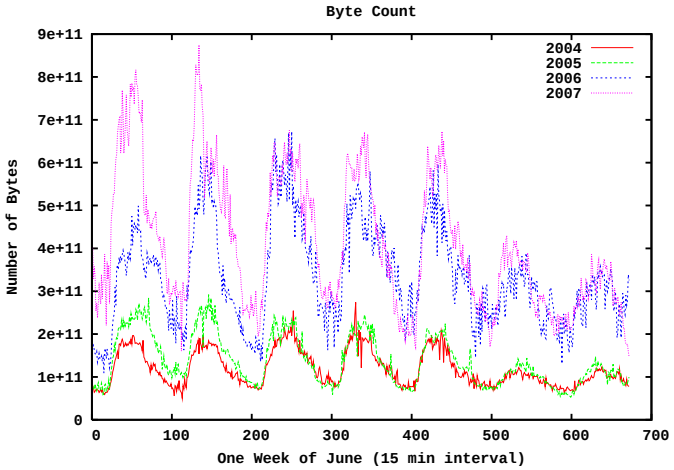
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Dominik Schatzmann

Analyzing network traffic from the SWITCH network from 2003 until today



Master Thesis MA-2007-12
April 2007 to September 2007

Tutor: Bernhard Tellenbach
Supervisor: Prof. Bernhard Plattner

Abstract

The analysis of the characteristics of network traffic is the basis for many areas of research such as traffic engineering, anomaly detection or application identification. In most cases, it consists of an analysis of the development of one or multiple traffic metrics in a certain time interval. While this interval is typically in the range of hours to days when investigating the characteristics of an isolated anomaly, its length can be up to several years when investigating seasonal effects or when performing other long term studies. The main problem with long term studies is that there is almost no seamless data set available that offers a complete view on the traffic of a medium to large scale network. Since we started collecting the Cisco NetFlow data from the border gateway routers of a medium scale Internet backbone network in 2003, we are now in a good position for doing such studies. Today (2007), our NetFlow data archive consist of 26 TiB of compressed flow records stored on a tape library. The main challenge with long (and short) term studies on this data set is to handle the huge amount of data in a convenient way. This means that we need a hard- and software infrastructure that allows for a time and resource efficient planing, supervising and processing of the network traffic data. To address this problem, we designed and implemented the Data Mole, an extensible framework for optimized access to tape archives and for large scale data processing on privacy compliant computer clusters. Other important design goals were ease of use and on-off capability. A web interface offers convenient monitoring and controll interfaces to the users of the framework. Furthermore, we optimized and integrated some basic modules for NetFlow data processing like a sorter for NetFlow records or a calculator for basic metrics like e.g. the number of flows per interval. Preliminary results from the data analysis projects of three different users show that the Data Mole framework is easy to use, is basically on-off stable and allows for a time and resource efficient processing of large scale data set.

Acknowledgment

Different people contributed to this Master Thesis and I would like to express my gratitude to them. I would like to thank . . .

- Prof. Dr. Bernhard Plattner for the supervision of my Master Thesis and for making this project possible.
- Bernhard Tellenbach for his support during my thesis. It was a great pleasure to collaborate with him. The discussions with Berhnard helped me to successfully finish this thesis.
- Dr. Rainer Baumann for various discussions that have inspired different parts of this work.
- The CSG for their general support and especially for using the coffee corner that help me to survive the last 6 months.
- My friends and colleagues for their support and their understanding for my absence during the last months.
- My parents for providing me with all I needed and for their unconditional support at anytime.

Dominik Schatzmann

Contents

List of Figures

1. Introduction

The analysis of the characteristics of network traffic is the basis for many areas of research such as traffic engineering, anomaly detection or application identification. In most cases, it consists of an analysis of the development of one or multiple traffic metrics in a certain time interval. While this interval is typically in the range of hours to days when investigating the characteristics of an isolated anomaly [10] [13] [14], its length can be up to several years when investigating seasonal effects or when performing other long term studies [11] [12].

The main problem with long term studies is that there is almost no seamless, unsampled data set available that offers a complete view on the traffic of a medium to large scale network. For example, the NetFlow [1] records collected by [6] of the GÉANT [5] Network are sampled with a rate of (1/1000). The NetFlow data of the Abilene Network [3] is collected by the Abilene Observatory program [4] is anonymized (lower-order 11 bits of IP address are set to zero).

Since we started collecting the unsampled Cisco NetFlow data from the border gateway routers of a medium scale Internet backbone network[2] [8] in 2003, we are now in a good position for doing such studies. Today (2007), our NetFlow data archive consist of 26 TiB of compressed flow records stored on a tape library.

1.1. Motivation

Our archive of unsampled NetFlow data is a valuable asset and puts us in a unique position for performing long term traffic analysis on the border traffic of a medium scale Internet backbone network. But if we want to do this type of analysis, we need to handle the huge amount of data in a convenient way. More precisely, we need a hard- and software infrastructure that allows for a time and resource efficient planing, supervising and processing of this data. But because our data set puts us in a unique position for doing such an analysis, it is unlikely that there already exists an off-the-shelf solution for such a hard- and software infrastructure. Hence, if we want to exploit our unique position for doing long term analysis, we have to design and implement a hard- and software infrastructure for a time- and resource efficient processing of our NetFlow data.

1.2. Problem Statement

To process the NetFlow traffic that is collected by the DDosVax project a modular and reusable management framework is needed. The management framework has to address the problem of data and job management. A simple interface for control and monitoring should be created. In a next step a set of basic traffic metrics should be evaluated with the help of this framework.

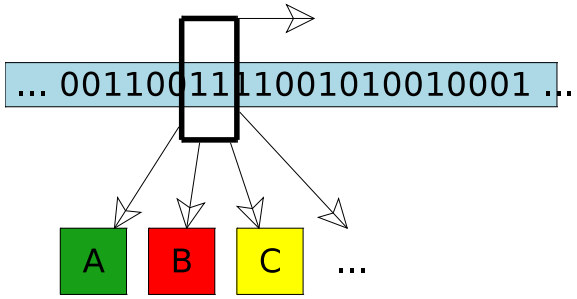


Figure 1.1: The data processing by UPFrame

Data Management The framework must be able to work with the full data set that is collected by the DDosVax project. The quantity and quality of the stored records is unknown (e.g missing, corrupt). The framework has to be able to cope with outages of the data set. The file download from the long term storage system to the computer cluster has to be optimized.

Job Management The framework has to be able to process several jobs in parallel. A measurement often has a run time of several weeks. For maintenance of the server cluster the framework has to implement the feature that on-going measurements can be stopped and resumed.

1.3. Related Work

To the best of our knowledge, we did not find an existing framework that is able to handle job management and data management of long term traffic analysis. However, there exist several programs and tools that are able to process NetFlow data. But most of them do not include job management or data management features. In the following we briefly present two popular tools for NetFlow data processing and one open source tool for job management.

UPFrame [9] is a powerful framework to process NetFlow data in real time. The main application of UPFrame is the optimized processing of subsequent NetFlow records with several programs at the same time. This is illustrated in Fig. 1.1. But for long term traffic analysis we are rather interested splitting the timeline into multiple parts and processing them in parallel by one or only a few different programs as illustrated in Fig. 1.2.

NFDUMP [15] is a toolset to capture, store, filter and replay NetFlow data. It is able to read NetFlow data from live streams or from files stored in folders meeting a special naming convention. Like UPFrame, NFDUMP is used to process data of a single timeline and has no built-in functionality for timeline splitting and parallelization.

There exist different generic frameworks for the issue of job scheduling like the one developed by the Condor Project [16]. Basically, Condor is a framework for high throughput computing (HTC). It detects idle personal computers and uses their resources to process any type of job. Furthermore, it has some data management features to distribute input data and to copy back local output of these jobs. This project could be a candidate for the job scheduling part

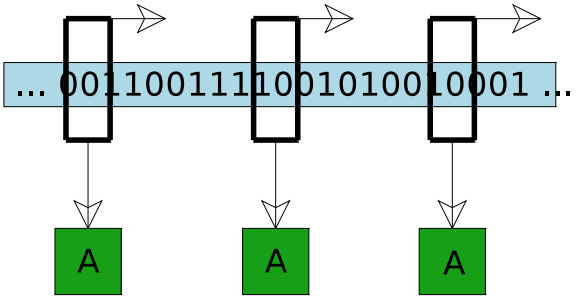


Figure 1.2: The data processing for a long term analysis

of our framework.

1.4. Outline

This work is structured into the following parts. In chapter 2 the problem statement of this work is analyzed based on the existing resources and the requirements. The design of the Data Mole is discussed in chapter 3. Then in chapter 4 the implementation of the different modules is presented. IN chapter 5 we discuss the evaluation of this framework. Chapter 6 summarizes the work and discusses the open issues.

2. Problem Analysis

In this chapter we work out some important key points that should be respected during design and implementation of the Data Mole project. In the first section we formulate the general requirements of this project and then the available infrastructure is presented. In a next step we review the infrastructure based on the requirements to work out the key points that should be respected. These results are then summarized in the conclusion section.

2.1. Requirements

The Data Mole project has to accomplish the following requirements:

- Modularity
  The framework has to be programmed in a modular manner. This allows to expand the framework or replace existing functional blocks without much effort. Further the cost for validation and debugging of the framework is decreased. This is achieved by the fact that each module can be validated and tested independently.

- Ease of Use
  The effort for an end user to handle the framework has to be very small. The end user is not willing to read more than 20 pages and to spend more than 5 hours to be able to perform measurements using the framework.

- Reusability
  There exists a large number of good analysis programs for NetFlow data processing. The framework should be able to integrate these programs with low effort. This reusability can be achieved by the choice of a robust and simple interface.

- On-Off Capability
  The time to perform a long term traffic analysis is in the range of weeks or months. Due to this long processing time the framework has to implement a feature that allows to stop and then resume the measurements. This feature is called 'On-Off Capability' and is required for example for maintenance of the computer cluster infrastructure.

- Performance
  The overall performance of the framework should be optimized. The critical parts of the data path and job path have to be identified and optimized.

2.2. Infrastructure

The Data Mole is primary used by the Communication Systems Group[7] (CSG). Therefore the Data Mole is optimized to run on the infrastructure that is provided by the CSG. But the design of the Data Mole has to be generic enough so that the framework can be adapted to any infrastructure with low effort. This section describes the infrastructure of the CSG. This knowledge is then needed for the further analysis of the problem.

2.2.1. Computer Cluster

The computer cluster for task processing consists of six computers called nodes. Each node has two AMD Opteron 275 processors with 8GB RAM. The AMD Opteron 275 processor is a dual core processor and therefore 4 jobs can be processed in parallel on one node. The local storage has a size of 4TiB. Due to the fact that the node performs a RAID 6 over the physical storage only 3TiB are available for the user. The nodes are connected with a Gigabit Ethernet (GbE) network.

Five nodes are used for general calculations. These nodes are called x01 to x05. One node is used as login and management server and it is called xFS. An other older computer called aw4 is used as backup management server. This is illustrated in Fig. 2.1. In near future the cluster will be expanded by a new file server, called fs1.

The root file systems of x01 to x05 are hosted by xFS and is mounted via Network File System (NFS). The local disk storage of each node can be accessed from each node over NFS.

2.2.2. Long Term Storage System

The CSG uses a long term storage system that is called Jabba [18]. This system is supported by the department of information technology and electrical engineering (ITET) of the ETH Zurich. Jabba consists of a Sun Fire V880 and of a IBM 3494 robot with 4 IBM 3592 FC tape drives. The storage system is connected with Gigabit Ethernet(GbE) to the ETH network. Jabba currently has a capacity of about 336 TiB and can be easily expanded.

2.3. Data Management

To achieve the requirement of an efficient framework critical resources have to be identified and the design of the framework has to optimized to cope with these critical resources. In this section we review the data path of the NetFlow records. We start our review with the Jabba storage server and will then focus on the cluster nodes.

2.3.1. Jabba Server

The Jabba server is a tape based storage server. To optimize the interaction with this server, we analyze in a first step how the data is stored on the Jabba system. Then we review how the data can be downloaded from The Jabba server. In the last part we discuss the limitations of the Jabba server.

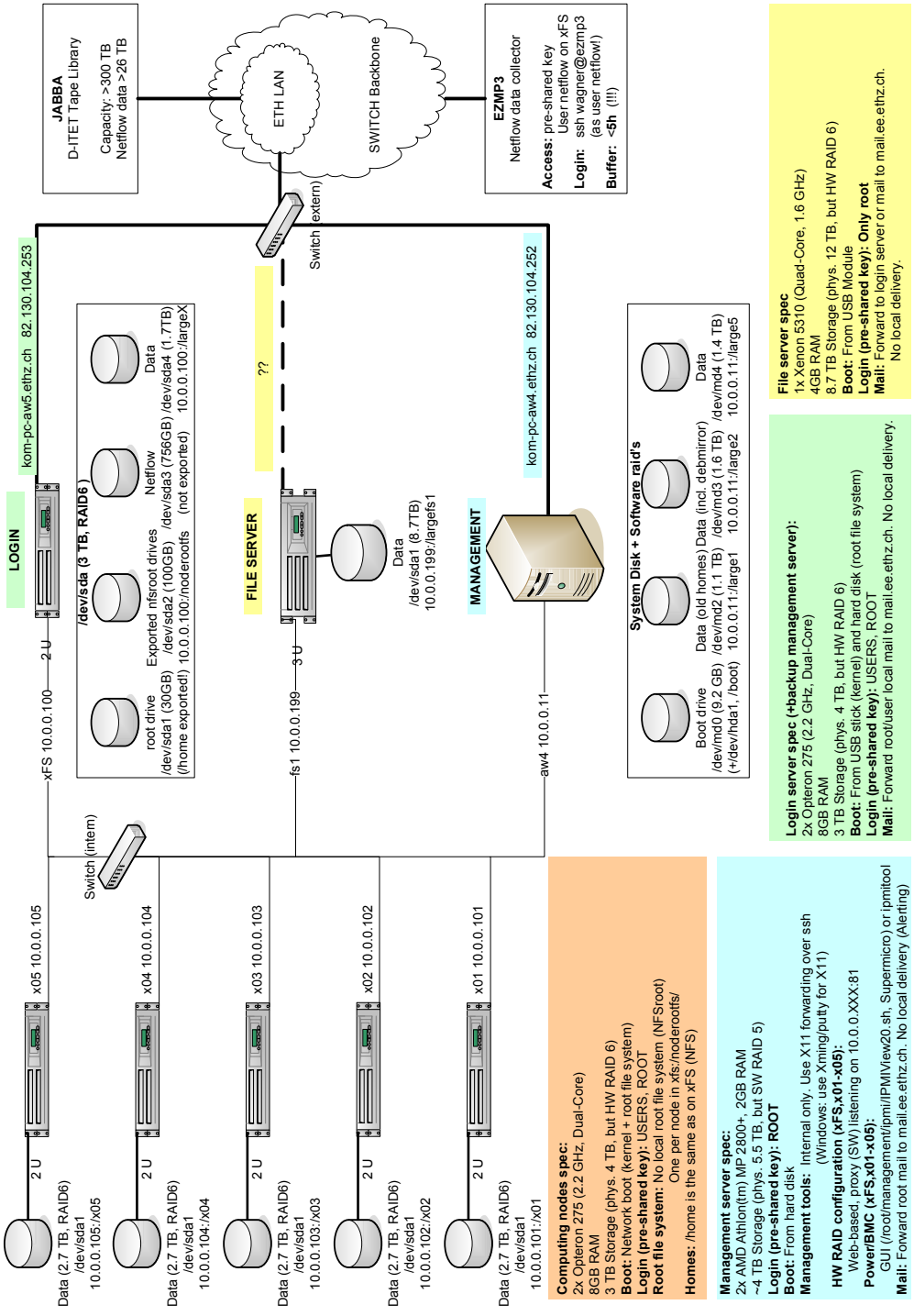


Figure 2.1: The computer cluster at CSG

File index The DDosVax project collects the NetFlow data from different border gateway routers of the SWITCH network. The NetFlow records from one hour are collected and aggregated to two bzip2 compressed hour files. These files are then saved on the Jabba server. Every month more than 700 files are stored on the Jabba server.

The data set is not complete. Several files with compressed NetFlow records are not stored on the Jabba server. These gaps can be explained by the fact that some system outages have stopped the recording process of the DDosVax Project.

In 2007 more than 160'000 files are stored on the Jabba server. The process to find and locate a file for a given hour by hand is quite painful. This process has to be automated.

An index of all files stored on the Jabba server has to be built. This index can be used to locate all files of a given time period. Due to the large number of files index generation has to be automated. The index generator has to be able to work with outages in the file set.

The Tape Device The time that is needed to download the data from the Jabba server to the cluster nodes has to be minimized to increase the performance of the system. IBM specifies the maximum download capacity of the IBM 3592 tape device [17] with 108 Mbps. The administrators [18] of the Jabba system announced that the drive has an IO-performance of 30 to 100 Mbps.

To increase the performance all files that are saved on the same tape should be downloaded at once. This reduces the chance that a tape must be loaded twice or more. To increase the download speed even more, several tapes can be downloaded in parallel. The table shows the expected download speeds for a minimal rate of 30 Mbps and a maximal rate of 108 Mbps. This optimization is only possible if the tape name of each file is stored in the file index.

