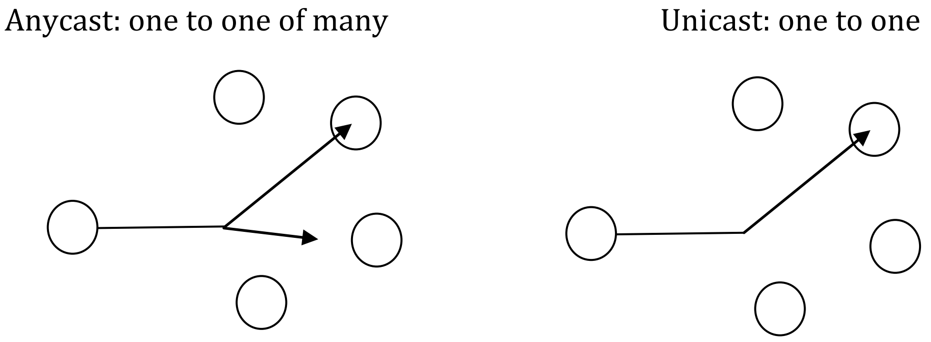


Fig 3.13 anycast and unicast 1->1

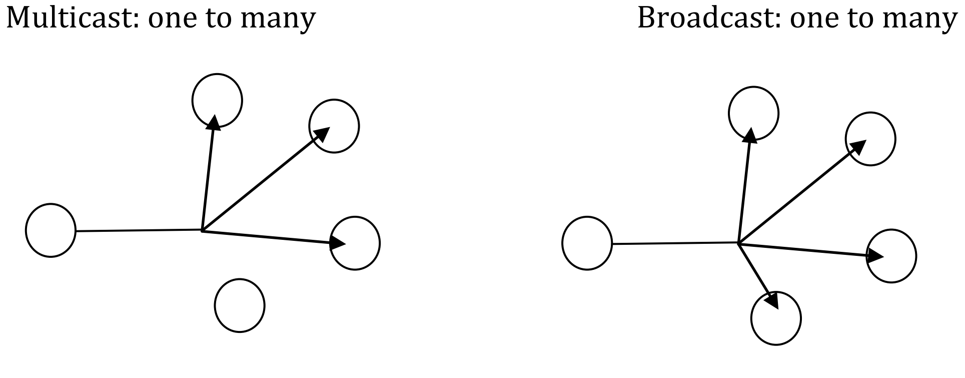


Fig 3.14 multicast and broadcast 1->n

Different Transmission Situations

In a network, maybe just one terminal transports data, maybe some terminals simultaneous transport data.

Therefore, the following transmission situations should be considered:

Firstly, for each terminal, when it transports data, there is no other terminals simultaneous transport data. Under this premise, three cases will be introduced.

1. The terminal runs an unicast. There is T-1 possibilities in all, where T is the number of corresponding terminals in the network.
2. The terminal runs a multicast. There is $\binom{2}{T-1}$+$\binom{3}{T-1}$+……+$\binom{T-1}{T-1}$ possibilities in all, where $\binom{2}{T-1}$ possibilities are for the case that the multicast is run from the terminal to arbitrary two terminals; $\binom{n}{T-1}$ possibilities are for the case that the multicast is run from the terminal to arbitrary n terminals … until the final item $\binom{T-1}{T-1}$=1 possibility is for the case broadcast.
3. The terminal simultaneous runs some unicasts. There is also $\binom{2}{T-1}$+$\binom{3}{T-1}$+……+$\binom{T-1}{T-1}$ kinds of possibility in all, where $\binom{2}{T-1}$ possibilities are for the case that simultaneous unicasts are run from the terminal to arbitrary two terminals; $\binom{n}{T-1}$ possibilities are for the case that simultaneous unicasts are run from the terminal to arbitrary n terminals … until the final item$\binom{T-1}{T-1}$=1 possibility is for the case that simultaneous unicasts are run from the terminal to all the other terminals. This situation is similar as the situation 2, but the available transport capacities of the terminal in the two cases are different. The difference is showed in Figure 3.15.

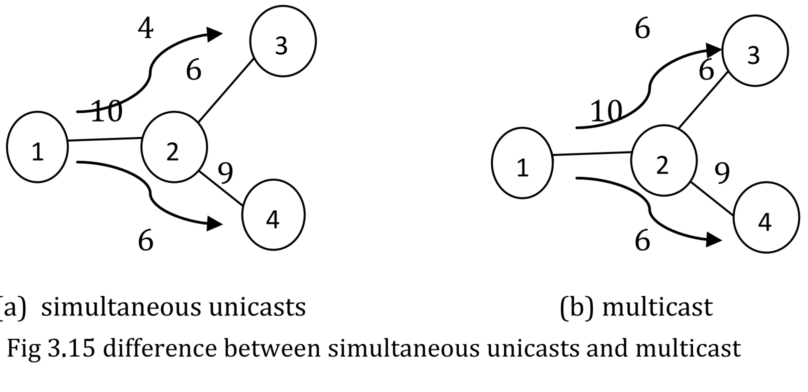


Fig 3.15 difference between simultaneous unicasts and multicast

Secondly, some of the terminals simultaneous transport data. The different transmission situations are located in two cases:

1. Arbitrary two terminals simultaneous transport data.
2. All the terminals simultaneous transport data.

That is to say, there are maybe arbitrary two or three or four or even all the terminals simultaneous transport data. And the transmission situation for each terminal is written in the first part.

3.5.2 Design of Node Classification Algorithm

There are two approaches for the design of Node Classification Algorithm. One is running under the assumption, that the routing information on all the nodes is known, while the other is running under the assumption, that the data will always transmitted on the shortest cost paths between each pair of nodes.

3.5.2.1 First Approach

In this approach, our assumption is that the routing information on all the nodes is known. That is to say, we know the next hop on each node for each communication.

3.5.2.1.1 Basic Definitions in the First Approach

For a communication between each pair of terminals, we consider that, all the transport paths between the two terminals are known.

For example, in the Figure 3.16, we can see all the transport paths from node T_1 to node T_2, where the arrow direction shows the routing direction from T_1 to T_2 and the digit on each link represents the bandwidth on the link in the arrow direction.

The routing paths form T_1 to T_2 are:
T_1->R_6->R_1->R_3->R_7->T_2
T_1->R_6->R_1->R_3->R_5->R_7->T_2
T_1->R_6->R_1->R_3->R_4->R_7->T_2
T_1->R_6->R_1->R_3->R_2->R_4->R_7->T_2
T_1->R_6->R_2->R_4->R_7->T_2

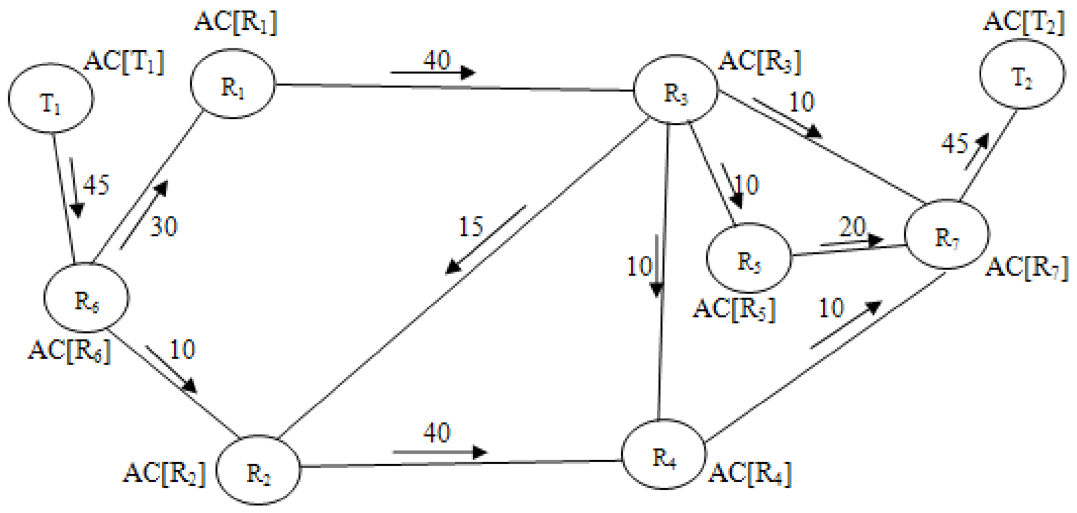


Fig.3.16 a small part network, where just the routing information from T_1 to T_2 is marked

The Array AC[r] on each router is the available transport capacity on the node, which represents the rest transport capacity on the outgoing links of a router after transporting data from the incoming paths to the outgoing links.

The sum of bandwidth on the outgoing links of the router r is $\sigma_r = \sum_i b_{rn_i}$, where node n_i is a node in the part network, and the sum of incoming transport size of the router r is $In_r = \sum_i I_{n_i r}$, where $I_{n_i r}$ represents the incoming transport size of the router r from the node n_i.

Thus, the definition of available transport capacity on a router r in formula is:

$$AC\,[r] = \begin{cases} \sigma_r - In_r, \text{if } \sigma_r > In_r \\ 0, \text{otherwise} \end{cases}$$

We know, in order to classify the nodes into groups, we need analyze the transport capacity on each terminal and each router. And the transport capacity on router is dependent on the transport capacities on the terminals.

So the question, how much data is transmitted from a terminal to the other corresponding terminals, is very important.

Before we calculate the transport capacity between terminals, the basic transmission rules on the intermediate routers are introduced.

The **basic transmission rules** are defined as following:

If In_r of a router r is known, then the node can be analyzed. The analysis is divided in three possibilities.

In the Listing 3.1, the basic transmission rules are written in pseudo code.

Listing 3.1 Basic Transmission Rules on a router r

```
// Out(r) is the sum of outgoing bandwidth of the router r
// In(r) is the sum of incoming data size of the router r
// Tr is the transport capacity between two terminals, which is analyzed here
// OV is the overflow size
If (Out(r) = In(r)) then
    Tr := Tr;
elseif (Out(r) < In(r)) then
    Tr := Tr – (In(r) – Out(r));
    OV := OV + (In(r) – Out(r));
elseif (Out(r) > In(r))then
    Tr := Tr;
    AC[r] := (Out(r) – In(r))
endif
```

3.5.2.1.2 Design of the First Approach

Transport Capacity of Terminal

Each terminal T_i has a vector $\Omega_{T_i} = \begin{pmatrix} \mu_{T_iT_1} \\ \mu_{T_iT_2} \\ \cdots \\ \mu_{T_iT_j} \end{pmatrix}$, where each item $\mu_{T_iT_j}$ means the data transport capacity from T_i to T_j.

Now in order to calculate every item $\mu_{T_iT_j}$, all the routing paths between them are considered. In the worst case, all the routers are in the routing paths, and then all the routers are analyzed.

According to the assumption, all the routing information for transmission of each pair of terminals is known. I.e. we know the incoming paths and outgoing paths of each router for each transmission of each pair of terminals.

Then $\mu_{T_iT_j}$ can be calculated in the following steps:

1. The initial transport capacity from T_i to T_j is set as outgoing bandwidth of T_i.
2. According to the basic rule defined before, all the neighbors of T_i will be analyzed.
3. Repeat the step 2 until T_j is reached. Now we have a value of transport

capacity between T_i and T_j as well as AC[r] on each router.

4. If the overflow size OS from T_i to $T_j > 0$, we should check the array AC[r]. If there are paths from T_i to T_j, on each of which the smallest AC[r] of every router is bigger than 0. Then the smallest AC[r] of each path will be added to the transport capacity from T_i to T_j. But the transport capacity from T_i to T_j cannot be bigger than outgoing bandwidth of T_i.

In order to understand better, a pseudo code is offered in Listing 3.2.

In the algorithm, we want to calculate a transport capacity from s to d. One input is $(G=(E,V,\gamma),s,d)$, where G is the graph of the part network, E is the set of nodes in the part network, V is the set of links in the part network and $\gamma(e_i e_j)$ is the bandwidth on the link from e_i to e_j, s is the source node and d is the destination node. The other input is PathSet, which is the set of routing paths from node s to node t. Furthermore, we known the Out(n), which represents the sum of bandwidth on the outgoing links of each node n.

Tr is the transport capacity from the source s to the destination d and OV is the overflow size in the part network.

Listing 3.2 Algorithm for Transport Capacity Calculation from s to d

```
var x,y nodes; OV,Tr float;
var TrIn,AC float;
α: array[1…|V|][1…|V|] of float;          (*real transport data size on a link*)
tn: array[1…|V|]of float;     (*in fact, how much data is transmitted through a node*)
In: array[1…|V|]of float;                  (*sum of incoming data size of a node*)
p: array[1…|V|]of nodes;                          (*previous node of a node*)
path: array[1…|E|]of paths;
B: set of nodes                                   (*nodes, which are analyzed*)
R: set of nodes       (*nodes, which are the neighbors of nodes in B and not in B*)
U: set of nodes                                               (*the rest nodes*)
B := {s}; R :=ϕ; p(s)=nil;                          (*initialization of B,R,U*)
Tr:=Out(s);                                         (*initialization of Tr*)
begin
    forall y∈V\{s}: {s,y}∈E do
        p(y) :=s;α(s,y) :=γ(s, y);
        In(y):=In(y) + α(s,y);
        insert (R,y, α(s,y));
    endfor
    U:=V\(R⊔{s});
    while d∉B do
```

```
        x:=nil;                                 (*look for a node x that data size on
        forall y∈R do                             all the incoming links is known*)
            cond:=true;
            for all (z,y)∈E do
                if $\alpha$(z,y)=0 then
                    cond:=false;
                endif
            endfor
            if cond:=true then
                x:=y;
                if In(x)=Out(x) then                  (*tn(n) is the sum of outgoing
                    tn(x):=In(x);                            data size of a node n*)
                elseif In(x)>Out(x) then
                    Tr :=Tr-(In(x)-Out(x));
                    OV:=OV+In(x)-Out(x);        (*recalculate the Overflow Size*)
                    tn(x):=Out(y);
                elseif In(x)<Out(x) then
                    AC(x):=Out(x)-In(x);
                    tn(x):=In(x)
                endif
                forall (x,z)∈E do                          (*calculate the outgoing
                    $\alpha(x,z)= \frac{\gamma(x,z)}{Out\ (x)} \times tn$;                data size on each link*)
                    In(z):=In(z)+ $\alpha$(x,z);       (*corresponding In(n) is modified*)
                endfor
                B:=B ⊔{x};
                R:=R \{x};                                       (*set R is updated*)
                forall y∈U: {x,y}∈E do
                    p(y) :=x;$\alpha(x,y) := \frac{\gamma(x,y)}{Out\ (x)} \times tn(x)$
                    In(y):=In(y) + $\alpha$(x,y);
                    insert (R,y, $\alpha$(x,y));
                endfor
            endif
        endfor
    endwhile
    forall path∈PathSet do                                             (*step 4*)
        If OV>0 then
            TrIn:=0;
            AC:=Float_MAX;
            forall y∈path do
                if AC<AC(y) then
                    AC:=AC(y);
                endif
            enddo
            if AC>0 then
                AC:=min(AC,OV)
                Tr:=Tr+AC;
                forall y∈path do
                    AC(y)=AC(y)-AC;
                enddo
                If Tr>Out(s) then
                    Tr:=Out(s);
                Endif
                OV:=OV-AC;
            endif
        endif
    endfor
end
```

For example:

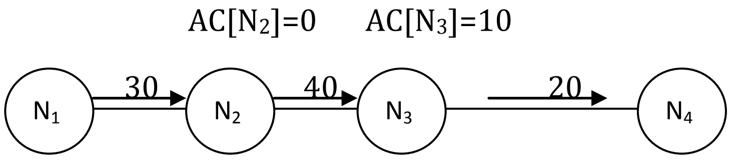


Fig. 3.17 the digit on the links means the bandwidth in the arrow direction

We want to calculate the transport capacity from N_1 to N_4, $\mu_{N_1N_4}$ in the Figure 3.17:

At the beginning, $\mu_{N_1N_4}$ is set to 30 because the sum of outgoing bandwidth of node N_1, $Out(N_1)$ equals 30.

