Security Incident Handling and Organisational Models

Hossein Hayati Karun

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2006

Acknowledgements

Thanks to my supervisor Professor Slobodan Petrovic for guidance and feedback during the work.

Thanks to Brita Vesterås for all her excellent help.

Thanks to Espen Aria and Lisa for having patience with me during last two years.

Audience

This master thesis focuses on calculating the efficiency in handling security incidents and is of main interest for personnel dealing with security organisation, security management and security policies.

The results achieved by this thesis can be used to get a better insight into the efficiency in handling security incidents in a hierarchical and matrix organisational model. One can also use the software prototype, developed during this thesis, to calculate the efficiency of other organisational models. Any changes like merging or dividing companies or department will create a new state in the organisation. By mapping the current organisations chart, at any time, our prorotype can calculate/compute the efficiency of the organisation in the most efficient way within few seconds.

Abbreviations/Definitions

Availability	- Availability is ensuring that information or data resources are present and applicable as necessary in accordance with agreed criteria[19].
Capacity	- The capacity of the flow
CIRT	- Computer Incident Response Team
Confidentiality	- The content of communications should remain secret: an unauthorized person should not be able to learn any private information. "Confidentiality is ensuring that information will not be available for unauthorised people or unapproved systems"[19].
C#	- C sharp, Programming language developed by Microsoft
Edge	- One line/arrow which connects two nodes
Graph	- A set of nodes connected with edges
Incident	- "Something unusual, serious, or violent that happens"[4]. "A group of attacks that can be distinguished from other attacks because of the attackers, attacks, objectives, items and timing"[14]. "A violation or imminent threat of violation of computer security policies, acceptable user policies, or standard computer security practices"[8]. An incident is any unexpected action that has an immediate or potential effect on the organisation.
Incident Handling	- "The mitigation of violations of security policies and recommended practices"[8].
Integrity	- Unauthorized changes to information transmitted between legitimate parties must be detected by the receiver. "Integrity is ensuring that information is not changed or destroyed in an unauthorised manner, that information is in agreement with reality and is consistent"[19].
Maximum Flow	- Maximum flow in a network
Network	- Set of nodes and edges
Network Flow	- Flow in a network
Node/vertice	- Any connecting point in a graph
Security Incident	- "Any adverse event whereby some aspect of computer security could be threatened: Loss of data confidentiality, disruption of data or system integrity, or disruption or denial of availability"[8]
Security Policy	- Guidelines to describe the policy within use of organisation's information system and assets.

Contents

Acknowledgements iii
Audience v
Abbreviations/Definitions vii
Contents ix
List of Figures xi
1 Introduction 1
1.1 Topic 1
1.2 Keywords 1
1.3 Problem description 1
1.4 Justification, motivation and benefits 2
1.4.1 Research questions 2
1.5 Method in this thesis 3
2 Preliminaries 5
2.1 Exploring the routines for handling security incidents 5
2.2 How to increase the efficiency 6
2.3 Incident 6
2.4 Security incident 6
2.5 Routines for handling security incident 7
2.5.1 Detecting The Threat 7
2.5.2 Locating The Threat 7
2.5.3 Reporting 8
2.5.4 Response to the threat 8
2.5.5 Forensic 9
2.5.6 Summary of routines for handling security incident 9
3 Evaluating efficiency of organisational models for security incidents handling **11**
3.1 Organisational models 11
3.1.1 Hierarchical model 12
3.1.2 Matrix organisational model 13
3.2 Graph 14
3.3 Karl Menger's theorem 15
3.4 The mathematical background of max flow - min cut 18
3.4.1 Flow in network 19
3.4.2 Feasible flow 19
3.4.3 Cuts 20
3.4.4 Relationship between flows and cuts 20
3.4.5 The maximum-flow and the minimum-cut problem 23
3.5 Algorithms for computing the max flow 24
3.5.1 Ford-Fulkerson's algorithm 24
3.5.2 Edmonds-Karp algorithm 24
3.5.3 Relabel-to-front 25

3.5.4 Comparing algorithms 26
3.6 Implementation of Ford-Fulkerson's algorithm 27
3.7 Measuring the readiness of handling security incidents 27
3.8 Developing the prototype 27
3.9 The prototype 28
3.9.1 The input file 28
3.9.2 Computing the max flow 29
3.9.3 The result of computing max flow 29
3.9.4 Graphical presentation 30
4 The experiment 31
4.1 Reliability and validity 31
4.2 Organisation charts 32
4.2.1 Hierarchical structure 33
4.2.2 Matrix structure 37
4.3 The result of our experimental work 41
4.3.1 Relation between nodes and edges 42
4.3.2 Tasks: Max flow - Hierarchical structure 43
4.3.3 Task: Max flow - Matrix structure 44
4.3.4 Report: Max flow - Hierarchical structure 45
4.3.5 Report: Max flow - Matrix structure 46
5 Discussion 47
5.1 Comparison of max flow for solved security incidents 47
5.2 Comparison of max flow for reported security incidents 48
5.3 Summary 49
6 Conclusion 51
7 Further Work 53
Bibliography 55
Appendix 1 - Ford-Fulkerson's algorithm 59
Appendix 2 - Edmond-Karp's algorithm 61
Appendix 3 - Relabel-to-front algorithm 63
Appendix 4 - Max flow algorithm from our prorotype 65

List of Figures

1 Function based organisation chart 12
2 Market based organisation chart 12
3 Hierarchical organisation chart 13
4 Matrix organisation chart 13
5 Example Graph 14
6 Example Graph - Menger's definition of a graph 15
7 **Max flow - min cut**; left: capacities, right: one used path 16
8 **Max flow - min cut**; left: computed max flow, right: all nine minimal cuts 18
9 Cuts 20
10 Graph subsets 21
11 Cut capacity 22
12 Sample graph: O estimates 26
13 Sample text output result 29
14 Displaying the Maximum Flow 30
15 Sample graphic output result 30
16 1. Sample hierarchical organisation chart 34
17 2. Sample hierarchical organisation chart 35
18 3. Sample hierarchical organisation chart 36
19 1. Sample matrix organisation chart 38
20 2. Sample matrix organisation chart 39
21 3. Sample matrix organisation chart 40
22 Relation between nodes and edges 42
23 Max flow, tasks - Hierarchy 43
24 Max flow, tasks - Matrix 44
25 Max flow, reports - Hierarchy 45
26 Max flow, reports - Matrix 46
27 Max flow, Tasks 47
28 Max flow, Reports 48

1 Introduction

1.1 Topic

Every organisation is dependent on stable and reliable communication channels between people and machines. In order to gain and maintain reliability in communication protocols and channels it is necessary to have reliable routines.

Having these thoughts in mind, it is not difficult to see the need for a safe system, not only by means of networking etc., but also routines and communication channels. These routines, that vary from case to case depending on their nature, can often ease the pressure on people and eliminate the possibility of doubts. In order to assure the security, *which is a vital part of any organisation* not only within an information system but also in routines, one should think of the efficiency of these routines.

In this Master's Thesis we focus on routines for handling security incidents in two different organisational models, hierarchical and matrix model. In order to simulate these organisational models and demonstrate the efficiency of work flow as a result of routines for security incident handling, we develop a software prototype, in C#, that computes the maximum flow in a given network.

1.2 Keywords

Security policy, information security, confidentiality, integrity, availability, security incident, security incident handling, network, graph, vertice/node, edge, capacity, network flow, maximum flow, Menger's theorem, Ford-Fulkerson's algorithm, security perimeters, management.

1.3 Problem description

Security incidents have become a part of daily life within most organisations since dependency on communication and networking is an essential part of their daily activities. Taking advantage of possibilities that come from networking often offers new and unknown security threats which need to be handled. Handling these security incidents presents the need for reliable routines, which can assure immediate response.

Reliability within routines is a necessity for providing safe routines, but this alone is far from enough to assure immediate response in case of security incidents. To improve a reliable routine it is indeed important to explore the possibilities for increasing their efficiency. To improve efficiency of incident handling routines one can take advantage of traditional ways of how to improve them.

In order to be able to determine the efficiency in handling security incident routines we need an effective way to do this, because such a calculation is a time consuming process. It is also important to use a robust and reliable method that can guarantee the same result for the same input, at any given time. This is a good reason for finding a trustworthy and reliable method to employ in this thesis. Based on this need, we choose to explore and implement the Menger's theorem of edge-connectivity in a network context. This theorem in a network context helps us to compute the maximum flow in a network, where the network can be ported/mapped/converted into an organisation chart.

By implementing Ford-Fulkerson's algorithm, max flow - min cut, which is based on Menger's theorem, in a software prototype we can measure the ratio of effectiveness within routines for handling security incidents. This prototype can demonstrate the efficiency of these routines within different organisation models, of the same size.

1.4 Justification, motivation and benefits

According to Norwegian law of personal information, POL §13 (Information Security) and POF §§2-3 (Security Management), the management is responsible to meet the requirements for assuring proper handling of security measures.

Having proper security incident handling routines not only benefits the immediate handling and elimination of the threat but also increases the efficiency and profit. Regardless of the organisation's structure in private or public sector, productivity, lower expenses and higher profit are always in focus. In order to attain, maintain and improve these goals, management has a significant role and responsibilities, which demand a more active participation in activities within the organisation. A challenging area within any organisation is handling security incidents, which emphasises the importance of adequate and effective routines for it.

By having these routines and procedures, the organisation and employees will be able to respond quickly to the threat when it is detected and located. The advantage of having effective and well functioning incident handling routines, is that the organisation will not only improve its performance and be more prepared for new situations, but also be able to handle sudden changes within the organisation. A well planned and smart security incident handling routine should dynamically handle sudden changes within the organisation. Having such routines as tools will ease the management's decision making about changes in the organisation's structure. This kind of incident handling routines can be used by a variety of parties like managements and security officers, both in private and public sectors.

1.4.1 Research questions

Whether or not routines for incident handling in different organisational models are employed according to their security policy or other rules, it is important to see if these routines oblige to expectations, are carried out the way they are supposed to, are efficient and how to increase the efficiency within these routines.

In this case we have chosen to implement combinatorial optimisation algorithms in a prototype of a system capable of evaluating efficiency of security incident handling procedures. This prototype is used to measure the efficiency within routines for incident handling by calculating the max flow in each organisational model.

By studying two organisational models, six sample organisation structures and their max flow were examined. We wanted to see if there is any significant difference between these organisation structures. These results are used to choose a preferable organisational model because of its efficiency.

We also take a look at the costs of conversion of less efficient routines in an organisational model into more efficient ones, and calculate the benefit of reorganising to a more efficient organisational model.

This leads to the following research questions, which we attempt to answer in this master thesis:

1. How to measure the efficiency of routines for security incident handling in two organisational models?
2. How to increase the efficiency of routines for handling security incidents?

1.5 Method in this thesis

The goal of this thesis is to examine the efficiency of two different organisational models regarding handling security incidents. In order to do this, we create three different organisational charts of each kind of model, with different sizes. These charts are converted into graphs, so that we can use existing graph theory to get our results. We also create a prototype in order to measure each organisation chart's efficiency. We then run the charts through the prototype, and compare the results. This method is a qualitative approach based on graph theory.

2 Preliminaries

Continuous use and dependence on networking increase the number of threat situations, which bring into focus security incident handling and their routines.

Security incident handling can be divided into several phases:

- detecting the threat
- locating the threat
- reporting
- responding to the threat
- forensic work
- countermeasures to avoid similar incidents in the future.

These phases can vary in organisations, depending on their structure, dimension and branch. Regardless of size, branch or organisational model, it is indeed of great importance to have safe routines for handling security incidents. There are a variety of routines and procedures that describe step by step how to proceed when a security incident occurs. In this chapter we describe literature and related work regarding efficiency of routines for handling security incidents and two chosen organisational models.

2.1 Exploring the routines for handling security incidents

Efficient routines for handling security incidents are important means for minimising the damage caused by security incidents. In order to have a better understanding of routines for handling security incidents we take a brief look at the main characteristics of these routines.

The employees' knowledge of these routines, their role, attitude and relation to them is important for the organisation's security and handling of security incidents. To assure the follow up of the security policy and also routines for handling security incidents, it is important to have a close relation to these routines. For instance, a weekly reminder, perhaps with information about the recent security incidents and their consequences etc., and parts of the security policies that can be used as countermeasures. To ease the responsibilities and work load for the management and make sure that these routines are followed, it would be a good idea to involve junior managers and other personnel in this process.

Strong security awareness of the employees leads to a better understanding of security issues and provides more security knowledge to be used in their daily work. This can decrease the number of successful attacks, ease the detection of threats and speed up the recovery work[21].

2.2 How to increase the efficiency

According to NIST's guideline to security incident handling[24], "organizations must create, provision, and operate a formal incident response capability". This guideline also describes in detail other measures of security incidents handling and how to organise the capacity of security incidents handling within an organisation[24].

When the elementary requirements of security measures defined in[25] are in place, an organisation is equipped with security policies, routines and procedures for obtaining security and safety in the daily operation. In order to maintain and achieve a higher capability in handling security incidents it is necessary to increase the efficiency of these routines.

2.3 Incident

An incident is defined in Information Security Management Handbook[9] as "any unexpected action that has an immediate or potential effect on the organisation".

Depending on the incident's character, the problem caused by the incident will vary based on how many people are affected and what is the consequence of the incident[9]. Depending on the impact caused by the incident's character an incident will belong to one of the following categories: global, regional and local[10].

An incident can be caused by authorized or unauthorized personnel, process, hardware or software. It can be an accident as well as a planned malicious action.

2.4 Security incident

A security incident is an action that endangers the safety and stability of an information system. There are several different definitions of security incidents; one is "A violation or imminent threat of violation of computer security policies, acceptable use policies, or standard computer security practices"[8], another definition describes the security incident as "any event that may threaten or compromise the security, operation or integrity of computing resources"[23]. In other words a security incident is a state of violation of security policy in an organisation and the security of their information system. However, it is not easy to invent only one multipurpose definition of security incident to cover all security incidents.

Security incident is a general concept that covers any kind of security violation regardless of location, the level of the threat or the extent of it. But the common known factors of Security Incidents are events and actions that jeopardise one or more basic elements of information security; Confidentiality, Integrity and Availability (CIA) of information systems.

Most of the definitions of security incident in the literature are mainly related to information systems and Information Technology(IT)/Information and Communication Technology(ICT) and are often referred to as IT-Security Incidents.

2.5 Routines for handling security incident

A variety of literature on Handbook for CSIRT[17], CSIRTs[17, 27] are available. These references are useful when an organisation has the basic routines based on a well defined and functioning security policy. What the literature does not offer in this stage is one or more sets of predefined routines for handling security incidents. This indicates the urgent need for routines for handling security incidents and the importance of including these as a primary part of any security policy within any organisation.

Routines for handling security incidents must be thoroughly considered, designed, implemented and tested. These routines must be pro-active in order to be able to go back to an early secure state for obtaining and maintaining the security in activities within the organisation.

Having proper routines for handling security incidents decreases the down time and limits the expenses caused by the security incident when it occurs.

2.5.1 Detecting The Threat

An important part of handling security incidents starts with detecting an incident. If a security incident is detected in an early phase, it is possible to minimise the damage caused by the threat and eliminate it before any further consequences. Detecting a threat before it occurs helps to reduce the vulnerability and increase ability for deteration of attacks. This involves identifying and characterising the threat and implement countermeasures to stop the attack or reduce its impact[5].

The literature often refers to detection as *detection of intrusion* which indicates lack of pro-activity in dealing with threats. It also describes that organisations are not adequately prepared to deal with intrusion before a security breach occurs. It is important to emphasise that the detection of a Security Incident does not necessarily need to be an intrusion. It could also be a weakness in the information system, hardware, software, security policy or other elements in the organisation.

2.5.2 Locating The Threat

In order to eliminate the threat when a security incident occurs, it is important to locate the threat fast, preferably before it causes any damage. There are several ways of detecting a threat which are described in literature on forensic. Locating a threat varies from case to case, depending on its nature, the infrastructure of information system and organisation etc.

2.5.3 Reporting

Reporting security incidents is a part of the routines for handling these incidents. Having routines for reporting, however, does not guarantee follow up, since it can cause negative focus on the employee who reports or other staff members. Based on this, there are many employees who deliberately have a second thought in reporting security incidents[3]. Lack of reporting causes less reliable statistical data of the actual extend of security incidents within organisations[2].

A technical report from Carnegie Mellon University gives guidelines on how to proceed when an incident occurs[16].