Parallel Downloads	1	2	3	4
per hour (min / max) [GiB]	12.6 / 45.3	25.1 / 90.5	37.7 / 135.8	50.29 / 181.0
per day (min / max) [TiB]	0.3 / 1.0	0.6 / 2.1	0.9 / 3.2	1.2 / 4.2

The Network Connection The Jabba server is connected with Gigabit Ethernet (GbE) to the ETH network. The traffic from the Jabba server crosses this network and then entresr the cluster on the xFS.

The capacity limit of a GbE Network is around one gigabit per second. This corresponds to a rate of 0.11 GiB per second or 419 GiB per hour. This rate is up to 10 times higher than the output of the tape library.

The Fig. 2.2 shows the network load on xFS during a file download phase when two tapes are downloaded in parallel. A throughput of around 150 Mbps is achieved. This corresponds to 71 % of the maximum rate. The gaps in the plot correspond to time periods when the download has been stopped to show the normal network traffic load.

It can be concluded that the network connection from the computer cluster to the Jabba server does not limit the download and is therefore not a critical resource.

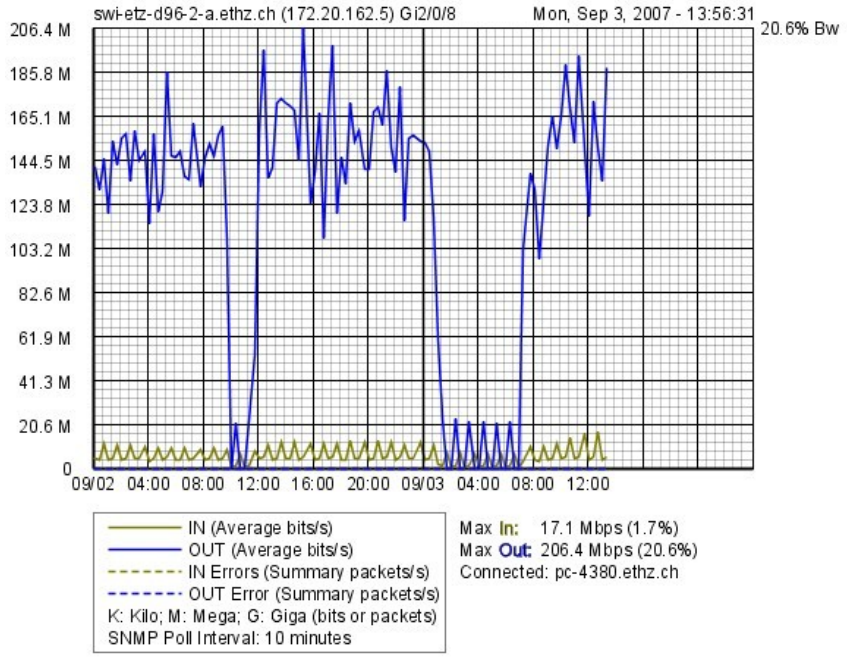


Figure 2.2: The network load during a file download at xFS

The Limits The Jabba server has four tape devices. The maximum download speed of the system is achieved if and only if all four devices are used. But the Jabba server is not only used by this project. The Jabba server is used by all the different institutes of the ITET. Therefore all tape devices of the Jabba server have to be shared with the other institutes. Therefore not more than two tape devices should be used in parallel.

2.3.2. Cluster Node

This section reviews the data path on the cluster node. First we focus on disk size, which limits the job size that can be processed. Then we discuss disk speed, wich limits the number of input streams that can be processed in parallel. Furthermore memory limitations of the node are reviewed.

Disk Size The hard disks of one cluster node are combined with the help of a hardware RAID controller to a logical disk with the capacity of the size of 3GiB.

The size of the compressed NetFlow data has increased over the years. For example a file with the NetFlow records of one hour in the year 2007 is twice the size of such a record in 2003. The Fig. 2.3 shows an overview over the average hour file size averaged over one month. To reduce the error from corrupted files only files with a minimal size larger than 20 MByte are used for the calculation.

We can conclude that in 2007 the local storage of the nodes is sufficient to store at least 12 weeks of compressed NetFlow data. A standard job runs over the collected data of several hours up to weeks. Therefore the disk space is large enough and is not a critical resource.

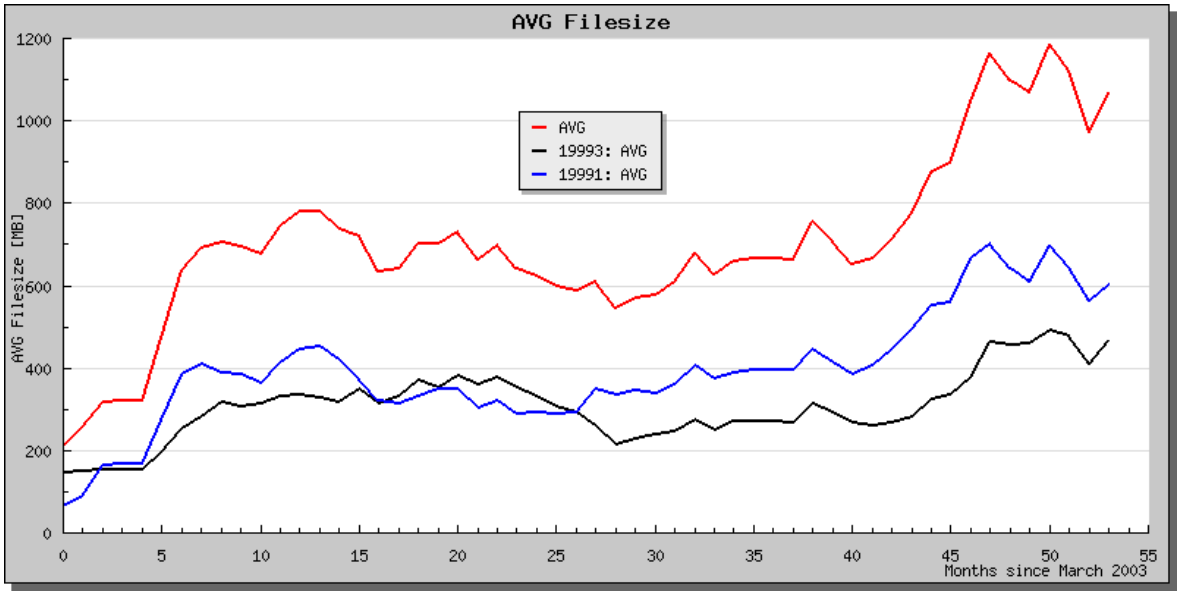


Figure 2.3: The development of file size

The data can be downloaded first and can then be processed by the analysis program without a problem.

Disk Speed The hardware RAID controller concatenates the local hard disks with a RAID 6 [19] to a larger logical capacity. In this mode the data is spread over different disks. This provides fault tolerance and increases the read and write performance of the local capacity.

The disk array performance can be estimated by a simple file copy operation. A node needs less than 25 minutes to copy different files with a total size of 68 GiB. This corresponds to performance of 46 MiB/s.

The NetFlow data of the DDosVax project is compressed with a Bzip2 [23] file compressor. On the cluster we have measured a decompression rate of 5 MiB/s. The decompression process is CPU limited. Based on the results of the file copy measurement it can be shown that 9 decompression processes can be fed from the disk array of one cluster node.

It can be concluded that the disk array of one cluster can at least feed the four local CPUs with compressed NetFlow data without any problems. Disk speed is not a critical resource.

2.4. Job Management

In the first subsection we focus on the process of how smaller work packages can be created from longer measurements. Then we review how this work or these jobs can be efficiently processed by the computer cluster. Furthermore the topic of monitoring is discussed.

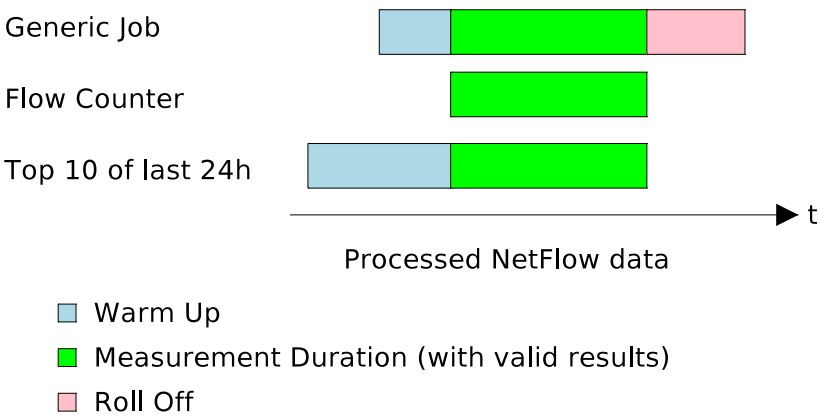


Figure 2.4: The measurement parameter

2.4.1. Job Creation

All the metrics that are used for network traffic measurements can be characterized by the following triple: Measurement duration, warm up duration and a roll off duration.

- Measurement duration
  The time it takes when the measurement process to create meaningful results.

- Warm up
  The additional information from the past (measurement as time interval) that is needed to bring the measurement process into a valid state.

- Roll Off
  The additional additional information from the future (measurement as time interval) that is needed finish the measurement.

For example, a simple flow counting metric needs no additional information of the past or of the future to create a meaningful output. Therefore the warm up and roll off duration can be set to zero.

On the other hand, a metric that collects the largest ten flows (based on the transmitted bytes) of the last 24 hours needs at least the NetFlow data of the last 24 hours before it is able to create meaningful output. The warm up phase is defined as 24 hours and is needed to bring the process into a valid state.

Fig. 2.4 shows a generic metric with roll off and warm up phase, the 'simple flow counting metric' and the 'largest ten flows metric'. The measurement duration of all metrics is the same, but the three processes have to run over different input data sizes to produce the desired measurement results.

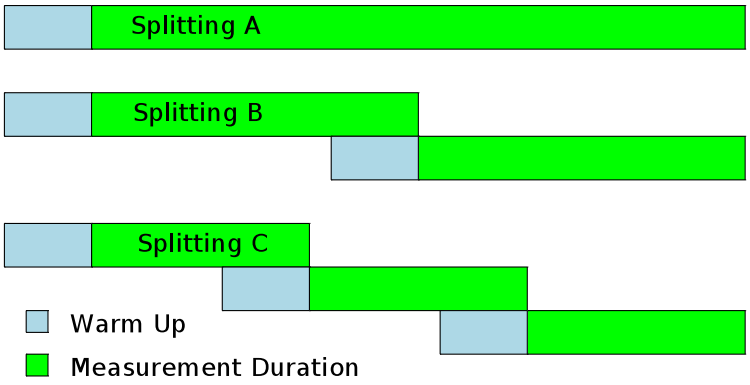


Figure 2.5: The measurement splitting

In a next step we have to split the long measurement into smaller parts, called 'jobs' in this work. This process of deviding up the longer measurement is often called 'time line split' in this work.

The division of the measurement along the time line is always possible. A problem arises only if the measurement needs a long warm up or roll off duration. The overhead of a measurement with non-zero warm up or roll off duration increases if the measurement duration decreases. This is visualized in Fig. 2.5.

To reduce the overhead the measurement interval can be increased. But this solution has one major drawback. When the measurement interval is increased, the time to process this interval increases. If the process is stopped during the processing of a longer interval, the results of the full period are lost and the measurement over this large interval has to be restarted. The trade off between this risk and the accepted overhead defines the optimal measurement interval.

To remove this drawback 'application checkpointing' [27] could be used. But checkpointing is a rather complex issue. Furthermore the advantage of checkpointing is limited to events that are caused by the environment, like power outages. But if the program crashes due to a memory leak or corrupted input data, the program will crash anyway and check pointing is not able to solve these problems.

It has been decided to accept these limitations in a first version and to cover application checkpointing in a later version of the Data Mole project. Then a module with checkpointing should be implemented to handle metrics with large warm up and roll off phases.

The Data Mole download divides the measurement into smaller blocks called jobs and processes these blocks in parallel. In our work, this processing type is called block processing and the unit that does this processing is called Generic Block Module (GBM).

2.4.2. Job Scheduling

In the previous section it was shown how longer measurements can be divided into smaller job. This section reviews the problem of how the different jobs can be processed efficiently in parallel on the computer cluster.

As long as the cluster can process more jobs than input data for new jobs can be downloaded, the job scheduling is very simple: Use the next free node. When the metric calculations take more time and probably need more resources, such as memory or disk space, an efficient job scheduling can increase the performance of the framework.

There exist different projects like Condor [16], LoadLeveler [20] or the TORQUE Resource Manager [21] that focus on the problematic of implementing a fair and efficient job scheduling.

The design of the job scheduler has to be modular and generic such that later the module can be replaced with an existing scheduler of an other project. But due to the limited time a simple job scheduler should be implemented first. The integration of an existing, more efficient job scheduler should be continued in a later phase of the Data Mole project.

2.4.3. Job Controlling

To accomplish the on-off capability the state of each job has to be stored. This allows to start and stop the measurement and resume it without any problem. Due to the large number of jobs an efficient method to manage the different states has to be selected.

2.4.4. Job Monitoring

The timeline is often divided into more than a thousand subparts. To keep track of all these jobs a visual interface is needed. This interface should indicate the state of each job in the 'go/no go' [22] form. A web interface that illustrates the states of each job with the help of colors could be used.

2.5. Conclusion

This chapter reviewed the problem of a long term analysis based on the infrastructure of the CSG group.

The analysis of the data path shows that an index of all saved compressed NetFlow files has to be implemented. This index allows to download from several physical tapes in parallel. This will increase the download rate. Further the local disk space of the cluster nodes is fast and large enough. Therefore the local disk space is not regarded as a critical resource.

The review of job management shows that a module for block based metrics should be designed. Furthermore a modular job scheduler has to be implemented, that can be replaced or improved in a later phase of the project. An efficient, scalable data structure to store the large number of jobs with their states has to be implemented.

3. Design

In this section we present the design that is used to build the Data Mole framework. This design is based on the problem analysis and the requirements of the last chapter. The knowledge that is presented in this chapter is needed to understand the implementation of the different modules.

In the first section the modular design of this framework is presented. Then the concept is shown that is used to reduce the implementation time of new metrics. After this step the data structure that is used by this framework and its handling is discussed. The chapter is closed with the presentation of the inter-module communication and a short survey of the Data Mole class library.

3.1. Modular Design

"Modular design means trying to subdivide an assembly in smaller parts (modules) that are easily interchangeably used" [24].

Based on this definition the Data Mole framework can be divided into four major parts. These are the downloader part, that fetches the data from the Jabba server, the job scheduler part, that processes the jobs on the cluster nodes, the control and monitor part, that is used by the user as interface to the modules and the fourth part is the 'metric universe'. This last part hosts the different metric modules like the IP entropy measurement module. All these parts consist of several modules that are used to implement the needed functionality. This is illustrated in the figure 3.1.

The measurement modules use a common interface that allows them to stack different modules together. This allows us to build some basic modules like a flow sorter module, that is then reused by an other metric module. This feature should reduce the time needed to develop new metrics modules.

Due to the modular design the implementation of new features or the replacement of existing modules can be handled in a short time. A module is an executable file and can be started independent of other modules on any cluster node. The features of the different modules are discussed in chapter 4.

3.2. Generic Metric Module

An important design decision of the Data Mole project is the strict separation of management functions and the traffic processing functions. The management functions are similar for all

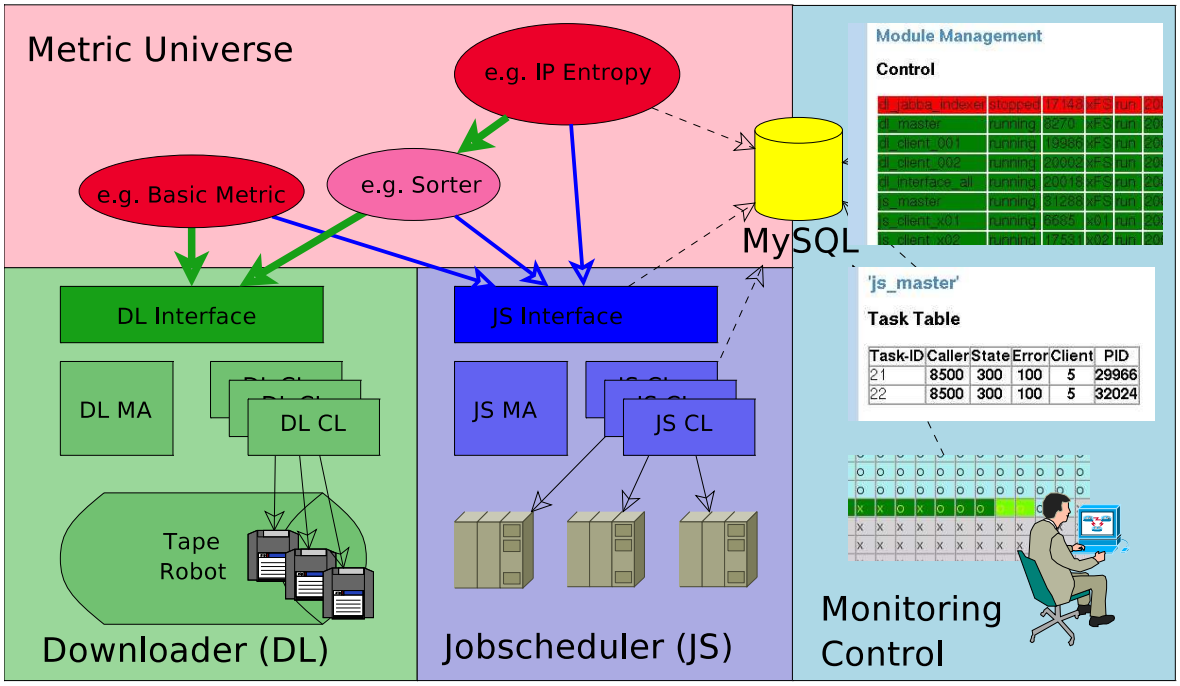


Figure 3.1: The modular design of the Data Mole project

metrics and can be reused for each metric module. The pure processing of the input data and the metric calculation is different for each metric module and therefore this part has to be redesigned and reimplemented for all different metrics modules.