Then we analyze the neighbor node N_2. For node N_2, the sum of data size on the incoming links of the node N_2, $In(N_2)$ is as big as $Out(N_1)$, which equals 30. The sum of the outgoing bandwidth of N_2, $Out(N_2)$ is 40. Therefore, according to the basic rule three, $\mu_{N_1N_4}$ is not changed. It is still 30. And the available capacity on the node N_2, $AC[N_2] = Out(N_2)-In(N_2) = 10$; the sum of data size on the outgoing data links on N_2, $tn(N_2)= In(N_2)=30$. Therefore, the outgoing data size on the link from N_2 to N_3, $\alpha(N_2, N_3) = \frac{\gamma(N_2,N_3)}{Out(N_2)} \times tn(N_2) = \frac{40}{40} \times 30 = 30$. And $In(N_3) = \sum_i \alpha(N_i, N_3) = \alpha(N_2, N_3) = 30$.

After that, the node N_3 is analyzed. As an input, $In(N_3)$ equals 30 while $Out(N_3)$ is 20. According to the basic rule two, $\mu_{N_1N_4}$ will be decreased. $\mu_{N_1N_4} = \mu_{N_1N_4} - (In(N_3) - Out(N_3)) = 20$. The overflow size, $OV = OV + (In(N_3) - Out(N_3))= 10$. And

$tn(N_3)=Out(N_3)=20$; $\alpha(N_3, N_4) = \frac{\gamma(N_3,N_4)}{Out(N_3)} \times tn(N_3) = \frac{30}{30} \times 20 = 20$. And $In(N_4) = \sum_i \alpha(N_i, N_4) = \alpha(N_3, N_4) = 20$.

Finally, we come to the destination node N_4. We can find out, that OV > 0. So like what is written in step four, we will check AC[r] for each router in each routing path from N_1 to N_4 now. There is just one routing path here, N_1->N_2->N_3->N_4. On the path, AC[N_2] = 0, and the AC[N_3] = 10, so the available transport capacity on the path is min(AC[N_2],AC[N_3]) = 0.
So the final result is:
$\mu_{N_1N_4} = \mu_{N_1N_4}$ + min(AC[N_2],AC[N_3]) = 20 + 0 = 20.

After that, for each terminal, we have a vector, in which the transport capacities from this terminal to other corresponding terminals are written. Through the compare of vectors the terminals can be divided in different groups.

Transport Capacity of a Router

In this approach, in order to get a better result of measuring, the calculation method of transport capacity of a router is different from the definition in the previous section.

The transport capacity of a router is defined as the difference of transport capacities between the case that the router is in the part network and the case that the router is not in the part network.

For each router r, there is a vector$\begin{pmatrix} |T_{a1} - T_{b1}| \\ |T_{a2} - T_{b2}| \\ \cdots \\ |T_{an} - T_{bn}| \end{pmatrix}$, where $T_{b1}, T_{b2}, \ldots T_{bn}$ are the transport capacities between corresponding terminals in the part network, when r is in the network and $T_{a1}, T_{a2}, \ldots T_{an}$ are the transport capacities, when r is not in the network.

For example the router N_2, N_3 in Figure 3.17

When N_2 is in the part network, the transport capacity from N_1 to N_4 is 20. When N_2 is not in the part network, the transport capacity from N_1 to N_4 is 0. The transport capacity of router N_2 is $(|T_{a1} - T_{b1}|) = (|0 - 20|) = (\ 20\ )$. And for N_3 is also$(\ 20\ )$.

3.5.2.1.3 Time Complexity

In this design, we will consider all the paths between two communication terminals. So in the worst case, all the routers are considered.

The time complexity for the calculating the transport capacity of a terminal is $O(T^2 \times R)$, where T is the number of terminals in a part network and R is the number of routers. In the worse case, there is communication between each pair of terminals; each communication goes through all the routers. The number of possibilities for pairs of communication terminals is T^2. The number of router is R. So the time complexity is $O(T^2 \times R)$.

For the calculation of the transport capacity of a router, the time complexity is even $O(T^2 \times R^2)$, because for each router, in the worst case, all the communication between any pair of terminals in the part network will be calculated once. That is $O((T^2 \times R) \times R) = O(T^2 \times R^2)$

However, the assumption, that routing information on all the nodes is known, is very hard to touch. Usually, we do not know so much routing information. So we will look for another design.

3.5.2.2 Second Approach

In this approach, we do not know all the routing information on each node. Then how can we get the routing information. In order to solve this problem, a new assumption is given.

3.5.2.2.1 Real Time Communication Case

In this assumption, a communication parameter p is used for a definition of a real time communication case, which is located between two extreme communication cases in the network: extreme high communication and extreme low communication in the network.

The real time means that, the communication parameter p can be changed along with different time point.

First Case: Extreme Low Communication in the Network

In this case, we assume that, just one terminal transmits data. The shortest path between two terminals is always considered as the communication path between them. The shortest path is calculated from the transmission cost model, represents the path with shortest cost between them. It can also be named shortest cost path. In such a case, the data from one terminal to another is always going along the shortest path between these two terminals. It does conform to the routing rules.

Second Case: Extreme High Communication in the Network

In this case, we assume that, all the terminals are simultaneously transporting data. The shortest path between two terminals is still always considered as the communication path between the two terminals. The reason is that, the network is overload everywhere in such a case. So the data, which is transported from one node to another, cannot flow to other nodes, which are not in the shortest path.

Thus, under this assumption, the communication between two terminals is always limited in the shortest path of the two terminals.

We keep the communication parameter p between 0 and 1, where the value 0 represents the case of extreme low communication in the network and the value 1 represents the case of extreme high communication in the network.

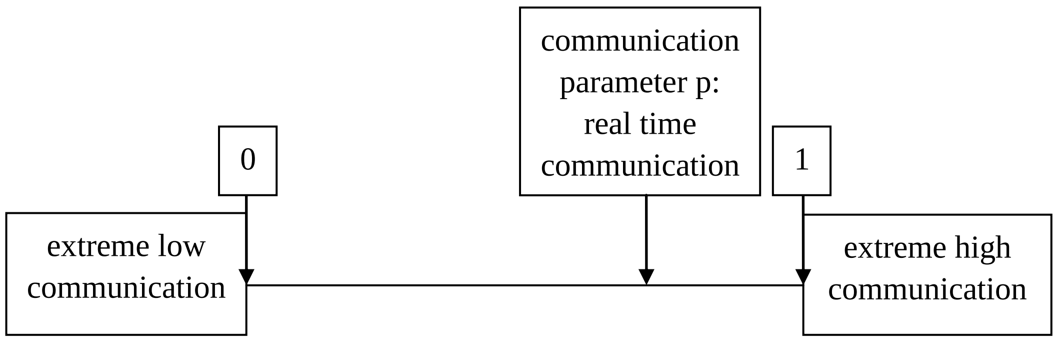

communication parameter p: real time communication
0
1
extreme low communication
extreme high communication

Fig. 3.18 real time communictaion

$$\begin{aligned}&\text{the transport capacity on a terminal}\\&= \text{transport capacity in the case of extreme low communication in the network}\\&\times \text{p}\\&+ \text{transport capacity in the case of extreme high communication in the network}\\&\times (1 - \text{p})\end{aligned}$$

With this formula the transport capacity on each terminal is calculated.

The calculation of transport capacity on each router is based on the results of the transport capacity on each terminal. It has been already described in previous section, Analysis for Data Transport Capacity.

3.5.2.2.2 Design of Second Approach

As what is in section Basic architecture of the load model of real world network defined, the nodes have been already divided in two classes, and a node of class terminal will never be similar as a node in class router due to the different functions of them. Therefore, for a node classification, only the nodes in the same class are compared.

In addition, in order to reduce the runtime of the operation in the algorithm, the whole network has already been divided into some part networks, which has already been discussed in Section network clustering.

So all the nodes in the same class in all the part networks will be compared together.

Total Process of the Node Classification

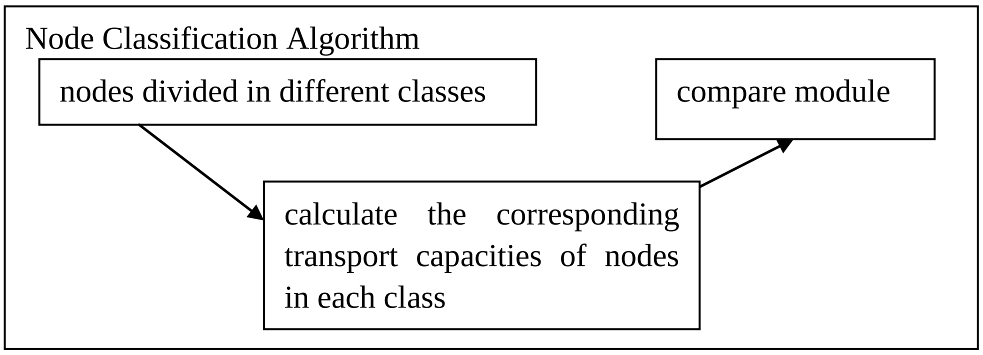

Node Classification Algorithm
nodes divided in different classes
compare module
calculate the corresponding transport capacities of nodes in each class

Fig 3.19 total process of the node classification

As the process showed in the Figure 3.19, in the Node Classification Algorithm, at first, nodes are separated into different classes, each of which has different function. After that, the corresponding transport capacities of nodes in each class are calculated. Finally, we divide the nodes into different groups in each class through comparing the transport capacities of nodes. The compare between nodes in one class is independent of the compare between nodes in another class. The compare module is running independently in each node class.

Nodes in Different Classes

In the section preparation work, it is defined that, the nodes in a network have been divided in two classes.

Furthermore, the class terminal will be divided in three part classes: class server, class client and class p2p point, which can acts as a server as well as a client. Concerning the software on each node, we can know which node belongs to which class.

Due to different functions of nodes in class servers, class clients and class p2p points, there are four classes now in all.

Data Transport Capacities of Nodes in Each Class

Above all, some **basic traffic rules** are defined:

In the network, server just transports data to client while client also just transports data to server. There is no communication between servers or between clients. P2p nodes can communicate with all the other p2p nodes, because a p2p node can act as a server as well as a client.

Normally, a network is either a p2p network or a client server network. We assume that, either all the terminals in a network are p2p nodes, or all the terminals are client and server nodes.

Transport Capacity Calculation in Different Classes

After that, the different calculation methods for nodes in different classes will be defined in details. The definitions are described respectively in two different network types: client server network and p2p network. Furthermore, in each type of network, the data transport capacities of all the nodes in all the classes will be calculated in two cases: extreme low communication in the network and extreme high communication in the network.

In a Client Server Network

Case1: Extreme Low Communication in the Network

In this case, we assume that, there is just one terminal transports data.

At first we analyze the terminals in **class client**. A client transports data just to one server at a time point. The possibility of transmission from a client to each server is the same. So the average data transport capacity from a client C to a server is defined as $A_C = \frac{\text{Sum}_C}{n_s}$, where n_s is the number of servers in the part network and $\text{Sum}_C = \omega_{CS_1} + \omega_{CS_2} + \cdots + \omega_{CS_{n_s}}$ represents the sum of maximal transport capacities from the client C to a server.

Thus, the transport capacity of a client $\Omega_C = \begin{pmatrix} \frac{\omega_{CS1}}{\text{Sum}_C} \times A_C \\ \frac{\omega_{CS2}}{\text{Sum}_C} \times A_C \\ \cdots \\ \frac{\omega_{CS_{n_s}}}{\text{Sum}_C} \times A_C \end{pmatrix} = \begin{pmatrix} \frac{\omega_{CS1}}{n_s} \\ \frac{\omega_{CS2}}{n_s} \\ \cdots \\ \frac{\omega_{CS_{n_s}}}{n_s} \end{pmatrix}$.

For example, in Figure 3.20, the bandwidth is given. So the data transport capacities of the two clients C_1 and C_2 can be calculated.

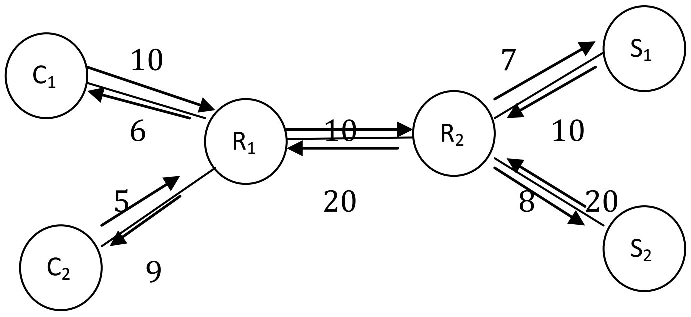


Fig 3.20 a simple network topology with two clients, two servers and two routers, where the digit on the arrows means the bandwidth of the link in the arrow direction.