Adequate routines for reporting improves the efficiency within handling security incidents and employees' awareness of the advantages that come with these routines. Good routines focuses on "how to eliminate the threat" rather than focusing on who caused it in the first place.

These reporting routines must include enough data about the incident's origin like:

- what to report and who to report to
- when the incident occurred
- primary systems involved
- the type of attack
- the impact of it in the organisation.

Defining these elements, type of reports, how to use them and educating employees must be included in the security policy[12].

2.5.4 Response to the threat

In order to handle any security incident it is important to have skilled staff to recognise the threat, eliminate it and return the system to normal and ideal state. By defining procedures for handling security incident as part of the security policy, and responsibilities and roles as a part of routines, one can increase the efficiency and decrease the time for responding to security incidents and solving the problem[12].

Any reported security incident will be handled by a Security Incident Response Team. With focusing on responding to security incidents the literature often points to Computer Security Incident Response Team(CSIRT)[19].

The study of handling security incident response often advices to set up a team to assist in responding to security incidents. This team is referred to as CIRT (Computer Incident Response Team) or a CIRC (Computer Incident Response Center, Computer Incident Response Capability)[24].

How to set up a CIRT team depends on the organisations capacity and capability of handling security incidents. There are several approaches to set up such a team which is described in the literature. Richard L. Rollason-Reese gives a description of how security incidents can be dealt with by using an IRT unit (Incident Response Team)[19]. The literature also provides guidelines on response and recovery which describes how to reduce vulnerabilities, implement countermeasures, respond to an attack and reduce the impact of it[5].

2.5.5 Forensic

The literature often refers to forensic as computer forensic with many definitions. The concept of forensic which means "*bring to the court*" or "*usable as evidence in a court of law*" in relation with computer forensic focuses on violated data.(dates back to 1248 C.E.[22]).

CERT describes computer forensic as the identification and extraction of stored or recorded digital/electronic evidence[20]. Another definition describes the goal of computer forensic to recover, analyse and present computer based material in such a way that it is useful as evidence in a court of law[22].

A growing use of computers and computer crimes has invented the concept of computer forensic which has developed into a disciplined field with a variety of guidelines, checklists, routines, procedures, roles, tools and even laws. Many of these tools are open source and compatible with Unix/Linux and Microsoft operating systems, while others are license protected products. A report by International Data Corporation (IDC) indicates an investment of 1.45 billion US dollars in 2006 within security software in USA[22].

2.5.6 Summary of routines for handling security incident

The (sections 2.5.1 - 2.5.5) are important steps, which are to be taken in order to achieve high efficiency in handling security incidents. These sections describe how each one of them can contribute in both solving security incidents and increasing the robustness of an organisation against security threats.

In order to assure easier detection, prevention solving and reporting security threats we study the efficiency of two organisational models, namely hierarchical and matrix model, in responding to security incidents. The means to carry out this is to determine flow of information about security incidents. Exploring this procedure will decrease the time spent in solving security incidents and avoid major damage, which can be caused by lack of good security incidents routines. This way the organisation's capability of handling security incidents increases without any extra costs.

3 Evaluating efficiency of organisational models for security incidents handling

The main goal of this work is to compare efficiencies of two basic organisational models of security incidents handling - hierarchical and matrix model. Our idea is to express efficiency in the schemes of this kind by relating them to capacity. This is justified by the need to propagate information about security incident handling into the fastest possible way through the organisation. Experience from the companies, where very often a small number of persons have to solve the majority of security incidents handling, makes us think that in a better organisational model these tasks could be carried out in a more efficient way.

In order to present our security incident handling efficiency evaluation algorithm, we first define some basic concepts regarding organisational models and network flow.

3.1 Organisational models

Every entity is organised in an organisational structure. This structure is concerned with distribution of work and authority, and is a system for government, coordination and control of many tasks in the organisation. The organisational structure is often associated the organisational chart that describes how the organisation is built up. The boxes in the organisational chart indicate how positions and functions are distributed in the organisation, and reflect how the distribution of tasks has built an organisation with specialised units working with different tasks. The organisational structure also specifies how information flows through the organisation. A good system for exchanging information is crucial for having a strong working environment in the organisation[7]. In this thesis, we want to prove the same for the organisational model related to security incident handling and reporting.

An organisational structure has three effects on the employee's behaviour. These are:

- stability: each employee has specified tasks that are solved in a particular way.
- limitation: the structure also defines what an employee should *not* do.
- coordination: by coordinating several employee's tasks, one gets better results than each employee could produce individually.

Breaking organisational tasks into smaller processes and distributing these to different employees will cause a specialisation in the organisation. Each employee gets to concentrate on one single task and further develop his/her skills in this particular area. This way the job may be done faster and with a higher quality than if each employee gets a new/different task each time. On the other hand, this specialisation means more coordination work in order to get a satisfying total result.

Division of tasks can be done in two different ways:

- Function based: similar tasks are gathered in the same organisational unit. Fig. 1
- Market based: all tasks associated with the same product or area (either based on customer criteria or geography) are gathered in the same organisational unit. Fig. 2

Both of these divisions have advantages and disadvantages. The two methods could give a better result if they were combined, and some of the structures take this into account.

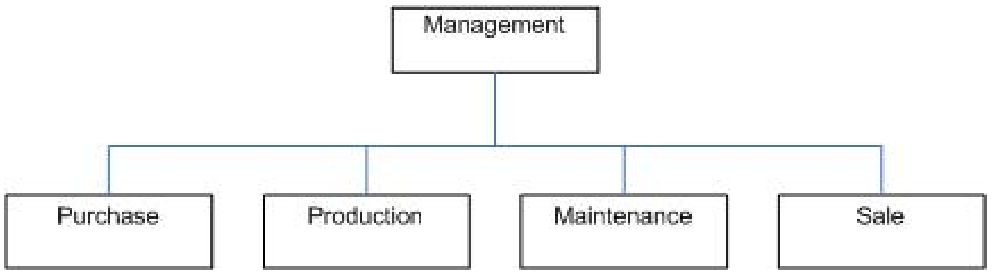


Figure 1: Function based organisation chart

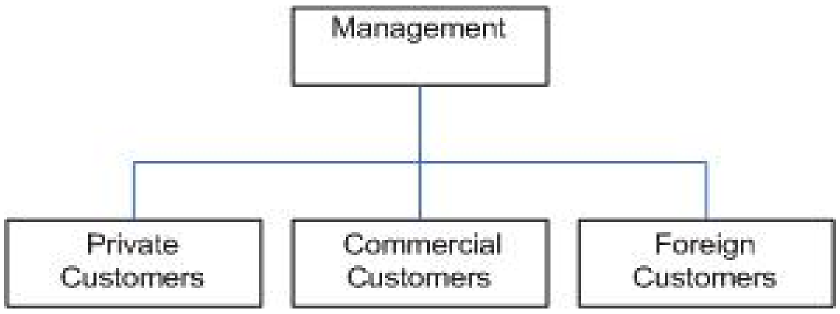


Figure 2: Market based organisation chart

An organisational structure can be placed into one of several organisational models. We will here describe two of these models.

3.1.1 Hierarchical model

The hierarchical model is the classical way of building up an organisation. It has a slight resemblance to a pyramid, with a top management and several levels of middle management[7].

The organisation structure is concerned with building up a hierarchy of positions. Personnel in higher positions have authority over employees in lower positions. This is a method in order to gain *direct control* of employees. The limitation in this model is how many employees you can put under one supervisor; how many people he/she can look after is limited.

The advantage with this model is that it is easy to give direct instructions and to get good feedback for control.

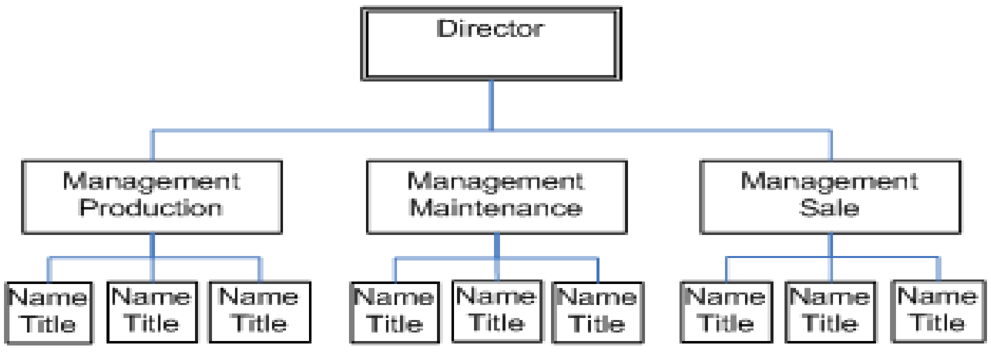


Figure 3: Hierarchical organisation chart

3.1.2 Matrix organisational model

A Matrix organisational model is a way of connecting a function based and a market based division. It is based on the fact that one values gathering similar tasks in the same unit on the same level as forming specific units to serve different products or market segments.

The advantages with a matrix structure is that employees get to explore more of their possibilities with both specialised tasks and more wide tasks, and that the organisation develops skills both regarding product areas and specific tasks. In a matrix structure, each employee has two superiors he or she needs to report to. This may cause a dilemma in how to balance his/her work between these two superiors.

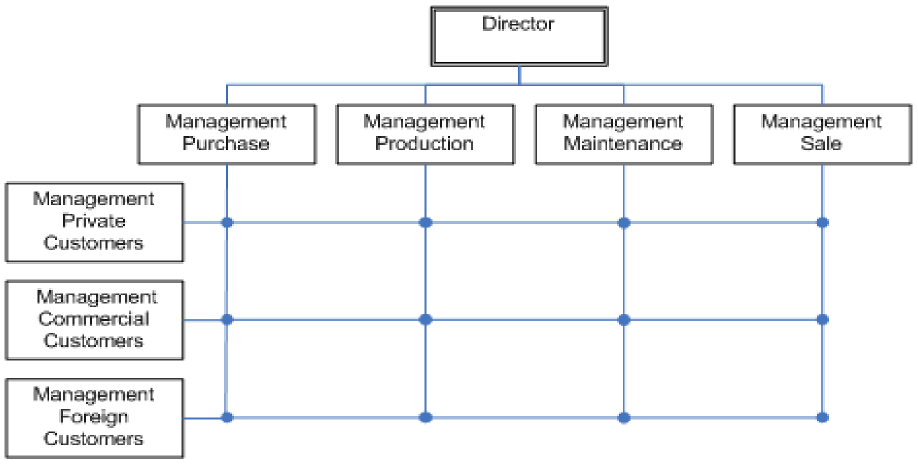

Director
Management Purchase
Management Production
Management Maintenance
Management Sale
Management Private Customers
Management Commercial Customers
Management Foreign Customers

Figure 4: Matrix organisation chart

The matrix model is a dynamic and flexible way to structure the organisation[7]. There exist two different kinds of matrix models[11]:

- the permanent matrix, divided in functional units and product/market segments.
- project matrix, one of the dimensions consists of the current projects.

3.2 Graph

We present here a brief description of graph, vertices and edges. A graph is a set of nodes that is connected by edges. A graph can be either directed or undirected. In a directed graph, also known as digraph, all edges are assigned an arrow which shows the flow direction. While an undirected graph, also known as bidirected, the flow direction is both ways. [1, 6, 18, 34]
Any graph consists of a set of nodes and edges. Neighbour nodes are connected by edges. We later consider nodes as employees, while edges describe both the nodes' capacity and the direction of solving and reporting security incidents.

A mathematical definition of a graph:
A graph $G = (V, E)$ is a mathematical structure consisting of two finite sets V and E. The elements of V are called vertices (or nodes), and the elements of E are called edges. Each edge has a set of one or two vertices associated to it, which are called its endpoints[15].

The Fig. 5 shows an example graph. The set of nodes $V = \{u, 1, 2, 3, 4, 5, 6, 7, 8, v\}$ and the set of edges $E = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n\}$. The graph shown here is an example of a directed graph, since all edges are pointing in the same direction and the flow is from u to v.

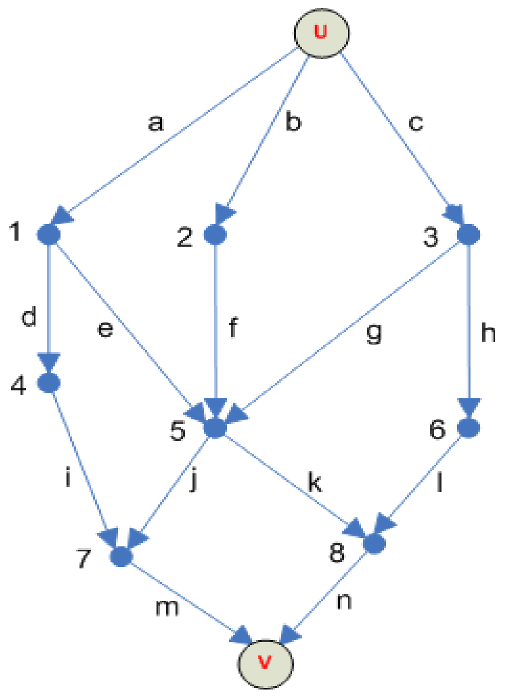


Figure 5: Example Graph

3.3 Karl Menger's theorem

Karl Menger's theorem from 1927, is one of the most important discoveries regarding edge-connectivity in finite undirected graphs. This theorem, which was later known as max flow - min cut theorem, is an important part of optimising network flows[29]. Since in this thesis we study flows in organisational models and try to opptimise them, it is important to quote the Menger's theorem here.

Menger's theorem:
Let u and v be distinct, non-adjacent vertices in a connected graph G. Then the maximum number of internally disjoint u-v paths in G equals the minimum number of vertices needed to separate u and v.[15]

Non-adjacent vertices are two nodes which are not connected to one another by only one edge. The minimum number of vertices needed to separate u and v is the set of vertices whose removal from the graph disconnect u and v. Internally disjoint paths are two paths with no internal edges in common.

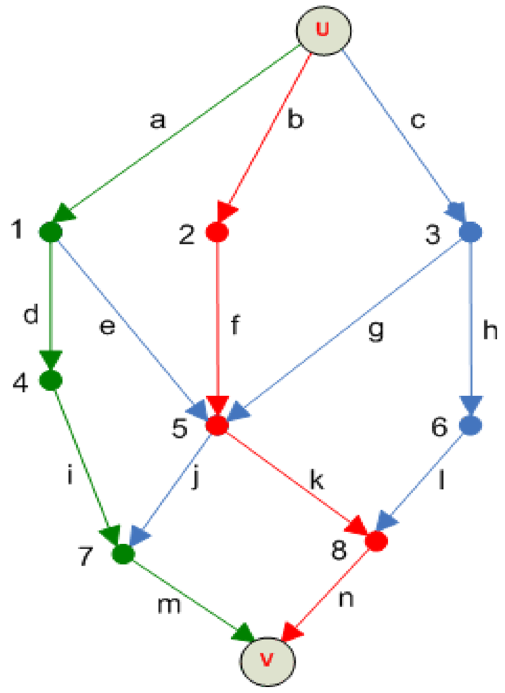


Figure 6: Example Graph - Menger's definition of a graph

The graph in Fig. 6 shows that vertex 1 is non-adjacent with vertices 3,6 and 8. In order to separate u from v one only need to remove two vertices, namely 7 and 8. According to the Menger's theorem this sample graph contains only two internally disjoint paths. These paths are indicated with green and red colours.

Menger's theorem can be interpreted in the network flow context in the following way:

Network flow theorem:
The maximum amount of flow in a network is equal to the capacity of a minimum cut.[28]

Converting the graph in Fig. 6 from Menger's point of view into a network context will cause some few changes in presenting the graph. This is shown in Fig. 7, where vertices u and v are replaced by s and t. Vertex s represents the source and vertex t represents the sink. The s *(source)* indicates the start of flow, and the t *(sink)* indicates the flow direction and last vertice in the graph. In addition one can see that edges are assigned positive integers, which describes the edge's capacity.

Fig. 7 shows two directed graphs consisting of 10 vertices and 14 edges each. In the graph on the left side, each edge is assigned a capacity as a positive integer. The capacity indicates the maximum flow through each edge, where the flow starts from the top vertex called s *(source)* and ends in the bottom vertex called t *(sink)*.

The graph on the right side of Fig. 7 demonstrates how to calculate the max flow of the graph. As it is shown in the graph on the right side, each edge has been assigned two values. The value on the right side of '/' is the original capacity as indicated in the graph on the left side, while the value on the left side of '/' indicates the used capacity of each edge.