The management functions that are needed to process a block oriented metric (see 2.4) are summarized to one generic module that is called Generic Block Module (GBM). The concept of this GBM is illustrated in figure 3.2. The GBM can be reused to build a new module for any block based metric. The GBM is configured over an XML file to fit the user's needs.

A traffic analysis that is performed with the help of the GBM can be summarized in the following steps. In a first step the GBM divides the timeline into different jobs. Then the GBM organizes the data that should be analyzed for each job (request this data by a download interface or ask an other GBM for this data). This input data is then sent with a user binary, that implements the traffic analysis routine, to a cluster node (the search for a free host is done by the job scheduler). There the user binary is executed and the result of the traffic analysis is stored in the job folder. The GBM is discussed in more detail in section 4.2.

This separation of management functions and the traffic processing functions has the following advantages:

- Stability
  If the user binary crashes for example due to a segmentation fault the management part of the module is not affected.

- Reusablity
  The management functions of the module can be reused for any new block based

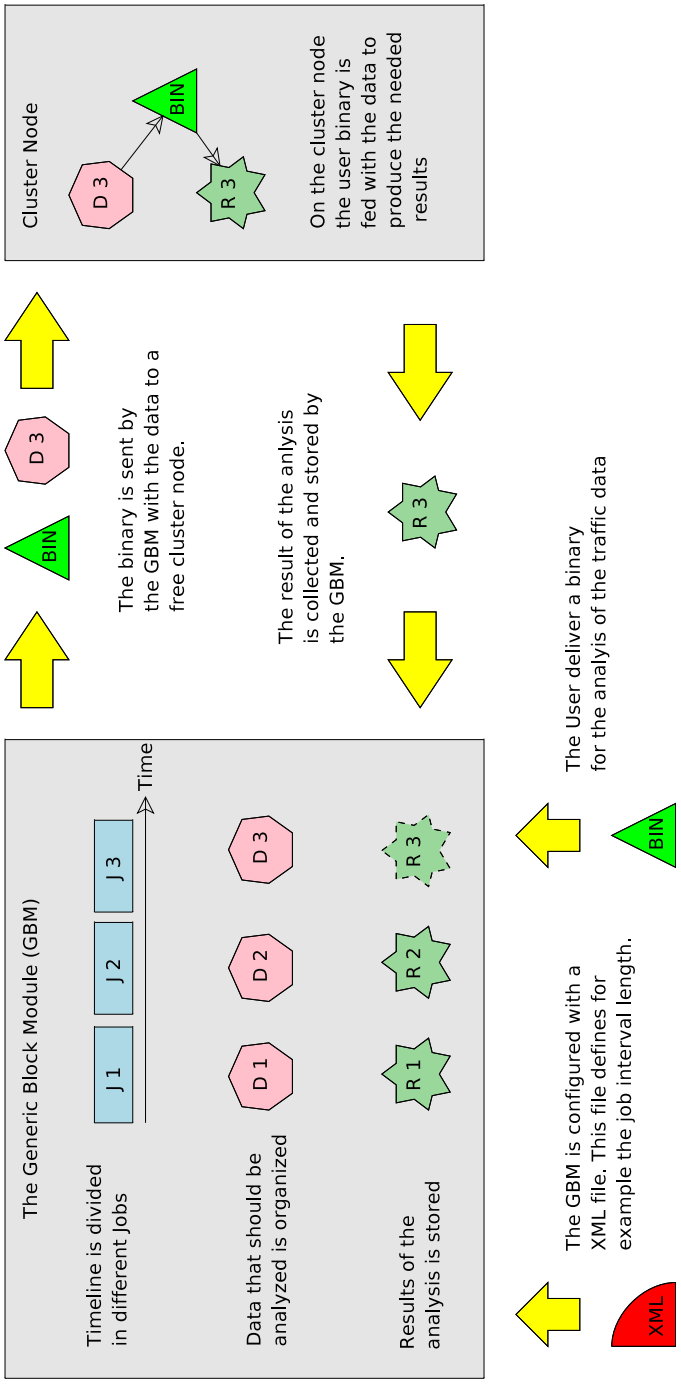


Figure 3.2: The Generic Block Module

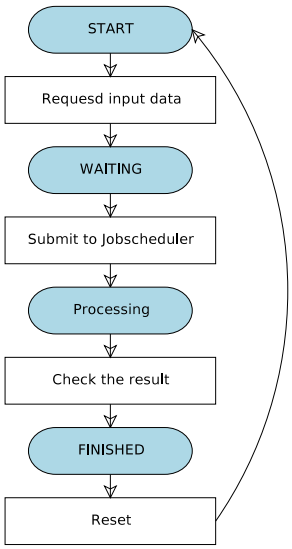


Figure 3.3: The finite state machine of the job data structure

metric. The user only has to adapt the XML configuration file and to create a new analysis binary.

3.3. Data Object Model

The different modules have to handle a large set of data objects. For example, the Generic Block Module (GBM) divides the timeline into thousands of jobs and the module has to work on these thousands of job data objects. In this section we discuss how the data objects are handled and processed by the modules.

To simplify the text the discussion is done based on the GBM. But the result and the concept of this section can be applied to any other module in the Data Mole framework.

3.3.1. Objects have states

The GBM divides the timeline into different jobs. Each job is modeled as a data structure that can be in different states. For example, when the job is processed by a cluster node the data object ’job’ is in the state ’processing’. When the data processing is finished the job is in the state ’finished’. All data objects of the Data Mole project have a finite number of states. Therefore the data processing of the Data Mole project can always be modeled as a finite state machine [25]. The figure 3.3 shows the simplified finite state machine of the job data structure. The state of a data object is carried in the data structure itself as a local variable called ’state’.

3.3.2. Where to store the objects

To accomplish the requirement of on-off capability the data objects have to be saved in persistent storage. The Data Mole project uses a database (MySQL) to store the data objects. The use of a database as storage server has the following advantages:

- Provided basic features
  The database already provides a set of standard features like searching, indexing or caching that can be used to work on data sets.
- Accessibility
  The data objects in a database can be accessed from anywhere over the network.
- On-Off Capability
  If a module is stopped or has crashed the data objects are still accessiblale on the database server.
- Multiple User capability:
  The database server has a built in multi user capability.
- Scalability
  A database server is built to handle thousands of queries on large tables.
- Interface
  An implemented library to access the database server exists for all major programming languages.

3.3.3. Work on Objects

The processing of data object is solved with the help of two fundamental class types. The table class, that is the keeper of the data objects, and the worker class that implements the transitions of the finite state machine.

The table class encapsulates and hides the MySQL database. The worker uses the table class to get a local copy of the data object and to set or update the variables of a data object on the database server. This is illustrated in figure 3.4

3.4. Inter-Module Communication

Inter-module communication is implemented with shared data objects. These data objects are saved on the MySQL server. The different modules use the table classes to access the shared objects. This is illustrated in figure 3.5.

The common solution for inter-process communication (IPC) [31] is the use of shared memory [30]. It is possible to interpret the inter-module communication via the MySQL server as a global shared memory concept. Therefore we claim that the Data Mole uses a global persistent shared memory for inter-module communication. The advantages of using a database server as shared memory are:

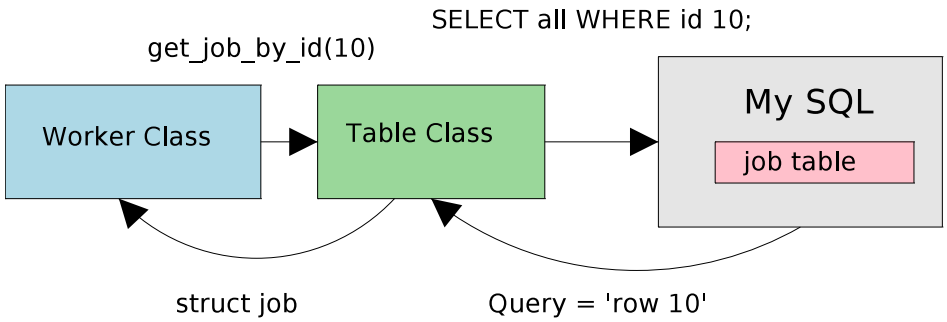


Figure 3.4: The data object model of the Data Mole

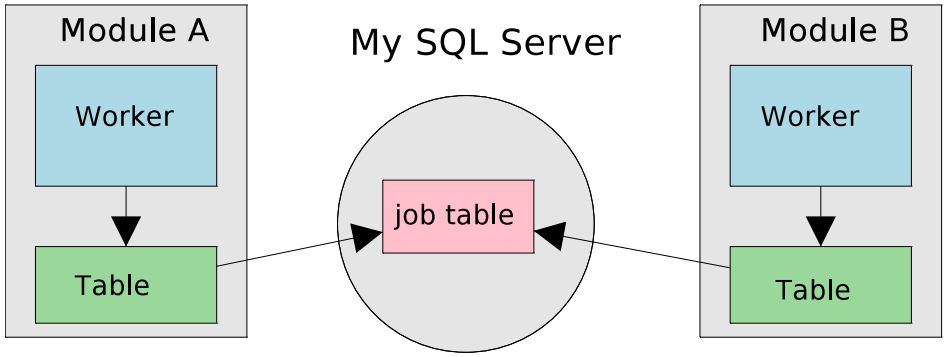


Figure 3.5: The inter-module communication

- On-Off Capability
  The database is accessible even if the module itself is stopped. Therefore one module is able to post a command/message even if the other module is stopped.
- Accessibility
  The shared memory can be easiliy accessed over the network from anywhere. The database handles multiuser access for us.
- Generic
  There exists a large set of libraries in different programming languages to access the database.
- Monitoring and Debugging
  We are able to monitor the communication between the modules with the help of existing tools like the phpMyAdmin [32] browser. This simplifies the software deployment.

3.5. Class Library

All of the modules share some common attributes. We have designed a class library in C++ to use this fact to reduce the implementation time. In addition the class library is used to provide the modules with common interfaces. Figure 3.6 shows an overview over the class library.

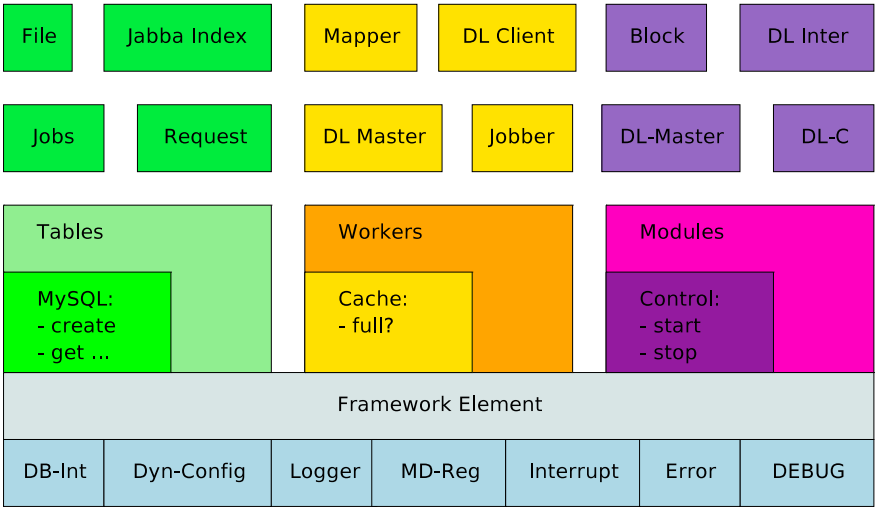


Figure 3.6: The Data Mole class library

All modules of the framework are children of the framework element class. The framework element class is an object composition [33] of different basic classes. These basic classes provide features like database connection, a generalized logging system and access to the module configuration.

The table class implements the required data structures that are used by the Data Mole. As we have discussed in section 3.3, the table class is used to access the different data objects from the MySQL Server.

The worker class implements the transactions of the finite state model. The worker interacts with the different tables to get, for example, a local copy of the data object.

The module class implements the 'main function' of each module. This class combines different workers and tables to one executable module. This class triggers the different workers of these modules to perform their work. The figure 3.7 shows some examples of modules of the Data Mole project with their internal classes structure.

With the help of this class library new modules can be built and old ones can be maintained with low effort.

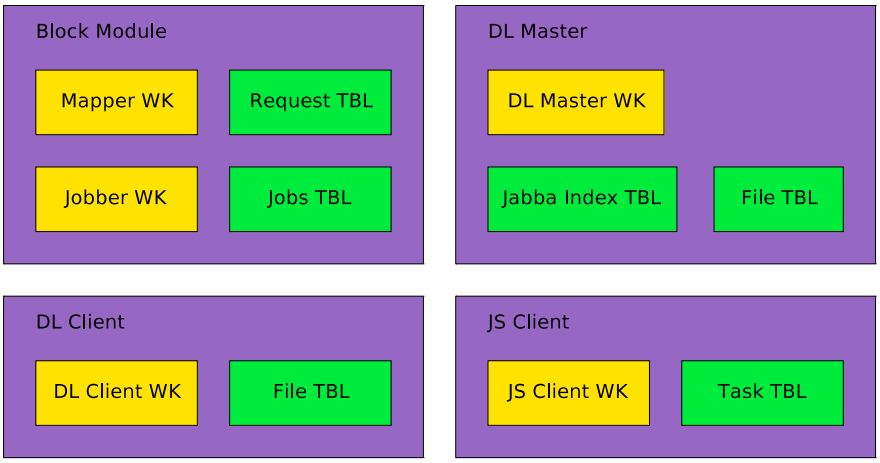


Figure 3.7: The module forming process

4. Implementation

In the Data Mole project the modules can be grouped into three different parts: the downloader part, the job scheduler part and the metric universe. These three categories are visualized in the Fig. 4.1. The fourth part is the web interface but this part has no modules of it own. In the following chapter these main parts with the corresponding modules are presented.

4.1. Overview

4.1.1. Metric Universe

The user modules are hosted in the metric universe. At the moment only one type of template exists to create new user modules. This is the Generic Block Module (GBM). The GBM is configured by an XML file to process a customized analysis. The GBM can be connected to the downloader interface or to an other GBM. If the module is connected to the downloader interface, the module will process the data form the DDosVax project. In the other case the GBM uses the output of the other GBM as its own input data. This allows the combination of different GBM's to a larger chain.

4.1.2. Downloader

The downloader part consists of five modules:

- Jabba Indexer
  The compressed files of NetFlow data are stored on the Jabba server. The jabba indexer module scans the Jabba storage server for all stored NetFlow data files and produces an index of these files. In this index the information about file path, file size, physical tape id or, for example, the router id is saved.

- Download Client
  The download client module downloads the files from the Jabba server to a local file cache. Normally multiple download clients work in parallel. This increases the download speed as long as they work on different physical tapes.

- Download Master
  The download master module manages the local file cache. If some files of the Jabba server are requested, this module will instruct the download client to download the requested files. The download master tries to distribute the work based on the physical tapes to the different download clients. The download master acts as 'garbage collector'

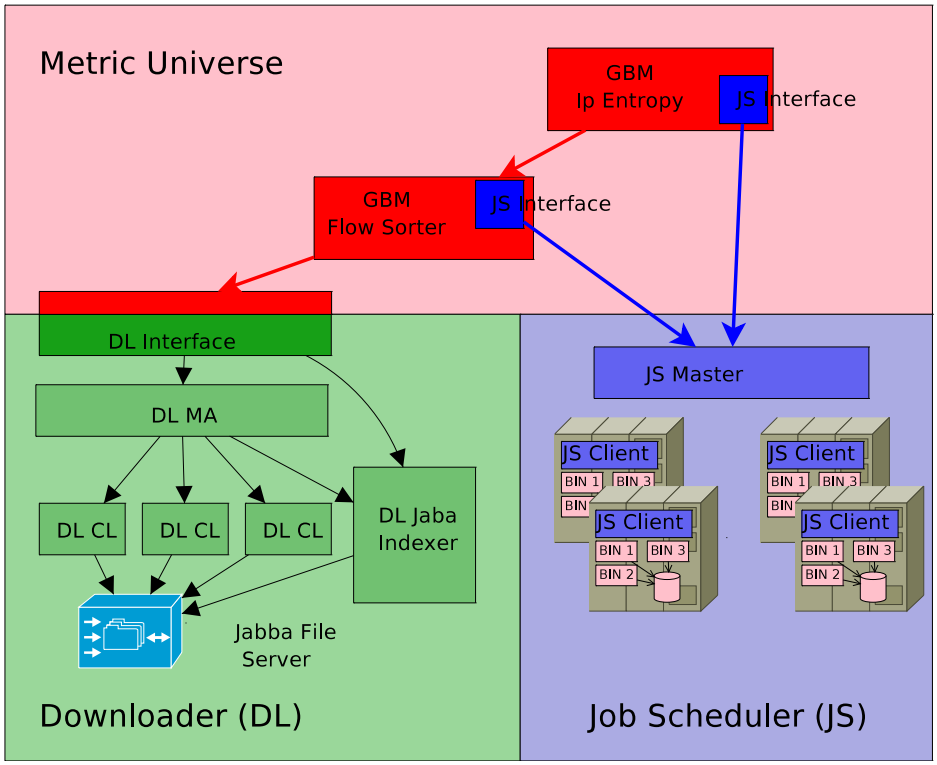


Figure 4.1: The module overview

on the local file cash. When all the space of the local file cache is used, the module tries to identify unused files and removes them form the local file cache.