For client C_1:

$\omega_{C_1S_1} = 7$, $\omega_{C_1S_2} = 8$, $n_s = 2$;

Therefore, the transport capacity of the client C_1

$$\Omega_{C_1} = \begin{pmatrix} \frac{\omega_{C_1S_1}}{n_s} \\ \frac{\omega_{C_1S_2}}{n_s} \end{pmatrix} = \begin{pmatrix} \frac{7}{2} \\ \frac{8}{2} \end{pmatrix} = \begin{pmatrix} 3.5 \\ 4 \end{pmatrix}$$

For client C_2:

$\omega_{C_2S_1} = 5$, $\omega_{C_2S_2} = 5$, $n_s = 2$;

Therefore, the transport capacity of the client C1

$$\Omega_{C_2} = \begin{pmatrix} \frac{\omega_{C_2S_1}}{n_s} \\ \frac{\omega_{C_2S_2}}{n_s} \end{pmatrix} = \begin{pmatrix} \frac{5}{2} \\ \frac{5}{2} \end{pmatrix} = \begin{pmatrix} 2.5 \\ 2.5 \end{pmatrix}$$

Then we will analyze the terminals in **class server**. In this case, data is transported from a server S to all other clients at the same time. Here we will introduce a new definition, bottleneck factor of a link.

The bottleneck factor of a link from n_1 to n_2 can be defined in the following form:

$$o_{n_1n_2} = \begin{cases} \frac{\theta_{n_1n_2}}{b_{n_1n_2}}, \text{if } \theta_{n_1n_2} > b_{n_1n_2} \\ 1\text{ , otherweise} \end{cases}$$

where $\theta_{n_1n_2} = \sum_{i,j} \omega_{T_iT_j}$ through the link from n_1 to n_2, $(\forall i, j, T_i, T_j \in T)$, is the sum of maximal transport capacities in the part network through the link from n_1 to n_2, each of which is determined by the communication between a pair of corresponding communication terminals in the part network and $b_{n_1n_2}$ is the bandwidth of the link.

In this case, $\theta_{n_1n_2}$ is the sum of maximal transport capacities through the link from n_1 to n_2, each of which is determined by the communication from a server S to a corresponding client in the part network.

The bottleneck factor of a link is the bottleneck on the link. When the bottleneck factor on a link equals 1, it means that, there is no bottleneck on the link.

Furthermore, we can write

The bottleneck factor of a path between two terminals T_1 and T_2

$$O_{T_1T_2} = \max(o_{T_1n_1}, o_{n_1n_2} \ldots\ldots o_{n_xT_2})$$

where nodes n_1, n_2…… n_x are the intermediate nodes on the shortest path from T_1 to T_2.

Then in this case, the transport capacity of a server S is

$$\Omega_S = \begin{pmatrix} \dfrac{\omega_{SC_1}}{O_{SC_1}} \\ \dfrac{\omega_{SC_2}}{O_{SC_2}} \\ \cdots \\ \dfrac{\omega_{SC_{n_c}}}{O_{SC_{n_c}}} \end{pmatrix}$$

Where n_c is the number of clients in the part network

Now we analyze the topology in Figure 3.20. In this example, in this case, the bottleneck factor on each link for server S1 is calculated as following:

Sum of the max transport capacity on the link from n1 to n2 : $(\theta_{n_1n_2})$	bandwidth	bottleneck factor on the link
$\theta_{S_1R_2} = 6 + 9 = 15$	$b_{S_1R_2} = 10$	$o_{S_1R_2} = \frac{\theta_{S_1R_2}}{b_{S_1R_2}} = \frac{15}{10} = 1.5$
$\theta_{R_2R_1} = 6 + 9 = 15$	$b_{R_2R_1} = 20$	$o_{R_2R_1} = 1$
$\theta_{R_1C_1} = 6$	$b_{R_1C_1} = 6$	$o_{R_2S_1} = 1$
$\theta_{R_1C_2} = 9$	$b_{R_1C_2} = 9$	$o_{R_2S_2} = 1$

Then each bottleneck factor on the shortest path from Server S1 to each other client is:

	Shortest path	bottleneck factor on the path
From S1 to C1	S1->R2->R1->C1	$O_{S_1C_1} = \max(o_{S_1R_2}, o_{R_2R_1}, o_{R_1C_1}) = 1.5$
From S1 to C2	S1->R2->R1->C2	$O_{S_1C_2} = \max(o_{S_1R_2}, o_{R_2R_1}, o_{R_1C_2}) = 1.5$

Therefore, the transport capacity of S1 is:

$$\Omega_{S_1} = \begin{pmatrix} \dfrac{\omega_{S_1C_1}}{O_{S_1C_1}} \\ \dfrac{\omega_{S_1C_2}}{O_{S_1C_2}} \end{pmatrix} = \begin{pmatrix} \dfrac{6}{1.5} \\ \dfrac{9}{1.5} \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix}$$

The bottleneck factor on each link for server S2 is:

Sum of the max transport capacity on the link from n1 to n2 : $(\theta_{n_1n_2})$	bandwidth	bottleneck factor on the link
$\theta_{S_2R_2} = 6 + 9 = 15$	$b_{S_2R_2} = 20$	$o_{S_2R_2} = 1$
$\theta_{R_2R_1} = 6 + 9 = 15$	$b_{R_2R_1} = 20$	$o_{R_2R_1} = 1$
$\theta_{R_1C_1} = 6$	$b_{R_1C_1} = 6$	$o_{R_2S_1} = 1$
$\theta_{R_1C_2} = 9$	$b_{R_1C_2} = 9$	$o_{R_2S_2} = 1$

Then each bottleneck factor on the shortest path from Server S_2 to each other client is:

	Shortest path	bottleneck factor on the path
From S_2 to C_1	S_2->R_2->R_1->C_1	$O_{S_2C_1} = \max(o_{S_2R_2}, o_{R_2R_1}, o_{R_1C_1}) = 1$
From S_2 to C_2	S_2->R_2->R_1->C_2	$O_{S_2C_2} = \max(o_{S_2R_2}, o_{R_2R_1}, o_{R_1C_2}) = 1$

Therefore, the transport capacity of S_2

$$\Omega_{S_2} = \begin{pmatrix} \frac{\omega_{S_2C_1}}{O_{S_2C_1}} \\ \frac{\omega_{S_2C_2}}{O_{S_2C_2}} \end{pmatrix} = \begin{pmatrix} \frac{6}{1} \\ \frac{9}{1} \end{pmatrix} = \begin{pmatrix} 6 \\ 9 \end{pmatrix}$$

Finally we will analyze the nodes in **class router**.

In this case, there is always only one terminal in transport mode.

So the transport capacity on each link = $\frac{\vartheta_{n1n2}}{n_c+n_s}$

Where $\vartheta_{n_1n_2}$ is sum of all the transport capacities of terminals in the case of extreme low communication through this link from n_1 to n_2, which can be defined as:

$$\vartheta_{n_1n_2} = \left(\sum_{i,j} \mu_{S_iC_j} + \sum_{i,j} \mu_{C_jS_i}\right) \text{through the link from } n_1 \text{ to } n_2, (\forall i, S_i \in S; \forall j, C_j \in C,$$

(All the transport capacities μ in the definition here are calculated in the case extreme low communication in the network.)

And the item $n_c + n_s$ represents the sum of number of servers and that of clients in a part network.

Because $\frac{1}{n_c+n_s}$ is the possibility for each terminal, that it is in the transport mode.

Thus, the outgoing transport capacity of a router r is

$$\Omega_{r_{out}} = \begin{pmatrix} \frac{\vartheta_{rn1}}{n_c+n_s} \\ \frac{\vartheta_{rn2}}{n_c+n_s} \\ \cdots \\ \frac{\vartheta_{rnn}}{n_c+n_s} \end{pmatrix},$$

And the incoming transport capacity of a router r is

$$\Omega_{r_{in}} = \begin{pmatrix} \frac{\vartheta_{n_1 r}}{n_c+n_s} \\ \frac{\vartheta_{n_2 r}}{n_c+n_s} \\ \cdots \\ \frac{\vartheta_{n_n r}}{n_c+n_s} \end{pmatrix},$$

where the nodes n_1, n_2...n_n are the neighbors of the router r.

As the topology in Figure 3.20 shown, the transport capacity on each node is:

$$\Omega_{C_1} = \binom{3.5}{4};\ \Omega_{C_2} = \binom{2.5}{2.5};\ \Omega_{S_1} = \binom{4}{6}; \text{on node } \Omega_{S_2} = \binom{6}{9}.$$

Obviously, the number of communication between communication terminals in this part network is $n_c + n_s = 2 + 2 = 4$.

For router R_1 and R_2:

At first, for each link from n_1 to n_2, the sum of all the transport capacities of terminals through this link from n_1 to n_2 $\vartheta_{C_1R_2}$ is calculated:

$\vartheta_{C_1R_1} = 3.5 + 4 = 7.5$	$\vartheta_{R_1C_1} = 4 + 6 = 10$
$\vartheta_{C_2R_1} = 2.5 + 2.5 = 5$	$\vartheta_{R_1C_2} = 6 + 9 = 15$
$\vartheta_{R_1R_2} = \vartheta_{C_1R_2} + \vartheta_{C_2R_1} = 7.5 + 5 = 12.5$	$\vartheta_{R_2R_1} = \vartheta_{S_1R_2} + \vartheta_{S_2R_2} = 10 + 15 = 25$
$\vartheta_{R_2S_1} = 3.5 + 2.5 = 6$	$\vartheta_{S_1R_2} = 4 + 6 = 10$
$\vartheta_{R_2S_2} = 4 + 2.5 = 6.5$	$\vartheta_{S_2R_2} = 6 + 9 = 15$

Therefore, for router R_1, the outgoing transport capacity on the router is:

$$\Omega_{R_{1_{out}}} = \begin{pmatrix} \frac{\vartheta_{R_1C_1}}{n_c+n_s} \\ \frac{\vartheta_{R_1C_2}}{n_c+n_s} \\ \frac{\vartheta_{R_1R_2}}{n_c+n_s} \end{pmatrix} = \begin{pmatrix} \frac{10}{4} \\ \frac{15}{4} \\ \frac{12.5}{4} \end{pmatrix} = \begin{pmatrix} 2.5 \\ 3.75 \\ 3.125 \end{pmatrix},$$

And the incoming transport capacity is:

$$\Omega_{R_{1_{in}}} = \begin{pmatrix} \frac{\vartheta_{C_1R_1}}{n_c+n_s} \\ \frac{\vartheta_{C_2R_1}}{n_c+n_s} \\ \frac{\vartheta_{R_2R_1}}{n_c+n_s} \end{pmatrix} = \begin{pmatrix} \frac{7.5}{4} \\ \frac{5}{4} \\ \frac{25}{4} \end{pmatrix} = \begin{pmatrix} 1.875 \\ 1.25 \\ 6.25 \end{pmatrix}.$$

For router R_2, the outgoing transport capacity is:

$$\Omega_{R_{2\,out}} = \begin{pmatrix} \frac{\vartheta_{R_2S_1}}{n_c+n_s} \\ \frac{\vartheta_{R_2S_2}}{n_c+n_s} \\ \frac{\vartheta_{R_2R_1}}{n_c+n_s} \end{pmatrix} = \begin{pmatrix} \frac{10}{4} \\ \frac{15}{4} \\ \frac{25}{4} \end{pmatrix} = \begin{pmatrix} 1.5 \\ 1.625 \\ 6.25 \end{pmatrix},$$

And the incoming transport capacity is:

$$\Omega_{R_{2\,in}} = \begin{pmatrix} \frac{\vartheta_{S_1R_2}}{n_c+n_s} \\ \frac{\vartheta_{S_2R_2}}{n_c+n_s} \\ \frac{\vartheta_{R_1R_2}}{n_c+n_s} \end{pmatrix} = \begin{pmatrix} \frac{6}{4} \\ \frac{6.5}{4} \\ \frac{12.5}{4} \end{pmatrix} = \begin{pmatrix} 2.5 \\ 3.75 \\ 3.125 \end{pmatrix}.$$

Case2: Extreme High Communication in the Network

In this case, we assume that, all the **terminals** will transport data simultaneously. Each client transmits data to a server, when each server transmits data to all the clients. Then maybe there is overflow in the network. We need the overflow factor on each link here.

The overflow factor of a link from n_1 to n_2 can be defined in the following form:

$$\gamma_{n_1n_2} = \begin{cases} \frac{\vartheta_{n_1n_2}}{b_{n_1n_2}}, \text{if } \theta_{n_1n_2} > b_{n_1n_2} \\ 1 \text{, otherweise} \end{cases}$$

In this case, $\vartheta_{n_1n_2}$ is the sum of transport capacities through the link from n_1 to n_2, each of which is determined by all the communication between corresponding communication nodes in the part network.

The overflow factor of a link is the overflow situation on the link. When the overflow factor on a link equals 1, it means that, there is no overflow on the link.

Furthermore, we can write

The overflow factor of a path between two terminals T_1 and T_2

$$\Gamma_{T_1T_2} = \max(\gamma_{T_1n_1}, \gamma_{n_1n_2} \dots\dots \gamma_{n_xT_2})$$

where nodes n_1, n_2…… n_x are the intermediate nodes on the shortest path from

T_1 to T_2. With it the transport capacity of a client C can be written in:

$$\Omega_C = \begin{pmatrix} \frac{\vartheta_{CS_1}}{\Gamma_{CS_1}} \\ \frac{\vartheta_{CS_2}}{\Gamma_{CS_2}} \\ \cdots \\ \frac{\vartheta_{CS_n}}{\Gamma_{CS_n}} \end{pmatrix}.$$

And the transport capacity of a server S can be written in:

$$\Omega_S = \begin{pmatrix} \frac{\vartheta_{SC_1}}{\Gamma_{SC_1}} \\ \frac{\vartheta_{SC_2}}{\Gamma_{SC_2}} \\ \cdots \\ \frac{\vartheta_{SC_n}}{\Gamma_{SC_n}} \end{pmatrix}.$$

In this case, concerning the topology in Figure 3.20, the overflow factor on each link is calculated as following:

Sum of the transport capacity on the link from n_1 to n_2 : $(\vartheta_{n_1n_2})$	bandwidth	overflow factor on the link
$\vartheta_{C_1R_1} = 3.5 + 4 = 7.5$	$b_{C_1R_1} = 10$	$\gamma_{C_1R_1} = 1$
$\vartheta_{C_2R_1} = 2.5 + 2.5 = 5$	$b_{C_2R_1} = 5$	$\gamma_{C_2R_1} = 1$
$\vartheta_{R_1R_2} = \vartheta_{C_1R_1} + \vartheta_{C_2R_1} = 12.5$	$b_{R_1R_2} = 10$	$\gamma_{R_1R_2} = \frac{\vartheta_{R_1R_2}}{b_{R_1R_2}} = 1.25$
$\vartheta_{R_2S_1} = 3.5 + 2.5 = 6$	$b_{R_2S_1} = 7$	$\gamma_{R_2S_1} = 1$
$\vartheta_{R_2S_2} = 4 + 2.5 = 6.5$	$b_{R_2S_2} = 8$	$\gamma_{R_2S_2} = 1$
$\vartheta_{S_1R_2} = 4 + 6 = 10$	$b_{S_1R_2} = 10$	$\gamma_{S_1R_2} = 1$
$\vartheta_{S_2R_2} = 6 + 9 = 15$	$b_{S_2R_2} = 20$	$\gamma_{S_2R_2} = 1$
$\vartheta_{R_2R_1} = \vartheta_{S_1R_2} + \vartheta_{S_2R_2} = 25$	$b_{R_2R_1} = 20$	$\gamma_{R_2R_1} = \frac{\vartheta_{R_2R_1}}{b_{R_2R_1}} = 1.25$
$\vartheta_{R_1C_1} = 4 + 6 = 10$	$b_{R_1C_1} = 6$	$\gamma_{R_1C_1} = \frac{\vartheta_{R_1C_1}}{b_{R_1C_1}} = 1.67$
$\vartheta_{R_1C_2} = 6 + 9 = 15$	$b_{R_1C_2} = 9$	$\gamma_{R_1C_2} = \frac{\vartheta_{R_1C_2}}{b_{R_1C_2}} = 1.67$

