DESIGN AND IMPLEMENTATION OF A NETWORK SEARCH SYSTEM

USMAN KHALID

MASTER’S DEGREE THESIS
STOCKHOLM, SWEDEN 2013
XR-EE-LCN 2013:006
Abstract

While modern communication networks have provided ease of connectivity and accessibility to vast resources, its operational and management task has become complex. Today's communication networks consist of network devices from various vendors, which produce enormous amount of operational data in the form of SNMP traps, syslog messages, NetFlow caches, etc. It has become increasingly difficult for operational and fault management applications to utilize all the necessary information from the vast pool of information of varying formats. This report presents a prototype called Network Search System (NSS) that aims at easing the development of these applications by providing a simple, uniform interface to obtain network information that require no knowledge of location of the information. NSS abstracts the information collection and generalization processes by providing a query interface to the management applications to retrieve intended information. The report presents several use cases implemented using the system to demonstrate its functionality. It also presents performance evaluations, which point out the crucial bottlenecks in the system and are subject for further improvement of the system.
Contents

1. Introduction ........................................................................................................................................6

2. Related Concepts ............................................................................................................................9
   2.1. Information Retrieval ................................................................................................................9
   2.2. Web Search ...............................................................................................................................11
   2.3. Search in the Web of Things ....................................................................................................11
   2.4. Related Works ........................................................................................................................12

3. Architecture of the Network Search System ..................................................................................15
   3.1. Object Representation and Querying Model ............................................................................16
   3.2. Functions ................................................................................................................................16
   3.3. Interfaces ..................................................................................................................................16

4. Implementation of the Network Search System ............................................................................19
   4.1. Implementation of a Search Node .............................................................................................19
      4.1.1. Maintaining the Local Search Databases ........................................................................19
      4.1.2. Retrieving Information from Database ............................................................................21
      4.1.3. A Query Interpreter for Network Search .........................................................................22
   4.2. Implementation of a Management Station ................................................................................23

5. Evaluation .......................................................................................................................................26
   5.1. Test bed Infrastructure ..............................................................................................................26
   5.2. Functionality of the Network Search System ..........................................................................26
   5.3. Performance of the Network Search System ..........................................................................27
      5.3.1. Database Management Latency .......................................................................................27
      5.3.2. Query Response Latency ..................................................................................................30
      5.3.3. Index Size of the Search Database ....................................................................................32
   5.4. Discussion ................................................................................................................................33

6. Conclusion ......................................................................................................................................36
   6.1. Experiences ..............................................................................................................................36

7. Bibliography ....................................................................................................................................38

Curriculum Vitae ..................................................................................................................................41
1. Introduction

Information and communication are two most important aspects of today’s digital era. The tremendous growth of information is reinforced by sharing it with others, which led to the progress in many research dimensions and development of various technologies. The information resources proved their usefulness by the ease of availability to the intended users. This has been achieved through an easy access of the information repositories to globally distributed users by means of communication networks. The Internet, which is a large network of numerous communication networks, has now become an essential part of today’s connected community.

The benefit of communication networks for ever growing number of devices is attributed to a number of challenges. The communication networks have already become very complex due to diverse access and core technologies, network devices from various vendors, security solutions and multimedia services, etc. These complexities are further increased in the wake of different management solutions and user interfaces, which make network management a cumbersome procedure. These solutions are mostly technology and provider dependent and have limited scope of interaction with others, which requires additional human resources and competence development for effective management.

Network devices hold and generate enormous amount of configuration and operational data. This data is available in various formats through standardized and proprietary sensing protocols and interfaces. Uddin et al. [1] envision bringing network information to a search process, which provides static and dynamic data, both current and historical, for network management. We predominantly focus on current information, both configuration and operational data, to achieve a search process for network information.

The network information can help in determining network-wide views of traffic patterns, network utilization based on available resources, evaluating Quality of Service (QoS) for Service Level Agreements (SLA), etc. For example, finding heavy flows in the network will return high bandwidth flows in the network that can help in determining traffic patterns, finding the excessively utilized resources in the network will return unexpectedly over-utilized network devices which can help in tracking the source of malicious activity and finding the resource availability of a network device to ensure that agreed QoS can be provided through that device, etc.

Although functionalities like these can be implemented by existing technologies, each of them has to be implemented as a separate system, with its own data collection and processing subsystems. Network search [1] aims to ease developing these functionalities by providing the intended information through a common search system. This can be visualized as disassociating individual functionalities from similar, yet burdensome tasks of sensing ever-increasing amount of heterogeneous information and providing it in a useable format.

The aim to bring network information from various sources into a generalized searchable form has various challenges attributed to it. The first challenge is to identify different protocols and interfaces through which network information can be sensed. The sensed information of various formats is then required to be generalized into a format that can be utilized afterwards. The intention to make information simultaneously available to different users advocates the use of persistent storage to
hold the information. A query language is finally required to access the desired information from the database.

These challenges are addressed by designing and developing a prototype we call Network Search System (NSS) using the concepts laid in [1]. It consists of a sensing system that collects information from various network devices using different sensing protocols and techniques like SNMP polling, capturing Command Line Interface (CLI) outputs and NetFlow. The gathered information is transformed into a uniform structure, which is then saved in a database. For formulating queries, NSS uses the query model presented in [1], which can be seen as a variant of the Boolean retrieval model [6]. The queries are processed on a distributed query processing system, which is implemented by extending the Lucene indexing and searching libraries [23].