- Download Interface
  The download interface module maps the time based requests of the metric modules to a file based request that can be handled by the download master.

4.1.3. Job Scheduler

The Data Mole implements a job scheduler on the basis of two module types:

- Job Scheduler Master
  The job scheduler master module collects all tasks of the block modules. These tasks are stored in a table called task table. The job scheduler master checks the workload of the different download clients and submits, if possible, the task to a free job scheduler client.

- Job Scheduler Client
  The job scheduler client module receives the runnable tasks from the job scheduler master. The module prepares and performs the execution of the user binaries. Then this module monitors the process until it has finished its work. After this the job scheduler client copies back the results and removes any temporary files.

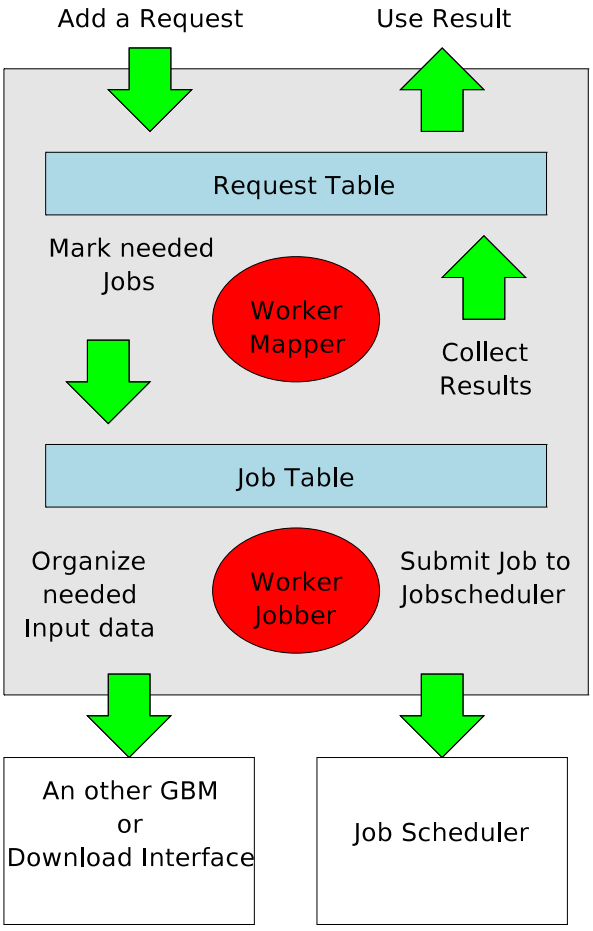


Figure 4.2: An overview of the Generic Block Module

4.2. Metric Universe

The metric universe is the part of the framework where user modules that are used to perform different traffic analyses will be stored. To simplify the process of creating new metric modules we have created a Generic Block Module (GBM). In this section the GBM is discussed.

4.2.1. Overview

The design of the GBM is simple and is illustrated in Fig. 4.2. The GBM consists of the two tables 'request table' and 'job table' and the two workers 'worker mapper' and 'worker jobber'.

The 'request table' of the GBM is used by the other modules to communicate with the GBM. The other modules use this interface to deposit their requests for this module and to check if their requests have been answered. A request can be interpreted as the following command: "Module, give me your results for the time period starting at time X and ending at time Y". The module then tries to answer these requests.

The 'job table' holds all possible jobs that can be processed by this module. The jobs are

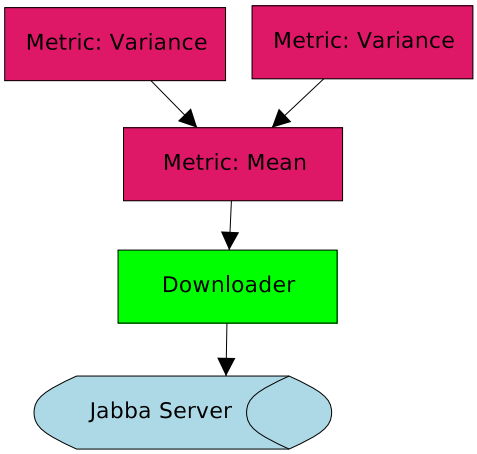


Figure 4.3: How to chain different modules

created based on the timeline division as discussed in section 2.4.

The worker 'mapper' identifies all the jobs that have to be processed to answer the requests. He marks these jobs in the 'job table' as requested (increment a request counter). Now the worker mapper has to wait until these jobs are processed. Then the worker collects the results of the different jobs and marks the request as finished. The modules that have placed the requests can then access the result and perform their analysis on this data.

The worker jobber is responsible for processing requested jobs. For each job the jobber organizes the needed input data. The input data and the user binary is then submitted to the job scheduler for processing. In a last step the results of the finished jobs are saved in the job folder.

4.2.2. Interface

A basic but important key idea of the Data Mole framework is that every module should be able to request and process the output from an other module. This allows the stacking of different existing modules together to get the needed result. The Fig. 4.3 visualizes this behavior. In this example a module that calculates the variance needs the input data from the module that calculates the mean. Furthermore it should be possible that multiple modules can use one module at the same time as their 'input producer'.

The interface that is used by these modules to order data from an other module has to be as generic as possible to support a large number of possible applications. The smallest common denominator of the different modules is the timeline because all modules produce results for a certain time duration. The form of this output id different for each module. But the output of all modules can always be saved into a data file.

The GBM of the Data Mole project uses a shared data object called 'request' as the interface. A request data object consists of a time duration and a path to a folder. The module submits the 'request' with the correct time period (time period of the requested data) to an other module. The submodule processes this request and saves the result into a folder. The path

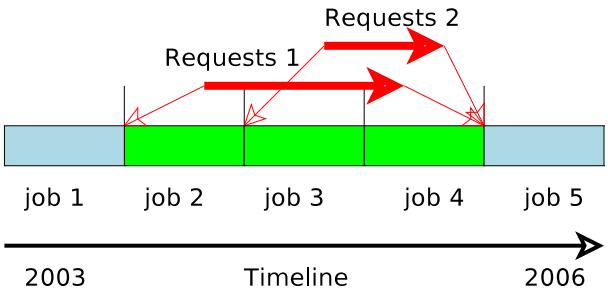


Figure 4.4: Mapping the request to jobs

of this folder is then saved into the request data object. The caller can then use this path information to access and process the provided result data.

4.2.3. Mapper

The mapper worker manages requests from the different modules and maps them request to the needed jobs.

More precisely the mapper sorts the requests in the request table by priority (FIFO) in a first step. Then it calculates for the first n requests the corresponding jobs that have to be processed. This selection is based on the principle that the provided answer of the module includes at least the requested time period. This mapping process is illustrated in Fig. 4.4. For example, to answer the request numbers 1 the job number 2, 3 and 4 have to be processed. The number of request processed in parallel is configured in the XML configuration file of the GBM.

In a next step the mapper sets the priorities of the requested jobs according to the priority of the corresponding requests. If the requests of different modules overlap, then the job inherit the highest priority of all requests.

Then the mapper has to wait until all jobs of the request are processed. In a next step the mapper links (or copies) all results of the different jobs into one result folder. During this process the mapper checks for each job if it was successfully processed and writes this information into an XML based log file. The path to this result folder is saved in the request table. In a last step the request is marked as finished.

When the calling module is finished with the processing of the data of the result folder, it marks the request as 'removable'. The mapper decreases the request counter of these jobs and removes the result folder. In a last step the request is removed from the request table.

The complexity of this full process is further increased due to the fact that the caller module is allowed to mark his request as 'removable' at any time during the processing. The full finite state machine of the mapper worker is illustrated in Fig. 4.5.

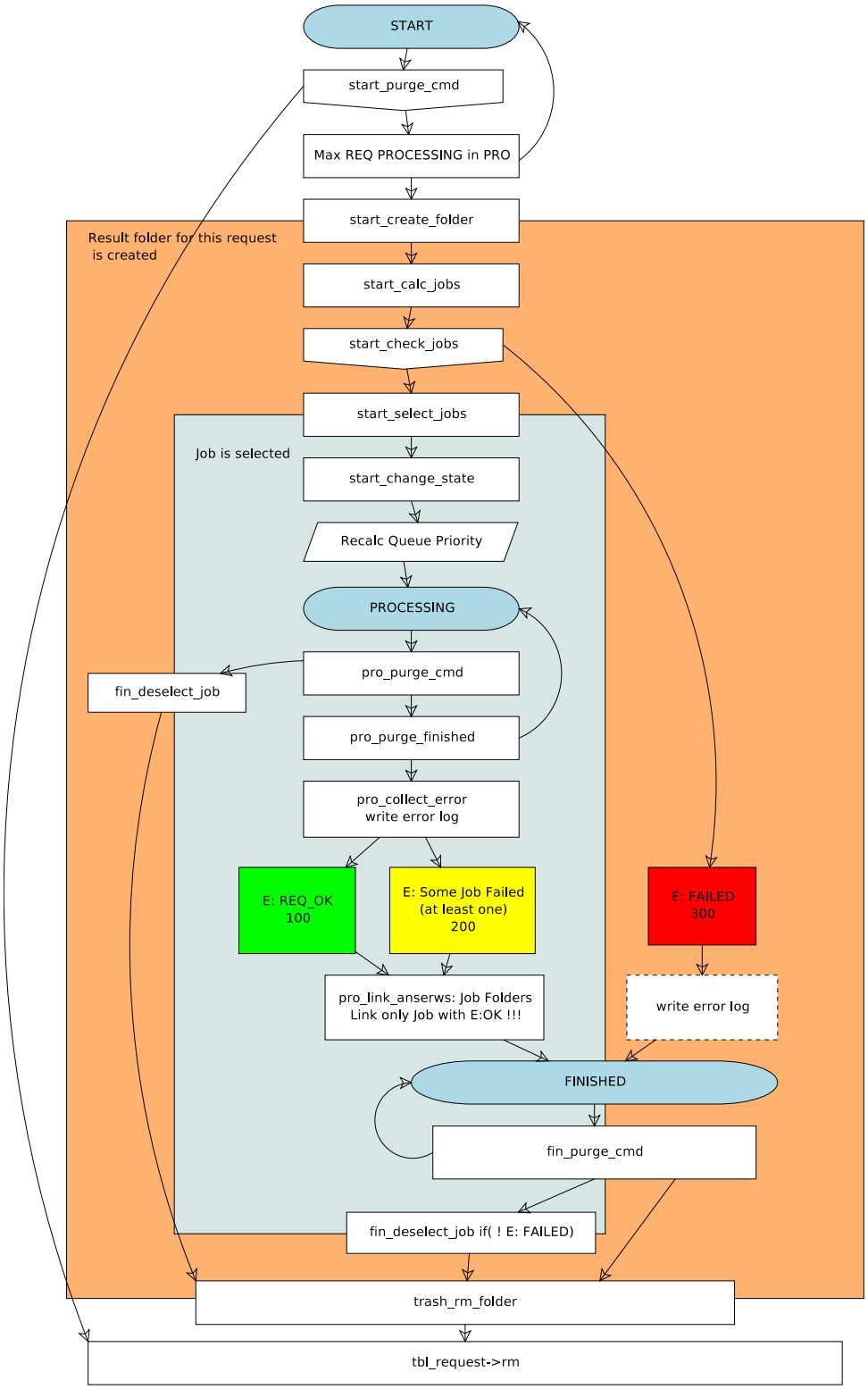


Figure 4.5: The finite state model of the worker 'mapper'

4.2.4. Jobber

The worker 'jobber' is responsible for processing the requested jobs. The jobber is, as are all workers of the Data Mole framework, designed as a finite state machine. The full state machine of the worker 'jobber' is illustrated in Fig. 4.6. The major part of this state machine is discussed in this section.

The timeline is divided as discussed in section 2.4 into different jobs. The parameter to split the timeline can be configured in the XML configuration file. Each job is identified with a unique number called job id, a warm up, start, stop and roll off time. The data structures of all jobs are saved as rows on the MySQL server and are accessed by the class job table.

Each job has a request counter. This counter is increased and decreased by the worker 'mapper' to indicate if the job is needed to answer a request. The worker 'jobber' tries to process all requested jobs. The results of each job are saved in a folder and are kept as long as the request counter of this job is larger than zero.

In a first step the module organizes the input data needed to process the requested job. The jobber adds a request for the input data of the required time period including warm up and roll off in the request table of this submodule (a download interface or an other GBM). The jobber then has then to wait until the request is answered. The job is then in the state 'waiting'.

The jobber polls the request table of the submodule to check when the request is answered. Then the job is sent to the job scheduler. At this time the job is marked to be in the state 'processing'. The job scheduler is responsible for process is the job on a free cluster node with the help of the user delivered binary. The job scheduler collects and stores the results in the job result folder. The jobber wait until the job is processed. Then the job is pushed into the state 'finished'.

The input data required to process this job is now no longer needed and therefore the worker 'jobber' marks the request that was submitted to the submodule as 'removable'. After this step the submodule has the permission to remove this request.

4.3. Downloader Part

Different modules are used to implement efficient access to the Jabba server. In this section we present the interaction that is needed to download the compressed hour files with the NetFlow data from the tape server. Fig. 4.7 displays the different modules that are used.

At a glance the module 'download indexer' creates a static index called 'jabba index' of all files of the DDoxVax project that are stored on the Jabba server. The module 'download master' organizes the download of the files. The download master uses the table 'file table' to store the dynamic data of the files, like the information if a file is requested or if the file is in the local file cache. The download interface is used to map the time-based request from the generic block module to the different files in the file table. The download interface increases or decreases the request counter in the file table to indicate if a file is needed or not. The download master creates a list of all files that are requested that are saved on the same physical tape and submits this list to a download client. The download client downloads the files into the local

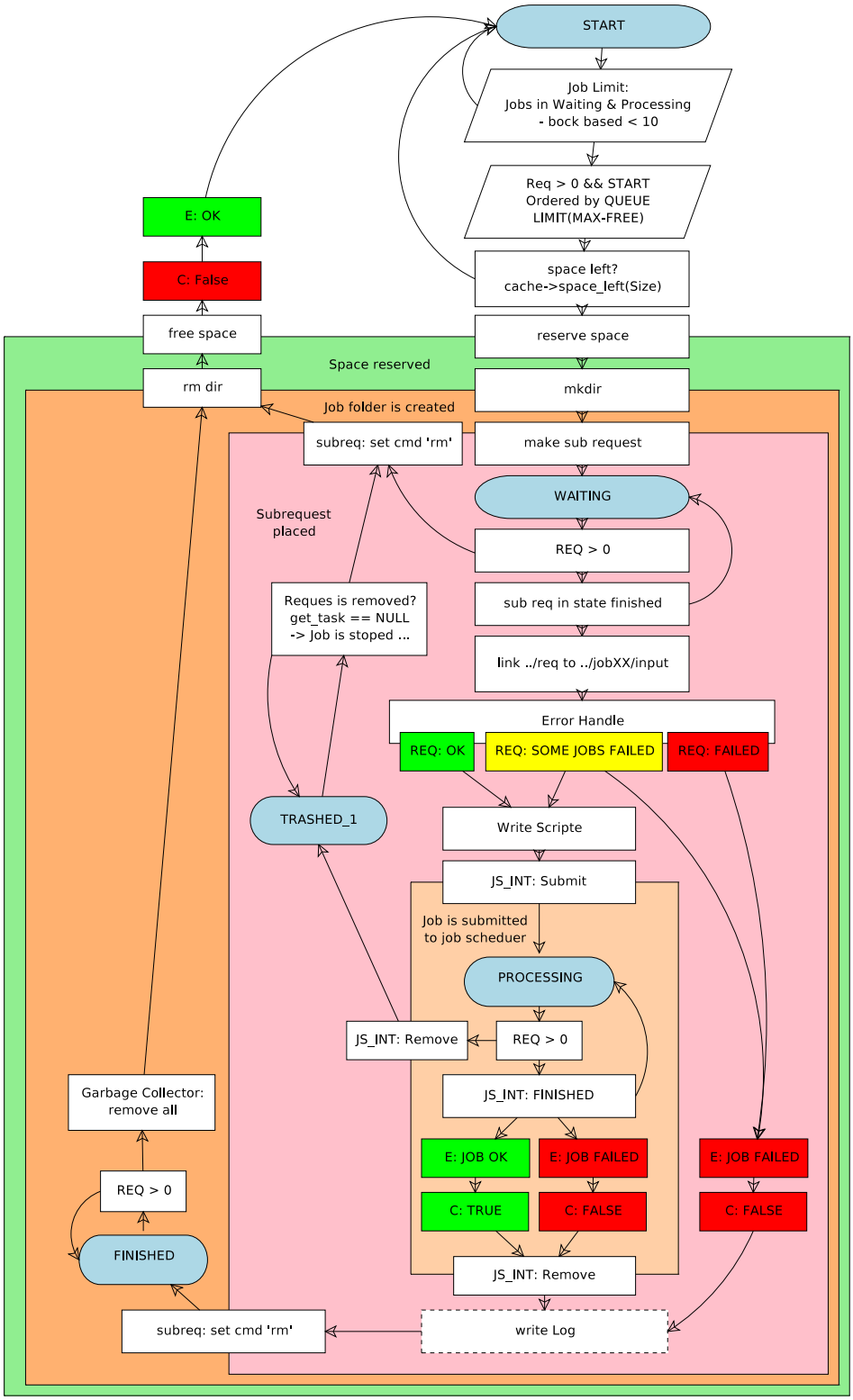


Figure 4.6: The finite state model of the worker ’jobber’