Then each overflow factor on the shortest path from each server and client to corresponding communication nodes is:

	Shortest path	overflow factor on the path
From C_1 to S_1	C_1->R_1->R_2->S_1	$\Gamma_{C_1S_1} = \max(\gamma_{C_1R_1}, \gamma_{R_1R_2}, \gamma_{R_2S_1}) = 1.25$
From C_1 to S_2	C_1->R_1->R_2->S_2	$\Gamma_{C_1S_2} = \max(\gamma_{C_1R_1}, \gamma_{R_1R_2}, \gamma_{R_2S_2}) = 1.25$
From C_2 to S_1	C_2->R_1->R_2->S_1	$\Gamma_{C_2S_1} = \max(\gamma_{C_2R_1}, \gamma_{R_1R_2}, \gamma_{R_2S_1}) = 1.25$
From C_2 to S_2	C_2->R_1->R_2->S_2	$\Gamma_{C_2S_2} = \max(\gamma_{C_2R_1}, \gamma_{R_1R_2}, \gamma_{R_2S_2}) = 1.25$
From S_1 to C_1	S_1->R_2->R_1->C_1	$\Gamma_{S_1C_1} = \max(\gamma_{S_1R_2}, \gamma_{R_2R_1}, \gamma_{R_1C_1}) = 1.67$

From S_1 to C_2 S_1->R_2->R_1->C_2 $\Gamma_{S_1C_2} = \max(\gamma_{S_1R_2}, \gamma_{R_2R_1}, \gamma_{R_1C_2}) = 1.67$
From S_2 to C_1 S_2->R_2->R_1->C_1 $\Gamma_{S_2C_1} = \max(\gamma_{S_2R_2}, \gamma_{R_2R_1}, \gamma_{R_1C_1}) = 1.67$
From S_2 to C_2 S_2->R_2->R_1->C_2 $\Gamma_{S_2C_2} = \max(\gamma_{S_2R_2}, \gamma_{R_2R_1}, \gamma_{R_1C_2}) = 1.67$

Thus, the transport capacities of C_1, C_2, S_1 and S2 are:

$$\Omega_{C_1} = \begin{pmatrix} \frac{\vartheta_{C_1S_1}}{\Gamma_{C_1S_1}} \\ \frac{\vartheta_{C_1S_2}}{\Gamma_{C_1S_2}} \end{pmatrix} = \begin{pmatrix} \frac{3.5}{1.25} \\ \frac{4}{1.25} \end{pmatrix} = \begin{pmatrix} 2.8 \\ 3.2 \end{pmatrix}; \ \Omega_{C_1} = \begin{pmatrix} \frac{\vartheta_{C_2S_1}}{\Gamma_{C_2S_1}} \\ \frac{\vartheta_{C_2S_2}}{\Gamma_{C_2S_2}} \end{pmatrix} = \begin{pmatrix} \frac{2.5}{1.25} \\ \frac{2.5}{1.25} \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix};$$

$$\Omega_{S_1} = \begin{pmatrix} \frac{\vartheta_{S_1C_1}}{\Gamma_{S_1C_1}} \\ \frac{\vartheta_{S_1C_2}}{\Gamma_{S_1C_1}} \end{pmatrix} = \begin{pmatrix} \frac{4}{1.67} \\ \frac{6}{1.67} \end{pmatrix} = \begin{pmatrix} 2.4 \\ 3.6 \end{pmatrix}; \ \Omega_{S_2} = \begin{pmatrix} \frac{\vartheta_{S_2C_1}}{\Gamma_{S_2C_1}} \\ \frac{\vartheta_{S_2C_2}}{\Gamma_{S_2C_1}} \end{pmatrix} = \begin{pmatrix} \frac{6}{1.67} \\ \frac{9}{1.67} \end{pmatrix} = \begin{pmatrix} 3.6 \\ 5.4 \end{pmatrix}.$$

Then we will analyze the nodes in **class router**. In this case, all the terminals are in transport mode.

The sum of all the transport capacities between terminals in the case of extreme high communication is defined in form:

$\vartheta'_{n_1n_2} = \left(\sum_{i,j} \mu_{S_iC_j} + \sum_{i,j} \mu_{C_jS_i}\right)$ through the link from n_1 to n_2, $(\forall i, S_i \in S; \forall j, C_j \in C,$

(All the transport capacities μ in the definition here are calculated in the case extreme high communication in the network.)

Therefore, the transport capacity on each link = $\vartheta'_{n_1n_2}$;

Thus, the outgoing transport capacity of a router r is

$$\Omega_{r_{out}} = \begin{pmatrix} \vartheta'_{rn_1} \\ \vartheta'_{rn_2} \\ \cdots \\ \vartheta'_{rn_n} \end{pmatrix},$$

And the incoming transport capacity of a router r is

$$\Omega_{r_{in}} = \begin{pmatrix} \vartheta'_{n_1r} \\ \vartheta'_{n_2r} \\ \cdots \\ \vartheta'_{n_nr} \end{pmatrix},$$

where the nodes n_1, n_2...n_n are the neighbors of the router r.

For example,

According to the results in the page 52, the outgoing transport capacity of the router R_1 in Figure 3.20 is:

$$\Omega_{R_{1_{out}}} = \begin{pmatrix} \vartheta'_{R_1C_1} \\ \vartheta'_{R_1C_2} \\ \vartheta'_{R_1R_2} \end{pmatrix} = \begin{pmatrix} 2.4+3.6 \\ 3.6+5.4 \\ 2.8+3.2+2+2 \end{pmatrix} = \begin{pmatrix} 6 \\ 9 \\ 10 \end{pmatrix},$$

The incoming transport capacity of the router R_1 is:

$$\Omega_{R_{1_{in}}} = \begin{pmatrix} \vartheta'_{C_1R_1} \\ \vartheta'_{C_2R_1} \\ \vartheta'_{R_2R_1} \end{pmatrix} = \begin{pmatrix} 2.8+3.2 \\ 2+2 \\ 2.4+3.6+3.6+5.4 \end{pmatrix} = \begin{pmatrix} 6 \\ 4 \\ 15 \end{pmatrix}.$$

In a P2P Network

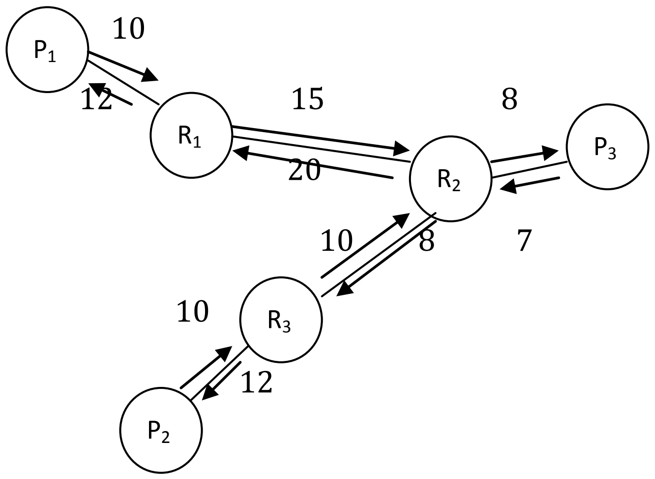


Fig 3.21 a simple network topology with three p2p nodes and three routers, where the digit on the arrows means the bandwidth of the link in the arrow direction.

For p2p point, either in the case extreme low communication or in the case extreme high communication in the network, all the **p2p nodes** will transport data as much as possible. So the transport capacity is limited by the bottleneck factor in the network.

And the definition of transport capacity on a p2p node p is:

$$\Omega_p = \begin{pmatrix} \frac{\omega_{pp1}}{o_{pp1}} \\ \frac{\omega_{pp2}}{o_{pp1}} \\ \cdots \\ \frac{\omega_{pp2}}{o_{pp n_p}} \end{pmatrix},$$

where nodes p1, p2 ... p_{n_p} are the p2p nodes in the p2p part network.

As showed in Figure 3.21, the maximal transport capacities on each p2p nodes are:

$\omega_{p_1p_2} = 8$; $\omega_{p_1p_3} = 8$; $\omega_{p_2p_1} = 10$; $\omega_{p_2p_3} = 8$; $\omega_{p_3p_1} = 7$ and $\omega_{p_3p_1} = 7$.

Then the bottleneck factor on each link in each direction can be calculated:

Sum of the max transport capacity on the link from n1 to n2 : ($\theta_{n_1n_2}$)	bandwidth	bottleneck factor on the link
$\theta_{p_1R_1} = 8 + 8 = 16$	$b_{p_1R_1} = 10$	$o_{p_1R_1} = \frac{\theta_{p_1R_1}}{b_{p_1R_1}} = \frac{16}{10} = 1.6$
$\theta_{p_2R_3} = 10 + 8 = 18$	$b_{p_2R_3} = 10$	$o_{p_2R_3} = \frac{\theta_{p_2R_3}}{b_{p_2R_3}} = \frac{18}{10} = 1.8$
$\theta_{R_1R_2} = 8 + 8 = 16$	$b_{R_1R_2} = 15$	$o_{R_1R_2} = \frac{\theta_{R_1R_2}}{b_{R_1R_2}} = \frac{16}{15} = 1.07$
$\theta_{R_3R_2} = 10 + 8 = 18$	$b_{R_3R_2} = 10$	$o_{R_3R_2} = \frac{\theta_{R_3R_2}}{b_{R_3R_2}} = \frac{18}{10} = 1.8$
$\theta_{R_2P_3} = 8 + 8 = 16$	$b_{R_2P_3} = 8$	$o_{R_2P_3} = \frac{\theta_{R_2P_3}}{b_{R_2P_3}} = \frac{16}{8} = 2$
$\theta_{P_3R_2} = 7 + 7 = 14$	$b_{P_3R_2} = 7$	$o_{P_3R_2} = \frac{\theta_{P_3R_2}}{b_{P_3R_2}} = \frac{14}{7} = 2$
$\theta_{R_2R_1} = 10 + 7 = 17$	$b_{R_2R_1} = 20$	$o_{R_2R_1} = 1$
$\theta_{R_2R_3} = 8 + 7 = 15$	$b_{R_2R_3} = 8$	$o_{R_2R_3} = \frac{\theta_{R_2R_3}}{b_{R_2R_3}} = \frac{15}{8} = 1.875$
$\theta_{R_1P_1} = 10 + 7 = 17$	$b_{R_1P_1} = 12$	$o_{R_1P_1} = \frac{\theta_{R_1P_1}}{b_{R_1P_1}} = \frac{17}{12} = 1.42$
$\theta_{R_3P_2} = 8 + 7 = 15$	$b_{R_3P_2} = 12$	$o_{R_3P_2} = \frac{\theta_{R_3P_2}}{b_{R_3P_2}} = \frac{15}{12} = 1.25$

Then each bottleneck factor on the shortest path is:

	Shortest path	bottleneck factor on the path
From P1 to P2	P1->R1->R2->R3->P2	$O_{P_1P_2} = \max(o_{P_1R_1}, o_{R_1R_2}, o_{R_2R_3}, o_{R_3P_2}) = 1.875$
From P1 to P3	P1->R1->R2->P3	$O_{P_1P_3} = \max(o_{P_1R_1}, o_{R_1R_2}, o_{R_2P_3}) = 2$
From P2 to P1	P2->R3->R2->R1->P1	$O_{P_2P_1} = \max(o_{P_2R_3}, o_{R_3R_2}, o_{R_2R_1}, o_{R_1P_1}) = 1.8$
From P2 to P3	P2->R3->R2->P3	$O_{P_2P_3} = \max(o_{P_2R_3}, o_{R_3R_2}, o_{R_2P_3}) = 2$

From P_3 to P_1 P_3->R_2->R_1->P_1 $O_{P_3P_1} = \max(o_{P_3R_2}, o_{R_2R_1}, o_{R_1P_1}) = 2$
From P_3 to P_2 P_3->R_2->R_3->P_2 $O_{P_3P_2} = \max(o_{P_3R_2}, o_{R_2R_3}, o_{R_3P_2}) = 2$
Thus, the transport capacities on p_1, p_2 and p_3 are:

$$\Omega_{P_1} = \begin{pmatrix} \frac{\omega_{P_1P_2}}{O_{P_1P_2}} \\ \frac{\omega_{P_1P_3}}{O_{P_1P_3}} \end{pmatrix} = \begin{pmatrix} \frac{8}{1.875} \\ \frac{8}{2} \end{pmatrix} = \begin{pmatrix} 4.27 \\ 4 \end{pmatrix};$$

$$\Omega_{P_2} = \begin{pmatrix} \frac{\omega_{P_2P_1}}{O_{P_2P_1}} \\ \frac{\omega_{P_2P_3}}{O_{P_2P_3}} \end{pmatrix} = \begin{pmatrix} \frac{10}{1.8} \\ \frac{8}{2} \end{pmatrix} = \begin{pmatrix} 5.56 \\ 4 \end{pmatrix};$$

$$\Omega_{P_3} = \begin{pmatrix} \frac{\omega_{P_3P_1}}{O_{P_3P_1}} \\ \frac{\omega_{P_3P_2}}{O_{P_3P_2}} \end{pmatrix} = \begin{pmatrix} \frac{7}{2} \\ \frac{7}{2} \end{pmatrix} = \begin{pmatrix} 3.5 \\ 3.5 \end{pmatrix}.$$

In a p2p network, all the p2p nodes transport data at the same time either in the case extreme low communication or in the extreme high communication. Therefore, the calculation of the transport capacity of a **router in p2p network** is same as the calculation of the transport capacity of a router in client server network in case extreme communication.

3.5.2.2.3 Time Complexity

In this design, we consider the shortest cost paths from a terminal to corresponding communication terminals. And the time complexity is O (n^3), where n is the number of nodes in a part network. So it is not scalable for very large network. We need make a network clustering at first.

In next chapter, the analysis of the time complexity will be introduced with codes in detail.

