Management by Network Search

January 10, 2012

Misbah Uddin, Rolf Stadler
ACCESS Linnaeus Center
KTH Royal Institute of Technology
Email: {ahmmud,stadler}@kth.se

Alexander Clemm
Cisco Systems
San Jose, California, USA
Email: alex@cisco.com

Abstract—While networked systems hold and generate vast amounts of configuration and operational data, this data is not accessible through a simple, uniform mechanism. Rather, it must be gathered using a range of different protocols and interfaces. Our vision is to make all this data available in a simple format through a real-time search process which runs within the network and aggregates the data into a form needed by applications—a concept we call *network search*. We believe that such an approach, though challenging, is technically feasible and will enable rapid development of new management applications and advanced network functions. The paper motivates and formulates the concept of network search, compares it to related concepts like web search, outlines a search architecture, describes the design space and research challenges, and reports on a testbed implementation with management applications built for exploratory purposes of this new paradigm.

I. INTRODUCTION

Any networked system holds and generates vast amounts of configuration and operational data in configuration files, device counters and data collection points. This data is partitioned in the sense that it is kept in various formats and needs to be accessed through different protocols and interfaces, including SNMP, NetFlow, CLI, etc. By making all this data available to a general search process in real-time across the network—a concept we call network search—novel management applications and advanced network functions can be envisioned.

Network search can be seen as a functionality that extends or complements monitoring, whereby state information and monitoring variables are not explicitly addressed by network location but characterized as content. While we envision that the concept includes searching static and dynamic data, current as well as historical information, we focus in this paper primarily on dynamic data, as we see the most innovative aspects in this area. Another way of understanding network search is as “Googling the network in real-time.” For instance, the search query “find information about the flow $\langle src, dest, app \rangle$ ” may return a list of systems that carry the flow in their caches, the interfaces the flow

uses, the list of firewalls the flow traverses, the list of systems that impose traffic policies on this flow, etc.

We anticipate that the capability of network search will accelerate the creation of novel network management functions. It will spur, for instance, the development of new tools that will assist human administrators in network supervision, maintenance and fault diagnosis. The tools will allow search queries, such as:

- *Search for information about the system with IP address $a.b.c.d$* which may return links to systems that list it as BGP neighbor, to firewalls or NATs that include its address, or to event logs where its address appears.
- *Search for excessive resource utilization*, which may return links to systems and components whose CPU, memory or communication links are highly utilized.
- *Search for information about a session with given endpoints*, which may return the paths of flows associated with the session and information about the supporting virtual and physical infrastructure

We believe that a network search functionality will allow the rapid development of new classes of network control functions and applications, for instance:

- *Tracking virtual networks and virtualized resources*, whereby virtual systems and resources are dynamically discovered and their states, physical and logical dependencies, and configurations tracked.
- *Service level diagnostics*, whereby critical session flows that are subject to service level agreements are discovered and analyzed in real time for potential detriments to performance.
- *Peer ranking*, whereby hosts are ranked for peer selection in the context of p2p applications peering, based on the hosts’ locality, distance, lifetimes, etc., which is discovered and collected through the search process.

While it is certainly feasible to engineer the above functions with available technology, they require a con-

siderable effort in devising the individual subsystems that identify, localize and collect the network data in real-time. Our point is that a general network search functionality would significantly accelerate the development time for such functions. Today, their development struggles not only to keep up with ever-increasing heterogeneity, but also to deal with the increasing volume of data that network elements are able to generate.

In the past monitoring generally involved occasional polling of key MIB objects, as well as listening to traps and syslog messages. Today, however, often large amounts of Netflow data are exported, with numerous per-flow statistics, and we anticipate that services will soon take periodic snapshots in rapid succession of most operational state. This trend makes real-time exploration and analysis of network data increasingly costly and requires significant infrastructure outside the network.

Network search will require more resources than monitoring, as the search process involves discovery, not only data retrieval from known location. However, search can be coupled with aggregation of partial results, which allows some degree of processing of network data as part of the search process. To achieve scalability and fast response times to queries, we believe that network search must be performed inside the network infrastructure, supported through a decentralized, self-organizing search architecture.

Recent advances in research, as well as in emerging technologies, when leveraged properly, make network search feasible for today's network environments. The development of scalable search functions can draw upon results in resource discovery, in-network aggregation, distributed real-time monitoring and scalable storage—often using tree-based or gossip-based distributed algorithms—which have been produced over the last decade by the networking, management and distributed systems communities. Also, query processors have been developed for very large databases, as exemplified through Google's Dremel system [28]. The instrumentation that network search requires has not been supported by off-the-shelf technology until very recently, since networking devices were considered closed systems, and management functions were provided off-box.