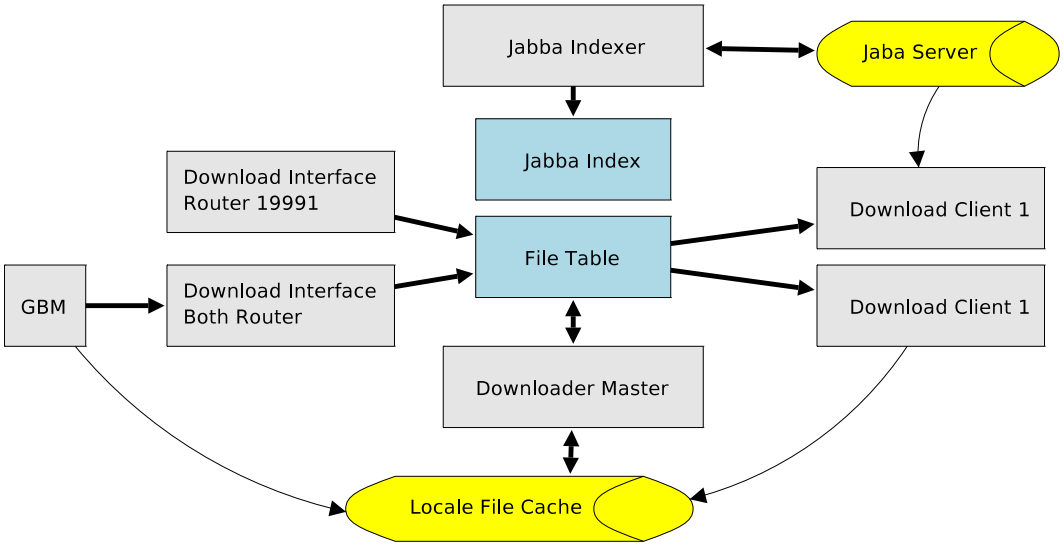


Figure 4.7: An overview of the downloader part

disk cache and updates the path information in the file table. The download interface checks if all files required to answer the request are downloaded. If so then the request is marked as 'answered' and the caller is able to run his analysis over this data set.

4.3.1. Jabba Indexer

The compressed NetFlow hour files from the DDosVax project are saved every hour on the Jabba tape server. The file names of these hour files are built form a router id, a consecutive number and the save time, for example 19993_00040897_1190494800.stat.bz2. The router id is 19991 or 19993 and identifies the border gateway router that has collected the NetFlow data. The consecutive numbering was started at 0 at the project start and is incremented by one at each file export. The save time corresponds to the time in Unix time when the file was exported from the border gateway router. This time can be used to estimate the content of the file.

The Jabba server uses SAM-FS as file system. The command 'sls' can be used to obtain information about the files on this file system, for example the physical tape id where the data is saved. The module 'jabba indexer' scans all folders of the DDosVax project with the command 'sls -D *' to obtain all the meta information about all existing files of the project.

In a next step the jabba indexer uses the unique consecutive number to build the file index. The module search as the meta file information for eache number. If the information is found, the module parses the data into the file index. If no information is found, the file is marked as missed.

4.3.2. Download Interfaces

The main task of the download interface is to map the time based request of a Generic Block Module to the corresponding files.

For example a Generic Block Module requests all flows that were collected during a given time A to B. Then the download interface uses the jabba index to create a list of all files that were collected during this time. Then all these files are marked as requested. This is done by increasing the request counter in the file table of all these files.

The download of the files is organized by the download master after this point. The download interface checks periodically if all files are downloaded and then marks the request as finished.

The DDosVax project collects the NetFlow data from different routers and saves them in different files. The download interface can be configured to download only the NetFlow data of one specific router or of all the routers. Different download interfaces with different configuration, can be run in parallel. This allows the Generic Block Modules to select the download interface that provides them it with the required NetFlow data.

The download interface consists of a request table and a worker 'mapper' which are identical to the mapper and request table of the Generic Block Module. This allows the GBMs to address the download interface just like a normal GBM. Furthermore the download interface consists of a download jobber worker and a standard job table. The download jobber worker initializes the job table with jobs that have the same duration as the compressed hour files of the DDosVax project. The mapper worker therefore automatically maps the time-based request to the files.

The download jobber worker maps the request of the job table to the file table. The full state machine of the worker 'download jobber' is illustrated in Fig. 4.8. The complexity of the state machine is increased due to the fact that the download interface is able to work in a so called copy mode, where the hour files are copied into the local cache of the donwload interface.

4.3.3. Download Master

The download master is the heart of the download part. This module checks the file table for files that should be downloaded and manages the local file cache. The process flow of the download master is illustrated in Fig. 4.9

The files from the Jabba server are downloaded of the download client into the local file cache. The size of this cache is limited by the configuration of the download master to avoid a disk overflow. When new files should be downloaded, the download master checks if there is enough space in the local file cache to store these requested files. If the local file cache is full, a garbage collector is started. The garbage collector removes all files from the local file cache that are no longer used (request counter smaller than one). If after this operation the free space in the local file cache is still too small, the download is paused until more files could be removed. Otherwise the download master submits the files for download to a free download client. For this purpose the download master updates the field 'client' with the id of the download client that should download this file.

The download master uses the information about the physical tape id of each file from the Jabba index to group files that are saved on the same physical tape. To optimize the download speed a group of files that lies together on the same tape is always submitted to one download

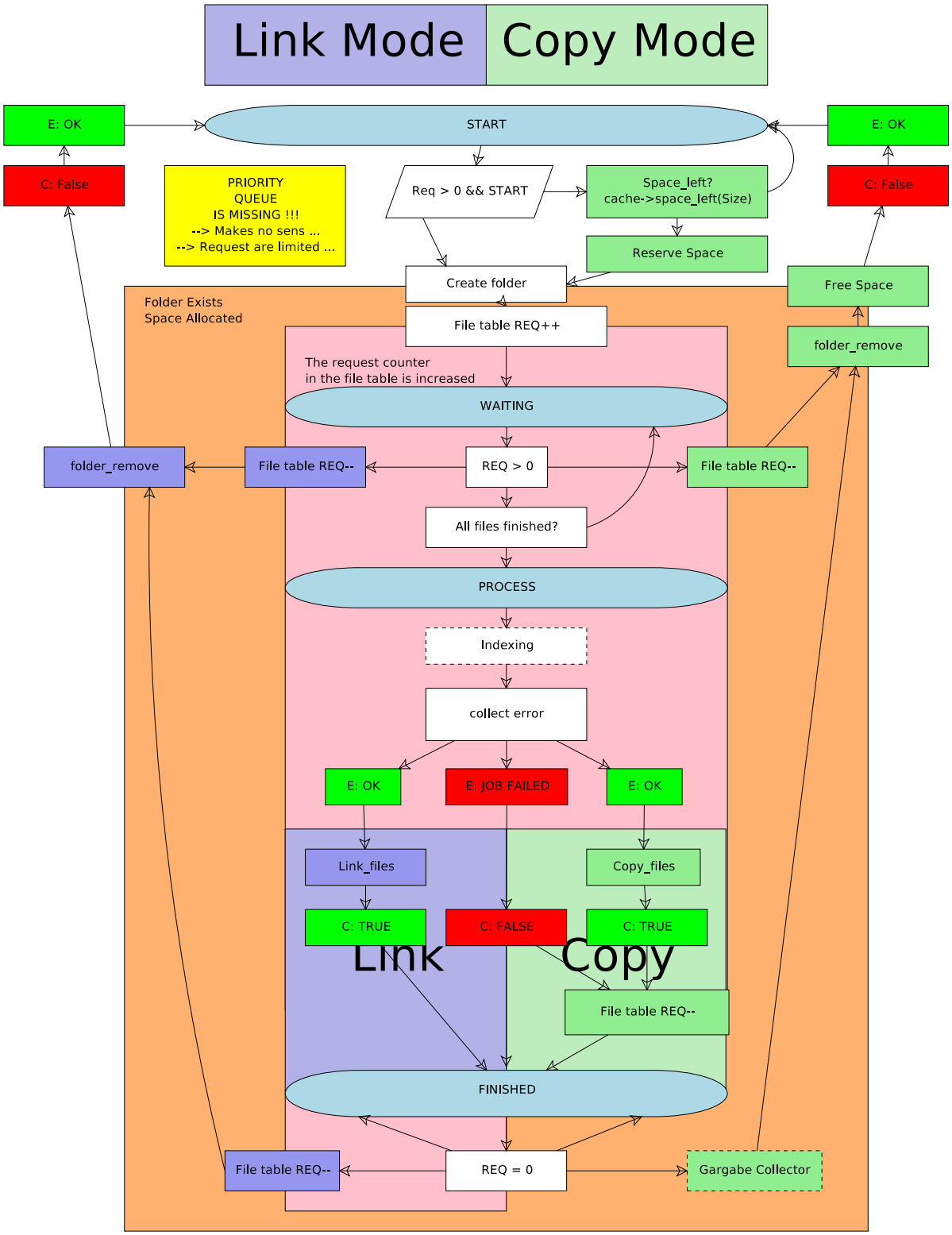


Figure 4.8: The finite state model of the worker 'download jobber'

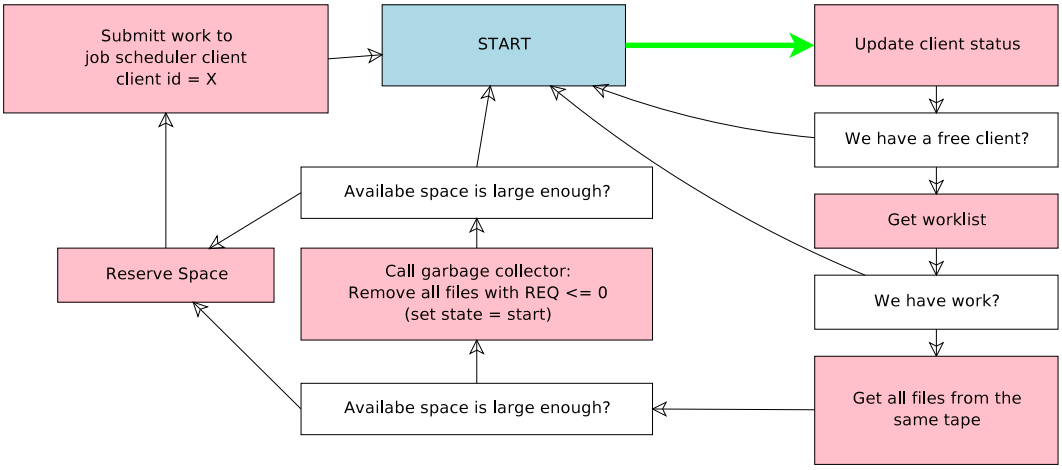


Figure 4.9: The process flow of the worker 'download master'

client.

4.3.4. Download Client

The download client searches the file table for files that should be downloaded. In a first step the download client uses the command 'stage' on the Jabba server to mark all files that should be downloaded. This command will force the tape server to copy the files from the tape to the hard disk of the Jabba server. In a second step the files are copied with the command 'scp' into the local file cache of the cluster. The path to the file in the local file cache is then saved in the file table. The file is marked as cached and finished. At this point the download client has finished its work and searches the table for new work.

To optimize the download rate several download client modules are used. Each module has its own unique id. A download client module processes only files that are tagged with his id. The download master uses this tag to distribute the workload over all download clients.

The finite state machine of the download client can be found in the appendix A.2

4.4. Job Scheduler Part

The job scheduler is used by the modules to process tasks on the computer cluster. In this section all modules that are used to implement the job scheduler are presented.

The job scheduler of the Data Mole consists of three major different elements. The job scheduler interface, that is used by the modules as interface to submit a job, the job scheduler master, that manages the queue of all tasks that can be processed and distributes this work to the job scheduler clients, and job scheduler clients, that process the task. The three elements use the table 'task' to communicate with each other. This is illustrated in Fig. 4.10.

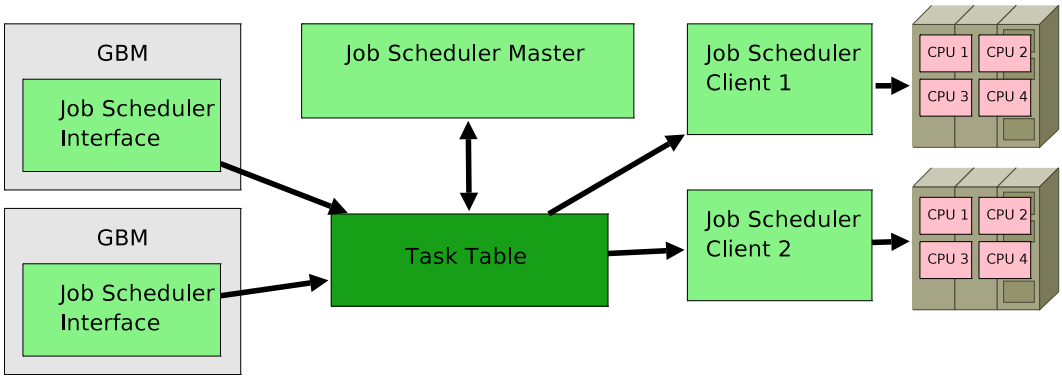


Figure 4.10: An overview of the job scheduler part

4.4.1. Job Scheduler Interface

The job scheduler interface is used by the Generic Block Module to interact with the job scheduler part. The job scheduler interface can be used to submit jobs to the table 'task' and can be used to check if the job have been processed.

The use of this interface class allows us to modify the job scheduler part at a later time without much effort, due to the fact that only this interface class has to be adapted to the new job scheduler design.

4.4.2. Job Scheduler Master

The job scheduler master distributes the runnable tasks among the different job scheduler clients that then process the work.

Each submitted task uses a bit field called 'client_mask' to indicate which clients can be used to process this task. The client_mask is a 64 Bit variable and therefore 64 different clients can be addressed. This makes it possible that tasks from different modules can be processed by different job scheduler clients.

The job scheduler master maintains a list with the usage of all job scheduler clients. If one job scheduler client has unused processing capacity the job scheduler master tries to find a task that can be processed on this client taking into account the client_mask restrictions. The task is submitted to the client by setting the 'client' field of this task to the id of the job scheduler in the task table . The job scheduler client will then process this task.

The process flow of the job scheduler master is illustrated in appendix A.2.

4.4.3. Job Scheduler Client

A job scheduler client is started on the physical node to which it was assigned and processes the individual tasks. The client will periodically check the task table to identify new tasks that should be processed. In a first step the job scheduler client creates a temporary folder in the local disk space and copies the input data into this folder (if requested). Then the parameter for the call of the user binary is created. The standard output and the standard error file

handler are redirected into a local file. Then the job scheduler starts the user binary and waits until the user process is finished. In a last step all the results are copied back and the local task folder is removed.

4.5. Web Interface

A web interface was programmed to manage all the different modules of the Data Mole project. It can be used to install and remove modules from the framework. Furthermore the web interface is used to control and monitor the different modules.

Each module is defined by an XML file. To install a new module the XML file has to be uploaded via the web interface into the module manager. After this step the module is registered in the module table and can be controlled over the web interface.

The names of all installed modules are visible on the left site of the web interface. A specific module menu is reached with a left mouse click on a module name. The module menu has at least the following three entries configuration, control and log.

The configuration menu is used to edit the parameters of this module. The module registers these changes after the next restart. The control menu is used to start, stop or pause the module. The log menu displays the last log entries of the module. To use this feature the DB-Logging target has to be activated in the XML configuration.

Different module types have additional menu entries. For example a module that is based on the Generic Block Module has a request menu. The request menu is used to manage all requests of a specific module. This menu can be used by a user to make a module perform a certain measurement over a given time.