3.6 Compare Module

In the previous section, the transport capacity of each node has been already calculated; now each terminal has a vector and each router has two vectors.

Each node, whether they are in the same part network or not, need to be compared with other nodes, which are in the same class. That is to say, the compare is running in the whole network.

3.6.1 Standard vector of a group

When the first node is put into a new group, the vector of the node becomes the standard vector of a group. For each further compare, the node will be compared with the standard vector of each exist group. If a node is put into an existing group, then the standard vector of that group will be recalculated, the value becomes the average value of the vectors of all the nodes in the group.

3.6.2 Compare method

The compare method is running as following:

Compare within **terminals**:

In order to make a compare between two terminals, each item in a vector of a terminal will be compared in order from top to bottom with that of the other terminal. If the difference between vectors of these two terminals is smaller than a value, within a certain range, then the two terminals are in the same group.

Compare within **routers**:

Each router has two vectors: the incoming transport capacity vector and outgoing transport capacity vector. So In order to make a compare between two routers, each item in an incoming transport capacity vector and in an outgoing transport capacity vector of a router, will be compared in order from top to bottom with those of the other router. If both the difference between incoming transport capacity vectors and the difference between two outgoing transport capacity vectors of the two routers are smaller than a value, within a certain range, then the two routers are in the same group.

3.6.3 Sort the items in each vector

Before the compare is carried out, all the items in each vector are sorted by values in descending order from top to bottom.

For example:

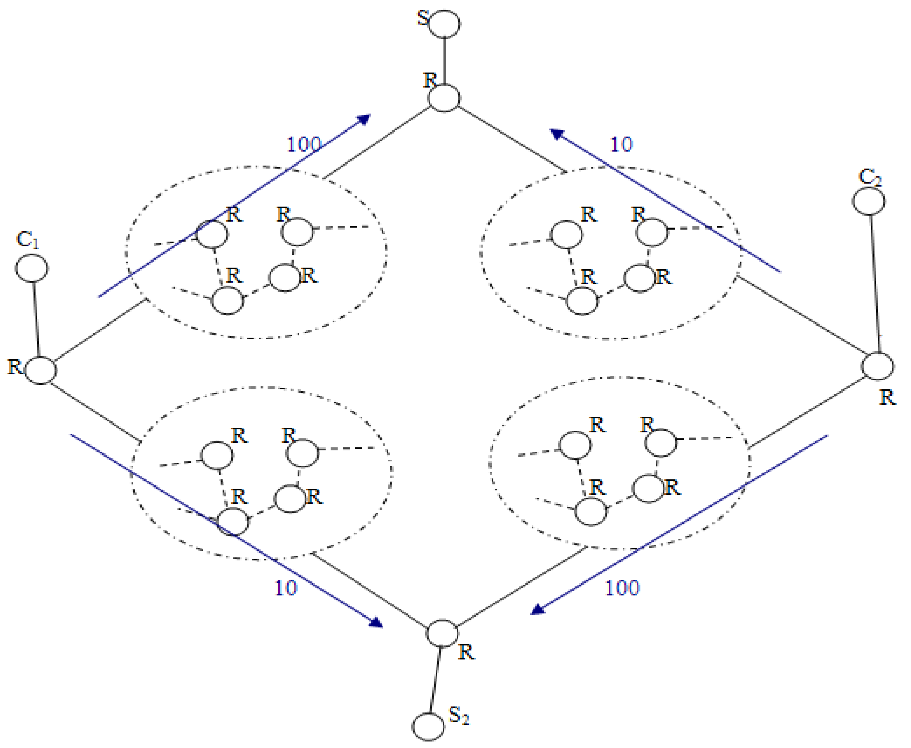


Fig 3.22 a network topology with two servers and two clients

In Figure 3.22, the transport capacities on client C_1 and C_2 are:

$$\Omega_{C_1} = \begin{pmatrix}100\\10\end{pmatrix} \qquad \Omega_{C_2} = \begin{pmatrix}10\\100\end{pmatrix},$$

Where the first item of each vector is always the transport capacity from the client to S_1 and the second item of each vector is the transport capacity from the client to S_2.

If we directly compare the vector C_1 with C_2 without sorting, we will get the result that the two clients are not in the same group; however, practically the transport capacity on C_1 is as big as that on C_2.

After the sorting, the transport capacities on client C_1 and C_2 are:

$$\Omega_{C_1} = \begin{pmatrix}100\\10\end{pmatrix} \qquad \Omega_{C_2} = \begin{pmatrix}100\\10\end{pmatrix},$$

In fact, for a client, the most important thing is the size of transport capacity, not the destination of each transport.

3.6.4 Compare Parameter

The compare parameter is a value, which is used for compare among nodes in the same class. If the difference between vectors of two nodes is within a range, which is determined by the compare parameter, then we could say the two nodes are in the same group.

The compare parameters for compare among nodes in different classes are different.

The compare parameter, which is used for compare among servers, is named server compare parameter; the compare parameter, which is used for compare among clients, is named client compare parameter; the compare parameter, which is used for compare among p2p nodes, is named p2p compare parameter; the compare parameter, which is used for compare among routers, is named router compare parameter.

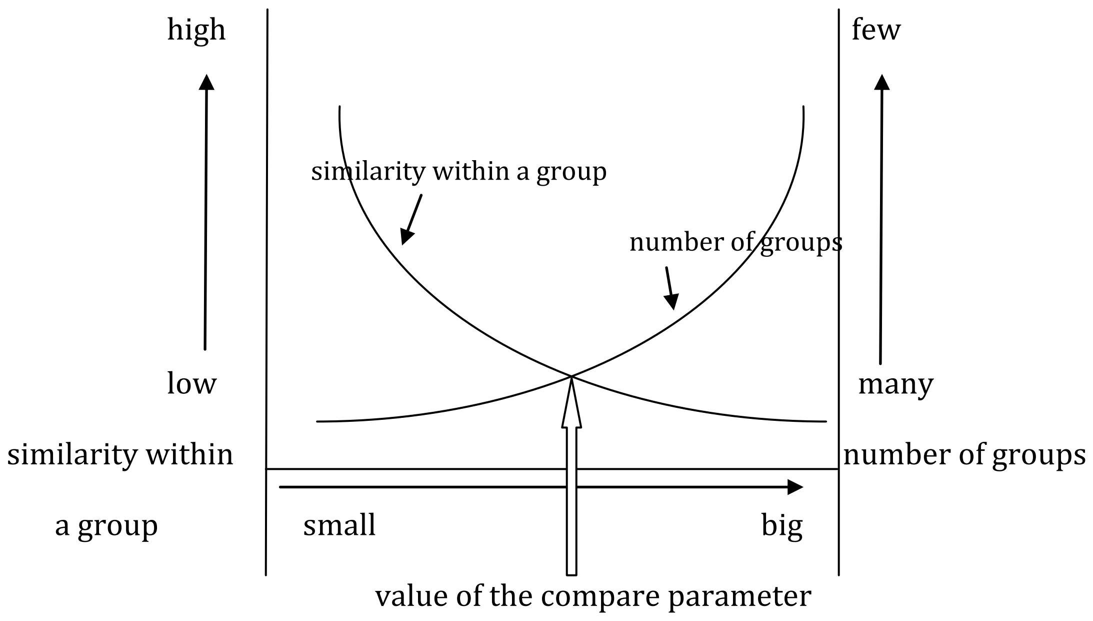


Fig. 3.23 compare parameter

The value of each kind of the compare parameter is adaptive. As showed in Figure 3.23, the smaller the compare parameter is, the more similar the nodes within a group is. But if the value is too small, then the nodes will be in too many groups divided. In this way, too many manual inquiries will be called. It cannot be accepted. We are hunting for a balance point, where the arrow is located in the picture.

3.7 Assignment Method

In the end, an assignment method is carried out.

At the beginning of the method, the CPU load on a node and the data rate on the outgoing links of the node will be inquired in each group.

According to the node classification algorithm described in the previous section, the nodes in the same group are in the same class: (two possibilities)

1. All the nodes in a group are routers.
2. All The nodes in a group are terminals.(class server, client and p2p belong to class terminal)

3.7.1 Assignment of Terminal

If nodes in a group are terminals, then we can directly assign the inquired CPU load on the terminal to other terminals; assign the inquired data rate on the outgoing link of the terminal to the outgoing link of other terminals. The reason is that, for each terminal, there is just one outgoing link.

3.7.2 Assignment of Router

If nodes in a group are routers, the assignment of CPU load of a router is same as that of a terminal. However, the assignment of data rates on the links is different, because the router in the same group may have different connection grads.

Therefore, the following rules are needed:

1. The sum of the data rates on the outgoing links of a router is same as the sum of data rates on the outgoing links of the inquired router.
2. The data rates will be divided into the corresponding links according to the weight of the links. (The link, whose cost is small, has a high weight.)

Chapter 4

Implementation

In this chapter the implementation details of the design, which has been described in the previous chapter, will be represented. The implementation is consisted of several parts: network clustering, shortest (cost) path algorithm, transport capacity calculation and compare module, where the last three parts belong to Node Classification Algorithm. All the components of the algorithm are written in language C.

4.1 Architecture of Implementation

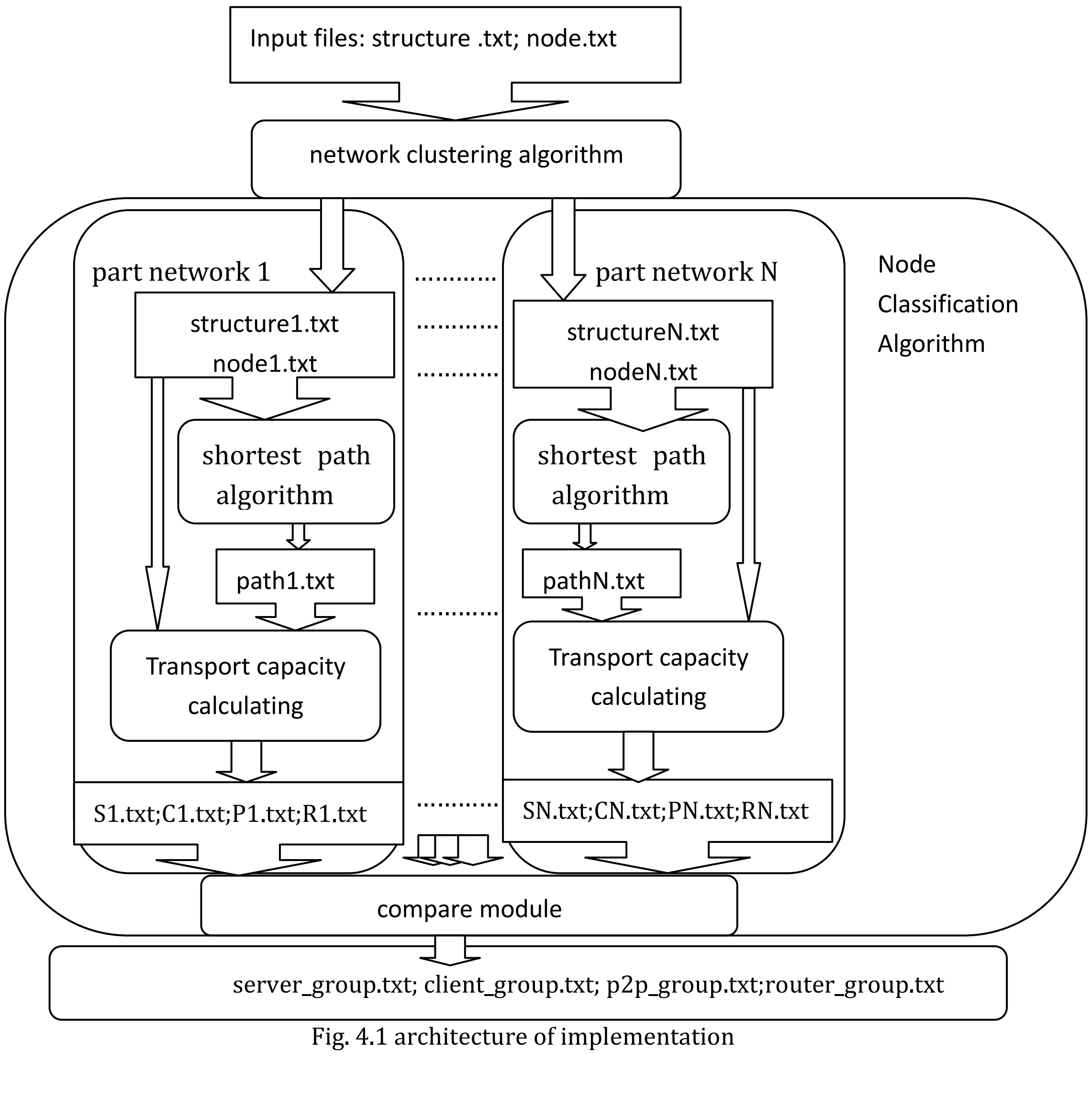

Input files: structure .txt; node.txt
network clustering algorithm
part network 1
structure1.txt
node1.txt
shortest path algorithm
path1.txt
Transport capacity calculating
S1.txt;C1.txt;P1.txt;R1.txt
part network N
structureN.txt
nodeN.txt
shortest path algorithm
pathN.txt
Transport capacity calculating
SN.txt;CN.txt;PN.txt;RN.txt
Node Classification Algorithm
compare module
server_group.txt; client_group.txt; p2p_group.txt;router_group.txt

Fig. 4.1 architecture of implementation

As what is in Figure 4.1 showed, as input, two files is given, one is named sturture.txt, in which the structure of links in the network is written; while the other is named node.txt, in which the information of nodes in the network is recorded.

In the end, the output files are server_group.txt, client_group.txt, p2p_group.txt and router_group.txt. From the name we can easy know, they are the results of the nodes classification for different node classes.

The whole process of implementation is running as following:

At the beginning, with the structure of links in the network, the network can be clustered into many relative small part networks (in Figure: from part network 1 to part network N), each of which has its own link structure and node information (in Figure: for example, files sturture1.txt and node1.txt are for part network 1; and files sturtureN.txt and nodeN.txt are for part network N).

After that, the shortest paths between each pair of nodes in every part network are calculated. It is the routing information for each pair of communication terminals, which is saved in pathn.txt, where n means that the routing information is for part network n. (for each $1 \leq n \leq N$)

Then the most important module in the Node Classification Algortihm is carried out. According to different cases, the data transport capacity of each node is calculated. And the corresponding data is saved in different files, where S1.txt means the transport capacity of server in part network 1 and C2.txt means the transport capacity of client in part network 2 and so on.

At last, we will use the compare module to compare all the nodes in the same class in the whole network. For example, the transport capacity of a server in S1.txt will be compared with that of each server in all the files S*.txt. As a result, we get a file group_server.txt, in which, each server belongs to which group is written. And the compares among clients, p2p nodes and routers are same as that among servers.