The report is organized as follows. Chapter 2 discusses the concepts related to network search. Chapter 3 presents the architecture and fundamental components of NSS. Chapter 4 describe the implementation of the NSS. Chapter 5 demonstrates the functionality and performance of the NSS prototype. Chapter 5 presents the conclusions.
2. Related Concepts

One of the most important tasks that a computer user performs is searching information in an information repository, ranging from personal computer to the Internet. These information repositories contain information in the form of text documents, scientific papers, newspapers, web pages, blogs, multimedia files like pictures, audio and video etc. Information available in these repositories enables users to search for information they are interested in. However, huge amount of already available information in wide range of repositories and its ever increasing trend makes it difficult to find information relevant to the user need. The enormous amount of available information pertaining to similar topics creates this difficulty and gives rise to abundance problem. The abundance problem occurs when the number of documents returned as responses to a user query is far too large for a human user to consider [2]. Search systems are so developed and used to ease the search process in the wake of the abundance problem. We discuss now information retrieval and related concepts, which lays the foundation for search systems to be able to cope with the abundance problem especially within the network search.

2.1. Information Retrieval

Information Retrieval (IR) provides useful concepts and techniques that can be used and implemented by search systems. According to Baeza and Ribeiro [3], IR is defined as:

“An information-retrieval model is a quadruple \(D, Q, f, R(q_i, d_j)\) where

1. \(D\) is a set composed of logical views (or representations) for the documents in the collection.
2. \(Q\) is a set composed of logical views (or representations) for the user information needs. Such representations are called queries.
3. \(f\) is a framework for modeling documents representations, queries, and their relationships.
4. \(R(q_i, d_j)\) is a ranking function which associates a real number with a query \(q_i \in Q\) and document representation \(d_j \in D\). Such ranking defines an ordering among the documents with regards to the query \(q_i\).”

There are many IR models, each having their own logical views of documents and queries as well as query matching and result ranking scheme, e.g., Bag-of-words, Boolean model, Vector Space model, etc. We discuss a well-known IR model, the vector space model, in terms of the concepts defined above.

Vector Space Model

The vector space model represents a document as a vector of weighted terms. Each dimension of the vector represents a distinct term, and its associated weight represents the term’s importance in the document. The terms are weighted according to two factors: (a) the term frequency and (b) the collection frequency [4].

The term frequency is generally used as the basis of a weighted document vector [5]. Term frequency factor of term \(t\) in document \(d\), denoted by \(f_{td}\), is computed by counting the occurrences of the term normalized by the occurrence count of the most frequent term in the document [6]. For
a document, \( tf_{dt} \) is higher for frequent terms and lower for rare terms, justifying the importance of frequent terms in the document.

The collection frequency factor of a term gives the global importance of the term in the document collection. Collection frequency factor of term \( t \) can be computed by inverse document frequency, denoted by \( idf_t \). \( idf_t \) of a term is calculated by taking the logarithm of the ratio of the total number of documents to the number of documents in which a specific term appears [6]. \( idf_t \) is higher for rare terms and lower for frequent terms in the collection set. Both \( tf_{dt} \) and \( idf_t \) are used in \( tf-idf \) weighting scheme for assigning the term weight according to its importance in the collection set.

\[
(1) \quad tf-idf = tf_{dt} \times idf_t
\]

The length normalization factor can also be considered for calculating the term weights. Occurrence probability of a particular term is more likely to be higher in a long document than a short document. This probability makes the long documents more likely to be retrieved than short documents for the same user query. To normalize this effect, the length normalization factor is used along with term and collection frequency factors.

User queries are considered as short text documents containing single or multiple terms. To match a query against the document collection, the documents are indexed as posting lists. A posting list is an ordered set of \{document id, term weight\}-pairs that is associated with a unique term in the document collection. While matching, the posting lists associated to the query terms are retrieved and documents IDs in the lists are intersected. This process returns all the documents that contain all the query terms.

For ranking, a similarity score is computed for each matched document by utilizing the associated term weights. For each matched document \( d_j \) and query \( q \), the similarity score is computed as follows:

\[
(2) \quad sim(d_j, q) = \sum_{i=1}^{f} w_{ij} w_{iq}
\]

whereby, \( w_{ij} \) and \( w_{iq} \) indicate the weight of term \( i \) in document \( j \) and query \( q \) respectively. The weights of terms in query \( q \) are computed by some weighting scheme such as \( tf-idf \). A document with high similarity score means that the document is highly relevant to the query. Therefore, the documents are sorted in descending order according to their similarity scores. The user receives this ranked list of documents as a response to the query.

**Boolean Model**

The Boolean model represents a document as a set of keywords. Each distinct document term is a distinct keyword in the set, and each keyword is equally weighed. The user query is a Boolean expression of query terms connected by logical operators like AND, OR and NOT etc. The matching function is an element of Boolean algebra over the set of documents and query expression. A document is considered relevant if and only if it satisfies all the Boolean conditions in the query expression. The Boolean model doesn’t provide result ranking as it considers the document relevant or not relevant.