Initially driven by the need to automate management workflows, e.g., through supporting scripts in network elements that are triggered by network events, devices are increasingly offering APIs that can be used to significantly extend their functionality. Also, additional processing capacity is becoming available within network elements, for instance in form of service blades, such

as Cisco's Network Analysis Module (NAM), originally used for passive measurements, deep packet inspection and real-time traffic analysis, but programmable for other uses. Furthermore, with the proliferation of multi-core CPUs on network blades, some cores are available for other purposes than traffic forwarding and signaling.

The contributions of the paper are as follows. We motivate and formulate the concept of network search and discuss related concepts. We outline an architectural framework for a distributed search system and describe its design space and associated research challenges. We report on a testbed implementation set up for exploration of this paradigm, where a simple search system with a search interface for an administrator has been built, together with an application for detecting traffic anomalies, which relies on network search functions.

Section II of this paper discusses concepts and work related to network search. Section III provides a proposed architectural framework for network search, and Section IV presents a testbed and a prototype we built for exploratory research. Section V discusses the design space and challenges of realizing network search functions, while Section VI presents the conclusions.

II. RELATED CONCEPTS

On a conceptual level, network search has similarities with the general field of *Information Retrieval*, which deals with searching for documents that match certain criteria in a large document space [27]. Each document is represented by an index, and a search query is matched against all indexes of the document space using a query processing system. The result of a query is a ranked list of indexes, whereby a higher ranked index indicates a closer match of the index with the query.

Web search can be seen as an application of information retrieval technology to the web. As exemplified by Google's search engine, web search is performed by matching keywords against the universe of web pages. Search results are presented as ranked lists of links to web pages, which can be accessed through a web browser. Matching is performed on a distributed inverted index of the web pages, using a dedicated search infrastructure outside the web. The index database is populated through so-called crawlers, which continuously navigate the web following the hyperlinks, i.e., the links between pages [19]. The ranking of the pages is computed through keyword matching and through the analysis of their hyperlink structure, for which several techniques have been developed, including HITS [21], PageRank [19] and HillTop [18]. More recent approaches consider the search history of social groups for computing the ranking [25]. With the advent of social tagging systems,

the concept of web search has been extended. In addition to web pages, the search space contains now people, movies, blogs, etc., and, in addition to hyperlinks, the ranking considers relationships, recommendations, sharing of links, etc. [39].

Recently, the search process has been adapted to enable *live search* on news, blogs, forums, and the like. Instead of using crawlers, the search database is updated through RSS feeds [13] [15]. The ranking of results in live search is computed differently from the ranking in traditional web search and uses reliability of sources, trends in keywords popularity, locality of information, freshness of data items, and similarity of objects. A concept for live search, which researchers from Google call “query-free-news search” and which primarily computes the rankings according to object similarity, is presented in [24]. An example of a search system that ranks blogs using several of the above mentioned criteria is BlogScope [15].

Over the last years, the research area in the Internet of Things, which investigates the internetworking of physical entities, which generally contain sensors and actuators, gave rise to an effort called *Web of Things*, which includes adapting of web search to search within the space of Internet of Things. Entities in the Web of Things are realized as web pages, which means they can be identified through URLs and accessed through web protocols. Further, these entities span a wide range of real-world objects, from rooms in buildings to mobile phones. These entities contain sensors and thus have associated quantities, from occupancy levels, over temperature, video captures to locations. A survey of concepts in the Web of Things and their implementations can be found in [31].

While it is possible to use traditional web search systems to search the Web of Things, additional matching schemes have been developed. They include comparing measured quantities against specific values, allowing to search, for instance, for an empty room in a building or the last known location of a cell phone. In addition, specialized ranking schemes have been devised. The search system is generally realized as a hierarchy of processing nodes, which perform the search and collect the results. Various implementations in the contexts Web of Things are presented in [23], [30], [38], [40], [41].

An active, established field that relates to network search is the area of query processing in *very large databases*. A key topic of this research is assuring the scalability of query response times, which can be achieved through various forms of weak consistency or through optimizing the system for certain types of

queries. The Dremel system is an example of the latter [28]. It enables fast response times for queries on tables with large numbers of columns, whereby only a small number of column attributes appear in the query. Queries are executed in a distributed fashion on servers that hold data, which is uniformly structured and potentially replicated. In [28], a prototype is reported on, which executes within 20 seconds a query on a single attribute in a database of 85 billion records that are horizontally partitioned over 3000 nodes.