Now we will turn to each module of the implementation.

4.2 Network Clustering

In this section, the network clustering is described. In order to reduce the runtime of NETclassify, number of nodes in a part network is limited to1000.

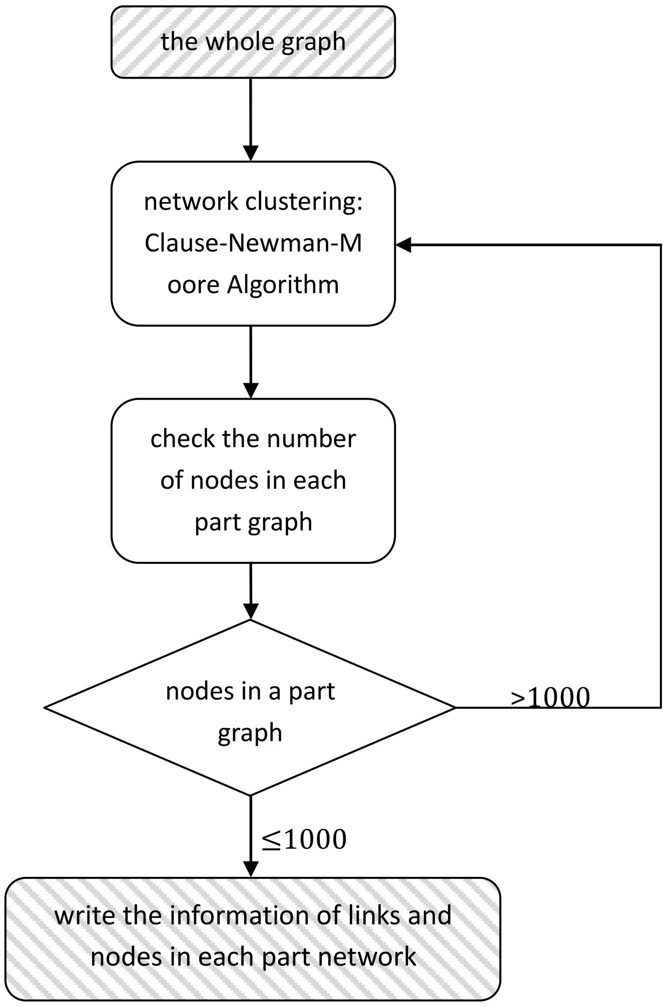

the whole graph
network clustering: Clause-Newman-Moore Algorithm
check the number of nodes in each part graph
nodes in a part graph
>1000
≤1000
write the information of links and nodes in each part network

Fig. 4.2 flow chat of the network clustering

As showed in Figure 4.2, the network clustering is running in this way. As the output of network clustering, we have got all the information of links and nodes in each part network.

4.2.1 Clauset-Newman-Moore Algorithm

The core of the network clustering is the Clauset-Newman-Moore Algorithm, the source code of which has been already written by the SNAP group in university STANFORD. [14]

Listing 4.1 Clauset-Newman-Moore Algorithm[14]

```
Input: Graph = (E, V)
Begin
    Matrix = Graph.get_adjacent Matrix();
    CmtyV = Graph.get_community();
    // maximize modularity
    Find (Matrix.exist_best_modularity()){
        //reconstruct communities
        CmtyV. reconstrut ();
    }
    return Matrix;
}
```

As showed in Listing 4.1, the algorithm is running in two steps. At first, the maximal modularity in the network is found; then the network is reconstructed according to the division, when the network has a maximal modularity.

4.3 Shortest Path Algorithm

As we all known, there are many shortest path algorithms in the world, which could be distinguished from each other in the following generalizations: single-source shortest path algorithm, single-destination shortest path algorithm and all-pairs shortest path algorithm.

Now we do need the third approach. All the terminals in a part network may transmit data to other corresponding terminals. We need the routing paths between each pair of corresponding terminals.

A famous algorithm of this approach is Floyd algorithm, [15] in which the shortest paths between all pairs of nodes are calculated. As a result, we get the cost of shortest path between each pair of nodes. But it is not useful for us; we want the path information between any two nodes. So a modified Floyd algorithm is used here, in which the forward sequence on the shortest path between each pair of nodes is recorded. [16] The source code is showed in listing 4.2.

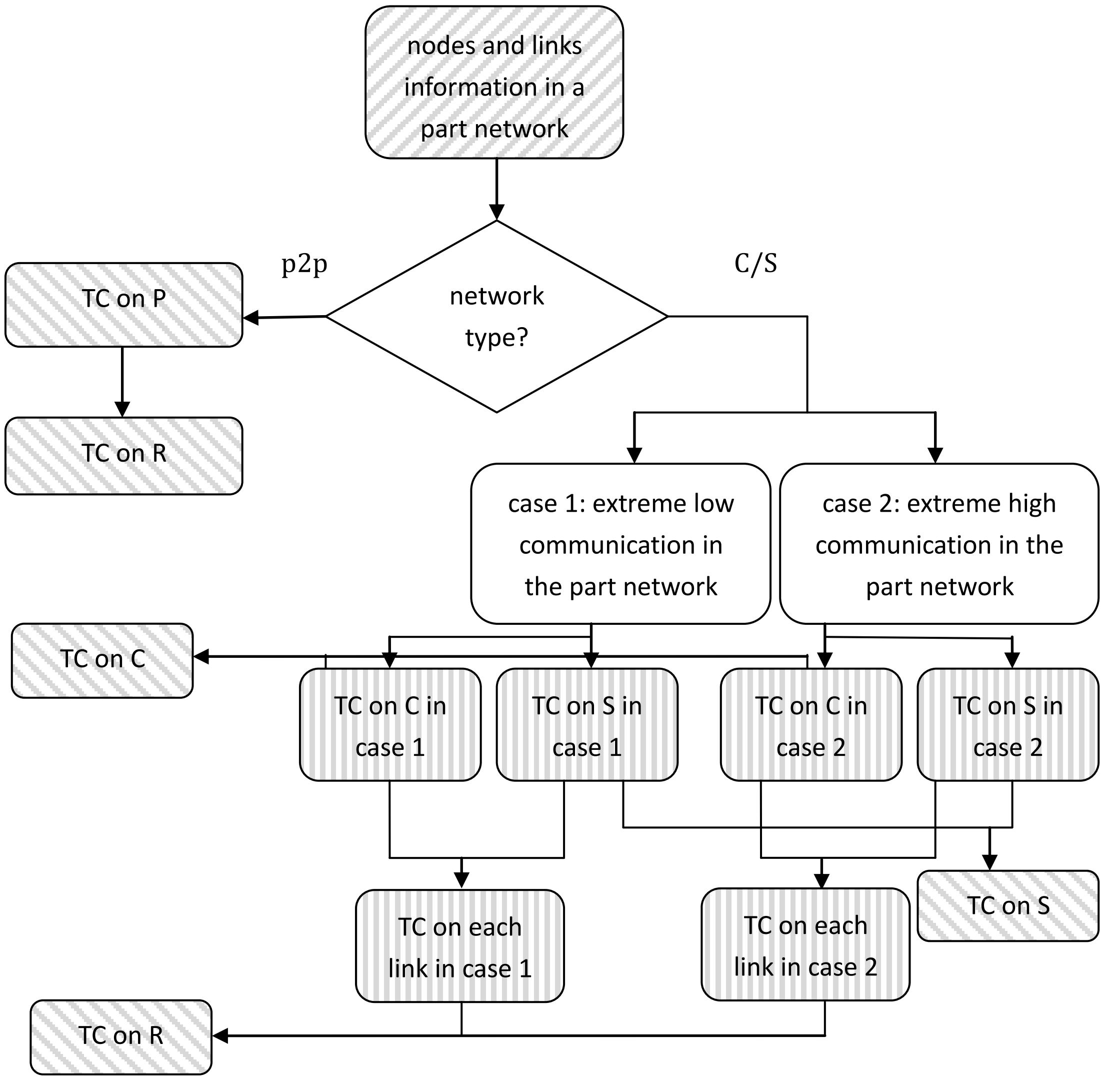
Listing 4.2 modified Floyd Algorithm

```
//the the forward sequence on the shortest path is saved in path[][]
void floyd(int dist[ ][ ], int path[ ][ ], int n)
{
    int i, j, k;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            path[i][j] = i;
    for (k = 0; k < n; k++)
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
```

```
                    if (dist[i][j] > dist[i][k]+dist[k][j])
                    {
                          path[i][j] = path[k][j];                    (*record the path*)
                          dist[i][j] = dist[i][k]+dist[k][j];
                    }
}
```

4.4 Transport Capacity Calculation

The calculation of transport capacity on each node is the most important part in the Node Classification Algorithm.



nodes and links information in a part network
network type?
p2p
C/S
TC on P
TC on R
case 1: extreme low communication in the part network
case 2: extreme high communication in the part network
TC on C
TC on C in case 1
TC on S in case 1
TC on C in case 2
TC on S in case 2
TC on each link in case 1
TC on each link in case 2
TC on S
TC on R

Figure 4.3 flow chat of the transport capacity calculation
In this figure, the TC means transport capacity calculation, R means router, C means client, P means p2p point and S means server

As showed in Figure 4.3, above all, we need know the network type of each part network, whether it is a p2p network or a client server network. The processes of analysis on nodes are different in different network types.

In a client server network, at first the calculations of transport capacities on clients and servers in each case (extreme low communication in a part network or extreme high communication in a part network) are running respectively. For each client or server, we have two vectors, each of which is calculated in an extreme case. After that, as what is said in the previous chapter, transport capacity on each server or client is produced with the communication parameter p. Finally, concerning the transport capacities on servers and clients, the transport capacity on each router could be calculated.

In a p2p network, the calculation of transport capacity is simpler than the one in client server network. Each p2p node transmits data always to other p2p nodes in any case. Therefore, we directly calculate the transport capacity on each p2p node. And concerning the results the transport capacity on each router is also calculated.

It had been elaborated in the previous chapter how to calculate each kind of Transport capacity concretely, so now I will not introduce the code again.

4.5 Compare Module

The transport capacity of a node is written in a vector. So the vector of a node in this section means the transport capacity of a node.

The compare is running just within the same class. (within class clients, servers, p2p nodes or routers.)

In each class, as showed in Figure 4.4, the first node is assigned to the first group, and the vector of the node becomes the standard vector of the group. Then from the second node, we can make a compare between the node and the standard vector(s) of the existing group(s). The vector of each node is compared with the standard vector in each group. If the difference between the vector of a node and the standard vector in an existing group is within a range, which is determined by compare parameter, then the node is assigned to this group, and the standard vector of this group is recalculated. If the difference between the vector of the node and the standard vector in each existing group is not within this range, then the node is assigned to a new group. After that, the standard vector of the new group is also generated.

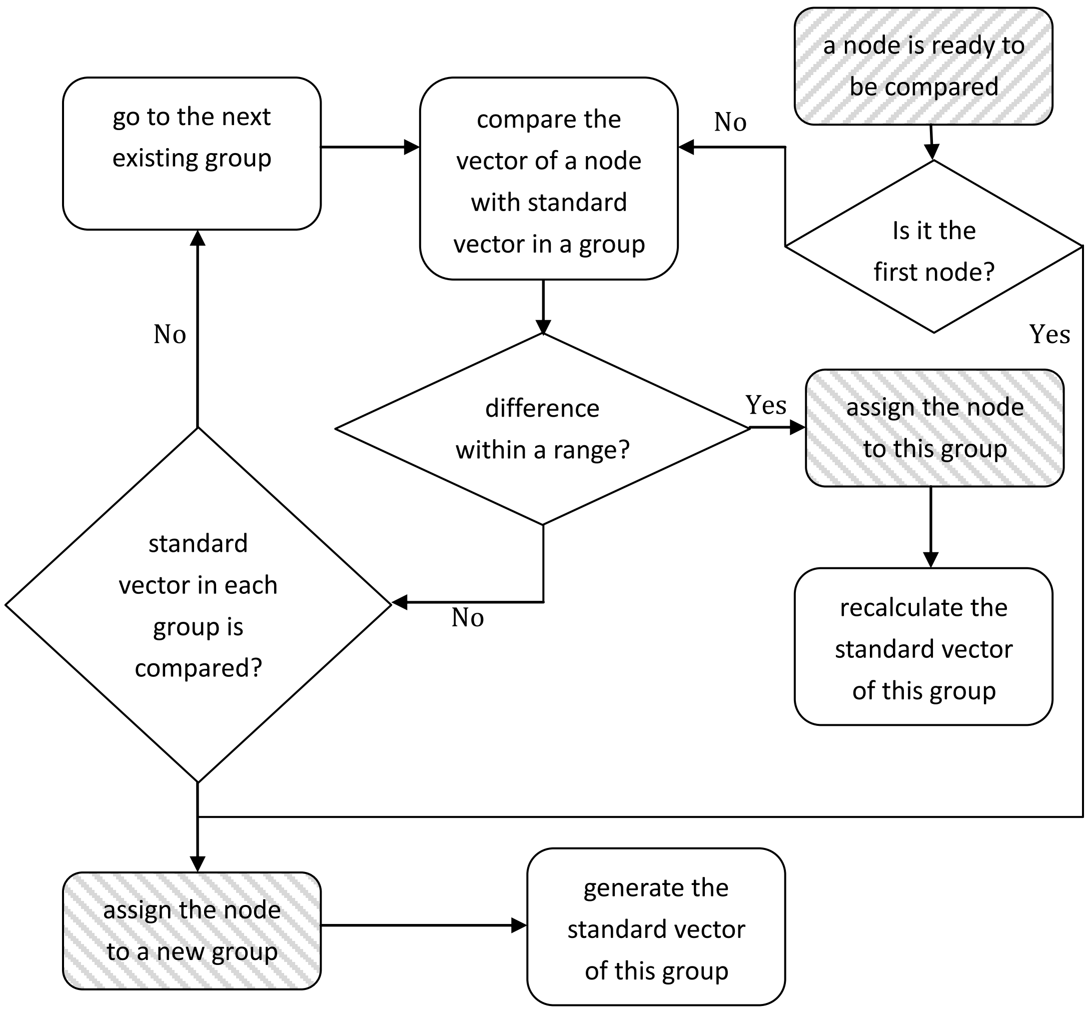

a node is ready to be compared
Is it the first node?
No
Yes
go to the next existing group
compare the vector of a node with standard vector in a group
difference within a range?
Yes
No
assign the node to this group
recalculate the standard vector of this group
No
standard vector in each group is compared?
assign the node to a new group
generate the standard vector of this group

Figure 4.4 flow chat of the compare module: make a compare among nodes in each class

4.5.1 Compare Method

In our approach, the numbers of items in the vectors of two routers are different, because the numbers of links on two routers may be different.