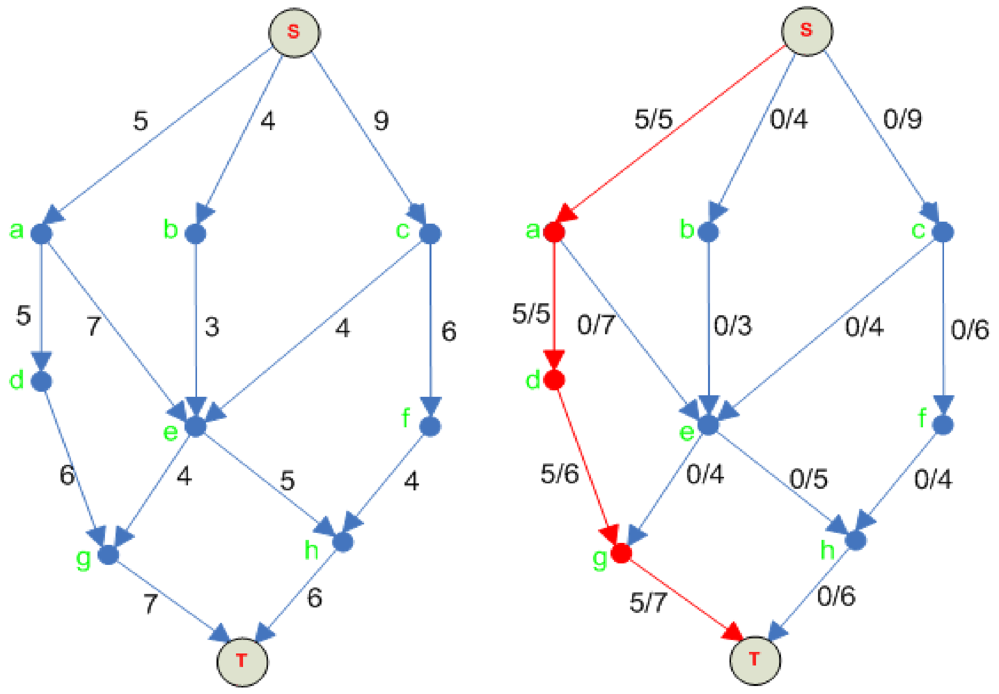


Figure 7: **Max flow - min cut**; left: capacities, right: one used path

As shown in the graph on the right side of Fig. 7, a path in calculating the max flow is found and marked red, and the used capacity of each edge in the path is indicated on the left side of '/'. The maximum flow capacity from s to t through this path is equal to 5. This value is equal to the lowest capacity of at least one of the edges within the path.

The process of finding a path from s to t which still has some capacity available continues. The found paths are marked red, and the value on the left side of '/' (used capacity) is updated. When it is not possible to send more through the graph, the calculation process stops. By now the max flow is equal to the total amount of flow from s to t as shown in table 1.

From node	**To node**	**Capacity**
S	a	5
a	d	5
d	g	5
g	T	5
S	b	3
b	e	3
e	h	3
h	T	3
S	c	2
c	e	2
e	g	2
g	T	2
S	c	2
c	e	2
e	h	2
h	T	2
S	c	1
c	f	1
f	h	1
h	T	1
		13

Table 1: Calculating all the possible paths and their capacity

The graph on the left side of Fig. 8 shows all the possible paths in calculating the max flow of the graph. The total flow capacity of this graph is equal to 13.

As shown in the graph, not all the edges' capacity is used in calculation of the max flow. In these cases, the values on the left side of '/' is lower than the value on the right side. The edge coloured with blue has an unused capacity of 7, which is not used at all. This is shown by the value on the left side of '/', which is 0.

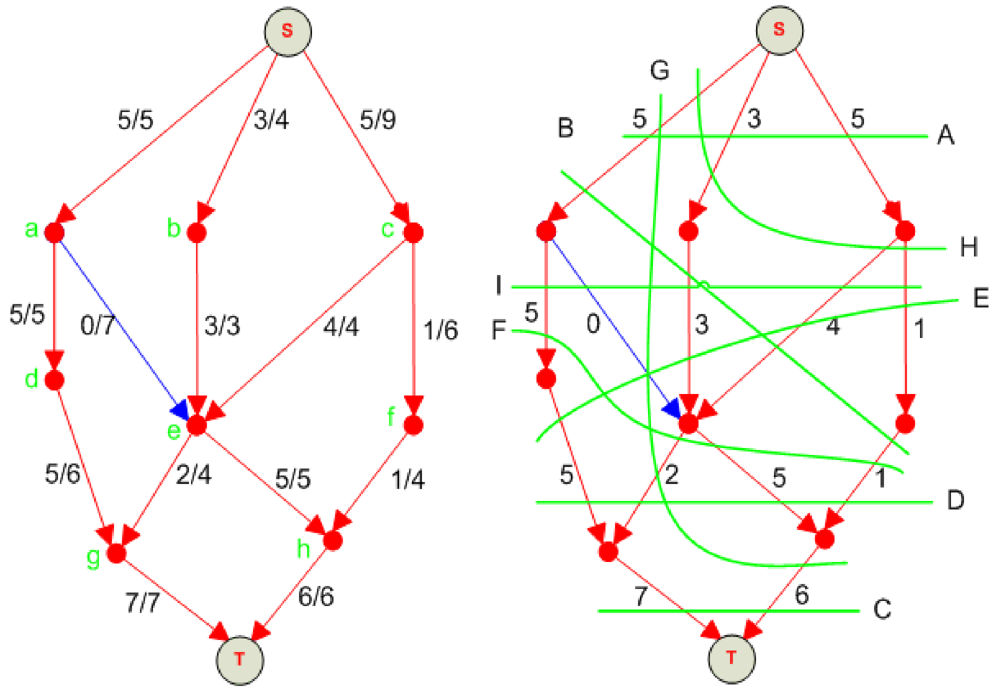


Figure 8: **Max flow - min cut**; left: computed max flow, right: all nine minimal cuts

As stated in the *network flow theorem* above, there is a relation between the maximum flow and the minimum cut. The value of maximum flow is equal to the value of minimum cut. A minimum cut of a network is defined as a way of dividing the graph into two subgraphs, where s and t are in different subgraphs, with the minimum capacity. In this graph there are nine ways to do this. These minimum cuts are indicated by the green lines in the graph on the right side of Fig. 8. The values in this graph indicate the used capacity of each edge (the value on the left side of '/' in the graph on the left side). By following one green line and adding the values of all edges the line crosses, one gets the value of the minimum cut, which is 13. This is also equal to the maximum flow of the graph.

3.4 The mathematical background of max flow - min cut

In this section we present a more detailed descriptions about the network flow, and the max flow - min cut problem. This gives a better mathematical understanding of some of the theory related to our work.

3.4.1 Flow in network

Flow in a network can be any kind of instance that is passed through the edges of the network. Some examples are water in a pipeline system, electricity in a power grid, phone calls through a phone net, packages transported on specified truck routes etc[15]. In our case, the flow in the network is information about security incidents through an organisation chart.

The capacity of an edge is the maximum amount of flow that can flow through the corresponding section per unit of time[15].

An organisational chart is a special kind of network, where all the flow goes from one source (the huge amount of incoming requests) that all end up at one source (the top manager). This kind of network is called a single source-single sink network.
The definition of a single source-single sink network is[15]:

- A single source-single sink network is a connected digraph that has i distinguished vertex called the source, with nonzero outdegree, and a distinguished vertex called the sink, with nonzero indegree.
- A single source-single sink network with source s and sink (or target) t is often referred to as an s-t network.

In a directed graph/network, the flow direction is specified. As an example, let e be an edge between the nodes a and b. The flow direction of e could then be from a to b, or from b to a. Any node in the graph will have at least one edge connected to it, with a flow direction either from or to the node. We can specify the sets of edges with a flow direction *from* and to the node, respectively, in the following way[15]:

Let v be a vertex in a graph N.

The out-set of v, denoted *Out(v)*, is the set of all edges that are directed *from* vertex v. That is,

$$\text{Out(v)} = \{\text{e} \in E_N | tail(e) = v\} \quad (3.1)$$

The in-set of v, denoted *In(v)*, is the set of all edges that are directed *to* vertex v. That is,

$$\text{In(v)} = \{\text{e} \in E_N | head(e) = v\} \quad (3.2)$$

3.4.2 Feasible flow

When we talk about flow, we mean the maximum amount that we could send along a path in a given time unit. This is defined as[15]:

Let N be a s-t network. A feasible flow f in N is a function $f = E_N \rightarrow R^+$ that assigns a non-negative real number $f(e)$ to each edge such that:

1. (capacity constraints) $f(e) \leq cap(e)$, for every edge e in network N
2. (conservation constraints) $\sum_{e \in In(v)} f(e) = \sum_{e \in Out(v)} f(e)$, for every vertex v in network N, other than the source s and sink t.

The first property states that the flow of the edge cannot be larger than the capacity of that edge, defined as $cap(e)$. The second property says that the total amount that comes into the node is the same amount that flows out of the node.

The value of flow f in a network, denoted $val(f)$, is the net flow leaving the source s, that is[15]:

$$val(f) = \sum_{e \in Out(s)} f(e) - \sum_{e \in In(s)} f(e) \qquad (3.3)$$

A *maximum flow* f^* in a network N is a flow in N having the maximum value, $val(f) \leq val(f^*)$, for every flow f in N[15].

3.4.3 Cuts

The *network flow theorem* (section 3.3) also describes *cut*. A cut is a removal of some of the edges in a s-t network. The definition of a cut is[15]:

Let N be a s-t network, and let V_s and V_t form a partition of V_N such that source $s \in V_s$ and sink $t \in V_t$. Then the set of all edges that are directed from a vertex in set V_s to a vertex in set V_t is called an s-t *cut* of network N and is denoted $\langle V_s, V_t \rangle$

The sets $Out(s)$ and $In(t)$ for an s-t network N are the s-t cuts $\langle \{s\}, V_N - \{s\} \rangle$ and $\langle V_N - \{t\}, \{t\} \rangle$, respectively. In Fig. 9, $Out(s) = \{a, b, c\}$ and $In(t) = \{m, n\}$.

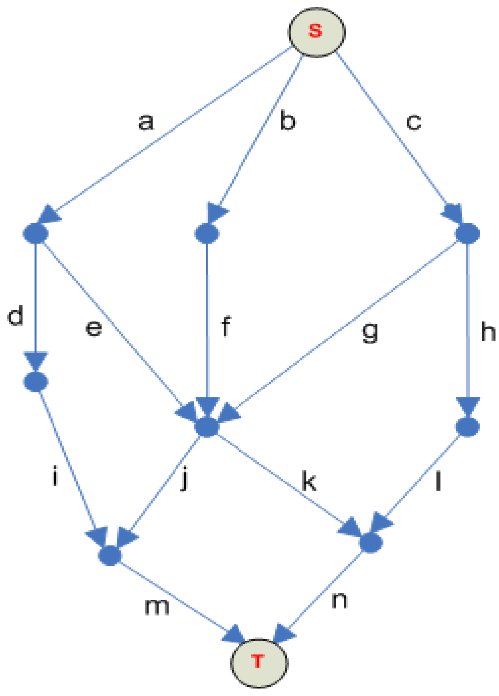


Figure 9: Cuts

3.4.4 Relationship between flows and cuts

The *network flow theorem* (section 3.3) states that there is a connection between flows and cuts. As we in the previous paragraph viewed the set $Out(s)$ of edges directed from source s as the s-t cut $\langle\{s\}, V_N - \{s\}\rangle$, we can view the set $In(s)$ as the set of "backwards" edges in this cut, namely $\langle V_N - \{s\}, \{s\}\rangle$. We may then rewrite the definition of the value of flow from equation 3.4.2 so that we get[15]:

$$\text{val(f)} = \sum_{e\in\langle\{s\},V_N-\{s\}\rangle} f(e) - \sum_{e\in\langle V_N-\{s\},\{s\}\rangle} f(e) \qquad (3.4)$$

Equation 3.4.4 states that the value of any flow equals the total flow across the edges of the cut $\langle\{s\}, V_N - \{s\}\rangle$ minus the flow across the edges of $\langle V_N - \{s\}, \{s\}\rangle$[15]. Now we want to generalise this to be used for all s-t cuts.

For any vertex $v \in V_s$, each edge directed from v is either in $\langle V_s, V_s\rangle$ or in $\langle V_s, V_t\rangle$. Similarly, each edge directed to v is either in $\langle V_s, V_s\rangle$ or in $\langle V_t, V_s\rangle$[15]. This can be illustrated by the example in Fig. 10, where we let node 1 act as v. We see that edge a in $\langle V_s, V_s\rangle$ is directed to node 1, and d and e in $\langle V_s, V_t\rangle$ is directed from node 1.

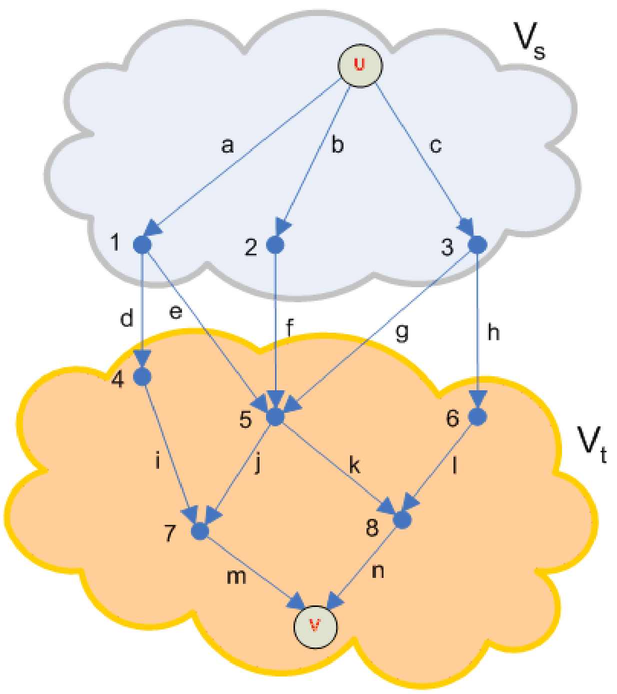


Figure 10: Graph subsets

If we move v around for all the nodes in V_s, we find the out-set for the partition V_s of the total network as[15]:

$$\bigcup_{v\in V_s} Out(v) = \langle V_s, V_s\rangle \cup \langle V_s, V_t\rangle \qquad (3.5)$$

and the in-set for the partition as[15]:

$$\bigcup_{v\in V_s} In(v) = \langle V_s, V_s\rangle \cup \langle V_t, V_s\rangle \qquad (3.6)$$

We want to use cut-terminology in the expression on how to calculate flow value. From equation 3.4.2 we have $val(f) = \sum_{e \in Out(s)} f(e) - \sum_{e \in In(s)} f(e)$. Since we know that the amount of flow that flows into a node is the same amount of flow that flows out from the node, we may state that $\sum_{e \in Out(v)} f(e) - \sum_{e \in Out(v)} f(e) = 0$. Based on this, we get[15]:

$$\begin{aligned} \text{val(f)} &= \sum_{v \in V_s} \left(\sum_{e \in Out(v)} f(e) - \sum_{e \in In(v)} f(e) \right) \\ &= \sum_{v \in V_s} \sum_{e \in Out(v)} f(e) - \sum_{v \in V_s} \sum_{e \in In(v)} f(e) \end{aligned} \tag{3.7}$$

If we now apply the equations in 3.4.4 and 3.4.4, the expression in 3.4.4 can be rewritten as[15]:

$$\begin{aligned} \text{val(f)} &= \left(\sum_{e \in \langle V_s, V_s \rangle} f(e) + \sum_{e \in \langle V_s, V_t \rangle} f(e) \right) \\ &- \left(\sum_{e \in \langle V_s, V_s \rangle} f(e) + \sum_{e \in \langle V_t, V_s \rangle} f(e) \right) \\ &= \sum_{e \in \langle V_s, V_t \rangle} f(e) - \sum_{e \in \langle V_t, V_s \rangle} f(e) \end{aligned} \tag{3.8}$$

To illustrate this, we use the graph in Fig. 11. Each edge is assigned two values, the first indicates the flow that is sent through it, and the second value indicates the capacity. As we see, we have five edges that goes from V_s to V_t. These are the edges from a to d, a to e, b to e, c to e and c to f. The value of flow in the cut in this example is $(5 + 0 + 3 + 4 + 1) - (0) = 13$.