The process of *mining event logs* from a network is related to network search. Event logs are created by network elements and collected in data collection points. Mining tools process either a database of event logs or a real-time stream of events. The processing is generally performed in a centralised way, whereby one mining process acts on one data store. Data mining is often used on event logs for anomaly detection. In [35]–[37], for example, frequent itemset mining is applied (a) to netflow logs in order to identify network hotspots and (b) to network IDS logs in order to classify network alerts in real time.

Having reviewed concepts underlying, and technologies developed for, Information Retrieval, Web Search, Web of Things, very large databases, and data mining, we ask the question: to which extent we can these be applied for the purpose of engineering a network search system?

Information Retrieval offers an approach where a document is represented as an object of simple structure and characterized by an index. The search language is limited, and a search query is expressed using index terms. The search semantic is captured in the matching and ranking functions. Information Retrieval is applicable to domains with a large number of objects. It offers a simple query language, and queries can be efficiently executed over large data sets. What makes Information Retrieval potentially attractive for network search is the large number of network objects that could be covered. The downside is that those objects must be of simple structure.

Web Search and its extensions discussed above offer an approach to information retrieval whereby the object space is distributed across the Internet. Sophisticated versions of matching and ranking functions have been developed to reduce the result set of search queries, which may be helpful in developing network search concepts. In addition, protocols and tools have been engineered that are of potential interest to network search developers. In the context of Web of Things, strategies have been devised to allow for distributed search, by

distinguishing between index nodes and data collection points, for instance, which may be helpful in designing a scalable network search system.

The field of very large databases, as exemplified above through concepts realized in the Dremel system, has made advances in scalable query processing for certain classes of queries. Such queries, which are based on relational algebra, can be much more expressive than Information-Retrieval-type queries, at the expense of a higher computational complexity for query processing.

Log mining can provide a higher level of abstraction or more complex processing of network data than database processing and certainly Information Retrieval can. The difficulties of directly using log mining techniques for network search are that these techniques generally rely on a centralized data store and are not designed for real-time use.

III. A FRAMEWORK FOR NETWORK SEARCH

A. Modeling and Querying Network Information

For the purpose of modeling and querying network information, we advocate using concepts that underlie the above reviewed fields of Information Retrieval, Web Search and Web of Things. Our choice is motivated by the objective of having a network search capability suitable for real-time search in a vast object space that is distributed over a large networked system. In addition, we want to make data from legacy systems available for search in a straightforward, efficient way, and we want the search system to adapt to a changing network and easily cover new network elements.

Following our approach, designing a model for network search includes devising a scheme for object identification (i.e., a naming scheme) and a simple object representation model. In addition, a query language must be developed, together with matching and ranking functions, which define the semantics of the query process. In the following we outline a simple model for network search which meets to the above objectives.

In our framework, the object space for search is a collection of network objects that represent physical and logical entities of varying lifetime, including servers, routers, flows, virtual-machines, and device-counters. An object naming scheme should define names that are location-independent and expressive in the sense that they convey useful information about the nature of the objects. These two requirements exclude schemes such as oblivious integers (as used in [32]) and Uniform Resource Locators (URLs) [17]. A possible naming scheme could be based on SNMP OIDs, but we choose here a simpler alternative, namely Uniform Resource

Names (URNs) [29]. For example, an object representing an IP-flow with flow-ID ‘ $\{IP_A, IP_B, PORT_A, PORT_B, PROTO_X\}$ ’ can be identified by the URN ‘urn : network-search : ipflow : $IP_A, IP_B, PORT_A, PORT_B, PROTO_X$ ’.

A simple representation of an object is a bag of attribute-value pairs, where one pair contains the object name, i.e., its URN. This means that an IP-flow can be represented as ‘(oid, urn : network-search : ipflow : $IP_A, IP_B, PORT_A, PORT_B, PROTO_X$), (tcp-flag, $FLAG_F$), (bytes, CNT_{C1}), (packets, CNT_{C2}), (timestamp, $TIME_{T1} : TIME_{T2}$)’}. Note that the object model we outline here has less structure than either a relational database model [16], [26], [34] or an object model based on ontologies would provide [12], [20], [22]. This simplicity implies that significantly less meta information has to be maintained and that legacy data, for example from MIBs, can be more easily incorporated, although some structural information may be lost.