Furthermore, the nodes in a class will be compared with all the other nodes in the same class. The nodes may be come from different part networks. Due to the different numbers of clients, servers and p2p nodes in different part networks, the numbers of items in two vectors of terminals in same class maybe also different.

However, the compare between two vectors is running under the condition, that the numbers of items in two vectors are same. Therefore, the vector, the number of whose items is less, is reconstructed.

For example, two vectors:

$$\text{vector I} = \begin{pmatrix} i_1 \\ i_2 \\ \dots \\ i_m \end{pmatrix} \text{ and vector J} = \begin{pmatrix} j_1 \\ j_2 \\ \dots \\ j_n \end{pmatrix},$$

where m<n, the number of items in vector I is less than that in vector J.

Then the vector I is reconstructed:

$$\text{vector I} = \begin{pmatrix} i_1 \\ i_2 \\ \dots \\ i_m \\ i_{m+1} \\ \dots \\ i_n \end{pmatrix},$$

where all the items i_m, i_{m+1} ... i_n equal 0.

As said in the previous chapter, before compare, all the items in each vector are sorted by values in descending order from top to bottom. The last items in the vector, i_m to i_n, which equals to 0, are already in this order.

Compare Parameter

Compare parameter is the measuring metric. The difference between two vectors is calculated in the formula:

$$D_{IJ} = \begin{pmatrix} d_1 \\ d_2 \\ \dots \\ d_n \end{pmatrix}$$

Where vector I is the standard vector of a group, and $d_n = \begin{cases} i_n - j_n, & i_n \geq j_n \\ j_n - i_n, & i_n < j_n \end{cases}$

Then we can compare the $\sum_n(\frac{d_n}{i_n} \times i_n) = \sum_n(d_n)$ with the compare parameter, where $\frac{d_n}{i_n}$ is the proportion of difference value between an item in the node vector and that in a standard vector in a group to that in the standard vector in the group. And $\times i_n$ here means that, the weight of the proportion of difference value is considered.

4.5.2 Recalculation of Standard Vector in a Group

If a node is assigned to a new group, then vector of the node becomes the standard vector in the new group. However, if a node is assigned to an existing group, the standard vector of the group needs be recalculated. The standard vector in a group is always the average value of the vectors of all the nodes in the group.

If a node n is assigned to an existing group, then the standard vector of this group becomes:

$$\frac{\mathrm{SV} \times \mathrm{weight} + \mathrm{vector\ of\ node\ n}}{\mathrm{weight} + 1}$$

Where SV is the standard vector of the group before node n is assigned to this group and weight is the number of nodes in the group before n is assigned to the group.

For example, the recalculation of the vectors of a router is written in Listing 4.3.

For each router r, there are two vectors: one represents the incoming transport capacity on a router and the other represents the outgoing transport capacity on a router.

Listing 4.3 Recalculation of the vectors of a router

```
Input:
//the vectors of the router
Analy_In[ ]; Analy_Out[ ];
//standard vectors of the group, where the router is assigned
SV_In[ ][ ];SV_Out[ ][ ];
Begin
  Grad = max (Analy_In[ ].get_number_of_items(), SV_In[ ].get_number_of_items())
  for (int i=0;i<Grad;i++){                    (*each item in the vectors are recalculated*)
      //each item in the standard vector of incoming transport capacity on a router
      SV_In[ ][i] = (standard_ In[ ][i] * weight + analy_In[i]) / (weight + 1)
      //each item in the standard vector of outgoing transport capacity on a router
      SV_Out[ ] [i] = (SV_Out[ ][i] * weight+analy_Out[i]) / (weight + 1);
  }
  weight = weight + 1; (*after the recalculation, the weight in the group is increased*)
End
```

Chapter 5

Evaluation

In this chapter, the evaluation of NETclassify is presented. At first, the goal of the evaluation of the algorithm is explained. Then we will introduce the platform, where NETclassify is running. Finally, the results of the evaluation is presented and discussed.

5.1 Evaluation Goals

The goal of the evaluation is to examine if the implementation of NETclassify described in the previous chapter fulfills the main requirements of NETclassify which are described in Chapter 3.1.

A research in the evaluation is the runtime of the algorithm and the number of groups in the results of the nodes classification. We want to know that, whether an acceptable classification can be finished in an acceptable time.

Furthermore, we want to know, whether the nodes in the same group have similar transport characteristics.

5.2 Platform

As hardware infrastructure, a 12 nodes cluster server "curium" is set up. Each node is a Dual QuadXeon. Each QuaXeou has 8 CPUs, each of which is 3000GHz. Our implementation is running on it.

On the other hand, the software environment for the implementation is Linux system. And in this implementation, the program of NETclassify is running in line one by one.

5.3 Evaluation Results

Above all, the input network topologies are introduced. Nine network topologies are offered for this Evaluation.

They are described in the following table:

Description	Topology	Number of links (directions)	Number of routers	Number of terminals
a network topology from DSL supplier ATandT	ATandT	3895	753	1500
simplified vision of Caumpus2	Campus1	250	170	50
composed with 20 Campus network topologies	Campus2	5620	600	4880
a network topology from internet	Internet	6632	1454	659
snapshot of routers in internet	NetworkMap	4527	2376	800
generated by the topology generator BRITE [17]	TopoAS	2609	1024	500
generated by BRITE	Waxman1.25k	3000	1250	500
generated by BRITE	Waxman2.5k	5500	2500	500
generated by BRITE	Waxman5k	12500	5000	2500

In order to execute the implementation, three kinds of parameters are offered: network name, communication case parameter and compare parameter. Furthermore, for each network, 5 kinds of different link properties (bandwidth, maximal delay and packet loss) on each link are provided in this evaluation, which is determined by link parameter.

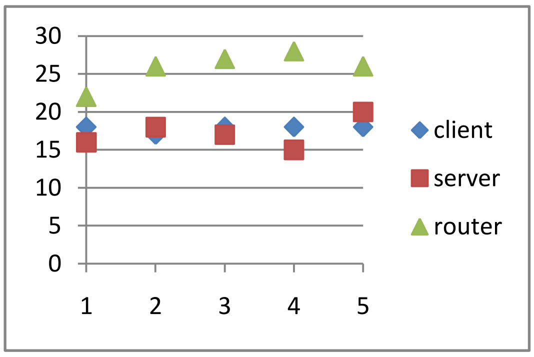
At first, we want to know, whether the link parameter has an influence on the results of classification, when the compare is made among nodes in the same network with same communication case parameter and compare parameter.

In each following chart the x-axis represents different link parameter, and the y-axis represents the number of groups for each class.

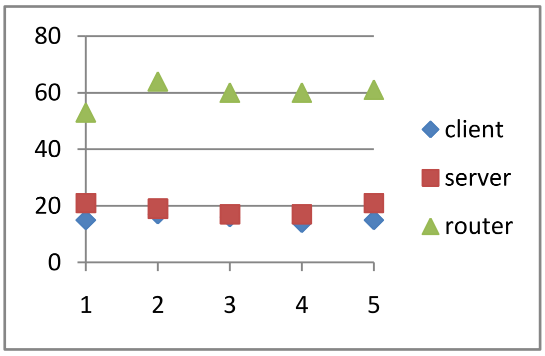
For each chart there is a description, the format is:

network name/communication case parameter/compare parameter/number of terminals in the network/number of servers in the network

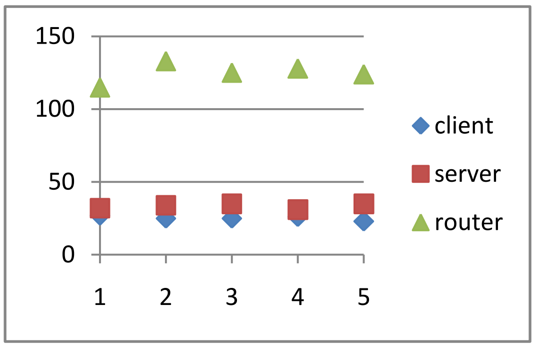
ATandT/0.25/0.5/1500/753



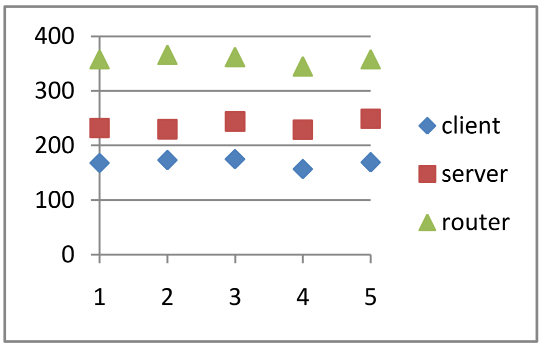
ATandT/0.75/0.4/1500/753



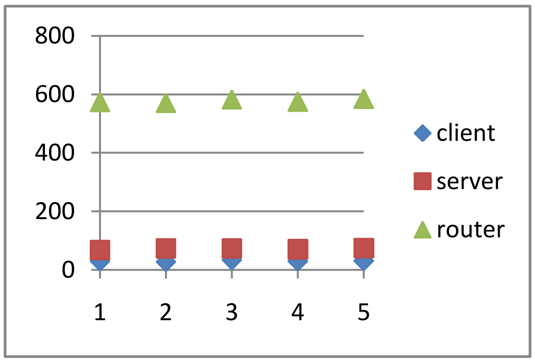
ATandT/0.5/0.3/1500/753



ATandT/0.0/0.2/1500/753



Waxman1.25k/1.0/0.1/1500/753



Internet/0.25/0.3/659/1454

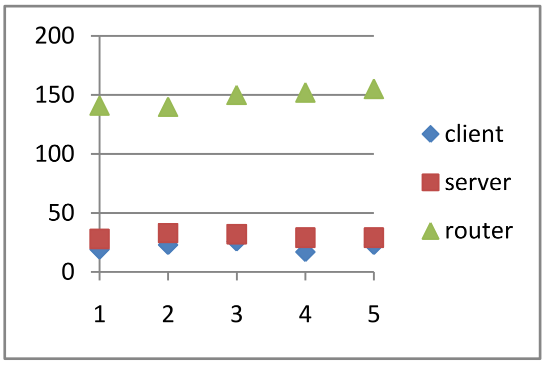


Fig 5.1 evaluation of the implementation with different link parameters

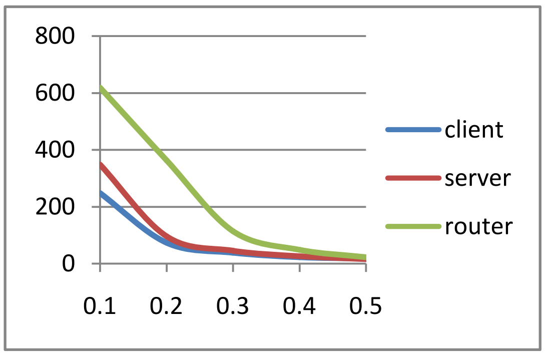
As shown in Figure 5.1, different link parameters of each network have just little influence on the results of classification, when the compare is made among nodes in the same network with same communication parameter and compare parameter.

Then we will make a research on the compare parameter. Therefore, now in each following chart the x-axis represents different compare parameters, and the y-axis represents the number of groups for each class.

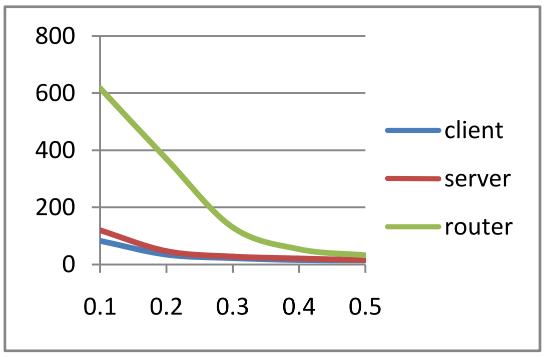
And the description of each chart here is in form:

network name/communication case parameter/number of terminals in the network/number of servers in the network

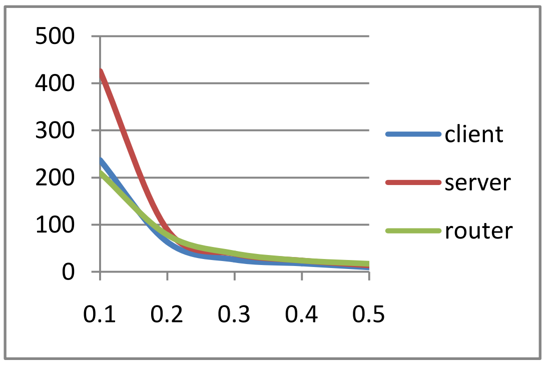
ATandT/0.25/1500/753



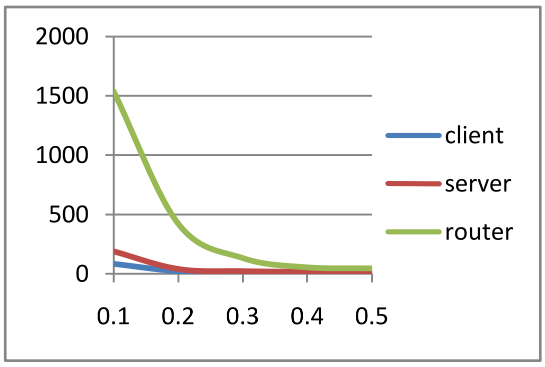
ATandT/0.75/1500/753



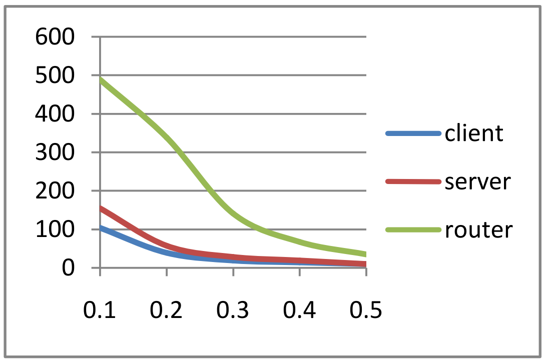
Campus2/0.25/4880/600



Waxman5k/0.5/2500/5000



Internet/0.25/659/1454



TopoAS/0.5/500/1024

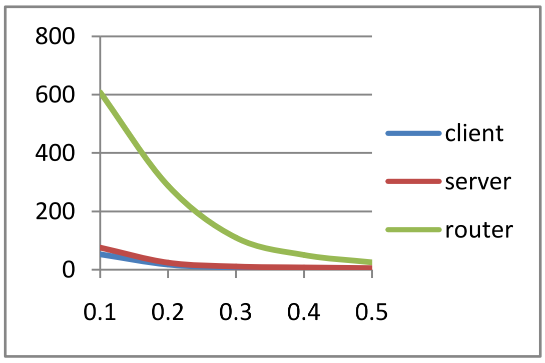


Fig 5.2 evaluation of the implementation with different compare parameters

As shown in Figure 5.2, compare parameter has a great influence on the results of classification, when the compare is made among nodes in the same network with same communication case parameter. The smaller the compare parameter is, the more groups after classification there will be. And likely, the increase of the nodes in the network will also be accompanied with a corresponding growth in the quantity of the groups.