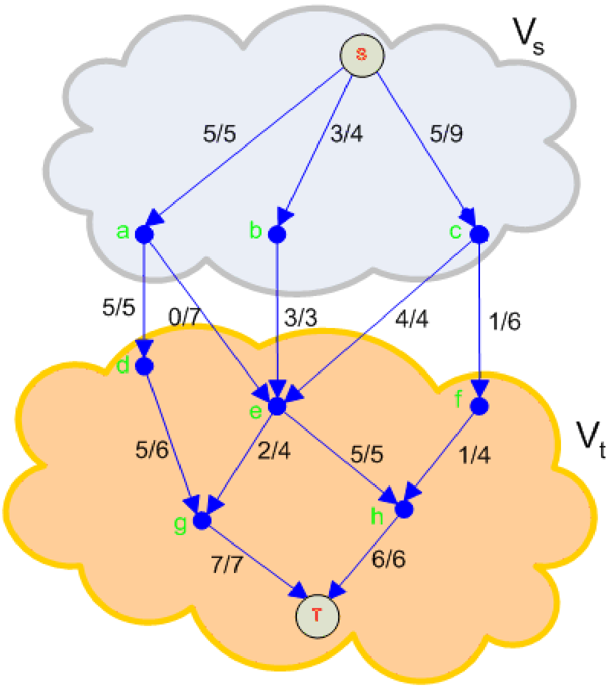


Figure 11: Cut capacity

We know that no node can store any amount of flow, just pass it further on, and no flow disappears on the way. This leads to the fact that in a s-t network, where there is only one source s and one sink t, the flow out of s is equal to the flow into t[15].

The *capacity of a cut* $\langle V_s, V_t \rangle$, denoted $cap \langle V_s, V_t \rangle$, is the sum of the capacities of the edges in cut $\langle V_s, V_t \rangle$[15]:

$$\mathrm{cap}\langle V_s, V_t \rangle = \textstyle\sum_{e \in \langle V_s, V_t \rangle} cap(e) \quad (3.9)$$

A *minimum cut* of a network N is a cut with the minimum capacity.

3.4.5 The maximum-flow and the minimum-cut problem

The problems of finding the maximum flow in a network N and finding a minimum cut in N are closely related. The maximum flow through *one* edge, is its capacity. When we try to find the maximum flow through a path (a number of edges), the expression is a bit more complicated, but the capacity is still the upper bound. From equation 3.4.4 we may derive the following expression[15]:

$$\begin{aligned} \mathrm{val(f)} &= \textstyle\sum_{e \in \langle V_s, V_t \rangle} f(e) - \sum_{e \in \langle V_t, V_s \rangle} f(e) \\ &\leq \textstyle\sum_{e \in \langle V_s, V_t \rangle} cap(e) - \sum_{e \in \langle V_t, V_s \rangle} f(e) \\ &= \mathrm{cap}\langle V_s, V_t \rangle - \textstyle\sum_{e \in \langle V_t, V_s \rangle} f(e) \\ &\leq \mathrm{cap}\langle V_s, V_t \rangle \end{aligned} \qquad (3.10)$$

If we let f^* be a maximum flow in a s-t network N, and K^* be a minimum s-t cut in N, then we get[15]:

$$\mathrm{val}(f^*) \leq cap(K^*) \quad (3.11)$$

Suppose we have a minimal cut K of a network N, and a flow f'. From equation 3.4.5 we see that if $f' \equiv cap(K)$, f' is the maximum flow. In the same way we can find the value of the minimum cut, when we know the max flow[15].

3.5 Algorithms for computing the max flow

In order to calculate the efficiency of handling security incidents, we need to use a suitable algorithm. There are many different algorithms, but we will closely examine three of them, respectively Ford-Fulkerson's, Edmond-Karp's and Relabel-to-front, since we consider them the most relevant for solving our problem.

Here we give a definition of three expressions used in these algorithms[1].

- Residual capacity: residual capacities is a way to represent the amount of capacity not consumed by a flow. In other words, residual capacity is the difference between the amout of the flow currently assigned to the path between u and v and the capacity of the path.
- Residual nerwork: a graph $G(V, E)$ with residual capacities $c_f > 0$. Consequently, edges in a residual network can sustain more flow.
- Augumenting path: any path from s to t in G.

3.5.1 Ford-Fulkerson's algorithm

Ford-Fulkerson's algorithm which is the first algorithm based on the edge-connectivity version of Menger's theorem later known as max flow - min cut, is used to solve the maximum flow problem in a network. The basic idea of this algorithm is to increase the flow in the network iteratively until it cannot be increased any further[15]. This algorithm uses three fundamental factors which are:

1. Augmenting paths, which describes the remaining usable paths from s to t.
2. Residual networks, which describes the remaining flow capacity in the network.
3. Cuts, a split of the nodes into two sets where the s and t are located in different sets (section 3.4.3).

Ford-Fulkerson's algorithm[28] (see appendix 1) calculates the maximum flow within a network. The algorithm searches for augmenting path through the network, recursively, from *s(source)* to *t(sink)*.

By adding the capacity of new augmenting paths, to already found path(s), it recursively calculates the maximum flow capacity in the network. The operation continues until no more augmenting paths are found. This algorithm is quite fast and requires little processor power. However, its drawback is that it sometimes does not terminate, but this cases are rare.

3.5.2 Edmonds-Karp algorithm

The Edmonds-Karp algorithm was first proposed in 1970's by a Russian scientist Dinic, and two years later the algorithm was once more published by Edmonds and Karp individually. The algorithm is almost similar to Ford-Fulkerson algorithm, except that the order of traversing the augmenting paths is predefined[31].

The algorithm always starts with the shortest path, and the next one and so on. After calculating each flow, the residual graph changes and the new shortest path will be selected until the max flow is reached (see appendix 2).

This algorithm, along with relabel-to-front 3.5.3, is independent of the max flow which distinguishes them from the original Ford-Fulkerson's algorithm (section 3.5.1). Although Edmonds-Karp's algorithm can be slower than relabel-to-front algorithm[31].

3.5.3 Relabel-to-front

This algorithm[32] has two types of operations called push and relabel. The flow is from U to V.

Push operation and its conditions:

The push operation forwards the incoming flow from(U) to outgoing flow(V) depends on three conditions.

1. More incoming flow than outgoing flow to the same node.
2. Available capacity through the same node.
3. The flow direction must be towards the *sink(t)*. Height (U) > Height(V)

Relabel operation and its conditions:

Relabeling node U means increasing its height until it is higher than at least one of the nodes that it has capacity to.

1. The first condition want more incoming flow than outgoing flow,
there must be a point in relabeling!
2. The second condition expects remaining node with capacity which
have a higher level than U.

The initial relabeling value is 0, but the new value used to relabel U is the lowest value which is larger than height(V). This still assumes available capacity from U to V (see appendix 3).

3.5.4 Comparing algorithms

A directed graph consist of n vertices, and has maximum $(n * n - 1)$ edges[30]. The density of the graph determines which algorithm is fastest. The sample graph in Fig. 12 is used to demonstrate the efficiency of these three algorithms. This graph consists of 10 vertices (V), 14 edges (E). The integer numbers indicates the edges' capacity, which is needed to calculate the efficiency of Ford-Fulkerson's algorithm. For calculation of efficiency of remaining algorithms, only the number of edges and vertices is needed.

As the results of calculation below shows, the Ford-Fulkerson's algorithm is the most efficient algorithm to be used for this sample graph. However if the capacity of edges increases by multiplying them with 10, total capacity 130, one can see this algorithm will be the second fastest.

1. Ford-Fulkerson: $O(E * f) = (14 * 13) = 182$
2. Edmonds-Karp: $O(V * E^2) = (10 * 14^2) = 5600$
3. Relabel-to-front: $O(V^3) = (10^3) = 1000$

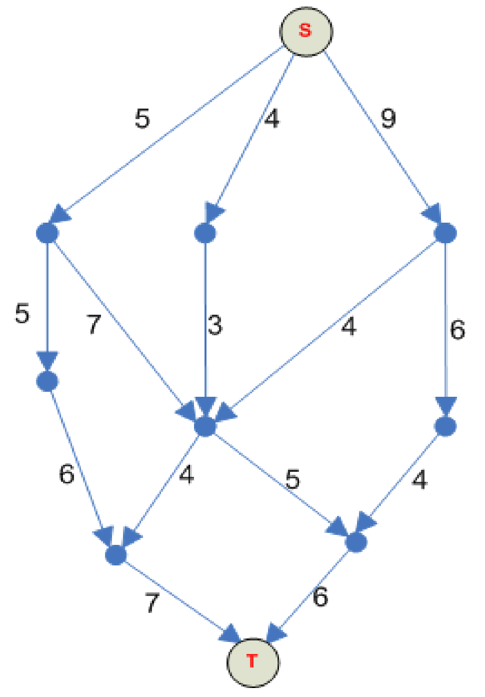


Figure 12: Sample graph: O estimates

A vital part of this work is based on the prototype which is used to demonstrate the effect of implementing an algorithm for measuring the efficiency of maximum flow within a network. Later will this prototype helps us to measure the efficiency of handling security incidents in organisation. The choice of algorithm depends on several factors, for instance the time used for implementing the algorithm, the algorithm's efficiency, it's capability and other advantages and disadvantages of the algorithm.

The comparison of the three chosen algorithms (section 3.5.4) demonstrates the efficiency of each algorithm. The efficiency of each algorithm will differ when changing the parameters in our example.

Since our primary focus is on the implementation of the algorithm we have chosen to implement the Ford-Fulkerson's algorithm. The advantages of this algorithm is the simplicity during the implementation, (section 3.5.4), and the high speed of the algorithm which requires little processor power.

The only drawback of Ford-Fulkerson's algorithm is the insignificant probability of not returning a value which means not being able to calculate the flow capacity (section 3.5.1). In our implementation, we took this possibility into account and made sure that the prototype never goes to eternal loop or crashes.

3.6 Implementation of Ford-Fulkerson's algorithm

Simplicity and effectiveness of this algorithm distinguishes it from the two other mentioned algorithms. This algorithm mainly consists of two parts. The first part recursively searches for flow augmenting paths and labels them, and the second part changes the flow value along the way. If there is no augmenting path there will be no flow capacity.

Since we always operate with a positive integer as capacity value, this algorithm will never run forever even when no augmenting path is found and the maximum flow is equal to zero. In this case the algorithm and the prototype will terminate after all possible iterations since the number of iteration is $O(E * f)$, where E is number of edges and f is max flow capacity.

3.7 Measuring the readiness of handling security incidents

The capacity of handling security incidents in any organisation depends on among others, the organisation's infrastructure, security policy, routines for handling security incidents and the extent of security incidents.

By knowing the number of employees involved in handling security incidents and each employee's capacity as a model for his/her capability of handling security incidents, we can make an organisation chart. This information can easily be used to measure the total capacity of handling security incidents.

One challenge in any organisation is being dynamic when changes within the organisation are made. Personnel are moved, new personnel are employed and new divisions are established. Such a change often causes some time needed to organise routines and assign roles and responsibilities to personnel which can decrease the efficiency of handling security incidents as well. In this case the challenge of measuring the organisation's capability of handling security incidents grows. Our algorithm should than be executed upon each change in the organisation.

3.8 Developing the prototype

In this section we cover measurement of the capability of handling security incident, computation of the possible maximum flow and demonstrate the efficiency of the Ford-Fulkerson's algorithm by developing our prototype.

Information gathered and labeled as an organisation chart in the previous section will be useful input in our prototype. By mapping and gathering information about the personnel handling Security Incidents and converting it into proper format, we constitute the necessary foundation which is needed for our prototype.

By using the C# (C sharp) programming language, we develop a prototype which runs in Microsoft Windows XP environment.

The main purpose of the prototype is achieved by implementing Ford-Fulkerson's algorithm for calculating the maximum flow in a network.

Our prototype includes some graphical elements to ease the visualisation. This way the results of displaying augmenting path, the flow path and computing the maximum flow is displayed in a more user friendly fashion.

3.9 The prototype

The prototype is basically meant to demonstrate the effect of implementing the Ford-Fulkerson's algorithm in handling security incident routines.

3.9.1 The input file

Information gathered about the number of employees handling security incidents and each employee's work capacity (section 3.7) will be stored in a text file with this format. An example of the input file is presented in table 2.

Nodes	**Edges**	
6	10	
From Node	**To Node**	**Capacity**
0	1	16
0	2	13
1	2	10
2	1	4
3	2	9
1	3	12
2	4	14
4	3	7
3	5	20
4	5	4

Table 2: Nodes, edges and their capacities

The content of the input file will be used to generate a graph consisting of nodes and edges. The file consists of two parts. The first line contains two integers describing the number of *nodes* and *edges*. The rest of the lines each contain three integers separated by a separator character, the values *from node*, *to node* and *capacity* in between these nodes.

The lowest node value *0* indicates the *s(source)* and the highest node value which is equivalent with the number of nodes-1, in this case *5* indicates the *t(sink)*.

The number of nodes corresponds to the number of employees handling security incidents and the number of edges corresponds with the number of connections between employees. The value of each edge is equivalent with the employee's work capacity. For example, capacity (C) from node *(x)* to node *(y)* $\Leftrightarrow C(x, y) \Rightarrow C(4, 5) = 4$.

3.9.2 Computing the max flow

In order to compute the max flow, these steps will be followed:

Step 1: Reading the graph input data file.

Step 2: Putting the value of number of nodes and edges into the variables *NoOfNodes* and *NoOfEdges*, respectively.

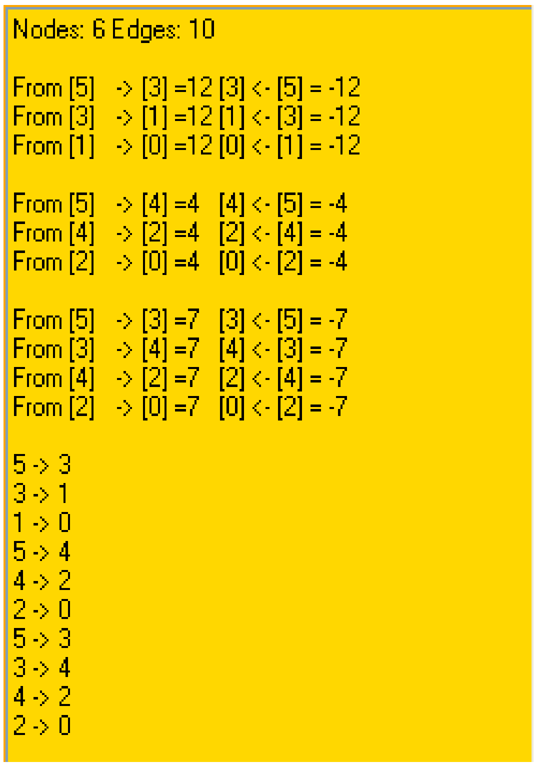
Step 3: Traversing through the read lines and placing the read values, each line consisting of three integer values, into the capacity array which is a two dimensional array of integer type.

Step 4: Traversing through the capacity array, while there still exists an augmenting path. When finding an augmenting path from *s(source)* to *t(sink)* then the capacity of the path will be added to the value *max_flow* (see appendix 4).

Step 5: When the calculation of Maximum Flow is done, the result will be presented in both text format and a simple graphic format.

3.9.3 The result of computing max flow

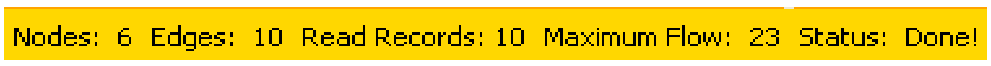
After running the max flow algorithm, the result stored in the FinalFlow-array and MaxFlow-variable will be displayed as text output on the left side of the prototype. The displayed result in Fig. 13 describes the number of nodes and edges, all possible augmenting paths and each path's capacity.



```
Nodes: 6 Edges: 10

From [5]   -> [3] =12 [3] <- [5] = -12
From [3]   -> [1] =12 [1] <- [3] = -12
From [1]   -> [0] =12 [0] <- [1] = -12

From [5]   -> [4] =4   [4] <- [5] = -4
From [4]   -> [2] =4   [2] <- [4] = -4
From [2]   -> [0] =4   [0] <- [2] = -4

From [5]   -> [3] =7   [3] <- [5] = -7
From [3]   -> [4] =7   [4] <- [3] = -7
From [4]   -> [2] =7   [2] <- [4] = -7
From [2]   -> [0] =7   [0] <- [2] = -7

5 -> 3
3 -> 1
1 -> 0
5 -> 4
4 -> 2
2 -> 0
5 -> 3
3 -> 4
4 -> 2
2 -> 0
```

Figure 13: Sample text output result

Each augmenting path contains at least one line describing the *from* node, *to* node and the belonging capacity. The sum of capacities of all augmenting paths are the final max flow, as shown in Fig 14.