A query for the above object model can be expressed as a list of tokens, separated by conjunction ($\wedge$) operators, whereby a token can contain an attribute-value pair with a comparison operator, an attribute name or an attribute value. More complex query can be formed using disjunction ($\vee$) of single queries. The following grammar describes the query language, where Q stands for a query, T stands for a token, A stands for an attribute, V stands for a value and OP stands for a comparison operator.

$$\begin{aligned} Q &\rightarrow Q \vee \dots \vee Q \\ Q &\rightarrow T \wedge \dots \wedge T \\ T &\rightarrow A \text{ } OP \text{ } V \mid A \mid V \end{aligned}$$

The query matching process evaluates a query statement against the objects in the search space and returns those that match the query. It is useful to include two special operators in the query language, namely, projection (π) and aggregation ($\mathcal{A}$). The projection operator reduces the object information from a query result to include only specified attributes. The aggregation operator aggregates into a single value the set of the returned objects from a query. An interesting case in query matching occurs when the query contains a URN, since we allow that a URN is only partially defined, for example, ‘*ipflow* IP_A ’ instead of ‘urn : network-search : ipflow : $IP_A, IP_B, PORT_A, PORT_B, PROTO_X$ ’. The partially defined URN is then matched against all URNs in the search space through substring matching. (This form of attribute matching realizes a flexible form of name resolution.)

Here are some simple queries expressed in this language.

Fig. 1. Proposed architecture for a network search system

1. ‘oid=*IP_A*’ (‘oid’ denotes attribute type for a URN.)

The query returns all objects in the search space whose attribute ‘oid’ has a value matching the pattern ‘*IP_A*’.

2. ‘oid=*ipflow* ∧ packets>10’

This query returns all IP-flow objects with a packet count larger than ‘10’.

3. ‘ $\pi_{oid}(oid=*ipflows* \wedge packets>10)$ ’

This query returns (only) the URNs of the matched IP-flow objects.

4. ‘ $\mathcal{A}_{count}(oid=*ipflow*IP_A* \vee oid=*ipflow*IP_B*)$ ’

This query returns the number of IP-flow objects containing either IP_A or IP_B in their respective URNs.

To deal with the potentially large number of objects that are returned after executing a query, the query process includes a ranking function whereby each object to be returned gets ranked according to “how well” the query matches its attributes and/or “how fresh” the object information is. We envision that only the top-k matched objects from a query are returned.

B. An Architecture for Network Search

Our architecture for a network search system is depicted in Figure 1. Its key element is the *search plane*, which conceptualizes the network search functionality. This plane contains a network of *search nodes*, which have processing and storage capacities. A search node can communicate with a set of neighbors, which are identified through links of the network graph. The design of this plane supports searching in a distributed and parallel fashion. A search node can be realized in various ways: it can be part of the management infrastructure

outside the managed system, it can be run as a standalone network appliance, or it can be integrated into a network element using a variety of technologies. Our current prototype implements the second option, while we envision the third option for future realizations of network search systems.

The bottom plane in Figure 1 represents the physical network that is subject to search. Each network element is associated with a search node, which maintains (or has access to) configuration and operational data from that network element. The top of Figure 1 shows the management plane, which includes the systems and servers running processes for network supervision and management.

There are two important interfaces in this architecture. The first is the query interface, which allows a process in the management plane to execute a network search operation. We envision that every search node is an access point for such a query. The second interface defines the interaction between a search node and a network element, which can be realized through polling or can be push based. This interface is technology-dependent and possibly proprietary.

C. Algorithms and Functions in the Search Plane

Each search node runs a process that communicates with the associated network element(s) from which it retrieves network data. A database function dynamically maps that data into the information model for network search and updates the local search database.

Search functions, invoked from the management plane through query invocation, are executed as distributed algorithms on the graph of search nodes. During the execution of a query on a search node, the local search database is accessed, the matching of the local query against stored indexes is performed, and the local search result is possibly aggregated with results from other nodes.

We envision that network-wide abstractions of network data, such as traffic matrices and other global objects, are dynamically constructed and maintained by processes in the search plane which aggregate data from local search databases across time and space and which store the results in the same databases, thereby making network-wide abstractions available to search queries from the management plane.

To develop distributed search functions, results from very large database research are potentially helpful. Furthermore, research efforts in the fields of Web Search and Web of Things have produced many approaches to query matching, result ranking and result aggregation, which are useful developing scalable search functions.