5. Evaluation

5.1. On-Off Capability

The modules of the Data Mole framework save their internal state during runtime in the MySQL database server. If a module receives the command to stop from the web interface, the module just finishes its works and returns to the parent process. No special backup routine has to be called. When the module is started again, it loads the latest internal states from the MySQL server and resumes its work.

But what happens if the module is stopped by a power outage of the computer? Due to the fact that the module updates the internal state during runtime, all information needed to resume work is already saved in the MySQL server. After the reboot of the cluster node, the modules can be restarted as if they were stopped.

This robust design was stressed during a node failure on 6. September 2007. During an ongoing measurement a cluster node was switched off by the internal monitoring card. All modules running on this node were killed. After the node was rebooted, the modules could be restarted as if they had been stopped.

The other modules of the framework were not effected by this power outage. This is possible due to the fact that the other modules post their requests for the killed modules directly on the MySQL server. The MySQL server will store these requests and the killed modules will process them when it is started again.

5.2. Ease of Use

An other important requirement on the Data Mole framework is usability. The user interacts with the Data Mole framework at two points in time: during the building process of a new user metric module and during the measurement phase when the user interacts with the web interface to control and to monitor the analysis process.

During this work three different metric modules were implemented and successfully used the Data Mole framework. The Counter-Strike Server traffic analysis of Lorenz Breu, Zattoo Observer of Daniel Koller and Basic Metrics Module of the developer of this framework. In all three projects the metric module was built on the top of the Generic Block Module as discussed in section 3.2. For all three project the XML configuration file to build the metric module was written in less than one day.

The web interface to control the modules was presented to different users. After a short introduction to the background of the framework the users were able to use the web interface

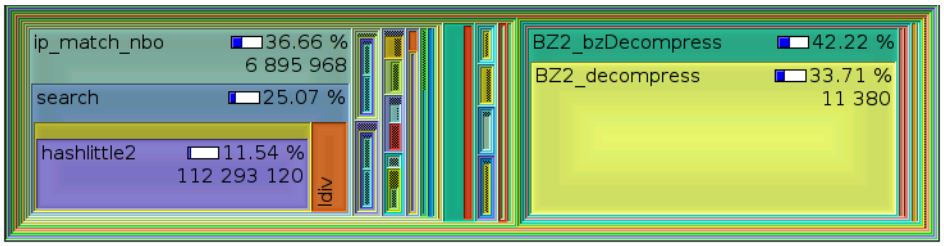


Figure 5.1: The processing time of the basic metrics module

to control and monitor the different modules in less than 2 hour.

Based on this experience we can conclude that the Data Mole fulfills the requirement of Ease of Use.

5.3. Performance

The throughput of the Data Mole network is limited by two main factors, the download speed from the Jabba server and the processing power of the cluster nodes.

The Download speed is bounded by the physical speed of the tape reader devices of the Jabba server. The theoretical limits and the possible optimizations are discussed in section 2.3.1.

The processing power of the cluster should be efficiently used to process the NetFlow data. To increase the effectiveness the runtime of each analysis process should be minimized. The processed metrics were designed with the help of the Software Optimization Guide for AMD64 Processors [29] and have been profiled with the valgrind [28] program to detected and remove performance bottlenecks. Fig. 5.1 shows a performance analysis of an optimized module that collects a set of basic metrics. This evaluation shows for example that 43 % of the processing time is spent on decompressing the NetFlow record files (bz2_bz_Decompress). 36 % of processing time is spent on a hashtable lookup that is used to resolve the autonmous system number of an IP address. Only 21 % of the processing time is used to sort of the netflow records and calculate of the metrics themself.

The charging level of the local file cache can be used to identify if throughput of the Data Mole is limited by the processing capacity of the nodes or the download. In Fig. 5.2 the relative cache usage during a measurement performed the Zattoo Oberserver of Daniel Koller is plotted.

After the request was placed in the metric module on Friday the data mole started downloading the necessary NetFlow records. Usage of the local file cache increased linearly form Friday to Saturday. During this time the throughput of the Data Mole framework was limited by the read performance of the Jabba server. On Saturday the download was stopped and the used cache size was stable. At this point the downloader module had organized enough input data that all cluster nodes were provided with runnable jobs. After this point the framework was limited by the processing capacity of the cluster. On Sunday midday the first jobs were

finished and the download module continued with the download. The maximal local cache size was then reached and the garbage collector was started. The garbage collector removes the unused NetFlow records from the cache. The event is visible as a rapid decrease of cache size in Fig. 5.2.

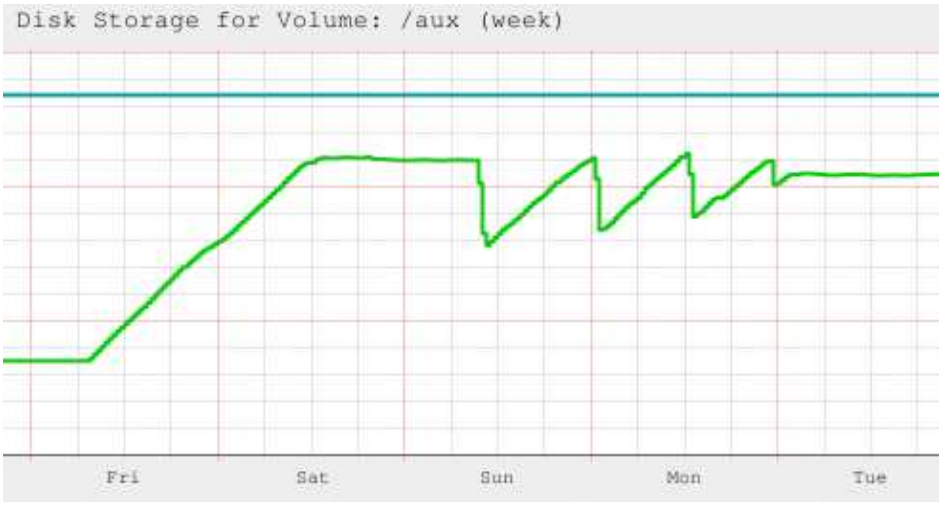


Figure 5.2: The size of the local file cache

After observation of activity of the download path and the CPU usage of the cluster during the measurements of these three projects, we can conclude that the performance of the Data Mole framework is only limited in the start time of the measurement by the download capacity of the Jabba server. Then the throughput is primarily limited by the computational resources.

5.4. Set of Basic Metrics

A module that collects a set of basic traffic metrics was designed and implemented during this work. This section presents the module and the results that were found by using this module.

5.4.1. Structure

The module is built on the top of the Generic Block Module. Therefore, as discussed in section 3.2, only the pure traffic analysis binary has to be built. The structure of the binary is illustrated in Fig. 5.3.

The binary is fed with the unsorted compressed NetFlow hour files from the DDosVax project. In a first step these files are decompressed and the timing of each flow is recalculated. Then the flows are submitted to the flow sorter.

The flow sorter performs a bucket sort of the incoming flows in a first step. The buckets of the last three hours are saved on the local disk. After the first flow of the fourth hour is detected the oldest hour is processed. The flow sorter sorts the buckets of this oldest hour according to the flow start time with a merge sort. To optimize performance all records of one bucket should fit into local memory. For 2007 a bucket size of about one minute is used.

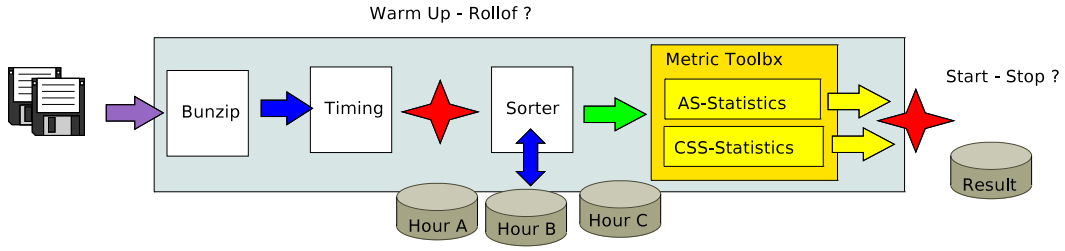


Figure 5.3: The structure of the basic metric user binary

The sorted flows are then submitted to the metric toolbox. In this toolbox different metrics that should be calculated can be activated. The results of these metrics are then saved as comma separated values into a local file.

5.4.2. Metrics

We have implemented four different metric plug-ins for the toolbox. The first plug-in, called 'trivial metrics' collects the overall byte, flow and packet counts. These metrics are useful to check if the input data has the desired characteristics. The plug-in 'IP metrics' collects different metrics like absolute counts, unique count and entropies based on the SRC and DST IP address of the transmitted flows. The plug-in 'AS metric' collects metrics based on the autonomous system number of each flow and the plug-in 'protocol metrics' collects metrics based on the port number of the protocol. All metrics are calculated separately for the different flow directions.

5.4.3. Results

The basic metric module is used to process a measurement on the traffic of the last four years. Due to limited time only two weeks of each year were processed. The module collected more than 728 different measurement points per interval (interval duration 15 min) over this period. In this section we discuss only the most important results from this analysis.

Fig. 5.4 shows the number of transmitted bytes over all border gateway routers of the switch network. The x-axis of these plots correspond to time index. On the left side we start with Monday midnight and end on the right side with Sunday midnight. The y-axis corresponds to the number of transmitted bytes in the last 15 minutes.

From this result we can conclude that the overall traffic volume has increased as expected over the last years. But the volume has not increased linearly. The results show that the traffic volume increased heavily between the year 2005 and 2007.

Fig. 5.5 shows the transmitted flows over all border gateway routers during the same time. The interesting point is that the number of flows was quite stable over the years 2003 to 2006. But in 2007 the number of flows per 15 minute interval increased heavily. Due to the fact that the traffic volume in 2007 is quite the same as in 2006, we can conclude that the additional flows did not significantly affect the traffic volume.

Based on these two results we are able to demonstrate that the traffic characteristic has

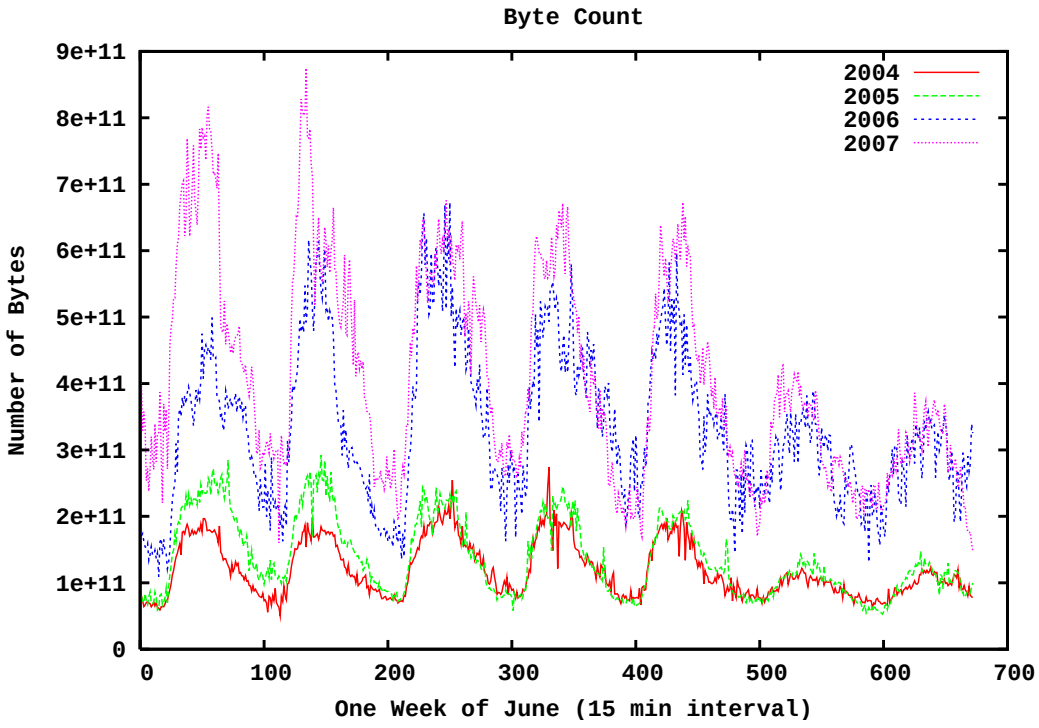


Figure 5.4: The byte counter metric

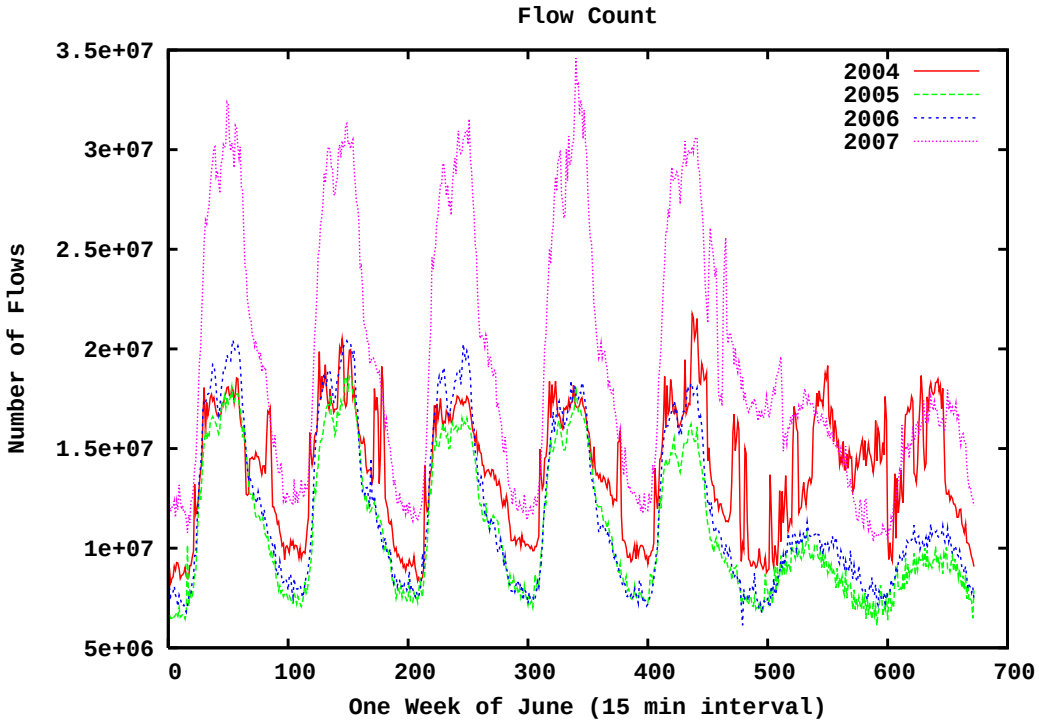


Figure 5.5: The flow counter metric

changed over the years. The change in traffic characteristic is likely to impact e.g. threshold based anomaly detection metrics and needs to bee assessed in more detail in future work.

6. Conclusion

This project started with the idea to perform a long term traffic analysis on archived NetFlow data. In the beginning of the project we realized that we need a tool set to manage the large number of files and jobs. Based on this need, we launched the Data Mole project. The result of this thesis is a first running version of the Data Mole Framework.

The next section summarizes the key achievements of this work. The section 'future work' discusses the major work packages that should be implemented to improve the Data Mole.

6.1. Summary

This work presents an efficient and modular framework for short and long term traffic analysis of Cisco NetFlow data.

The Data Mole provides a generic module that can be used for fast integration of block based traffic analysis. The effort required to create a new metric module based on this Generic Block Module is reduced to providing the pure traffic analysis routines and to adapting an XML based configuration file. This allows the integration of existing projects in less then one day.

The Data Mole automatically divides a job of long duration into smaller jobs. Each job is independently processed and monitored by the Data Mole. The framework is able to manage the state of more than 100'000 jobs in parallel. This is possible as a database system is used in the background to manage the jobs. This makes it possible to perform measurements over long time periods that are divided into thousands of small jobs.

The platform provides a web based monitor and control interface. This graphical interface provides the user with an overview of the progress of the measurements, allowing him to detect problems like failed jobs and crashed modules in just a few seconds.

The architecture of the Data Mole allows the starting and stopping of the framework during ongoing measurements. This feature makes it possible to switch off all or only some dedicated cluster nodes for maintenance without affecting the measurement. The data path of the Data Mole is optimized to provide an efficient analysis of the NetFlow data. Using an optimized download strategy the effect of the bottleneck of the file download can be limited. The analysis of traffic data from 2007 has shown that the bottleneck is shifted from the downloader path to the processing of the NetFlow data. Or in other words, we can download the NetFlow data faster than we are able to process it on the cluster.

6.2. Contribution

In this work an object oriented library in C++ has been designed and implemented that can be used to create different modules. Based on this library several modules have been designed and implemented that provide on the one hand a transparent and fast access to the NetFlow records on the long term storage system and on the other hand a basic job scheduler functionalities. In the end a Generic Block Module was created to allow fast integration of existing and new block oriented traffic analysis metrics.

A monitoring and control web interface based on an object oriented php5 library has been built.

Based on existing code, a new NetFlow processing template for long term traffic analysis has been created. The template includes a fast and robust stream based NetFlow record sorter. The interface of the template is adjusted to the Generic Block Module. Based on this template a module to collect basic traffic metrics like flow counts, IP entropy or protocol port entropy has been created. This template calculates more than 720 metrics.