Nodes: 6 Edges: 10 Read Records: 10 Maximum Flow: 23 Status: Done!

Figure 14: Displaying the Maximum Flow

3.9.4 Graphical presentation

The results presented as text output in Fig. 13 did not make a visual impression of connectivity within a graph, and required some degree of imagination and understanding to see the real result. To ease the presentation and visualization of the result a graphic environment was added to the prototype.

Fig. 15 presents the graph generated as a result of computation by the algorithm shown in Fig. 13. This graph consist of six nodes, ten initial connection between the nodes and three sets of augmenting paths from *source(s)* to *sink(t)*. Node number (0) represents the *source(s)* and node number (5) represents the *sink(t)*. White dotted arrows represent the initial connections between the nodes (not all of these connections are used in the computation of the Maximum-flow). The coloured arrows are used to show augmenting paths from *source(s)* to *sink(t)*.

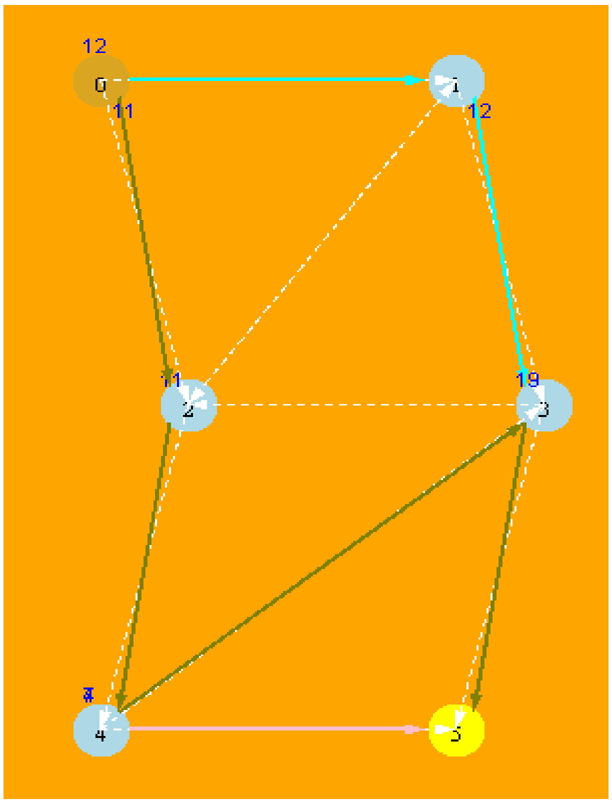


Figure 15: Sample graphic output result

4 The experiment

In this master thesis we aim to find out if we can measure the efficiency of routines for handling security incidents, and how to increase the efficiency of the routines. In order to answer these questions, we developed a software prototype, which was used to measure the efficiency of flow in different organisational models.

We want to find a way to improve the routines for handling security incidents in two different organisational models, regardless of the branch, public or private. In general our focus is the structure of the organisation. In this case we have chosen to take a closer look at the hierarchical and matrix structures, as the structure met in the practice most often.

4.1 Reliability and validity

In order to achieve two important aspects of any project, reliability and validity, we designed six organisational charts; three charts for each organisational structure. Each pair of organisational charts consists of the exact same number of employees and same total capacities. This way we can see the efficiency of each organisational model with the same size and compare the results of the outcome for our prototype.

Testing six anonymous organisation charts does not demand any specific needs for confidentiality. In this case we do not have any concern of keeping any information confidential.

The result of this master thesis is based on the output from our prototype. The prototype uses data files to calculate the max flow for each organisation chart. For this reason we carefully audit the number of nodes, value of each capacity and total capacity, for each organisation chart in addition to the content of each datafile for belonging organisation chart and comparing these values. This measure was taken to assure the reliability of the input data and the validity of the outcome of our research.

4.2 Organisation charts

We have created six organisation charts, three for each organisational model: The hierarchic model Fig . 16, 17 and 18, and the matrix model Fig. 19, 20 and 21.

The hierarchical organisation chart Fig. 16 corresponds to the matrix organisation chart in Fig. 19 since the number of employees (12), total capacity (110) and the number of users requests are identical. The total capacity is the number of tasks/reports the organisation can perform for one period of time, for instance one day.

The capacities for the management in the next six organisation charts represents the number of tasks they can solve. This capacity is considerably lower than their reporting capacity.

In order to be able to measure the efficiency within our six organisation charts and compare the efficiency between hierarchical and matrix organisation we also need to have some number of user requests. The number of user's requests are equal for each organisation chart with the same number of employees and capacity. These user's requests are only a theoretical value that can ease the process of measuring and comparing the efficiency in between the chosen organisational models.

The difference between the number of edges in these organisation charts is caused by the structure of each organisational model. In our examples, we assume all requests always start from the bottom node with the highest node number.

In the next step we assign capacities to all nodes in all organisation charts. Despite of different number of edges in these organisation charts, the total capacities in each organisation chart is exactly the same. This way we can assure the validity of our experiment.

The final step before testing is to create one data file for each organisation chart, each containing information about the number of nodes, edges, the capacity between two nodes and the number of user's requests. These files will be used as input files in our prototype.

In all organisation charts, we assume that the tasks only come from the outside of the organisation, which is a box labeled *Users incoming requests*. This is the *source(s)*, which is assigned the highest node number. The *sink(t)*, which is numbered 0, indicates the top management.

4.2.1 Hierarchical structure

The flow direction in the next three hierarchical organisation charts are only upward, from the node with the highest node number to the lowest node number. Each employee takes care of the same number of tasks corresponding to its own capacity.

The Fig. 16 below demonstrates our first hierarchical organisation chart with one top management, two managers and nine employees. Their total capacity, for one day, is equal to 110. Each employee is marked as a *node* with a node number and belonging capacity. The node number 12 represents both the user's requests flow direction and total number of user's requests. The total number of user's requests for this organisation chart is equal to 300 for one period of time.

The next Fig. 17 is our second hierarchical organisation chart with one top management, three managers and twelve employees. Their total capacity, for one day, is equal to 165. The node number 16 represents the user's requests which is equal to 400 requests per day.

The next Fig. 18 is our third hierarchical organisation chart with one top management, four managers and sixteen employees. Their total capacity is equal to 265. The node number 25 represents the user's requests which is equal to 500 requests a day.

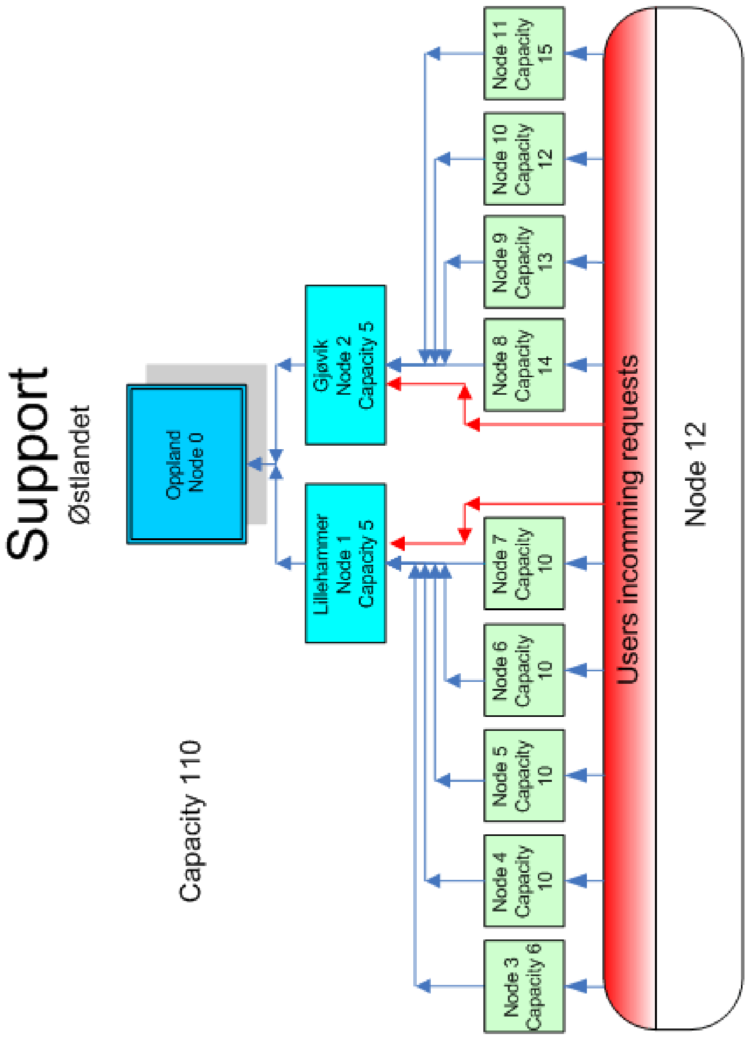


Figure 16: 1. Sample hierarchical organisation chart

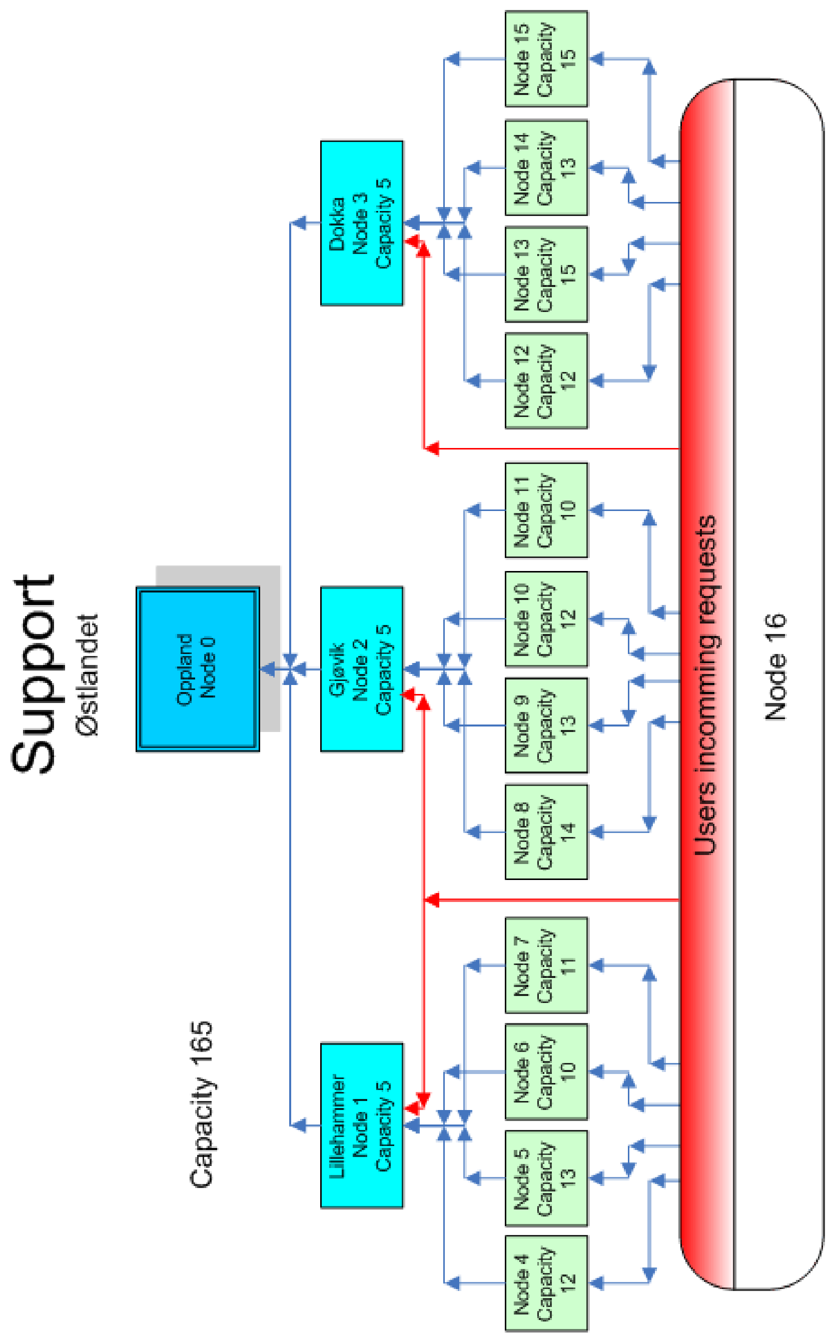


Figure 17: 2. Sample hierarchical organisation chart

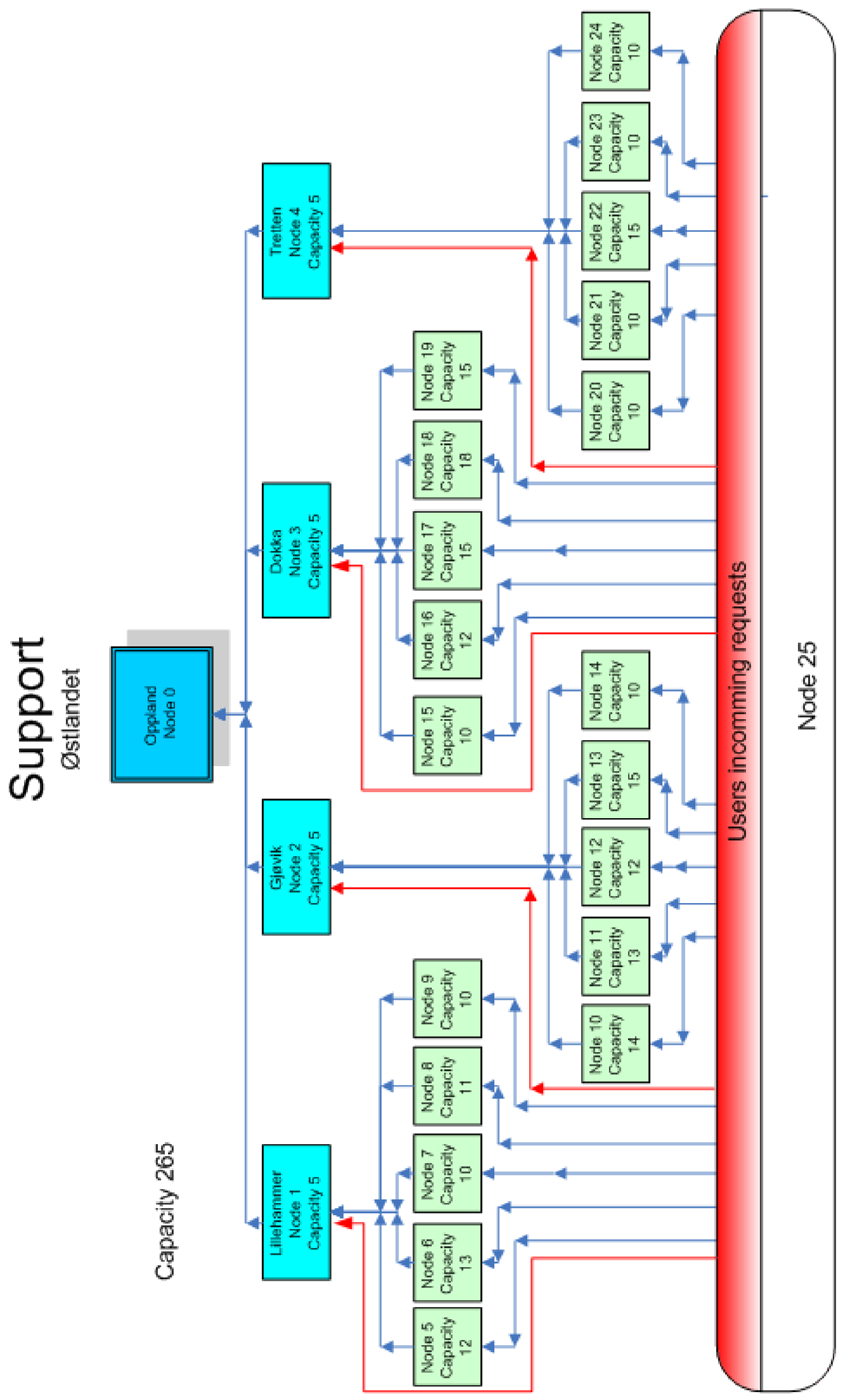


Figure 18: 3. Sample hierarchical organisation chart

4.2.2 Matrix structure

The flow direction in the next three matrix organisation charts, starts only upward but continues both vertically and horizontally towards left, since each employee reports to two managers and managers are placed both on the top and the left side of the employees in these organisation charts.