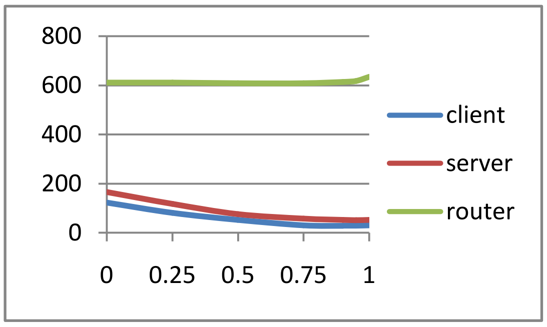
We can control the number of groups through change the adaptive compare parameter.

After that, we will focus on the communication case parameter. Therefore, in each following chart the x-axis represents different communication case parameters, and the y-axis represents the number of groups for each class.

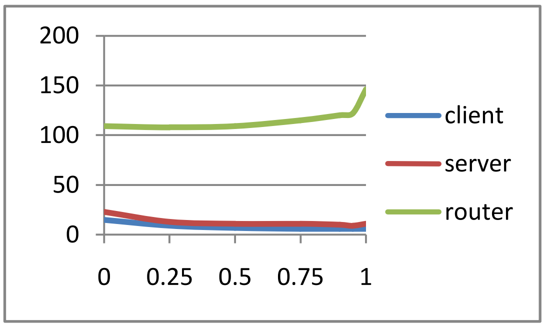
And the description of each chart here is in form:

network name/compare parameter/number of terminals in the network/ number of servers in the network

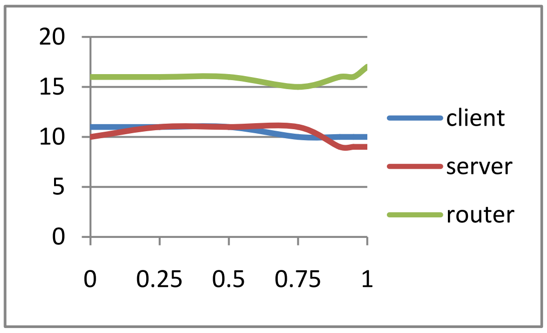
TopoAS/0.1/500/1024



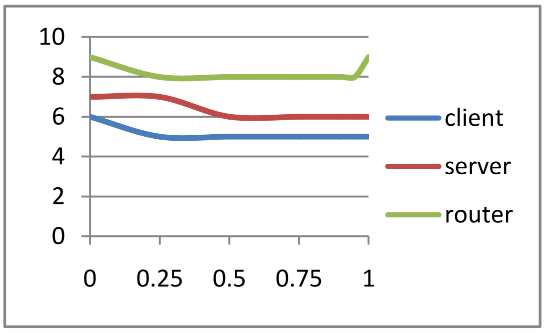
TopoAS/0.3/500/1024



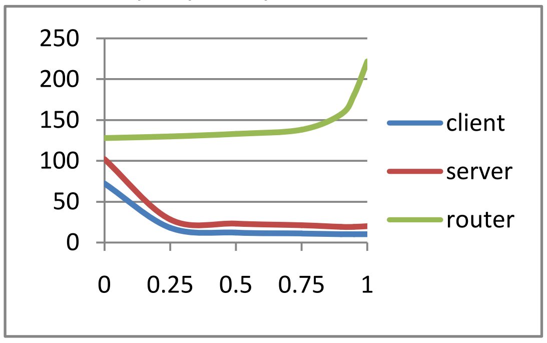
Compus1/0.1/50/170



Compus1/0.2/50/170



Waxman5k/0.3/2500/5000



Waxman5k/0.5/2500/5000

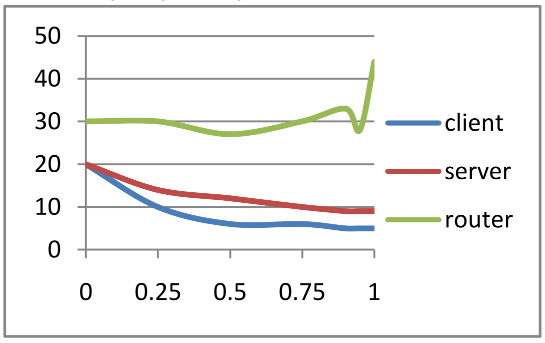
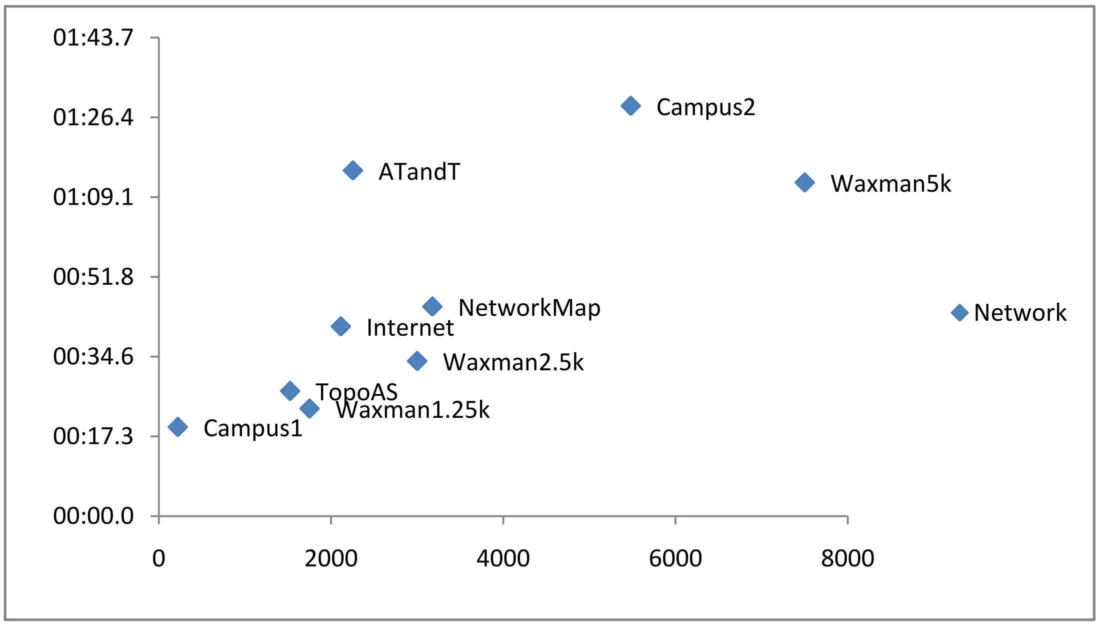


Fig 5.3 evaluation of the implementation with different communication case parameters

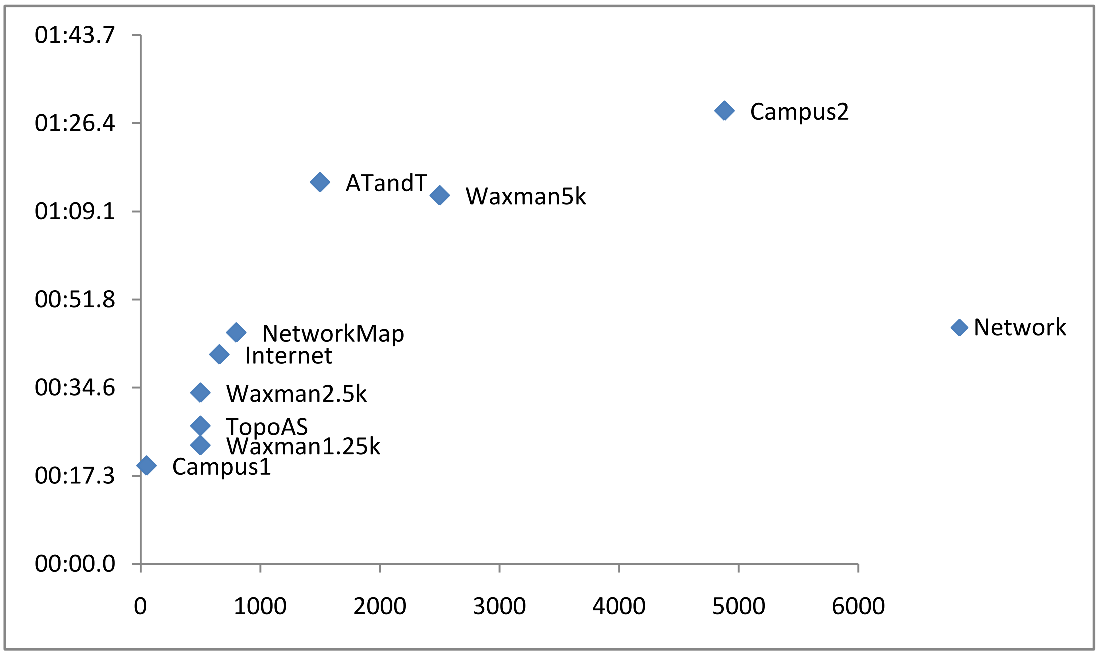
As showed in Figure 5.3, in the network, which are rich in nodes, the number of the groups fluctuates in line with different communication case parameters; on the contrary, when there are fewer nodes in the network, the number remains level in such a case. However, in the same network the effects of compare parameter are found negligible.

Finally, we will make a research on the runtime of the implementation. We want to know, whether the runtime of the implementation is acceptable.

The average runtime of different networks are showed in Figure 5.4 (a) and (b).



(a) x-axis represents number of nodes in a network



(b) x-axis represents number of terminals in a network

Fig 5.4 compare of runtime in different networks

As showed in Figure 5.4, for different networks, there is an upward trend in the runtime with regards of the number of the nodes (especially the number of terminals) in the network, but the rate of increase slow down. Therefore, the runtime of NETclassify is acceptable.

However, it is very hard to evaluate, whether the nodes in the same group have similar real-time transport capacity or not. In the further, this problem should be solved. A possible solution is through some measuring methods of bottleneck link capacity and available bandwidth in the complex transmission situation to calculate the available transport capacity of each node. Then with this result we can know the similarity of a group.

Chapter 6

Conclusion

In order to offer NETplace two important parameters: expected CPU load on a node and expected data rates on each data link, an algorithm NETclassify is designed in this diploma thesis.

NETclassify is consisted of two most important parts, network clustering and Node Classification Algorithm. In the network clustering, the whole network is divided into small part networks. Furthermore, before the execution of Node Classification Algorithm, nodes in each part network are divided into different classes. And the shortest paths between each pair of communication nodes are calculated in a transmission cost model, which is the routing information. Then the Node Classification Algorithm is carried out. Concerning the routing information in the part network and the different functions of nodes in different classes, the transport capacity on each node is calculated and in the compare method, the nodes in the whole network will be composed, if they within the same class. As the result, each node is assigned to a suitable group. Finally, a manual inquiry and assignment of the CPU load on a node and data rates on the outgoing data links of it in each group is given, and according to the manual input, the CPU load on other nodes and data rates on the outgoing data links of them in the same group are also automatically assigned.

In the evaluation, we can see that, the runtime of the NETclassify in different networks is acceptable (less than 2 minutes); even when the number of nodes in the network is more than 7000 nodes. And the results of the classification depend strongly on the compare parameter. So if the operator thinks that, the number of groups is too high, then he could set the compare parameter to a higher value. Then he can get the desired result.

However, we do not know how to evaluate the similarity of nodes in a group. In the further, this problem should be solved.

References

[1] NET-Project. http://net.informatik.uni-stuttgart.de/, 2008.

[2] Andreas Grau, Steffen Maier, Klaus Herrmann, Kurt Rothermel, Time Jails: A Hybrid Approach to Scalable Network Emulation, 22nd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2008)

[3] Andreas Grau, Klaus Herrmann, and Kurt Rothermel, NETplace: Efficient Runtime Minimization of Network Emulation Experiments, Proceeding of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'10) [Best Paper Award], 2010

[4] Satu Elisa Schaeffer, Graph clustering, Computer Science Review, Volume 1, Issue 1, August 2007

[5] M. E. J. Newman, Detecting community structure in networks, The European Physical Journal B - Condensed Matter and Complex Systems, Vol. 38, No. 2. (25 March 2004)

[6] Matthew J. Rattigan, Marc Maier, David Jensen, Graph Clustering with Network Structure Indices, ICML '07 Proceedings of the 24th international conference on Machine learning

[7] M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks, Physical review, E, Statistical, nonlinear, and soft matter physics, Vol. 69, No. 2 Pt 2. (February 2004)

[8] Lei, Tang, Community Detection in Social Networks, http:// www.public.asu.edu/~huanliu/dmml_presentation/2008/Communi ty % 20Detection%20in%20Social%20Networks.pdf

[9] Aaron Clauset, M. E. J. Newman, and Cristopher Moore, Finding community structure in very large networks, Physical Review E (2004), p. 1- 6.

[10] M. E. J. Newman, Analysis of weighted networks, Phys. Rev. E 70, 056131 (2004)

[11] M. Girvan, M.E.J. Newman, Community structure in social and biological networks, Proceedings of the National Academy of Sciences, USA 99 (2002) 8271–8276.

[12] P. Elias, A. Feinstein, C.E. Shannon, Note on maximum flow through a network, IRE Transactions on Information Theory IT2(1956) 117–119.

[13] L.R. Ford Jr., D.R. Fulkerson, Maximum flow through a network,

Canadian Journal of Mathematics 8 (1956) 399–404.

[14] SNAP, http://snap.stanford.edu/data/index.html

[15] Floyd algorithm, Volker Diekert, from teaching materials of "Entwurf und Analyse von Algorithmen", SS 2006

[16] http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorith m

[17] Brite, http://www.cs.bu.edu/brite/

[18] Blaauw, D.T.; Banerjee, P.; Abraham, J.A., Automatic classification of node types in switch-level descriptions, Computer Design: VLSI in Computers and Processors, 1990. ICCD '90. Proceedings., 1990 IEEE International Conference

[19] Ningning Hu, Peter Steenkiste, Evaluation and Characterization of Available Bandwidth Probing Techniques, IEEE Journal on Selected Areas in Communications, Vol. 21 (2003)

[20] Jin Cao; William S. Cleveland; Don X. Sun, Bandwidth Estimation for Best-Effort Internet Traffic, Statist. Sci. Volume 19, Number 3 (2004), 518-543.

Statement

I ensure that I have created this document on my own and only used those external sources in the references.

kai zhou