The network information is available in semi-structured format as documents of \{attribute, value\}-pairs. These documents are Boolean modelled to equally weigh the pairs as each pair is equally
important in the context of network management. The Boolean query model can be used to explicitly specify the terms with “AND” operator that must be present in the matched objects. The network information is considered to be relevant or irrelevant to the network management process which advocates the use of the Boolean matching function.

2.2. Web Search

Web search is a realization of information retrieval systems in the domain of the World Wide Web. Web search systems make use of crawlers, which navigate the web pages by following their hyperlinks and store the crawled pages in a database(s) of indexes. A search query, invoked by a user, is a list of keywords, which is matched against the index database, and a list of uniform resource locators (URLs) of the matched web pages are returned to the user. This list can possibly grow to the order of hundreds of thousands of URLs due to the abundance problem. To deal with this problem, web pages are ranked, usually through link analysis schemes such as PageRank [7] and HITS [8], etc.

PageRank and HITS algorithms calculate ranking scores for each page in the web by exploiting the hyperlink structure of the web. The score reflects the importance of each page in the World Wide Web graph. These algorithms give a high ranking score to a webpage that is linked to by many webpages and links to many other pages. A list of webpages in the decreasing order of their relevance and ranking score is returned to the user as a query result.

Topic-Sensitive Search

Web search algorithms rank webpages by keyword-matching scores like relevance measurement and link analysis metric like PageRank score. On one hand, even though keyword-matching scores ensure that a high-ranked page has higher frequency of query keywords, it may not focus on the query topic. On the other hand, link analysis scores are completely topic insensitive. Topic sensitive search enables the user to search information on the basis of a topic description in the document corpus. Topic aware search systems make use of context terms and extend the use of link structure of the web to find the topic-authoritative pages. Context terms refer to any supplementary information associated with the user query like “jaguar” can be concatenated with “car” or “cat” as context term. Context terms are modeled as vector space model, with associated weights of their importance in the context.

Kraft et al. [9] proposed the use of context terms in conjunction with the initial query to broaden the search pattern. Specifically, a query is reformulated by concatenating it with context terms and ranking weights to boost the ranking of matching documents. Krishna et al. [10] enhanced Kleinberg’s link analysis algorithm, HITS [8], to make it topic sensitive, where a broader query is suggested by concatenating the first few hundred words from all the documents in the root set. This approach increases the relevance of result pages to the query topic. To further elevate the relevance of result pages, nodes whose relevance to the query topic is lower than a certain threshold are pruned from the neighborhood graph [8].

2.3. Search in the Web of Things
The increasing trend of monitoring real-world objects with embedded sensors, connecting these sensors to the Internet and publishing their contents on the World Wide Web allows the users to search for real world objects with minimal delay. This trend is the precursor of Web of Things (WoT), which transformed the document centric web into a universal interface for real-world objects by giving them a web presence. Web presence of real-world objects poses new challenges to the search engines like entity discovery, i.e., finding real-world objects with the given state of the sensor connected to the object. Another key challenge posed to search engines by the web of things is the huge amount of highly dynamic information generated by the sensors. As the dynamic nature of sensors’ information is different in principle from the static or slowly changing contents of the document-centric Web, conventional web search engines are not suitable for rapidly changing contents. The search in WoT is synonymously treated as real-time search where information is readily available for search immediately after its generation.

Snoogle/Microsearch [11, 12] and MAX [13] are used to discover sensors, which are attached to real world objects and contain the textual description of these objects. Both systems are organized in a multi-tiered hierarchy of nodes named as mediators. Sensors push their changes to mediators, which can be queried by a list of keywords. The system returns the list of the top k real world objects, which match the given states specified in the query keywords. Global Sensor Networks (GSN) [14] provides a peer-to-peer approach to find virtual sensors in the Internet-based interconnection of heterogeneous sensors and sensor networks. In GSN, a virtual sensor represents either a physical sensor or a virtual entity, which represents one or multiple virtual sensors. Virtual sensors can be discovered by specifying the identifier of a virtual sensor or by metadata describing the sensor. Keller et al. [15] presents a real-time search engine called Dyser for WoT. Dyser uses a prediction model to calculate the probability of the sensors, which can produce the sought output at the querying time.

2.4. Related Works

We now discuss the search systems which are used to ease the process of finding network information. These and other search systems are developed on the four fields summarized in section 2.1. We will describe two of these systems, Fusio [16] and Weaver Query System [17], and describe similarities and differences between these systems and our search system.

Fusio

Fusio is a centralized search system for management information which holds large and diverse information resources. It provides search capability for unstructured text and Resource Description Framework (RDF) triples, \{subject, predicate, object\}: The predicate represents the relation between subject and object [16]. The subject and predicate are referable Uniform Resource Identifier (URIs) and object can be a referable URI or a literal value. The system accepts the set of query terms and returns the list of URIs which led to the data that matches all or part of the query terms. The system presents the ranked list of URIs together with the important predicates fetched from the corresponding data.

The network search system (NSS) is different to Fusio in searching the documents which have matching \{attribute, value\} pair(s) as compared to searching the documents which have matching terms or RDF triples in Fusio. Fusio queries are free-text strings while NSS accepts query tokens
based on the Boolean model. Like Fusio, the central search application of NSS will accept the queries and return the results.