The Fig. 19 below demonstrates our first matrix organisation chart with one top management, and five managers divided into two groups. One regional management and the other section management. And six employees which each reports to two managers. The total capacity of this organisation is equal to 110.

The digit 2 on the arrows from the *Users incoming requests* indicates one connection to each of the upper four nodes, one connection to each node.

The next Fig. 20 is our second matrix organisation chart with one top management, and six managers divided into two groups. One regional management and the other section management. And nine employees which each reports to two managers. The total capacity of this organisation is equal to 165.

The digit 3 on the arrows from the *Users incoming requests* indicates one connection to each of the upper five nodes. One connection to each node.

The next Fig. 21 is our third matrix organisation chart with one top management, and eight managers divided into two groups. One regional management and the other section management. And sixteen employees which each reports to two managers. The total capacity of this organisation is equal to 265.

The digit 4 on the arrows from the *Users incoming requests* indicates one connection to each of the upper six nodes. One connection to each node.

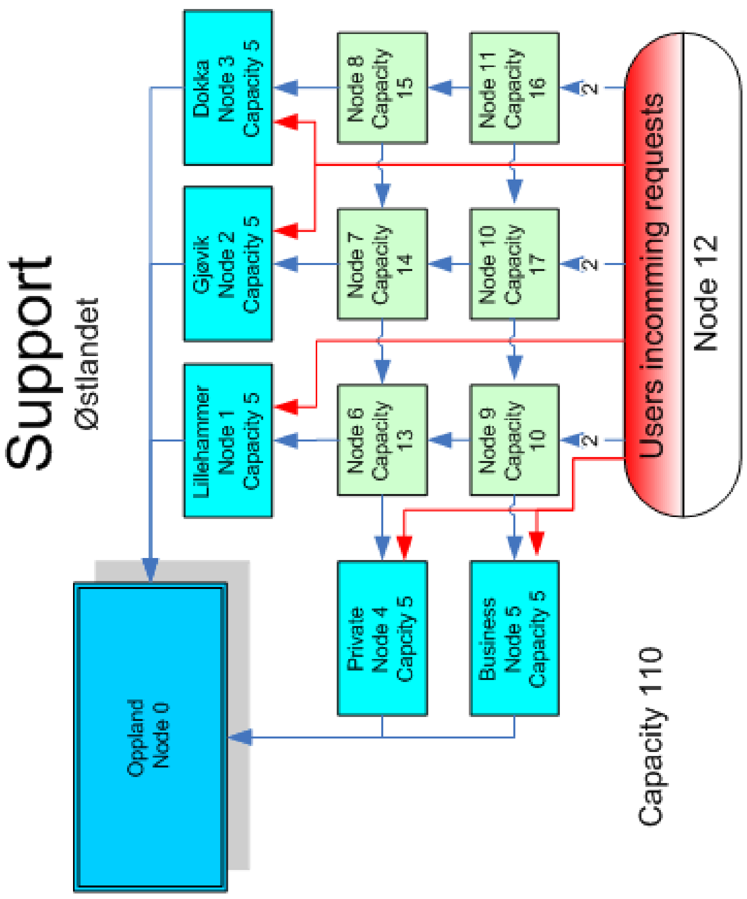


Figure 19: 1. Sample matrix organisation chart

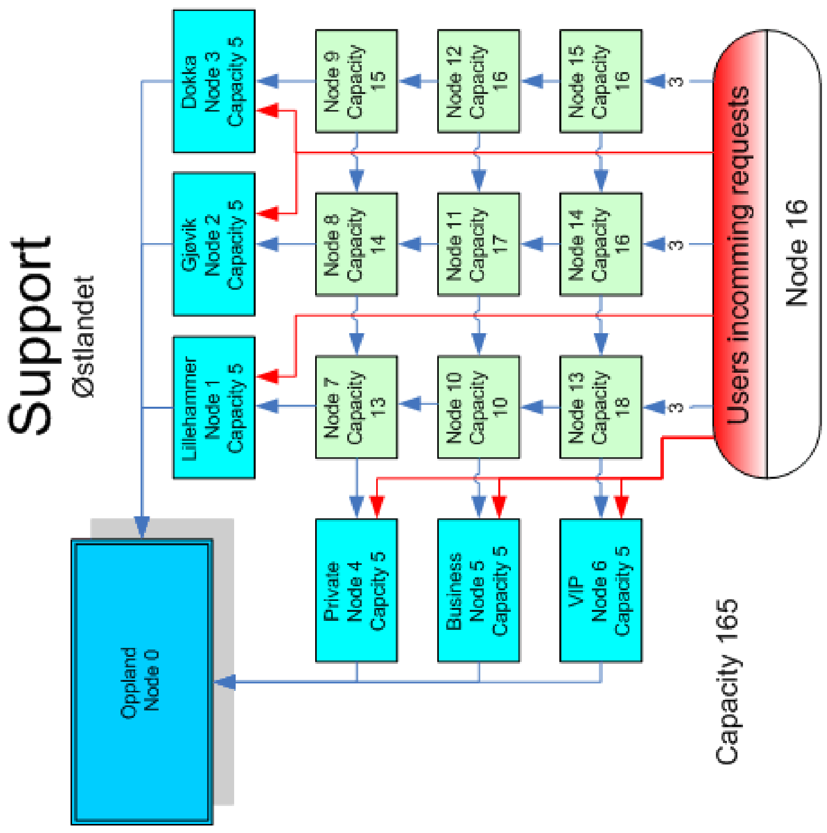


Figure 20: 2. Sample matrix organisation chart

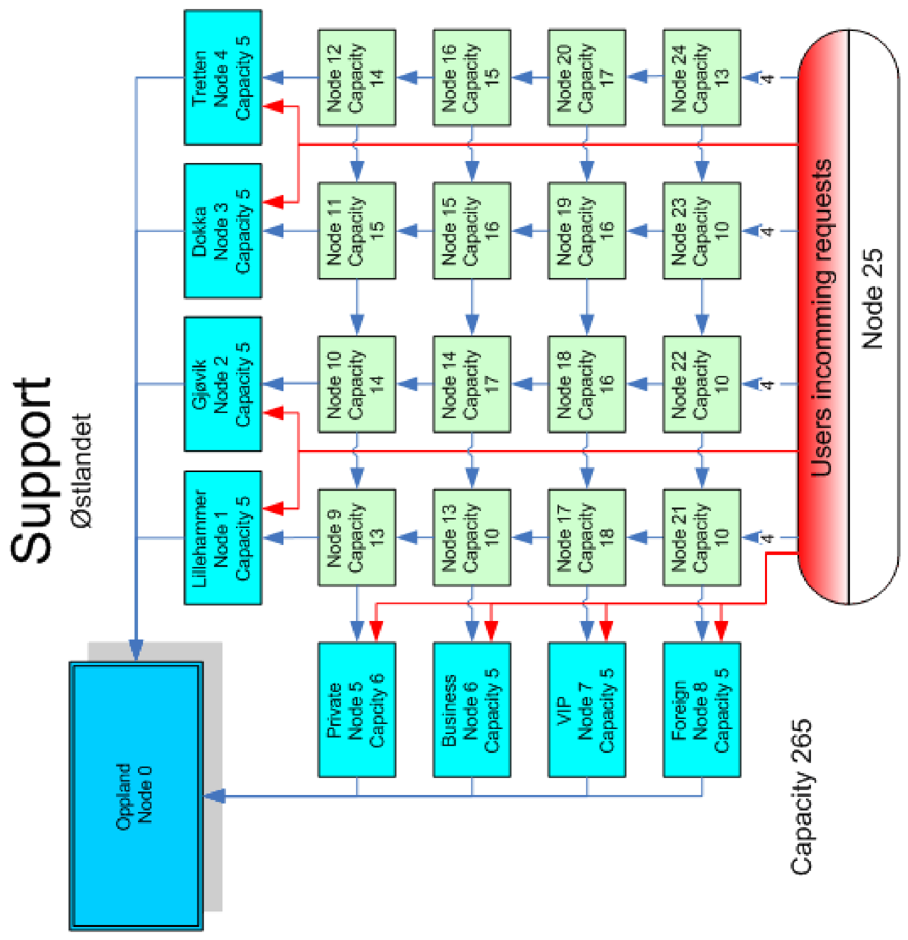


Figure 21: 3. Sample matrix organisation chart

4.3 The result of our experimental work

In this section we demonstrate the outcome of our prototype, the max flow in each organisation chart. The max flow is calculated twice for each organisation chart. We measure two different max flows in each organisation, one for reported and solved security incidents and one for reported but not solved security incident. This is because the reporting capacity at management level is considerably higher than solving security incident capacity. Therefor we will have twelve different max flow values.

First of all, we measure the max flow in solving user's requests in each organisation chart. In this case the capacities for managers are quite low. This low number combined with Ford-Fulkerson's algorithm will give a low max flow because the value of max flow is not larger than the value of min cut (section 3.5.1).

Secondly we will measure the max flow of reporting for each organisation chart. In this case we assume that the managers have a higher capacity to handle and forward reports. This will cause a higher value of max flow for each organisation chart.

In addition we will compare the achieved results for each organisational model. Since each pair of our organisation charts consist of the same number of employees, same total capacity and same number of user requests, we will compare these organisation charts with one another. This way we can see if one of the organisation charts has a higher max flow and if different size of organisation gives a different result.

Finally we will explore the possibilities of preferring one organisational model with a better performance.

4.3.1 Relation between nodes and edges

Because of the nature of our chosen organisational models, hierarchy and matrix, we will have different number of edges in organisation charts with the same number of node and capacity. This difference is easier to see in Fig. 22.

Fig. 22 represents the relation between the number of nodes and the number of edges in each organisation. Since we have two organisation charts of each organisational model, with the same number of nodes, we have categorised them into three categories, based on number of nodes. The edges indicator for each organisation chart with the same number of nodes are tied together. This way we can clearly see the differences between the number of nodes and number of edges.

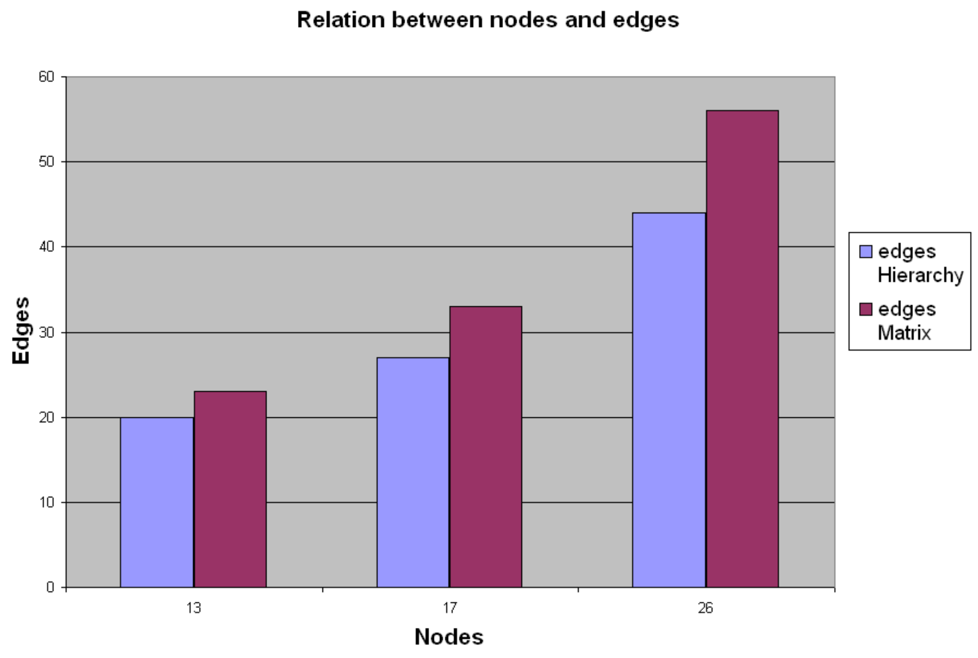


Figure 22: Relation between nodes and edges

We can easily observe the increasing number of edges in all matrix organisation charts. By increasing the number of edges in between nodes, it is possible to achieve a higher efficiency, which results in a larger max flow.

In the organisation charts with 13 nodes, the hierarchy chart has 20 edges, while the matrix chart has 23. In the organisation with 17 nodes, the hierarchy chart has 27 edges, while the matrix chart has 33. Finally, in the organisation with 26 nodes, the hierarchy chart has 44 edges, while the matrix chart has 56.

The difference between the number of edges increases as the organisations grow bigger. In the smallest organisations, the matrix organisation chart has 15 % more edges than the hierarchy organisation chart. The next matrix organisation chart has 22 % more edges than the hierarchy organisation chart. The largest matrix organisation chart has 27 % more edges than the hierarchy organisation chart of the same size.

By using our prototype and each organisation chart's input data file, we have calculated 12 max flow values, both for processed tasks and for reporting.

4.3.2 Tasks: Max flow - Hierarchical structure

Fig. 23 demonstrates the value of max flow for processed tasks in our three hierarchic organisation charts. We can see that the number of edges in these organisation charts are about twice as large as their max flow.

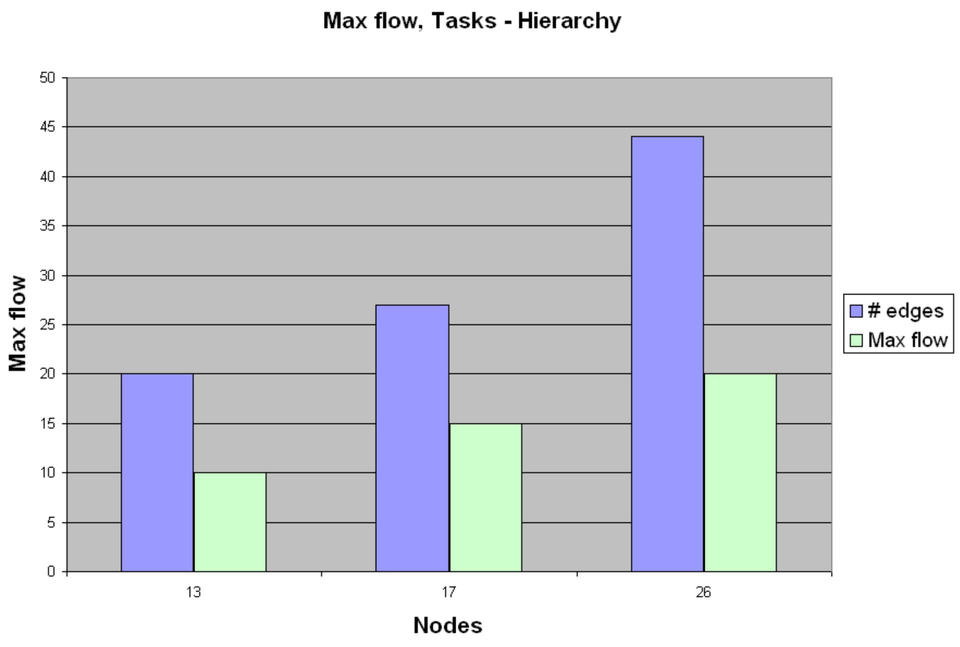


Figure 23: Max flow, tasks - Hierarchy

The relation between the number of edges and the max flow in the smallest hierarchic organisation chart is twice as big. In the nest hierarchic organisation chart, the number of edges is 80 % bigger than the max flow. While in our largest hierarchic organisation chart, the number of edges are 120 % bigger than the max flow.

The result and relation between the number of edges and the value of max flow in these three cases does not apply to all hierarchic organisations. We can test this theory by decreasing the capacity of lower nodes (employees), and increase the capacity of the higher nodes (managers), as shown later (section 4.3.4). It is obvious that increasing size of min cut increases the max flow.

4.3.3 Task: Max flow - Matrix structure

Fig. 24 demonstrates the value of max flow for processed tasks in our three matrix organisation charts. In our examples of matrix organisation, we can see that the gap between the number of edges and the max flow value of each organisation chart is considerably smaller than in the hierarchic case.