Additionally the integration and running of several other NetFlow evaluation have been supported:

- Counter-strike server traffic analysis, Lorenz Breu
- Zattoo Observer, Daniel Koller

6.3. Future Work

The Data Mole in its first version provides a simple job scheduler. This scheduler keeps a list of runnable jobs and distributes the jobs on the available computers. To decide if a job is to be submitted to a node or not only the number of already started jobs on this node is taken into account. This simple scheduler decision metric is not optimal. It can and has to be improved. For example a better approach would be based on the resource requirements of a job. These requirements, such as number of CPUs, size of memory and the size of the local disk space could then be compared with the available resources of the nodes. Based on this comparison the decision should be done.

An other work package should address the multi-user capability of the framework. At the moment all tasks from all users are handled with the same priority. A fair resource allocation based on different users has not yet been integrated.

Furthermore, a generic streaming module for the framework could be created in addition to the block processing module. This generic streaming module could be used to process metrics that e.g depends on state aggregated over the whole measurement interval. The key feature that should be implemented by this streaming module is application checkpointing.

A. Appendix

A.1. Task Description

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

February 26th, 2007
Bernhard Tellenbach, Daniela Brauckhoff

Master Thesis:

Analyzing network traffic from the SWITCH network from 2003 until today

for Dominik Schatzmann <schadomiee.ethz.ch>

1 Introduction

Since the foundation of the modern Internet in 1992, it was subject to significant structural and technical changes. On the one hand, these changes are inflicted by its massive growth in users and content. But on the other hand, they are a result of the fact that the Internet is responsible for a constant output of new services and applications. Nevertheless, there is also a ”dark side” of the Internet: It is a huge playground for criminals and grumblers. In the arms-race between security professionals and cyber-criminals, a lot of different technologies and methods to protect home-users and networks were developed. This includes intrusion detection/prevention systems, anti-virus software and (application-level) firewalls.

A significant amount of these technologies rely on some form of anomaly detection or behavioral patterns. Examples are: [8](clustering), [1](fast spreading worms) and [11](spam) for large scale applications and [7] or [13] for host/small scale applications. Furthermore there are multiple publications that investigate properties of different metrics using data that is restricted to a rather short extract of the timeline (hours, days and sometimes weeks). [4] investigates the influence of sampling on some metrics used for anomaly detection (mainly entropy [1]), [14] studies the persistency aspects of Internet flows (”heavy hitters”) and [3] studies characteristics of P2P flows. Another closely related field is the influence of the network type on selected traffic metrics/patterns. [2] investigates e.g. upstream traffic behavior based on data collected from Broadband Fixed Wireless (BFW) and Digital Subscriber Link (DSL) access services.

If we consider a changing environment like the Internet, it is likely that the definition of what is ”anomalous” has to be updated/revised continuously. The following methods could be used to perform the updates:

- Adapt the definition using manual inspection and plausibility checks whenever necessary. Problem: Time consuming and slow.

[1] see [9] for an information theoretical evaluation of the entropy metric

- Use the last X measurements of the metric to adapt the definition of ”anomalous” (e.g. used in early warning systems/signature generators that are based on byte frequency distributions). Problems are e.g.: Selecting a ”good” value for X or evasion if X and the anomaly detection algorithm is known to the attacker.

But our claim is that if we could improve our understanding of the mechanics and properties of the Internet at different levels of detail (autonomous systems, national, metropolitan and enterprise networks), it will be possible to derive models that include future changes in a more reliable way. Ideally, there is no need for updates at all.

Despite the fact that research was able to reveal a lot of interesting properties, little is known about their long term development. The reason for this is that collecting and storing the required data for medium scale national network is hard and becomes unmanageable for huge ASes. But knowing the past is a first step toward a better understanding of the mechanics and properties and might therefore help a lot in developing better models of what is anomalous. At least there is strong evidence in the form of a variety of algorithms in the area of anomaly detection that rely on the past to predict the future (e.g. [15]).

In the course of the DDoSVax [6] project, an infrastructure to collect and store information about the network traffic crossing the borders of the Swiss Education and Research Network SWITCH was set up at our institute. Furthermore, because the project started in 2003, it brings us in an unique position to look at the long term development of different properties of the Internet at the scale of a national educational network. Until now, only parts of our dataset have been investigated (each with a very specific focus).

In this thesis, we want to make a first step toward understanding the long term development of selected properties/metrics. Some of these properties/metrics will be new and have to be developed during the thesis.

The DDoSVax Project

In the joint ETH/SWITCH research project “DDoSVax” aggregated network traffic data (Cisco NetFlow [5]) is collected at all border gateway routers of the Internet backbone operated by SWITCH [12]. This data contains information about which Internet hosts were connected to which others and how much data was exchanged over which protocols.

The DDoSVax project provides archived NetFlow data as well as a near real-time framework (named UPFrame) with plug-in support for online processing of NetFlow data received from routers.

2 The Task

This master thesis consists of the following two major tasks:

- **Management tools:** Design, implementation and evaluation of a set of management tools to set up and control netflow data processing and evaluation.

- **Extraction of flow metrics:** Design, implementation and evaluation of a set of tools that extract different flow metrics.
- **Long term analysis:** Select two or more of the extracted metrics, analyze their long term development in detail and put it into the context of the development of the Internet in general.

These tasks and their subtasks are described in the following two subsections.

2.1 Management Tools

First, some facts about our NetFlow data set:

- **Time interval:** March 2003 - today
- **Hourly volume (compressed):** 500 MB
- **Total estimated volume (compressed) until 22.02.07:** 15 TB
- **Storage locations:**
  - Archive on jabba (tabe-library). Avg. download speed: 5 MBytes/s
  - Data of the last 20 days on aw3 (/aux/ddosvax/pulled/old/)
- **Completeness:** Only a few gaps (outages) or corrupt files (exact number unknown). A partial (incomplete) log already exists (Task: complete it).

And some facts about our (new) cluster:

- **1 Login-Server (kom-pc-aw4):** AMD Athlon MP 2800+ (dual core), 2.1 GHz, 2 GB RAM (old file server)
- **1 Fileserver:** 2x DualCore Opteron 275 with 8 GB RAM and approx. 2.5 TB data storage (redundant, RAID 6).
- **5 Nodes:** 2x DualCore Opteron 275 with 8 GB RAM and 4 TB data storage (no backup, no redundancy!).
- **Network:** 1 GB Ethernet

In order process our entire netflow data archive and with respect to the above infrastructure, the management tools have to accomplish the following tasks:

1. Download the required files from jabba and preprocess (e.g. sorting, splitting,...) them. This includes sanity checks for corrupt files or actions/alerting in case of missing files.
2. Initiate and control the processing of the downloaded files.
3. Initiate and control post-processing steps. E.g. graph generation or generation of mean, variance, std.deviation or later on, for other derived quantities defined in the ”long term analysis” task. [2]
4. Cleanup if necessary (local storage control) and backup results from already processed files.

Furthermore, it is important that these tools are designed in a way that they are easy to configure and to adapt to other tasks. A modular and robust design is the most important criteria for the assessment of the management tools.

[2] ’Thís task should be scheduled for the time when the processing is running. Therefore, ”other” quantities will not be available at the start of the data processing

2.2 Extraction of Flow Metrics

This task consists of the following subtasks:

- **Define the flow metrics to be extracted:** For the autonomous system (AS) stuff, see [10]. The required IP to AS tables should be provided by the tutor. The following flow metrics are mandatory(if feasible):
  - Packets/Flows/Bytes/ASes per interval
  - Unique SrcIP/DstIP (direction IN/OUT respectively) and unique SrcAS (direction IN) and DstAS (direction OUT) count per interval
  - Entropy: SrcIP/DstIP, SrcPort/DstPort, SrcAS/DstAS per interval (directions like for the unique count)
  - Peer-Tracker statistics (contact Lukas Haemerle)
  - Flow-length distribution per interval (curve fitting(?)/define classes)
  - Packet-size distribution per interval (curve fitting(?)/define classes)
  - Flows per port (1024+top XXX) per interval
  - Two or more additional metrics defined by the student.

  For the mandatory metrics this involves the definition of interval size, pre-filtering (protocol/subnet/router...) and storage format. Furthermore, estimations of complexity, required main memory and storage space have to be provided as a basis for the feasibility analysis. The feasibility analysis should be based on the limitations set by our infrastructure. If applicable, specify simplifications/abstractions of a given metric in order to make its extraction feasible.
- **Design the processing tools:** Propose a design for the software that extracts the metrics defined in the previous step. This includes at least the core algorithms, the format(s) of the results, a concept to store them in a feasible way, a concept for load distribution and robustness considerations.
- **Implement the processing tools:** In C or C++ (the basic toolset is written in C, but C code can be integrated in C++).
- **Evaluate the processing tools:** Run a set of tests using data from approx. one day. Functional tests: Are the calculated metrics correct? (Using sanity checks and/or manual inspection). Robustness: Memory leaks, not enough diskspace, file access error (e.g. broken connection to file-server).

2.3 Long Term Analysis

In this task, two or more of the extracted metrics should be selected. Furthermore, their their long term development should be analyzed in detail and it should be put into the context of the development of the Internet in general.

For the analysis of the long term development of time series (including forecasting), there exist multiple well-known techniques. Depending on the metric, these techniques might consist of trend analysis with respect to mean, variance and std. deviation. Other techniques might include identification of daily/weekly/monthly/yearly rhythms and/or estimation/investigation of parameters of process models for time series (see http://www.statslab.cam.ac.uk/ rrw1/timeseries/t.pdf for a headstart).

3 Deliverables

During this thesis the following deliverables will be produced:

1. A concise and detailed documentation of the conducted work.
2. The code and installation instructions for the data management tools.
3. The code and installation instructions for the data processing tools.
4. The code and installation instructions for the data evaluation tools.
5. A ready-to-use installation on our computing cluster along with its documentation.

3.1 Documentation structure

The report should contain the steps conducted along with the findings/consequences/results, lessons learned, summary and an outlook on future work [3]. All designs should be described in detail and design decisions should be motivated. Evaluations have to be academically sound. In case of measurements of stochastic quantities, this means e.g. multiple runs and indication of confidence intervals. Finally, the report should contain an addendum with installation and usage instructions for the developed tools.

The code of all of the in this thesis developed tools should come along with a detailed documentation. The code should follow a coherent and clean coding and commenting style so that the code is easy to understand and use. If applicable, the documentation could be generated using automated documentation tools (e.g. doxygen).

If important new research results are found, a paper might be written as an extract of the thesis and submitted to a computer network or security conference.

4 General Information

Dates/Meetings:

- This master thesis starts on Monday, 19.03.2007 and finishes on Monday, 19.09.2007. It lasts six months in total.
- Informal meetings with the tutors will be held at least once a week.

Presentations:

- Two intermediate informal presentations for Prof. Plattner and all tutors will be scheduled two and four months into the thesis.
- A final presentation at TIK will be scheduled close to the completion date of the thesis.

Supervisors:
Tutor: Bernhard Tellenbach, tellenbach@tik.ee.ethz.ch +41 44 632 70 06, ETZ G97
Co-Tutor: Daniela Brauckhoff, brauckhoff@tik.ee.ethz.ch, +41 44 632 70 50, ETZ G97
Co-Tutor: Arno Wagner, wagner@tik.ee.ethz.ch, +41 44 632 70 04 , ETZ G95

[3] Additional information and tips are available on the thesis wiki at `http://tikiwiki.ethz.ch/thesis/index.php/Main/HomePage`.

Literatur

[1] B. A. Wagner. Entropy based worm and anomaly detection in fast ip networks. In *14th IEEE International Workshops on Enabling Technologies Infrastructures for Collaborative Enterprises (WET ICE 2005)*, 2005.

[2] D. M. Amit Sinha, Kenneth Mitchell. Flow-level upstream traffic behavior in broadband access networks: Dsl versus broadband fixed wireless. In *IPOM2003*, 2003.

[3] L. H. Arno Wagner, Thomas Dübendorfer and B. Plattner. Flow-based identification of p2p heavy-hitters. In *International Conference on Internet Surveillance and Protection (ICISP) 2006,*, 2006.

[4] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina. Impact of packet sampling on anomaly detection metrics. In *Internet Measurement Conference 2006*, Rio de Janeiro, Brazil, 2006.

[5] Netflow services solutions guide. `http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/netflsol/nfwhite.htm`.

[6] Ddosvax. `http://www.tik.ee.ethz.ch/~ddosvax/`.

[7] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261, New York, NY, USA, 2003. ACM Press.

[8] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proceedings of ACM SIGCOMM 2005*, August 2005.

[9] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. IEEE Sumposium on Security and Priovacy, Oakland, CA, May 2001.

[10] J. Oberheide, M. Karir, and D. Blazakis. Vast: Visualizing autonomous system topology. In *Proceedings of ACM 3rd International Workshop on Visualization for Computer Security(VizSEC)*, November 2006.

[11] A. Ramachandran and N. Feamster. Understanding the networklevel behavior of spammers. 2006.

[12] SWITCH. Swiss academic and research network. `http://www.switch.ch/`, 2006.

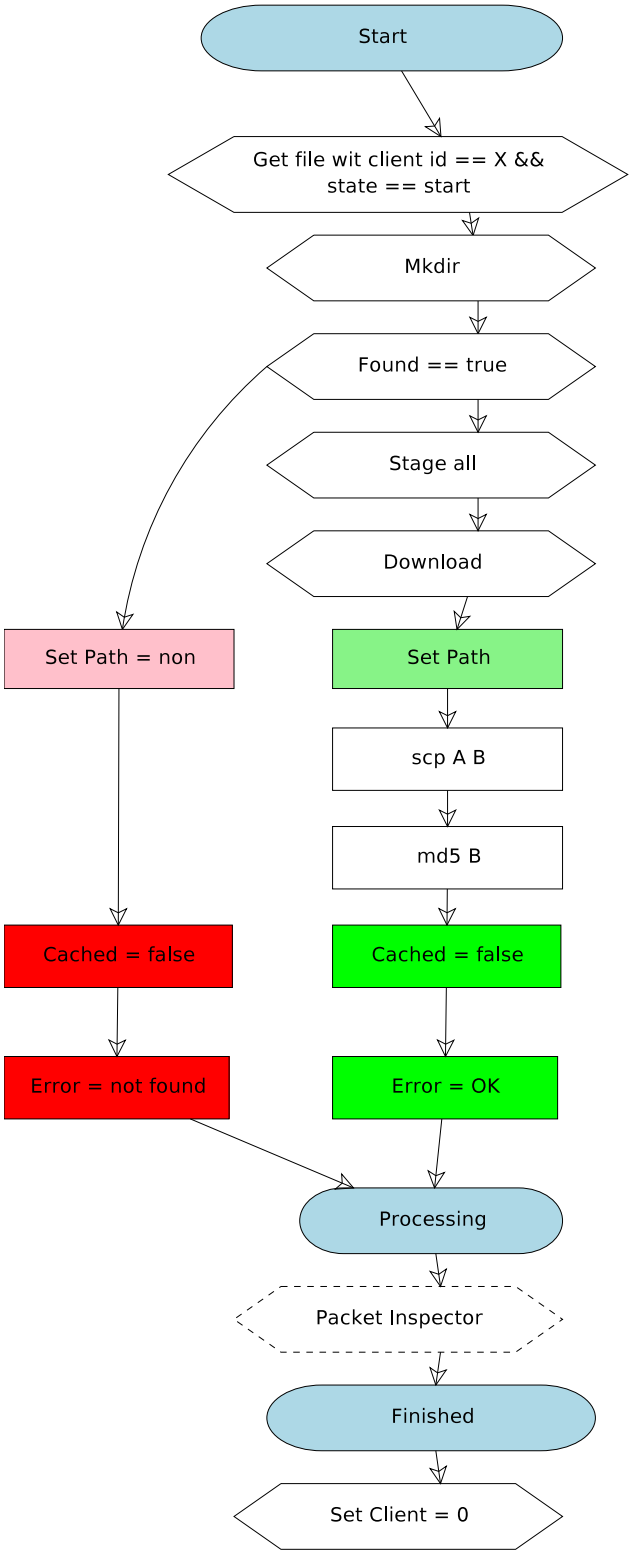
[13] Y. Tang and S. Chen. Defending against internet worms: A signature-based approach. In *Proceedings of IEEE INFOCOM*, March 2005.

[14] J. Wallerich, H. Dreger, A. Feldmann, B. Krishnamurthy, and W. Willinger. A methodology for studying persistency aspects of internet flows. *SIGCOMM Comput. Commun. Rev.*, 35(2):23–36, 2005.