**Weaver Query System**

The Weaver Query System (WQS) is a distributed querying system for monitoring traffic flows in near real-time. It enquires the network statistics which are held in structured tables maintained by Relational Database Management System (RDBMS). The system employs a navigation pattern called Echo [18] to propagate the queries and aggregate the returned results. WQS uses a declarative query language called Weaver Query Language (WQL), which is based on the Structured Query Language (SQL), to interact with the RDBMS. The WQL query returns an aggregated global view of the network or traffic statistic. The returned results can be sorted out by specifying it explicitly in the query.

WQS uses RDBMs for maintaining network information while NSS uses the Boolean model to maintain network information as index objects. WQS finds relation between network attributes and values based on SQL-like queries while NSS finds network tuples of {attribute, value} format. NSS is a centralized search system which is different to WQS distributed architecture of propagating queries into the network and presenting the aggregated results.
3. Architecture of the Network Search System

The information retrieval and related concepts in chapter 2 pointed out the important elements for a search system, which includes models for object representation and querying and functions for matching, ranking, query routing, etc. In this section, we present the design and architecture (figure 1) of a system for searching network information, which we refer as the Network Search System (NSS). We also describe important modules and functions for the search system.

![Network Search System Architecture](based on [1])

As shown in the figure 1, the system consists of three planes: a network plane, a user plane and a search plane. The network plane is positioned at the bottom of the architecture. This plane consists of network devices that hold and generate configuration or operational information in their logs, caches and counters. The user plane is positioned on the top. This plane contains network management processes, which invoke search queries to obtain information in the network plane. In the middle of the architecture, we have the search plane, which provides a query interface to the processes in the user plane to retrieve information for network management. This plane is responsible for collecting information from the network plane and creating search database from
this information. The plane is also responsible for retrieving information from the search database by matching search queries invoked from the user plane.

The search plane is realized by a set of nodes called search nodes. A search node is realized as a standalone network appliance. All search nodes possess the same set of functionalities. These functionalities enable them to collect information from the network devices in order to build a search database and retrieve desired information for the management processes in the user plane.

3.1. Object Representation and Querying Model

The object representation and querying models are based on the models presented in [1]. Network information from devices can be grouped and characterized as network objects. We represent these objects as bags of attribute-value pairs. For example, an object that represents a physical device with an Ethernet interface can be expressed as the bag \{(deviceID, 127.0.0.1), (deviceType, router), (numOfInterface, 1), (interfaceDescription, FastEthernet0/0), (interfaceIpAddress, 192.168.1.1)\}. A search database is formed by such bags.

A query for searching network information can be expressed as a list of comma-separated tokens. A token can either be an attribute name or an attribute-value pair with a comparison operator, e.g., ‘==’, ‘>’, ‘<’ and ‘!=’, in between. Complex queries can be formulated by simple queries that are enclosed in brackets and separated by commas.

The querying process involves matching queries against a search database. The matching is performed by evaluating the query tokens against the attribute-value pairs of objects in the search database. Such an operation returns only those objects that successfully evaluate all tokens of the query.

3.2. Functions

A network search process is realized through a set of interacting functions. These functions are data sensing, database, query management, query processing and query manager. The first four functions are placed in search nodes. The data sensing function enables a search node to collect information from an associated network device. When network information is collected, a database function formats this information into objects according to the model described in Section 3.1 and stores the objects in a database. The query management function receives queries from the management station and returns results. The query processor function is responsible for matching search queries against the database, retrieving matching objects and extracts the desired results from the matched objects. The query manager function runs in the management stations in the user plane. This function receives queries invoked by the management processes as inputs and propagates them to the search nodes. When a search node replies back, the function aggregates the responses and forwards the aggregated response to the invoking process.

3.3. Interfaces

Our search system has a number of interfaces through which messages are exchanged between different functional modules. The interfaces are sensor interfaces, database interface, search interface, query interface and user interface. The sensor interfaces are used by search nodes to sense information from the underlying network. The database interface is used by database modules
to store the network objects in the database. The search interface is used by the query processing module to execute the search query against the local database. The query interface enables the interaction between the query manager module and the query management module in the search nodes. Through this interface, the query manager module sends user queries and the query management module replies with local results. The user interface resides in the user plane. It enables interaction between the management processes and the query manager module in terms of user query and response.
4. Implementation of the Network Search System

The Network Search System is an IP network of search nodes and network devices. Each search node, a personal computer, is associated with a single network device, which in our implementation is a router. The implementation of NSS is divided in two parts: implementation of a search node and implementation of management station.

4.1. Implementation of a Search Node

We describe the implementation of a search node using the block diagram illustrated in Figure 2. We classify its implementation in two parts (a) maintaining the local search database and (b) retrieving information from the database.

4.1.1. Maintaining the Local Search Databases

At first, we describe the maintenance of a local search database. There are several tasks involved, namely, sensing data from network plane, formatting the sensed data and storing the formatted data in a local database.

We implemented our search system in such a manner that each search node is associated with a network device, in this case a router, in the network plane. We implemented a range of sensing
functions that enable a search node to collect different types of data from the associated router. The data types are associated with router configurations, routing tables, MIB counters, and Netflows. The functions for sensing router configurations, routing tables, and MIB counters are realized as Java programs that use WebNMS SNMP API [22]. The function that senses flow entries from the NetFlow caches is realized as a Linux shell script by using NetCat [24], which invokes CLI commands over telnet session to the routers.