IV. A PLATFORM FOR NETWORK SEARCH

A. The Testbed Infrastructure

Our testbed for experimentation, which is part of a larger laboratory infrastructure, includes 16 Cisco 2600 Series routers and 33 Hosts interconnected via four 100 Mbps FastEthernet switches. The routers run the OSPF protocol for routing. All hosts are rack-mounted Pentium-4 computers with a 2.8 GHz CPU and 1 GB RAM, running Ubuntu 10.04. 16 of the hosts are configured as traffic generators, which produce load using *pktgen* [6]. In addition, they use the *fping* and *hping* packages to inject anomalous traffic into the network [3], [4].

B. The Network Search Prototype

The routers in the testbed form the network plane on which search is performed (see Figure 1). With each router, a host is associated running a search node. There is one search node per router. The management plane currently includes a single management station on a dedicated host. Search nodes communicate with one another and with the management station through the routers in the network plane.

The search nodes dynamically populate their local databases with configuration data from the routers, such as interface descriptions, IOS version, and routing table (which change at slow time scales) as well as with operational data, such as network statistics from MIB variables and IP flow statistics from NetFlow caches (which change at fast time scales). The data is collected by polling SNMP MIB objects or by issuing CLI commands, at a rate reflecting the frequency of change of particular data items.

A database module on the search node converts the collected data to the object model of the search database described in Section III-A. The search database is kept in the memory of the search node to allow for fast access. It is organized as an inverted index and maintained using the indexer libraries of the Apache Lucene package [1].

A query module on the search node matches queries invoked on the management station with the content of the local database and returns the results. In our current implementation the results are not ranked. The query module uses Lucene's query libraries.

The network search function is realized in a simplistic way. Its functionality is centralized, and it executes on the management station. It is made up of two modules—the communicator and the aggregator. When the communicator receives a query from a management application, it forwards the query to all search nodes. The responses are then processed by the aggregator

module, and the result is handed to the application. While the centralized implementation of the network search function is sufficient for exploratory purposes on our testbed, it clearly does not demonstrate the scalability of the network search concept, which will be addressed in future work.

The modules of the search node are written in Java and shell script, and those in the management station in Python.

C. Application 1: An Interface for Network Search

We have implemented an interpreter for a query language that allows a management application to search for information in the network. The query language is a simple version of the one introduced in Section III-A and characterizes the interface between the management plane and the search plane.

The interpreter is written in Python [10]. In our current implementation, it runs on the management station, which processes a query by sending it to all search nodes, aggregating their responses, and returning the result to the invoker.

Here are some examples of using the query interface on the testbed. The examples are based on Python scripts.

Example 1: Search for the router with the highest load. In this use case, the script invokes the query $\pi_{oid,load}(oid=*router* \wedge load)$, which returns the URNs and the load of all router objects in the testbed. The script then identifies the router with the highest value.

Example 2: Search for flows that pass through two given routers.

The script invokes the query $\pi_{oid}(oid=*ipflow* \wedge router=A)$ and the corresponding query for $router=B$. Each query returns a set of URNs, one for each IP-flow traversing the routers *A* or *B*, respectively. The script then computes the result by intersecting the two sets.

Example 3: Search for SYN-flood attacks or flash-crowd events

The script invokes the query $\pi_{oid,tcp-flag}(oid=*ipflow*, tcp-flag=0x02)$, and the corresponding query $\pi_{oid,tcp-flag}(oid=*ipflow*, tcp-flag=0x11)$. These queries return the URNs of all IP-flows that either correspond to an initiated or recently terminated TCP session. The script then computes the number of session initiations and the ratio of session terminations to initiations for the network. An alarm is raised, if the number of session initiation is higher than a given threshold. If then furthermore the ratio is lower than a second threshold, the alarm indicates a possible SYN-flood attack. Otherwise, the alarm indicates a flash-crowd event.

The above examples illustrate that, in order to search for network data using our interface, neither the detailed structure of the information nor the location of the data items must be known. What we must provide as input to a query are solely attribute name(s) and value(s) that characterize the information we wish to retrieve.

D. Application 2: Network-wide Anomaly Detection

In this subsection, we present an application built on top of the simple search interface described in Section IV-C. The application is designed to detect network-wide anomalies, and we use it in experiments to find anomalies related to simulated network attacks on the testbed.

The application is written in Python and makes use of several open-source software packages. It uses operational network data associated with network flows and link utilization.

The anomaly-detection application periodically executes a series of steps as follows. Using the network search functionality, it queries the search plane for flow and utilization data. A Principal Component Analysis (PCA) component reduces the dimensionality of this data [9] to two dimensions. Then, a clustering component groups the data points into recognizable clusters. After that, unusual clusters, i.e., clusters that are far from the center of gravity of all data points, are identified. Then, a data mining technique, implementing a frequent-itemset-mining scheme, identifies flow patterns that are associated with the unusual clusters. Finally, these flow patterns are matched against possible attack signatures.