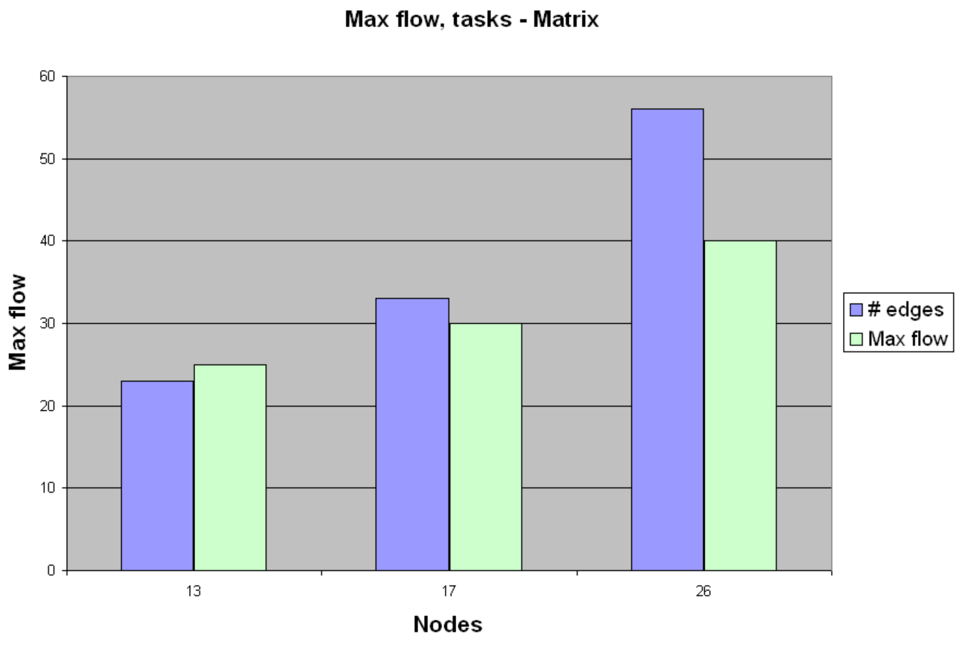


Figure 24: Max flow, tasks - Matrix

In general, we can see that these organisation charts have a bigger max flow than the hierarchic organisation chart, even though the relation between max flow and edges is not consistent.

In the smallest matrix organisation, the max flow is 9 % bigger than the number of edges. But in the next two matrix organisation charts, the number of edges is higher than the max flow, respectively 10 % and 40 % bigger.

4.3.4 Report: Max flow - Hierarchical structure

Fig. 25 demonstrates the value of max flow for reported solved tasks in our three hierarchic organisation charts. We assume that each lower level employee's capacity of reporting is equal to their capacity of processing tasks. The reason for substantial larger value of max flow here is caused by the manager's capacity of reporting. Their reporting capacity is several times larger than their problem solving capacities. For this reason we have increased all managers reporting capacities in both organisation charts.

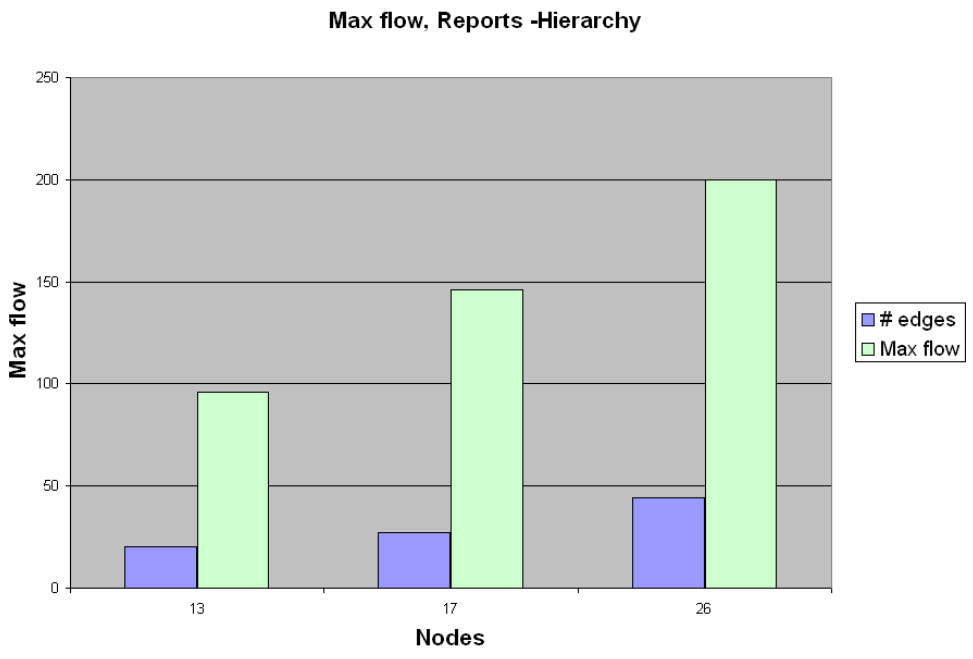


Figure 25: Max flow, reports - Hierarchy

As we see from Fig. 25, the max flows are in this case substantially higher than the number of edges. In the smallest hierarchic organisation charts, the max flow is 380 % bigger. In the next hierarchic organisation chart, the max flow is 440 % higher. Finally, the max flow of the biggest hierarchic organisation chart is 350 % higher than the number of edges.

4.3.5 Report: Max flow - Matrix structure

Fig. 26 demonstrates the value of max flow for reported solved tasks in our three matrix organisation charts. We have also in this case assigned a higher value of capacity to managers in all matrix organisation charts. The increased reporting capacity of managers cause substantial increase in max flow for reporting. Once more we can clearly see not only a higher max flow but several times more than hierarchic organisation charts.

Higher number of edges along with higher reporting capacity at management level (in addition to the nature/structure of matrix organisation) cause the big gap between the value of max flow in hierarchic and matrix organisations.

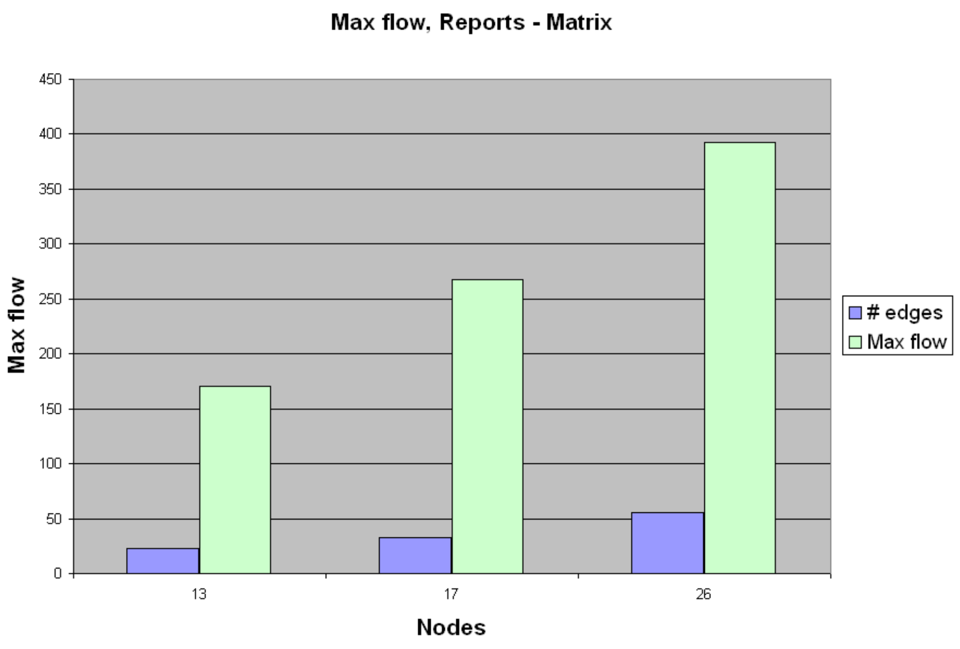


Figure 26: Max flow, reports - Matrix

Also in this case, the max flow is considerably higher than the number of edges. In the smallest matrix organisation chart, the max flow is 640 % higher than the number of edges. In the next two matrix organisation charts, the max flow is 710 % and 600 % higher than the number of edges, respectively.

5 Discussion

In this chapter, we will discuss the results from our experiment. These results are limited to our test data for the six organisation charts. It is possible to achieve other results by manipulating one or more parameter(s) like number of node, numbers of edges, node capacity or total capacity for each organisation. For this reason it is important to emphasise the relation between our organisation charts, their data files and the achieved results.

Our tests produced several results in different contexts. Depending on the nature of organisation size and it's capacity we can see the max flow of handling security incidents in each organisation. These results indicates which organisation chart and organisational model are more cost effective in handling security incident.

5.1 Comparison of max flow for solved security incidents

The Fig. 27 demonstrates the max flow for solved incidents in our six organisation charts. The max flow in each organisation chart with the same number of nodes are placed together to ease the comparison in between them. All bars on the left side of each pair indicates the max flows in hierarchic organisation charts and all bars on the right side of each pair indicate the max flow for matrix organisation charts.

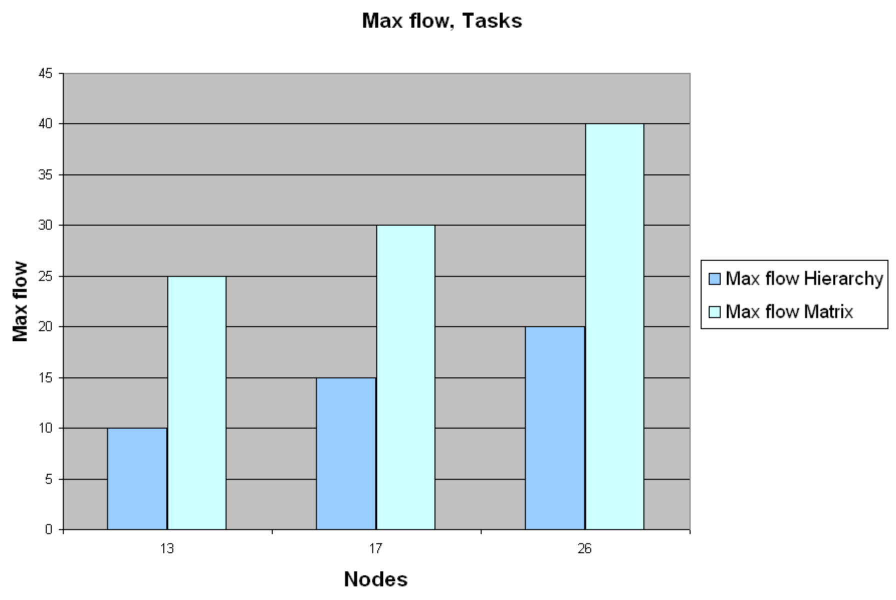


Figure 27: Max flow, Tasks

We see that regardless of size of the organisation, the matrix charts get a substantially higher max flow. The max flow for the smallest matrix organisation chart is 150 % bigger than the belonging hierarchic chart. In the next two organisation charts, the matrix organisation chart is exactly twice as big as the hierachic organisation charts.

We can also see that the smallest matrix organisation chart, with a lot less total capacity of 110, achieves a higher max flow than the largest hierarchical organisation chart with a total capacity of 265.

As shown in Fig. 27 the smallest matrix organisation chart have only 13 employees, while the largest hierarchical organisation chart have 26 employees, twice as much. Yet, the smallest matrix chart achieves a better performance, and therefore has a higher efficiency in handling security incidents. Having less number of employees and a low total capacity in an organisation does not necessarily equals to lower performance by any means. As the Fig. 27 shows, one can clearly see the proof of this claim.

What is important is to look for a better way of organising the resources in an organisation, rather than how big an organisation is or can be. In other words, the efficiency in an organisation is not dependent on the number of employees, but how to organise them, and which kind of organisation structure should be used? Fig. 27 shows that an organisation with matrix structure achieves 25 % higher efficiency in handling security incidents than an organisation with the hierarchical structure and twice as much employees (26). These results show that the matrix structure has definitively a better capability in handling security incidents than the hierarchical structure.

5.2 Comparison of max flow for reported security incidents

The Fig. 28 demonstrates the max flow for solved and reported incidents in our six organisation charts. The notable difference between this Fig. 28 and Fig. 27 is mainly caused by increased reporting capacity at manager level, since the managers can process more reports than solving incidents.

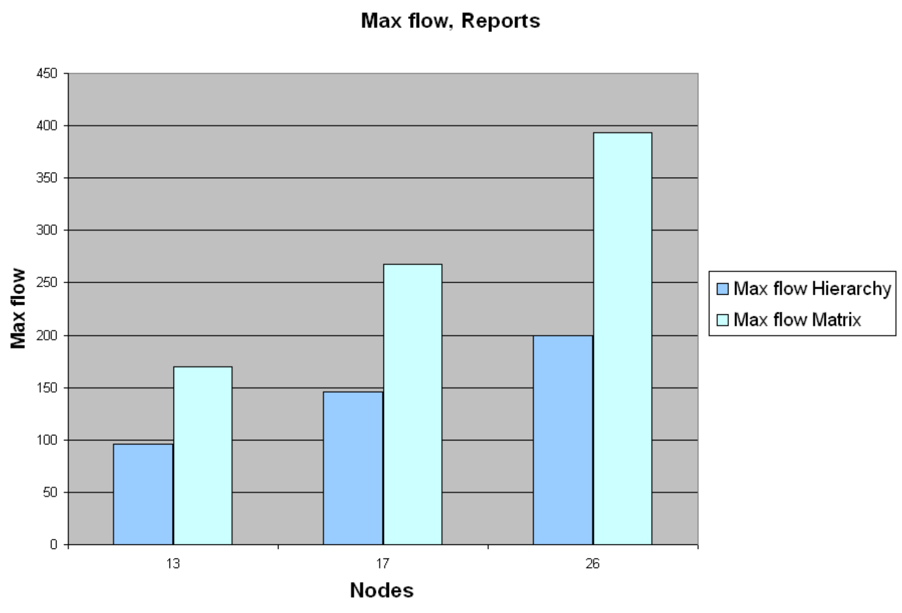


Figure 28: Max flow, Reports

Also in this case, the matrix organisation charts get a higher max flow than the hierarchic organisation charts. In the smallest organisation chart, the matrix max flow is 77 % higher than the hierarchic. For the next two organisation charts, the matrix max flow is 83 % and 96 % higher, respectively.

Once more, the smallest matrix organisation chart, with a total capacity of 110, has a max flow close to the max flow of the largest hierarchic organisation chart, which has a total capacity of 265.

5.3 Summary

Based on these results we can clearly see each organisational model's efficiency and see that one distinguishes from the other. Having these results, the process of choosing and recommending the organisational model with a higher max flow capability is a trivial task.

Having obtained these results, we will recommend the matrix organisational model rather than the hierarchical organisational model. In order to approve the validity of this experiment in other contexts, it is indeed necessary to collect information about existing matrix and hierarchic organisations, make data files for each one and run the prototype. This way one can compare our and new results to see if matrix organisational model still distinguishes from hierarchical organsational model.

If the results achieved by running existing organisations through our prototype distinguishes the matrix organisational model, then it is natural to see how to reorganise the hierarchical structure into matrix structure. Reorganising any organisation is an expensive process, that is why it is important to calculate the total costs, if possible, of such a change.

6 Conclusion

Handling security incidents is a part of daily life of most organisations, but how effectively they handle these increasing incidents has not been studied very often so far. This master thesis shows one way of computing the efficiency of handling security incidents routines in two different organisational models, namely hierarchical model and matrix model. However it will not be easy to implement this theoretical method in practice. We found out that the efficiency in handling security incidents does not depend on the number of employees, but rather how to organise them in which organisational model.

Our first research question was to find a way to measure the efficiency of routines for handling security incidents in two organisational models.

In order to find an answer to this research question we used our prototype and ran with each organisation charts' data file (12 files). The prototype computed each organisation charts' efficiency in handling security incidents. We achieved two sets of results in measuring the efficiency of routines for security incidents handling in these organisations. The first set of results showed us the efficiency in routines for solving security incidents, and the second set of results showed us the efficiency in routines for reporting security incidents.

Our prototype and these results helped us to measure the efficiency of routines for handling security incidents in both organisational models, hierarchical and matrix. We found that the matrix organisational model are more efficient organisational model than the hierarchical model, both in solving and reporting security incident.