A manager function manages these sensing functions. The function, named as the sensor manager, periodically runs the sensing functions at a different rate because the data types associated with the functions changes at different time-scales. The sensor manager is realized as a Linux shell script.

A data modeling function is coupled with each sensing function. The modeling function is triggered whenever an associated sensing function brings in a new set of data. The modeling function formats the newly sensed data into bags of attribute-value pairs as describe in Section 3.1. The function then stores and indexes the bags into a local database. These functions are realized as Java programs that use Apache Lucene API, specifically, the indexing libraries [23].

The sensing and database functions for configuration, routing and MIB counters information are implemented in separate Java programs. Each program performs data sensing, conversion and storage in one execution thread. The Java programs consist of 557 code lines for configuration information, 428 code lines for routing table information and 410 code lines for MIB counters information. The NetFlow sensing function is implemented in a Linux script, which consists of 31 code lines. The database function for NetFlow information is implemented in a Java program which consists of 200 code lines. NetFlow entries are converted and stored in database in separate but parallel execution threads. Figure 3 presents a high-level view of sensor manager, configuration sensing function, routing table sensing function, MIB counter sensing function, NetFlows sensing function and associated database functions.

![Figure 3: Abstract of Data Collection](image-url)
4.1.2. Retrieving Information from Database

The search nodes facilitate the management process in information retrieval from the local database. The information retrieval in the search node includes tasks like receiving queries from the management station, tokenizing and parsing the query, retrieving records (which is a bag of \{attribute, values\}-pairs as given in section 3.1) relevant to the query from the database and returning the results to the management station.

A query management function receives queries from the management station. A parser tokenizes the received queries into tokens. While parsing, the parser classifies these tokens either as match-tokens or as search-tokens. The match-tokens are used while matching a query with the local database and search-tokens are used to extract desired part of the matched records.

A query processing function uses match-tokens to create a local query and matches this local query in the search database. The matching is performed using the Boolean model, which states that each match-token should exist in the record to be considered a matched record [6]. The processing function returns parts of the matched records by using the search-tokens to the query management function.

The query management and query processing functions are implemented in a Java program, which consists of four Java classes and 370 code lines. The query management function is implemented as a server of server-client model. It starts a new execution thread, on reception of a query from management station, in which query processing function is executed. The query processing function uses Lucene’s search libraries. The query management function forwards the results, returned by the processing function, to the management station. Figure 4 present high level abstract of query management function and query processor function.

![Diagram of Query Processing](Diagram.png)

**Figure 4: Abstract of Query Processing**
4.1.3. A Query Interpreter for Network Search

The query interpreter in a search node is based on the query model presented in section 3.1. It uses different kinds of flags to interpret search queries.

A simple query is formed by query tokens separated by commas. There are two kinds of tokens: match and search. The match-tokens are attribute-value pairs, while the search-tokens are simply attribute names. These tokens are grouped accordingly inside query and the groups are identified by −s flag (for search-tokens) and −m flag (for match-tokens). A simple query is given as follows.

\[-s \text{attr}_A, \text{attr}_B -m \text{attr}_X:val_X, \text{attr}_Y:val_Y\]

This query will return attr_A and attr_B of all records in the search space that matches attr_X:val_X and attr_Y:val_Y pairs. The query will return the whole records if flag −sa is mentioned instead of flag −s (in such a case, no search tokens are used in the query). The query will return all records if flag −m is not mentioned in the query (in such a case, query processor will add a match-token router ‘*’, which will be matched to all network objects in the database).

More complex queries can be formed using multiple queries that are separated by enclosing parenthesis. While interpreting the complex queries, the interpreter simply treats query segments, delimited by the parenthesis, as individual queries. The management process assigns a unique identifier to each segment. The complex query returns the records that match any of these segments. The identifiers of each segment are attached in the returned result. Using these identifiers, query-invoking processes can separate the results for further processing. A sample complex query is given as follows.

\{(100 -s \text{attr}_A, \text{attr}_B -m \text{attr}_X:val_X, \text{attr}_Y:val_Y)(200 -sa-m \text{attr}_Z:val_Z)\}

This query contains two segments identified by unique identifiers 100 and 200. The query returns the records (which is a bag of {attribute, values}-pairs as given in section 3.1) that match any of the segments.

An interesting kind of query is persistent query. A persistent query is the same query as the previously defined queries but written with the flag −p. In such a case, the query will be matched against the database in a periodic manner. The period is by default 1 second but can be set using a flag −t. A sample persistent query is given as follows.

\[-p 5 -t 2 -sa-m \text{attr}_Z:val_Z\]

This query returns the matching records continuously at 2 second interval for five times.

The query interpreter provides some additional features. The results returned by a query can be saved at management station in a file by using the flag −f followed by filename. The number of matched records can be learned by using the flag −c. The unique results can be filtered by using the flag −u.

The query interpreter is divided into two separate sub-modules: local and global. The local module resides on the search nodes and interprets parts of a query that will be processed in local database, e.g., identifying tokens and query segments. The local module is a part of query processor function.
The global module resides on a remote station that receives the search query from the network administrators and network management processes. This module interprets parts of a query that will be applied on the results returned by the search nodes, e.g., counting, storing, filtering, etc. The global module also determines whether a query is ad-hoc or persistent.