For performing PCA and clustering, the application makes use of FactoMineR, an R-based open-source tool for multivariate data analysis ([2], [11]). For performing frequent itemset mining, it relies on the LogHound package [5].

We illustrate the operation of the anomaly-detection application through an experiment on our testbed. We inject TCP traffic into the network, which we consider a “normal” traffic pattern for the purpose of the experiment, using the traffic generators equipped with pktgen. In addition, we periodically inject a sequence of three traffic patterns, each of which simulating a network attack—a ping sweep (for 5 sec), a port scan (for 3 sec) and a synflood (for 3 sec) ([7], [8], [14]), which are some 30 sec to a minute apart from one another. The attacks are created using two network analyzing tools, fping and hping ([3], [4]).

During the experiment, a search query is executed every second, its result set is aggregated into a vector of 17 components, and the analysis is performed over a window size of the latest 20 such sets. Figure 2 gives

the output of the FactoMineR module at four distinctive times during the experiment. It shows the projection of network states in the first two principal components, after PCA and hierarchical clustering has been performed. Figure 2(a) refers to the network under normal operation, in which clusters 1 and 3 are marked as normal, and 2 and 4 as unusual. Figure 2(b) refers to the network during a port scan, in which cluster 2 is marked as normal, and 1 and 3 as unusual. Figure 2(c) refers to the network during a Denial of Service (DoS) attack, in which clusters 2 and 3 is marked as normal, and 1 and 4 as unusual. Figure 2(d) refers to the network during a ping sweep, in which clusters 1 and 3 are marked as normal, and 2 and 4 as unusual.

After identifying the unusual clusters, the application invokes the loghound module, which performs frequent itemset mining on the flow data associated with the data points in the unusual clusters. The module returns flow patterns of high frequency. In the data associated with Figure 2(a), it identifies several patterns, none of which matches any known attack signature. In the data associated with Figure 2(b), the module detects several patterns, for example *srcIP:192.168.5.150 dstIP:192.168.2.150 * * proto:06 pkt:1 freq:9.3%*. This pattern is consistent with the signature of a port scan, because it represents a large number of flows, each consisting of a single packet, sent to different ports of the same node. Similarly, the module detects patterns from the data associated with the unusual clusters in Figure 2(c) and (d), which are consistent with the signatures of a DoS attack and a ping sweep, respectively.

During the course of this experiment, our application correctly detected anomalies related to the three simulated attacks.

The application performs an analysis of the network state every second and produces an alarm, if an anomaly is detected, within some 250 milliseconds from the start of an analysis cycle.

V. DESIGN SPACE AND CHALLENGES

The task of engineering the search plane introduced in Section III and building novel applications, such as those described in Section IV, opens up many interesting problems. Due to lack of space, we limit ourselves to the design space and research challenges associated with devising efficient search algorithms. We leave out other issues, including searching in a multi-domain environment, privacy aspects of search data, securing the search infrastructure, handling search nodes with different capacities, etc., which will be discussed elsewhere.

The design goals for a search algorithm are short execution time, low overhead in consuming search plane re-

Fig. 2. Four snapshots of the network state at different times of the experiment. The dimensionality of the state-vectors from network search are reduced to two dimensions using Principal Component Analysis. The state-vectors are then clustered into a small set of recognizable groups, using a hierarchical clustering method. Clusters far from the center of gravity are identified as outliers, which need to be further investigated. (a) refers to the system under normal operation, (b) to a DoS attack, (c) to a ping sweep and (d) to a port scan.

sources, and scalability, which means sub-linear growth of these two metrics with increasing system size. In addition, a search algorithm should dynamically adapt to changes in the network configuration and provide results with high precision. Obviously, these metrics cannot all be jointly optimized, and, therefore, the tradeoffs need to be studied, and engineering solutions need to be developed to make them controllable.

A naive approach to search is flooding the search plane with local queries, possibly using a gossip algorithm. To avoid the initiator of the search query from being overloaded with answers from individual search nodes, a wave algorithm, such as echo, will likely perform better, as it allows aggregating the partial results us-

ing a spanning tree [33]. To further reduce the search overhead, exploiting domain-specific knowledge to guide the search process and applying heuristics to restrict the search space may prove effective. For example, knowing that IP addresses are subject to firewall rules, that they can designate source and destination addresses of flows, that they are associated with systems which have names and aliases, that they are assigned by DHCP servers out of address pools, etc., may help speeding up the processing of certain queries. When a query contains an IP address as a search term, the search algorithm may check Access Control Lists and flow entries and follow entries to the next hop a flow gets routed to, or the algorithm may check a system's DNS server

for corresponding domain names and the DHCP server for information about the IP's lease. Similarly, for a query with a parameter IP address, knowledge about the network topology and routing, both of which can be dynamically acquired, can be used to propagate the query and bound its search space.