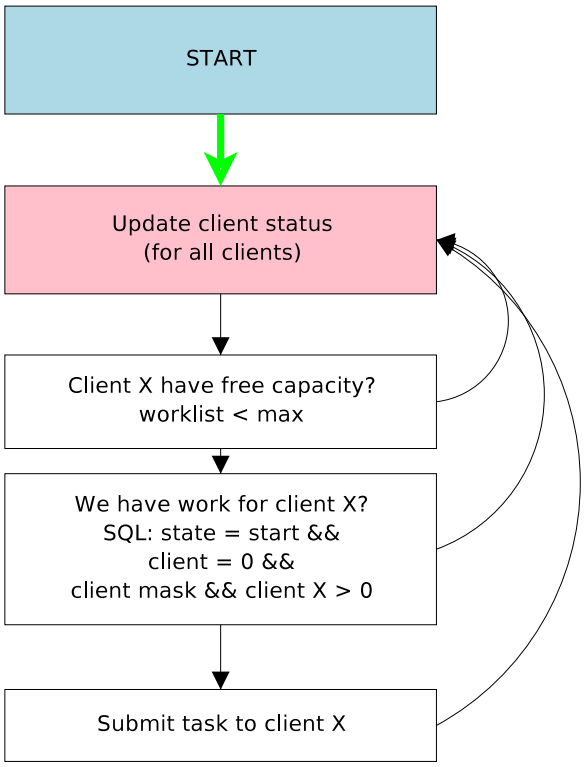
[15] Q. Wu and Z. Shao. Network anomaly detection using time series analysis. In *Proceedings of the IEEE Joint International Conference on Autonomic and Autonomous Systems 2005*, 2005.

A.2. Finite State Modules

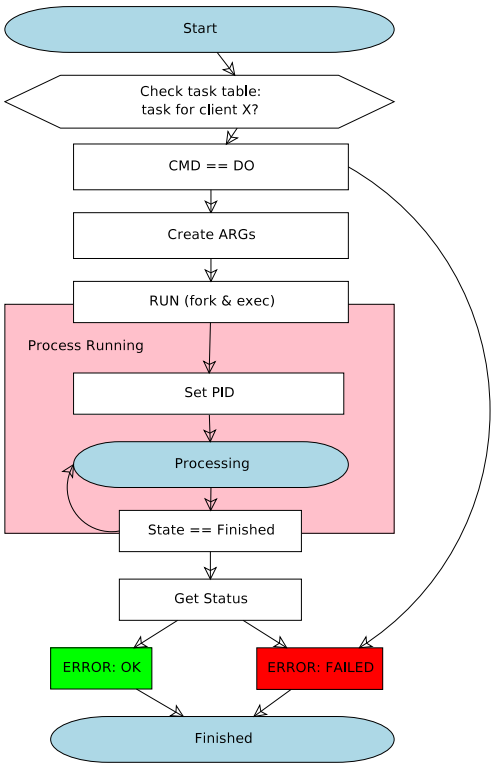
A.2.1. Download Client



A.2.2. Job Scheduler Master



A.2.3. Job Scheduler Client



A.3. GBM Example Configuration

Listing A.1: Multi-Page Java Code

```
<?xml version="1.0" encoding="UTF-8"?>
<jobdescription>
<module_config>
<!-- Warning: TYPE_LENGTH < 13 -->

<!-- Unique module id -->
<param type="module_id"   >7000</param>
<param type="module_name">basic_metric</param>
<param type="module_type">MD_Block</param>

<!-- Module owner -->
<param type="module_owner">schadomi</param>
<param type="module_group">studis</param>

<!-- Module Binary-->
<param type="MD_BIN_CMD" >/home/schadomi/dm/md/md_block -m basic_metric -h 10.0.0.100 -s

<!-- DEBUG -->
<!--
      D_STEP:
      Wait after each message on a user key interaction?
      'false': (Default)
      'true': Implements a very simple debugger
-->
<param type="D_STEP">false</param>

<!-- LOGGER -->
<!-- Log sources: -->
<param type="LS_framework" >true</param>
<param type="LS_dbc"       >true</param>
<param type="LS_dbq"       >false</param>
<param type="LS_config"    >true</param>
<param type="LS_logger"    >false</param>

<!-- Log level selection -->
<!-- LL_error is always on ... -->
<param type="LL_debug"    >true</param>
<param type="LL_warning"  >true</param>

<!-- Log Target Selection -->
<!-- Log Error & Warnings to the Systemlog? -->
<param type="LT_syslog"     >false</param>
<!-- Log to the console? -->
```

```
<param type="LT_console"     >false</param>
<!-- Log the exit code to the DB? -->
<param type="LT_db_log_err" >true</param>
<!-- Log the full 'msg' in the DB-Log? -->
<param type="LT_db_log_txt" >true</param>
<!-- Log to file -->
<param type="LT_file"        >false</param>
<param type="LT_file_p"      >/dev/null</param>
<!-- Log to error and warning as email [Not Implemented Yet]-->
<param type="LT_email"       >false</param>
<param type="LT_email_@"     >schadomi@ee.ethz.ch</param>

<!-- MD_Base-->
<!-- The number of seconds the module will sleep
     after a completed 'main' run. This coressponds
     to a the lower limit of the polling intervall.
-->
<param type="MD_B_poll_int"    >120</param>
<!-- The number of seconds the module will sleep,
     when it get's a wait cmd.-->
<param type="MD_B_wait_int"    >180</param>

<!-- MD_Block -->
<param type="MD_B_req"    >basic_metric_req </param>
<param type="MD_B_job"    >basic_metric_job </param>
<param type="MD_B_sreq"   >dl_interface_all_req </param>

<!-- TBL_Job -->
<!-- Describes the timeline to job mapping -->

<!-- if this option is set the job table will be filled
     with the jobs from job_start to job_stop. Otherwise
     you have to filled the table by your own, befor you
     can start the module. If you are unsure say "true"
     -->
<param type="TBL_Job_init"  >true </param>
<!-- This parameters are used to auto fill the table
     if TBL_Job_init == true.
     TBL_Job_start == First Job Start
     TBL_Job_stop  == Last Job Stop
     (will be adjust to fit on the job_du boundaries).
     TBL_Job_du    == The duration of one job.
     TBL_Job_duw   == The duration of the warm up.
     TBL_Job_dur   == The duration of the roll off.

     time_warmup    time_start         time_stop          time_rolloff
     |<- warm up -> |<- cout results -> | <- roll off ->|
```

```
Example: Job X has the following timings:
  time_warmup:   1000
  time_start:    2000
  time_stop:     3000
  time_rolloff:  4000

  To process the job X the framwork will request data
  from time_warmup to time_rolloff by the "submodule".
  This input data range will be delivered  to the Job Binary.
  The Job Binary should produce some output for the
  intervall time_start to time_stop.

  Tip: One day = 86400 s
-->

<!-- Trick: Job switch at 0600 and 1800 ...
     ... no overlapp over 1200 (rushhour) -->
<param type="TBL_Job_start" >1047621600</param>
<param type="TBL_Job_stop"  >1184842138</param>
<param type="TBL_Job_du"    >43200</param>
<param type="TBL_Job_duw"   >3600</param>
<param type="TBL_Job_dur"   >3600</param>

<!--
    If this option is set, all existing entries of the jobtable
    will be checked during the module start. This takes a lot of time.
    Otherwise just the first entrie is checked. If unsure say "false"
    -->
<param type="TBL_Job_sca"   >false</param>

<!-- WK_Mapper -->
<param type="WK_MAP_maxreq" >3</param>

<!-- WK_Mapper Cache "WK_MAPC " -->
<param type="WK_MAPC_path">/home/schadomi/dm/basic_metric/mapper</param>
<param type="WK_MAPC_slim"  >500</param>
<param type="WK_MAPC_hlim"  >550</param>
<param type="WK_MAPC_jsize" >1</param>
<param type="WK_MAPC_rman"  >false</param>
<param type="WK_MAPC_wari"  >3600</param>

<!-- WK_Jobber -->
<param type="WK_JOB_wijf"   >true</param>
<param type="WK_JOB_para"   >25</param>
<param type="WK_JOB_gc"     >true</param>
```

```
<!-- WK_Jobber Cache "WK_MAPC" -->
<!-- Cache Path -->
<param type="WK_JOBC_path">/home/schadomi/dm/basic_metric/jobber</param>
<!-- Cache soft limit (MB) -->
<param type="WK_JOBC_slim"  >500</param>
<!-- Cache hard limit (MB) -->
<param type="WK_JOBC_hlim"  >550</param>
<!-- Cache job size (MB) -->
<param type="WK_JOBC_jsize" >1</param>
<param type="WK_JOBC_rman"  >false</param>
<param type="WK_JOBC_wari"  >3600</param>

<!-- JS_Interface -->
<!-- Path where this file can be found ... -->
<param type="JS_IF_xml" >/home/schadomi/dm/md/basic_metric.xml</param>
<!-- Where to place the runnable task (Task Scheduler DB) -->
<param type="JS_IF_tbl" >tbl_task</param>
<!-- Client Mask: (BitMask to select the cluster nodes)
     Client selection:
     cl 1     2
     cl 2     4
     cl 3     8
     cl 4    16
     cl 5    32
     JS_IF = SUMME
     -->
<!-- Use client 1, 2, 3, 4 -->
<param type="JS_IF_cm">30</param>
</module_config>

<!--
############# BINARY CONFIGS ##############################
# This confnigs are processed by the jobscheduler client
# to create the Arg-Vector to call the binary
###########################################################
-->

<!-- input file pattern -->
        <pattern>
<!--
        Describes the input files that should be processed.
        Target selection =~= ls input/req_folder/<rel_folder>/< patter >.
        Multiple pattern can be defined. Just increment the "id"
-->
<list id="1" rel_folder="." desc="DataPakets">dat.bz2</list>
</pattern>
```

```
<!-- set the option for binary execution -->
<bin>
<!--
     The path to your binary. Don't forget to change the rights of the
     file, that the "apache-Webaerver" can execute your binary!
     -> chwon o+rwx <bin>
  -->
<param type="bin_path" >/home/schadomi/dm/md/basic_metric.bin</param>
<!--
    If this option is selected, the jobscheduler client will
    change the working path to the jobfolder. Otherwise the
    jobscheduler use the a local-tmp folder.
    If unsure say "false". This can reduce nfs load because
    the files (with realtive path) will be stored on a local disk.
-->
<param type="run_in_jobfolder" >false</param>
<!--
    Copy the input data in a temporary folder (<tmp>/input) on the node?
    The files of the 'filelist' will be replaced with the local copies.
    This option can reduce the NFS load of the "file server"
    and speed up your binary. If unsure say "true"
-->
<param type="cp_data_to_input" >true</param>
<!-- Copy the files that are stored in the <tmp>/output folder
  back to the jobfolder. If unsure say "true"
  -->
<param type="cp_output_back"   >true</param>
<!-- Copy the files that are stored in the <tmp> folder back to
  the jobfolder. If unsure say "false" (otherwise probably unused
  tmp files will be copied back ... )
  PS:
  If you have some problem with your binary, say yes here. Then the
  the standart out and standart err will be copied back.
  -->
<param type="cp_tmp_back"      >true</param>
</bin>

<!-- Copy the files that are stored in the <tmp> folder back to
  the jobfolder. If unsure say "false" (otherwise probably unused
  tmp files will be copied back ... )
  -->
<!-- set the option to create the correct arguments for your binary -->
<user_arg>
<!-- Examples:
     TYPE:
       TXT : A static text argument like "-v"
       TXT2: Like TXT but with to arguments like "-interval 900"
```

```
        PATH: Will be replaced by the correct path by the js_client
          opt = "tmp"          local tmp folder
          opt = "tmp_output"   local output folder (<tmp>/output)
          opt = "tmp_input"    local input folder (<tmp>/input)
        JOB: Will be replaced with the current job parameters
          opt = "warmup"       time warmup of this job
          opt = "start"        time start of this job
          opt = "stop"         time stop of this job
          opt = "rolloff"      time rolloff of this job
       LIST: Will add the lists, that are defined with the help of the "patterns"
          opt = "paralell"     <item1_id1><item1_id2>...<item2_id1> ....
          opt = "seriell"      <item1_id1><item2_id1>...<item1_id2> ....
-->
<param type="TXT"    opt=""             flag="--verbosen"      ></param>
<param type="TXT2"   opt=""             flag="--interval"      >900</param>
<param type="PATH"   opt="tmp"          flag="--tmp"           ></param>
<param type="PATH"   opt="tmp_output"   flag="--output"        ></param>
<param type="JOB"    opt="warmup"       flag="--time_warmup"   ></param>
<param type="JOB"    opt="start"        flag="--time_start"    ></param>
<param type="JOB"    opt="stop"         flag="--time_stop"     ></param>
<param type="JOB"    opt="rolloff"      flag="--time_rolloff"  ></param>
<param type="LIST"   opt="paralell"     flag=""                ></param>
</user_arg>
</jobdescription>
```

Bibliography

[1] NetFlow Services Solutions Guide.
http://www.cisco.com/en/US/products/sw/netmgtsw/ps1964/ . . .
products_implementation_design_guide09186a00800d6a11.html
10.09.2007

[2] The Swiss Education & Research Network.
http://www.switch.ch
10.09.2007

[3] Abilene Network.
http://abilene.internet2.edu
10.09.2007

[4] The Abilene Observatory.
http://abilene.internet2.edu/observatory
10.09.2007

[5] The GÉANT project.
http://www.geant.net
10.09.2007

[6] DANTE.
Netflow data captured on the GEANT network. *http://rtmon.gen.ch.geant2.net*
10.09.2007

[7] Communication Systems Group (CSG).
http://www.csg.ethz.ch
10.09.2007

[8] The DDosVax Project.
http://www.tik.ee.ethz.ch/∼ddosvax
10.09.2007

[9] Duebendorfer, T. (IWCIP 2005).
A Framework for Real-Time Worm Attack Detection and Backbone Monitoring
http://www.tik.ee.ethz.ch/∼ddosvax/publications/papers/ . . .
. . . iwcip2005-duebendorfer-upframe.pdf
10.09.2007

[10] A.Wagner. (WET ICE 2005).
Entropy based worm and anomaly detection in fast ip networks.

[11] Jörg Wallerich.
A methodology for studying persistency aspects of Internet flows.
www.net.informatik.tu-muenchen.de/∼hdreger/papers/CCR35_FlowPersistency.pdf
10.09.2007

[12] Anukool Lakhina. (BUCS-TR-2003-021).
Structural Analysis of Network Traffic Flows.
http://www.cs.bu.edu/techreports/pdf/2003-021-odflows.pdf
10.09.2007

[13] Daniela Brauckhoff. (IMC-2006)
Impact of Packet Sampling on Anomaly Detection Metrics.
www.imconf.net/imc-2006/papers/p16-brauckhoff.pdf
10.09.2007

[14] Seong Soo Kim and A. L. Narasimha Reddy. (ICDCSW '06).
Impact of Packet Sampling on Anomaly Detection Metrics.
dropzone.tamu.edu/∼skim/adsn2.pdf
10.09.2007

[15] NFDUMP tools.
http://nfdump.sourceforge.net
10.09.2007

[16] The Condor Project.
http://www.http://www.cs.wisc.edu/condor/
10.09.2007

[17] IBM System Storage Product Guide
http://www.ibm.com/storage/pguide/prodguidetape.pdf
10.09.2007

[18] Archiving - Department of Information Technology and Electrical Engineering
http://computing.ee.ethz.ch/infrastructure/archiving/index.en.html
10.09.2007

[19] Wikipedia contributors.
RAID
http://de.wikipedia.org/w/index.php?title=RAID&oldid=36349137
10. 09 2007

[20] Tivoli Workload Scheduler LoadLeveler.
Using and Administering
http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp? . . .

. . . topic=/com.ibm.cluster.loadl.doc/llbooks.html
10. 09 2007

[21] TORQUE Resource Manager.
http://www.clusterresources.com/pages/products/torque-resource-manager.php
10. 09 2007

[22] Wikipedia contributors.
Go/no go
http://en.wikipedia.org/w/index.php?title=Go/no_go&oldid=127844875
10. 09. 2007

[23] Julian Seward.
bzip2 and libbzip2.
http://www.bzip.org/
10. 09. 2007

[24] Wikipedia contributors.
Modular design
http://en.wikipedia.org/w/index.php?title=Modular_design&oldid=138005279
10. 09. 2007

[25] Wikipedia contributors.
Finite State Machine
http://en.wikipedia.org/w/index.php?title=Finite_state_machine&oldid=155400777
10. 09. 2007

[26] MySQL.
The world's most popular open source database
http://www.mysql.com/
10. 09. 2007

[27] Wikipedia contributors.
Application checkpointing
http://en.wikipedia.org/w/index.php? . . .
. . . title=Application_checkpointing&oldid=126648132
10. 09. 2007

[28] Valgrind.
A toolset for debugging and profiling Linux programs
http://valgrind.org/
10. 09. 2007

[29] AMD.
Software Optimization Guide for AMD64 Processors
http://www.amd.com/us-en/assets/content_type/ . . .
white_papers_and_tech_docs/25112.PDF
10. 09. 2007

[30] Wikipedia contributors.
Shared memory
http://en.wikipedia.org/w/index.php?title=Shared_memory&oldid=152113914
10. 09. 2007

[31] Wikipedia contributors.
Inter-process communication
http://en.wikipedia.org/w/index.php?title=Inter-process_communication&oldid=154663455
10. 09. 2007

[32] The phpMyAdmin Project
Effective MySQL Management
http://www.phpmyadmin.net
10. 09. 2007

[33] Wikipedia contributors.
Object composition
http://en.wikipedia.org/w/index.php?title=Object_composition&oldid=154383462
10. 09. 2007