### 4.2. Implementation of a Management Station

We describe the implementation of a management station that provides a search interface to the management processes using the block diagram illustrated in figure 3.

![Block Diagram of User Plane](image)

Figure 5: Block Diagram of User Plane

The management station implements the behavior of a user plane. That means, the station receives search query from a management process and returns the matching results. A user interface enables query invocations and receptions of result responses. A query interpreter listening to the interface (the global sub-module of the interpreter discussed in section 4.1.3) parses this query and forwards the parsed query to a (global) query manager module. This module interacts with the local query management module in the search nodes. The global manager module propagates the parsed query to the local management module, receives local results returned from the local modules and performs a global operation on all local results. The global manager then forwards the final results to the management process through the user interface.

The interface, the interpreter and the manager module in the management station are realized in the Python scripting language and it consists of 165 lines of code. Figure 6 present high level abstract of query interpreter and query manager function.
Figure 6: Abstract of Management Station
5. Evaluation

The main goal in designing Network Search System is to support searching information in the network. Other design objectives are to provide low execution time and low memory overhead. In this chapter, we present the results to demonstrate the capabilities of NSS to search the network information and to meet the design objectives. We categorize the results in three categories: functionality, performance and overhead. We present the functionality of the search system through use cases that we support on the system, the performance by the latency of the key involved processes, such as searching, sensing, etc., and the overhead by storage occupancy of the search repositories and processes. At first, we present the test bed infrastructure.

5.1. Test bed Infrastructure

Our test bed is part of the KTH laboratory infrastructure, which includes 16 Cisco 2600 series routers and 33 Linux hosts. The routers and hosts are interconnected via four Fast Ethernet switches. The routers use RIP for routing. These routers comprise the network plane that contains information to be searched (figure 1). All hosts are Pentium-4 computers with 2.8 GHz CPU, 1 GB RAM and Ubuntu 10.04 OS. 16 hosts are configured as search nodes, 16 hosts are configured as traffic generators and 1 host is used as a management station in the user plane. The traffic generators use pktgen package [19] to inject traffic associated to normal behavior and fping [20] and hping [21] tools to inject network attacks such as DoS, port scans, etc.

5.2. Functionality of the Network Search System

We start by demonstrating the usefulness of the search system by presenting the following five use cases. The use cases are based on the queries that are invoked by management processes on the management station.

Case 1: Find configuration of a device given its IP address

Query: (-sa –m router 192.168.1.10)

This query returns configuration of the device with IP address 192.168.1.10 such as router IOS version, number of interfaces, type of interface, physical and IP addresses, etc. The management process that invoked the query uses the returned information to maintain an inventory of network assets.

Case 2: Find the heaviest flow in the network

Query: (-s srcip dstip srcport dstport load)

This query returns recent IP flows in the network. The management process that invoked the query identifies the heaviest flow using the ‘load’ value. The management process uses this information to profile load-generating applications and users in the network.

Case 3: Find the highly loaded router and flows passing through it

Query: (100 -s routerID ifInOctets) (200 -s srcip dstip srcport dstport)
This query returns the number of octets received and recent IP flows observed by the routers. The management process that invokes the query identifies the highly loaded routers by using the ifInOctets value and marks the associated IP flows as flows passing through the highly loaded router. The process uses this information to profile traffic in the overloaded routers.

**Case 4: Find the flows that are passing between two given routers**

Query: (100 -s srclp dstl port -m router 1.1.1.1)(200 -s srclp dstl port -m router 2.2.2.2)

This query returns IP flows in the router 1.1.1.1 and router 2.2.2.2. The query invoking management process selects those IP flows that are present in the results of both query segments. The process uses this information to profile traffic between two nodes, for example, remote offices connected by VPN.

**Case 5: Find the over-utilized network elements**

Query: -p -t 1 (-s ifInOctets -routerID)

This query continuously returns octets received on the routers in the network at an interval of one second. The query invoking process computes several statistics such as average and standard deviation using octet values of a given window size for each router. Then the process detects over-utilized resources using the current load value and associated statistics. The process then further investigates the over utilized router(s) to detect anomalies in the network.

5.3. **Performance of the Network Search System**

We demonstrate performance of the system by reporting the latencies of two key processes of the system: database management and query response. These results are explained as follows.

5.3.1. **Database Management Latency**

We start by the database management latency measured in a search node. In our experiment, this latency includes time to sense information (from a network device, e.g., router, to a search node), time to format information (according to the object model given in section 3.1) and time to store the information in a local database. We collect four different kinds of data from a router: (a) configuration data, (b) routing table, (c) MIB counters and (d) NetFlows entries. For each kind, we collected fifty samples of database management latency and plotted a box-plot as given in Figure 7.