Achieved results of our prototype demonstrate the difference between these two organisational models' capability in handling security incidents, both in solving and reporting security incidents. These results distinguishes, without any doubt, the matrix organisational model to be the most efficient organisational model in this context and thereby the best organisational model in handling security incidents of the two studied ones. Based on these results, one can see that the advantages of using a matrix organisational model is far more than hierarchical model, even though the matrix model is a more complex organisation to set up. In other words, increasing the efficiency of routines for handling security incidents does not depend on the organisations' size, but rather the organisations' model. By reorganising a hierarchical organisation into a matrix organisation one can achieve a higher efficiency in routines for handling security incidents.

7 Further Work

In our thesis we have based our test on organisation charts with no relation to the real world. The next step will naturally be to target some existing organisations, both based on hierarchic and matrix, collect data about them, make data files and run trough the protocol, as mentioned (section 5.3).

This way one can, not only see how the prototype works and measure the efficiency in other organisations, but also see if the matrix structure still distinguishes from the hierarchic structure.

The next step will be to see if:

- it is practically possible to reorganise an organisation
- it is financially possible to do so
- it is professionally possible.

These are only a few aspects to consider. It would also be interesting to see if one could use this concept in measuring the efficiency in other organisational models, for instance network structure.

It is also interesting to see if this prototype can be used to calculate max flow in larger dynamic organisations with rapidly changing structure.

Bibliography

[1] Berge Claude. *Graghs and hypergraphs / translated by Edward Minieka*. Fagbokforlaget, 2 edition, 1976.

[2] CERT Coordination Centre publiserer statistikk over innrapporterte sikkerhetshendelser. Incidents reported. *Electronical version on: http://www.cert.org/stats/#incidents*, 2004.

[3] CERT CSO Magazine and Carnegie Mellon University Software Engineering Institute. 2004 E-CRIME WATCHTM SURVEY SHOWS SIGNIFICANT INCREASE IN ELECTRONIC CRIMES. *Electronical version on: http://www.csoonline.com/releases/ecrimewatch04.pdf*, 2003.

[4] Longman Dictionairy. Longman Dictionairy of Contemporary English Online. http://www.idoceonline.com.

[5] Eric A. Fischer. Creating a National Framework for Cybersecurity: An Analysis of Issues and Options. *Electronical version on: http://www.acm.org/usacm/PDF/CRS_cybersec.pdf*, 2005, February.

[6] Harary Frank. *Graph Theory*. Addison-Wesley, 1969.

[7] Ulla-Britt Waagaard Frøydis M. Petersen. Mellomlederens rolle i fag- og forskningsbibliotek. http://www.rbt.no/abmu/publisert/Skrifter/skrift71/nr710012.htm, 1994.

[8] Tim Grance Karen Kent Brian Kim. Computer Security Incident Handling Guide Recommendations of the National Institute of Standards and Technology NIST800-61. *Electronical version on: http://csrc.nist.gov/publications/nistpubs/800-61/sp800-61.pdf*, 2004.

[9] CISSP Harold F. Tipton, CISSP Micki Krause, editor. *Information Security Management handbook*. Aurebach Publication, 2002.

[10] CISSP Harold F. Tipton, CISSP Micki Krause, editor. *Information Security Management handbook Page 1853-1854*. Aurebach Publication, 2002.

[11] Høgskolen i Vestfold. Høringsnotat Organisasjonsgjennomgangen. *http://www.hive.no/adm/info/hve2003/gruppe6/ouhoringsutkast.pdf*, 2002.

[12] Nebraska Information Technology Commission. Incident Response and Reporting Procedure for State Government. *Electronical version on: http://www.nitc.state.ne.us/standards/security/ Incident_Reporting_Procedure_20020618.pdf*, 2002, April 5.

[13] Dag Ingvar Jacobsen og Jan Thorsvik. "*Hvordan organisasjoner fungerer - Innføring i organisasjon og ledelse*". Fagbokforlaget, 2 edition, 2002.

[14] Thomas A. Longstaff John D. Howard. A Common Language for Computer Security Incidents. *Electronical version on: http://www.cert.org/research/taxonomy$_9$88667*.pdf.

[15] Jay Yellen Jonathan L. Gross. *Graph Theory and ITS APPLICATIONS, second edition*. Chapman & Hall/CRC, 2006.

[16] Georgia Killcrece Klaus-Peter Kossakowski Robin Ruefle Mark Zajicek. State of the Practice of Computer Security Incident Response Teams (CSIRTs). *Electronical version on: http://www.cert.org/archive/pdf/03tr001.pdf*, 2003, October.

[17] Georgia Killcrece Klaus-Peter Kossakowski Robin Ruefle Mark Zajicek. Organizational Models for Computer Security Incident Response Teams (CSIRTs). *Electronical version on: http://www.cert.org/archive/pdf/03hb001.pdf*, December 2003.

[18] Bernhard Korte and Jens Vygen. *Combinatorial Optimization - Theory and Algorithm*. Springer, 3 edition, 2006.

[19] Tore Larsen Orderløkken. Security Incident handling and reporting. Master's thesis, Gjøvik University College, Department of Computer Science and Media Technology, 2005.

[20] Chris May Marie Baker Derek Gabbard Travis Good Galen Grimes Mark Holmgren Richard Nolan Robert Nowak Sean Pennline. Advanced Information Assurance Handbook. *Electronical version on: http://www.cert.org/archive/pdf/aia-handbook.pdf*, page 190, 2004, March.

[21] Carnegie Mellon University. CSIRT Services. *Electronical version on: http://www.cert.org/archive/pdf/CSIRT-services-list.pdf*, 2002.

[22] Richard Nolan Colin OSullivan Jake Branson Cal Waits. First Responders Guide to Computer Forensics CERT Training and Education. *Electronical version on: http://www.cert.org/archive/pdf/FRGCF_v1.3.pdf*, 2005, March.

[23] Richard L. Rollason Reese. Incident handling: an orderly response to unexpected events. *September 2003 Proceedings of the 31st annual ACM SIGUCCS conference on User services Electronical version on: http://delivery.acm.org/10.1145/950000/947496/p97-rollasonreese.pdf*, 2003.

[24] Brian Kim Tim Grance, Karen Kent. Computer Security Incident Handling Guide ,Recommendations of the National Institute of Standards and Technology NIST800-61. *Electronical version on: http://csrc.nist.gov/publications/nistpubs/800-61/sp800-61.pdf*, 2004, January.

[25] Dr. Udo Helmbrecht. The IT Baseline Protection Manual describe the elementary. *Electronical version on: http://www.bsi.bund.de/english/gshb/manual/download/modules.pdf*, 2004.

[26] Oxford University. *The Oxford Reference Dictionary*. Oxford University Press, 1986.

[27] Johannes Wiik and Jose J. Gonzalez. Limits to Effectiveness in Computer Security Incident Response Teams. Electronical version on: http://www.cert.org/archive/pdf/Limits-to-CSIRT-Effectiveness.pdf.

[28] unknown Wikipedia. Ford-Fulkerson's algorithm: Max-flow min-cut theorem. http://en.wikipedia.org/wiki/Max-flow_min-cut_theorem.

[29] unknown Wikipedia. Karl Menger's theorem. http://en.wikipedia.org/wiki/Menger

[30] unknown Wikipedia. Dense graph. *http://en.wikipedia.org/wiki/Dense_graph*, 2006.

[31] unknown Wikipedia. Edmonds-Karp algorithm. *http://en.wikipedia.org/wiki/Edmonds-Karp_algorithm*, 2006.

[32] unknown Wikipedia. Relabel-to-front algorithm. *http://en.wikipedia.org/wiki/Relabel-to-front_algorithm*, 2006.

[33] unknown Wikipedia. Ford Fulkerson's algorithm. *http://en.wikipedia.org/wiki/Ford-Fulkerson_algorithm*, 2006, 15 February.

[34] Øystein Ore. *Graphs and their uses*. Mathematical Association of America, 1963.

Appendix 1 - Ford-Fulkerson's algorithm

```
Implemented in Fortran:

let G be the input graph.
Initialize an array f such that f[e] = 0 for all edges e in G

While there exists a source - > sink path in the residual graph
    choose some such path p
    augment(f, p)
    update residual graph
    augment(array f, path p):
    c = lowest capacity of edges on p

For each edge e on p
    if e is a forward edge
        f[e] += c
    if e is a backwards edge
        f[e] -= c
```

Ford-Fulkerson's algorithm

Appendix 2 - Edmond-Karp's algorithm

```
Implementation in Python:

def edmonds_karp(C, source, sink):
    n = len(C) # C is the capacity matrix
    F = [[0] * n for _ in xrange(n)]
     # residual capacity from u to v is C[u][v] - F[u][v]

    while True:
        path = bfs(C, F, source, sink)
        if not path:
            break
        flow = Inf # traverse path to find smallest capacity
        for i in xrange(len(path) - 1):
            u,v = path[i], path[i+1]
            flow = min(flow, C[u][v] - F[u][v])
        # traverse path to update flow
        for i in range(len(path) - 1):
            u,v = path[i], path[i+1]
            F[u][v] += flow
            F[v][u] -= flow
    return sum([F[source][i] for i in xrange(n)])

def bfs(C, F, source, sink):
    P = [-1] * len(C) # parent in search tree
    P[source] = source
    queue = [source]
    while queue:
        u = queue.pop(0)
        for v in xrange(len(C)):
            if C[u][v] - F[u][v] > 0 and P[v] == -1:
                P[v] = u
                queue.append(v)
                if v == sink:
                    path = []
                    while True:
                        path.insert(0, v)
                        if v == source:
                            break
                        v = P[v]
                    return path
    return None
```

Edmonds-Karp's algorithhm

Appendix 3 - Relabel-to-front algorithm

Implementation in Python:

```
def relabel_to_front(C, source, sink):
    n = len(C) # C is the capacity matrix
    F = [[0] * n for _ in xrange(n)]
    # residual capacity from u to v is C[u][v] - F[u][v]
    height = [0] * n # height of node
    excess = [0] * n # flow into node minus flow from node
    seen   = [0] * n # neighbours seen since last relabel
    # node "queue"
    list   = [i for i in xrange(n) if i != source and i != sink]
    def push(u, v):
        send = min(excess[u], C[u][v] - F[u][v])
        F[u][v] += send
        F[v][u] -= send
        excess[u] -= send
        excess[v] += send
    def relabel(u):
        # find smallest new height making a push possible,
        # if such a push is possible at all
        min_height = height[u]
        for v in xrange(n):
            if C[u][v] - F[u][v] > 0:
                min_height = min(min_height, height[v])
                height[u] = min_height + 1
    def discharge(u):
        while excess[u] > 0:
            if seen[u] < n: # check next neighbour
                v = seen[u]
                if C[u][v] - F[u][v] > 0 and height[u] > height[v]:
                    push(u, v)
                else:
                    seen[u] += 1
            else: # we have checked all neighbours. must relabel
                relabel(u)
                seen[u] = 0
    height[source] = n   # longest path from source to sink is less than n long
    excess[source] = Inf # send as much flow as possible to neighbours of source
    for v in xrange(n):
        push(source, v)
    p = 0
    while p < len(list):
        u = list[p]
        old_height = height[u]
        discharge(u)
        if height[u] > old_height:
            list.insert(0, list.pop(p)) # move to front of list
            p = 0 # start from front of list
        p += 1
    return sum([F[source][i] for i in xrange(n)])
```

Relabel-to-front algoritm

Appendix 4 - Max flow algorithm from our prorotype

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Windows.Forms;
using System.ComponentModel;
using System.Data;
using System.Drawing;

using MaxFlow;

//MaxFlowClass
namespace Ford_FulkersonsAlgorithm_Graphics
{

    public class NetworkFlow
    {

        #region local variables
        public static int MaxNoOfNodes = 200; //1000
        public static int oo = 1000000000;

        //Declarations
        public static int NoOfNodes = 0;  // number of nodes
        public static int NoOfEdges = 0;  // number of edges

        // declare and initialize multidimensional array
        public static int[,] Capacity = new int[MaxNoOfNodes, MaxNoOfNodes];
      // capacity matrix
        public static int[,] Flow = new int[MaxNoOfNodes, MaxNoOfNodes];
      // flow matrix
        public static int[,,] FinalFlow = new int[1000,1,3];
        public static int FinalFlowCounter = 0;
      // Indicates the real number of flow, last value in FinalFlow

        // Single-dimensional arrays.
        public static int[] Color = new int[MaxNoOfNodes];
      // needed for breadth-first search
        public static int[] PreDestination = new int[MaxNoOfNodes];
      // array to store augmenting path

        //A Queue for Breadth-First Search
        public static int head, tail;
        public static int[] Queue = new int[MaxNoOfNodes + 2];
        public static int White = 0;
        public static int Gray = 1;
        public static int Black = 2;
        #endregion local variables
```

```
#region Max Flow
        public static void Enqueue(int x)
        {
            Queue[tail] = x;
            tail++;
            Color[x] = Gray;
        }

        public static int Dequeue()
        {
            int x = Queue[head];
            head++;
            Color[x] = Black;
            return x;
        }

        public static int BreadthFirst(int start, int target)
        {
            //Breadth-First Search for an augmenting path
            int u, v, y;

            // Breadth-First Search for an augumenting path
            // Initializing the PreDestination[] (Augumenting path)
            for (u = 0; u < NoOfNodes; u++)
            {
                Color[u] = White;
            }
            head = tail = 0;
            Enqueue(start);
            PreDestination[start] = -1;
            while (head != tail)
            {
                u = Dequeue();
                // Search all adjacent white NoOfNodes v. If the capacity
                // from u to v in the residual network is positive,
                // enqueue v.
                for (v = 0; v < NoOfNodes; v++)
                {
                    y = Capacity[u,v] - Flow[u,v];
                    if (Color[v] == White && y > 0)
                    {
                        Enqueue(v);
                        PreDestination[v] = u;
                    }
                }
            }
            // If the color of the target node is black now,
            // it means that we reached it.
            if (Color[target] == Black) return 0;
            return 1;
        }
```

```
public static int MaxFlow
                (int source, int sink, TextBox myTextArea, int _NoOfNodes )
          {
              //Ford-Fulkerson Algorithm
              int i, j, u;

              // Initialize empty flow.
              int max_flow = 0;
              NoOfNodes = _NoOfNodes;
              // Calculating Max flow.
              // Initializing the flow matrix[][]
              for (i = 0; i < NoOfNodes; i++)
                  for (j = 0; j < NoOfNodes; j++)
                      Flow[i, j] = 0;
              for (i=0; i < 100; i++)
              {
                  FinalFlow[i, 0, 0] = 0;
                  FinalFlow[i, 0, 1] = 0;
                  FinalFlow[i, 0, 2] = 0;
              }
              // While there exists an augmenting path,
              while ( BreadthFirst(source, sink) == 0)
              {
                  // Determine the amount by which we can increment the flow.
                  int increment = oo;
                  for (u = NoOfNodes - 1; PreDestination[u] >= 0;
                          u = PreDestination[u])
                  {
                      //increment = min(increment, y );
                      if (increment > Capacity[PreDestination[u], u] -
                          Flow[PreDestination[u], u])
                      {
                          increment = Capacity[PreDestination[u], u] -
                          Flow[PreDestination[u], u];
                      }
                      //increment = min(increment,
                          Capacity[PreDestination[u]][u] -
                          Flow[PreDestination[u]][u]);
                  }
                // Now increment the flow.
                for (u = NoOfNodes - 1; PreDestination[u] >= 0;
                        u = PreDestination[u])
                {
                    Flow[PreDestination[u], u] += increment;
                    Flow[u, PreDestination[u]] -= increment;
                    myTextArea.AppendText("From [" + u + "]\t-> [" +
                        PreDestination[u] + "] =" + increment);
                    myTextArea.AppendText("\t[" + PreDestination[u] + "]
                        <- [" + u + "] = -" + increment);
                    myTextArea.AppendText(Environment.NewLine);

                    //myTable.SetValue(startX, node, 0, 0);
                    //Array.Resize(ref FinalFlow, (myCounter + 1));
                    FinalFlow.SetValue(u, FinalFlowCounter, 0, 0);
                    FinalFlow.SetValue(PreDestination[u],
                        FinalFlowCounter, 0, 1);
                    FinalFlow.SetValue(increment, FinalFlowCounter, 0, 2);
                    FinalFlowCounter++;

                }
                myTextArea.AppendText(Environment.NewLine);
                max_flow += increment;
            }
            // No augmenting path anymore. We are done.
            return max_flow;
        }
        #endregion Max Flow


    } //end of class NetworkFlow

} //End of namespace
```