When engineering the search plane for efficient operation, the dynamics or life times of network data must be considered. Information related to physical system configuration or installed software licenses is fairly long-lived, which allows caching. Other information, on short-lived flows for instance, does not, as is changing too fast. (Certain local statistics may even be computed on demand, as continuous updates may be too expensive.) A related problem is the placement of index data, against which queries can be executed. The fact that much network data is transient suggests that indexes should be kept on or close to the search nodes. Centralizing and replicating index data to a certain degree will shorten the execution of queries, while, at the same time, will increase the resources in the search plane needed for updating the index. A possible approach that allows to control this tradeoff involves maintaining distributed index trees in the search plane, whereby individual indexes are pushed towards the root, depending on their particular lifetime.

VI. DISCUSSION

In this paper, we motivated and introduced the paradigm of management by network search. The paradigm addresses the problem of diversity in monitoring interfaces, by introducing a search mechanism that allows uniform access to network data in a simple format and in a way that is oblivious to the data location. The traditional “precise” monitoring interfaces are replaced by a single “less precise” query interface. The implication of this type of interface on the type of management tasks that are particularly suited for the paradigm needs further investigation.

Leveraging technology trends that allow more than ever customized in-network processing of management information, we advocate that network data is accessed and aggregated inside the network, which can reduce the infrastructure needed today for processing monitoring data outside the managed system.

Having access to a search query interface, as described in this paper, can accelerate the speed of developing management applications, specifically for those applications that require data from various sources at potentially unknown or changing location, possibly in aggregated form. We expect the emergence of novel solutions to applications with these requirements.

From the experimental work described in this paper, we draw the following conclusions. Comparing our concept of a network-search language to an SQL-based query language for networked systems [26], we note that queries in our language are stated in a simpler and “freer” form, without knowledge of the global schema. On the negative side, our approach cannot directly use the query processing framework that has been developed for SQL-based languages, which allows to efficiently process many classes of queries in a large-scale networked system [26]. Second, using public-domain software packages, we were able to develop a complex application for anomaly detection within a short period of time (approximately two weeks). A significant factor that made this possible was the availability of a network search system (although a very primitive one), which gave us uniform access to operational network data.

REFERENCES

- [1] Apache Lucene - Overview. Lucene. September 2011. <<http://lucene.apache.org/java/docs/index.html>>.
- [2] FactoMineR. September 2011. <<http://factominer.free.fr/>>.
- [3] fping. September 2011. <<http://fping.sourceforge.net/>>.
- [4] hping. September 2011. <<http://www.hping.org/>>.
- [5] LogHound - a tool for mining frequent patterns from event logs. LogHound. September 2011. <<http://ristov.users.sourceforge.net/loghound/>>.
- [6] Open Source Traffic Analyzer. pktgen. September 2011. <<http://tslab.ssvl.kth.se/pktgen/>>.
- [7] Ping sweep. Wikipedia. September 2011. <http://en.wikipedia.org/wiki/Ping_sweep>.
- [8] Port scanner. Wikipedia. September 2011. <http://en.wikipedia.org/wiki/Port_scanner>.
- [9] Principal component analysis. Wikipedia. September 2011. <http://en.wikipedia.org/wiki/Principal_component_analysis>.
- [10] Python programming language - official website. Python. September 2011. <<http://www.python.org/>>.
- [11] R project for statistical computing. R. September 2011. <<http://www.r-project.org/>>.
- [12] Rdf. January 1, 2012, <<http://semanticweb.org/wiki/RDF>>.
- [13] RSS 2.0 specification. RSS. September 2011. <<http://www.rssboard.org/rss-specification>>.
- [14] Syn flood. Wikipedia. September 2011. <http://en.wikipedia.org/wiki/SYN_flood>.
- [15] Nilesh Bansal and Nick Koudas. Blogscope: a system for online analysis of high volume text streams. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 1410–1413. VLDB Endowment, 2007.
- [16] Alexander Behm, Vinayak R. Borkar, Michael J. Carey, Raman Grover, Chen Li, Nicola Onose, Rares Vernica, Alin Deutsch, Yannis Papakonstantinou, and Vassilis J. Tsotras. ASTERIX: towards a scalable, semistructured data platform for evolving-world models. *Distributed and Parallel Databases*, 29(3):185–216, 2011.
- [17] T. Berners-Lee, L. Masinter, and McCahill M. Uniform resource locators (URL). IETF, RFC 1738, December 1994. [Online]. Available: <http://www.ietf.org/rfc/rfc1738.txt>.
- [18] Krishna Bharat and George A. Mihaila. When experts agree: using non-affiliated experts to rank popular topics. *ACM Trans. Inf. Syst.*, 20:47–58, January 2002.
- [19] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