The boxplot is a graphical representation of observation set through five numbers, which are the smallest observation, the lower quartile (Q1), the median, the upper quartile (Q3) and the largest observation. The smallest and largest observations are straight forward numbers, while Q1 represents lowest 25%, median represents 50% and Q3 represents highest 25% of values in the data set.
Figure 7: Latency of managing a local database

We consider the median to compare the latency of different kinds of data. The lowest latency in figure 7, 165 milliseconds, is observed for MIB counters. We collected eleven MIB counters in separate SNMP requests. All attribute-value pairs of MIB counters are saved in the database in one event. The configuration data consists of ten attribute-value pairs. All pairs are sensed in separate SNMP requests and saved in the database in one event. However, the latency of configuration data (i.e., 177 milliseconds) is higher than MIB counters because the values of three attributes in configuration data, e.g., routerID, engineID and physicalAddress, require formatting before they can be saved in the database. The next higher latency is observed for routing data, which shows 280 milliseconds. It is because of fifteen routing table entries sensed in separate SNMP requests and saved in separate database events. The highest latency, 607 milliseconds, is observed in the case of NetFlows, because NetFlow data is sensed by capturing the output of CLI commands, which by the way is comparatively slower than SNMP polling. We sensed thirty NetFlow cache entries on average in each collection sample. Each entry is separately formatted and saved in the database.

We collected each kind of data at a different refresh rate. The refresh rate reflects how fast information changes in the network plane. The refresh rate is lower for the data that changes at a slower rate and higher for the data that change at a faster rate.

The refresh rate for configuration data is one hour because it changes rarely, e.g., when new configuration is provided or existing configurations is updated. The routing information is collected at an interval of one minute. The routing information changes in the event of network device malfunction or connectivity problems. Since the routers are using RIP, routing information in the router can take several minutes to update. The MIB counters, which represent traffic counters, are changed according to the traffic load. The refresh rate of MIB counters is 200 milliseconds. The entries in the NetFlows caches can change very rapidly. We use the refresh rate of 700 milliseconds for this information, which is close to its sensing latency.
The latency of managing a local database is further divided into latencies of sensing function and database function. We again collected fifty samples of sensing latency and database latency for each kind of information and plotted in figure 8 and figure 9.

Figure 8: Latency of sensing functions

- (a) Configuration Data
- (b) Routing Table
- (c) MIB Counters
- (d) NetFlows

Figure 9: Latency of database functions

- (a) Configuration Data
- (b) Routing Table
- (c) MIB Counters
- (d) NetFlows
5.3.2. Query Response Latency

The query response latency includes the time to send a user query to a search node, time to execute the search query in the database, time to process search tokens against matched objects and time to send results to the management station. We used queries from the first four use cases in section 5.2 to measure the query response latency. For each query type, we collected fifty samples of query latency and plotted a box-plot as given in Figure 10.

![Figure 10: Latency of query](image)

We consider the median to compare the latency of different query types. The lowest latency (i.e., 28 milliseconds) is observed for the first query, which returned one result for each invocation. The next higher latency, 43 milliseconds, is observed for the second and third query, which returned 15 results on average for each invocation. The highest latency, 61 milliseconds, is observed for fourth query. The query returned 28 results for each collection sample. The increasing trend of latency is observed as the number of returned results is increased. This is due to the fact that higher number of matched objects incurs higher processing time and higher delay in transmitting the result from search node to the management station.

The latency of query is further divided into latency of query processing function and latency of query manager function. We again collected fifty samples of query processing latency and query manager latency for each kind of query and plotted in figure 11 and figure 12.
Figure 11: Latency of query processing function

Figure 12: Latency of query manager function
5.3.3. Index Size of the Search Database

We now discuss the overhead of the search system by presenting the memory usage of the databases. Our search system consists of four databases for each type of sensed data in section 4.1. For each database, we collected fifty samples of occupied memory size and plotted a box-plot as given in Figure 13.

![Box-plots of memory usage for different indexes]

We will again consider the median of memory usage of the indexes. The configuration data index occupied the lowest memory on the scale of 1385 kilobytes. The index holds one entry of configuration data at any time. The MIB counter index takes 1488 kilobytes of the memory space. The MIB counter index also holds one entry at a time but it occupy slightly more space compared to configuration data index. This is because of eleven attribute-value pairs in each MIB counter index entry compared to ten attribute-value pairs in configuration data index entry. The routing table index takes 3170 kilobytes of the memory and holds fifteen routing entries in each index instance. The NetFlow data index occupied the highest memory space, 4045 kilobytes. The index holds thirty NetFlow entries on average in each index instance. The increasing trend of memory occupancy is observed as the index entries are increased.

We collected index occupancy of each type of data where each sample is taken by adding the same data to the index. We collected fifty samples of such indexes and plotted in the figure 14.
5.4. Discussion

We demonstrated the usefulness of the Network Search System with the help of use cases, its performance in terms of response times and memory utilization. In this section, we analyse the experimental results presented in section 5.2 and 5.3 as follows.

We reported five use cases to show the capability of Network Search System in network management tasks. These use cases were demonstrated in terms of search queries that management process invoked through user interface.

We then reported latencies of key functions involved in the NSS operations and observed identical patterns in the statistics. The increasing trend of latency was observed when higher numbers of processing functions were involved compared to similar operation with lower processing. The high database management latency of NetFlows was seen as performance bottleneck due to comparatively higher number of data polling request, data formatting and/or data storage operations. The query response latency was also seen to be increasing when invoked query leads to higher number of database search operations, higher number of processing operations on matched objects and higher transmission time of large result sets. The key to provide lower latencies is to implement fast sensing functions, simple formatting functions and parallel database functions, which will save the bulk of network objects in one event.