- [20] C. Dannewitz. Netinf: An information-centric design for the future internet. In *Proceedings of 3rd GI/ITG KuVS Workshop on The Future Internet*, May 2009.
- [21] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets*, chapter 14, pages 397–435. Cambridge University Press, 2010.
- [22] Metin Feridun and Axel Tanner. A search engine for systems management. In *NOMS*, pages 697–710, 2010.
- [23] Christian Frank, Philipp Bolliger, Friedemann Mattern, and Wolfgang Kellerer. The sensor internet at work: Locating everyday items using mobile phones. *Pervasive Mob. Comput.*, 4:421–447, June 2008.
- [24] Monika Henzinger, Bay-Wei Chang, Brian Milch, and Sergey Brin. Query-free news search. In *Proceedings of the 12th international conference on World Wide Web, WWW '03*, pages 1–10, New York, NY, USA, 2003. ACM.
- [25] Anne-Marie Kermarrec. Challenges in personalizing and decentralizing the web: An overview of gossip. In Rachid Guerraoui and Franck Petit, editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 5873 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin / Heidelberg, 2009.
- [26] Koon-Seng Lim and Rolf Stadler. Real-time views of network traffic using decentralized management. In *9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*, Nice, France, May 2005.
- [27] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [28] Sergey Melnik, Andrey Gubarev, Jing J. Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive analysis of Web-Scale datasets. In *The 36th International Conference on Very Large Data Bases*, volume 3, September 2010.
- [29] R. Moats. URN syntax. IETF, RFC 2141, May 1997. [Online], Available at: <http://datatracker.ietf.org/doc/rfc2141/>.
- [30] B. Ostermaier, K. Romer, F. Mattern, M. Fahrmaier, and W. Kellerer. A real-time search engine for the web of things. In *Internet of Things (IOT), 2010*, pages 1–8, 29 2010-dec. 1 2010.
- [31] K. Romer, B. Ostermaier, F. Mattern, M. Fahrmaier, and W. Kellerer. Real-time search for real-world entities: A survey. *Proceedings of the IEEE*, 98(11):1887–1902, nov. 2010.
- [32] Alan Shieh, Emin Gün Sirer, and Fred B. Schneider. NetQuery: a knowledge plane for reasoning about network properties. In *Proceedings of the ACM CoNEXT Student Workshop, CoNEXT '10 Student Workshop*, pages 23:1–23:2, New York, NY, USA, 2010. ACM.
- [33] G. Tel. *An Introduction to Distributed Algorithms*. Cambridge University Press, 2000.
- [34] Jeffrey Ullman and Jennifer Widom. *First course in database systems*. Prentice-Hall Inc., 1997.
- [35] R. Vaarandi. Mining event logs with slct and loghound. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 1071–1074, april 2008.
- [36] R. Vaarandi. Real-time classification of ids alerts with data mining techniques. In *Military Communications Conference, 2009. MILCOM 2009. IEEE*, pages 1–7, oct. 2009.
- [37] R. Vaarandi and K. Podins. Network ids alert classification with frequent itemset mining and data clustering. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 451–456, oct. 2010.
- [38] Haodong Wang, C.C. Tan, and Qun Li. Snoogle: A search engine for pervasive environments. *Parallel and Distributed Systems, IEEE Transactions on*, 21(8):1188–1202, aug. 2010.
- [39] Sihem Amer Yahia, Michael Benedikt, and Philip Bohannon. Challenges in searching online communities. *IEEE Data Eng. Bull.*, 30:2007.
- [40] Tingxin Yan, Deepak Ganesan, and R. Manmatha. Distributed image search in camera sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems, SenSys '08*, pages 155–168, New York, NY, USA, 2008. ACM.
- [41] Kok-Kiong Yap, Vikram Srinivasan, and Mehul Motani. Max: human-centric search of the physical world. In *Proceedings of the 3rd international conference on Embedded networked sensor systems, SenSys '05*, pages 166–179, New York, NY, USA, 2005. ACM.