We also presented the memory utilization of the NSS storage, which is the index representation of the network objects in the search plane. We again observed the similar pattern of higher memory utilization in case of a large set of network objects. The index occupancy or memory utilization was seen to increase when higher numbers of network objects were put in to the database or when a network object contained a large amount of information in terms of a higher number of object
attributes. The key to restrict the overhead of NSS from growing to enormous data collection depends on the NSS characteristics and model. As NSS provided a current network view, its overhead was very small. This small overhead can grow very large if NSS has to provide historic network views as well. This growing overhead can be reduced if network views within specific time window are kept in the database.
6. Conclusion

A network search system is important in the context of network management when a good number of management processes require access to a wide range of information. The key contribution of this thesis is the implementation of such a system for exploration. We implemented a basic network search system (NSS) that enabled management processes to access information easily through a query interface. Through use cases, we demonstrated that NSS promotes exploration of network management functions. NSS is lightweight in terms of CPU utilization and memory footprint and gives mildly quick query response.

The current NSS prototype has several limitations. Its query language is limited in the sense that it does not support comparison operators such as greater than or less than. In addition, it only allows matching queries to objects in the search database in exact fashion and does not allow ranking of returned objects. Moreover, our functions are centralized, which is not ideal for large-scale systems. Finally, we have observed several performance problems particularly in managing and updating search databases.

In our future work, we intend to address the above issues. We intend to enhance query language to support more operators. We want to add another form of query, we call cascaded queries. In these queries, multiple queries are embedded together in nested fashion, and results of inner queries can be used in the outer queries. We want to implement the search function in a distributed fashion, which can be done by wave algorithms such as Echo [18]. We aim to explore partial matching and result ranking concepts and bring them in our network search systems. A possible approach can be using such concepts in information retrieval models such as vector space model [4, 5, 6]. We aim to improve performance bottlenecks like database management latency in NSS. This can be achieved by using new sensing functions based on the push model instead of the NSS pull model. We also aim to improve the database management latency by exploring new database technologies like Mnesia database which is appropriate for continuous operations and exhibits soft real time properties [25].

6.1. Experiences

We now conclude this report with experiences during our work. We initially designed the document model to support historic network views as well as current views. This proved to be challenging because of difficulties in distinguishing historic and current views during object retrieval from database. The database was also growing very large with many considerations which were difficult to handle in the beginning. During our initial work, data collection, formation and data storage processes were completely decoupled from each other due to which the management process was able to get current views after about five second of its generation. We, in the end, have been able to get the network views within 700 milliseconds of its generation when data sensing, formation and storage have been completely coupled with each other. We have also seen issues in the implementation of the query processor due to the Java programming language, being used. These limitations can easily be handled in the Erlang programming language if the query processor is implemented from scratch. This will also help enriching the query language with capabilities of expressing complex queries. Overall, it has been a good learning experience and we are looking forward to improve it in our future work.
7. Bibliography

4. THOR center for Neuroinformatics, “An Introduction to Multimedia Text Mining”.
14. K. Aberer, M. Hauswirth, and A. Salehi, “Infrastructure for data processing in large scale interconnected sensor networks, Proceedings of Mobile Data Management (MDM ’07), Mannheim, Germany, May 07”.
Curriculum Vitae

ADDRESS: Hasselgatan 55, Lgh 1102, 194 37, Upplands Vasby, Stockholm
DOB: 27-November-1986, PHONE: (+46) 0727242766 • E-MAIL: ukhalid@kth.se

OBJECTIVE
Aiming to excel in the field of telecommunication to render my knowledge, skills and potential in a professional environment.

WORK EXPERIENCE
- Software Developer (since 2011) in Core and IMS department, Ericsson AB, Sweden.

CERTIFICATIONS & PROFESSIONAL SKILLS
- Cisco Certified Network Associate (CCNA)
- Cisco Certified Network Professional (CCNP)
- Cisco Certified Internet Professional (CCIP-BGP training)
- GSM MSS training at UET-ZTE center.
- CDMA2000 BSS training at UET-ZTE center.
- Programming Languages: Erlang/OTP, Java, Python, Linux Shell Scripting, NesC/TinyOS
- Platforms: Unix/Linux, Windows, SUSE, Ubuntu
- Tools: MATLAB, OPNET Network Simulator, Maple, Wireshark, VMware, ECLIPSE etc.
- Conceptual Understandings: LTE, IP Networks, WAN, Protocols (TCP/IP, UDP, RIP, OSPF, EIGRP, SIP, FTP, TFTP, DNS etc.), WIFI, BLUETOOTH etc.

EDUCATION
- M.Sc Network Services & Systems
  Kungliga Tekniska Hogskolan, KTH, Sweden (2009-2012)
- Cisco Certifications
  Corvit Systems, Lahore, Pakistan (2008-2009)
- B.Sc Electrical Engineering
  University of Engineering & Technology, Lahore, Pakistan (2004-2008)

EXTRA CURRICULAR ACTIVITIES, INTERESTS & HOBBIES
- Theatre
- Trekking
- Travelling
- Photography

REFERENCE
- Janos Haris, Line Manager at Feature Development Centre 2, Development Unit Core & IMS (DUCI), BNET, Ericsson AB, Sweden, Email: janos.haris@ericsson.com
- Mr. Rolf Stadler, Professor at School of Electrical Engineering, KTH Royal Institute of Technology, Stockholm, Sweden, Email: stadler@kth.se