DIPStorage: Distributed Storage of IP Flow Records

Cristian Morariu, *Student Member IEEE*, Thierry Kramis, Burkhard Stiller

***Abstract*—The storage of IP traffic traces increasingly grows more complex, since data flows tend to increase largely over time. Network operator's backbones generate each day hundreds of Gigabyte of IP flow records that need to be stored and analyzed. Handling such amount of data requires a high-performance hardware and software. One way to leverage performance requirements of a storing and analysis of traffic data is to design a distributed platform to replace centralized solutions existing today. This paper designs a scalable storage platform for IP flow records. The evaluation of the implemented prototype shows that such an approach can offer a good and practical solution for storing and retrieving high amounts of IP flow records.**

***Index Terms*—IP Flow, Flow Data Storage, P2P**

I. Introduction

During the last decade IP networking has constantly grown in popularity. Traditional services that have been deployed over dedicated network infrastructures for a long time, such as telephony, radio, or television, have slowly switched to IP (Internet Protocol). The "everything-over-IP" approach has a direct impact on network operators, as with each new service offered the amount of carried traffic increases. Such traffic increases impact highly the ability of network operators to perform traffic analysis, which is required by operators to understand the type of traffic they carry in order to provide better quality services, charge users based on the traffic they generate, plan network upgrades, or detect malicious traffic.

The measurement of IP traffic generated by a device connected to an IP network is the first step of several key operations in network management: load balancing, intrusion detection, performance monitoring, generation of traffic statistics, or charging for connection time, bandwidth used or volume transferred. Depending on the granularity of the data collected as well as on the type of link on which traffic is measured the information gathered may range from a few kilobyte per hour up to gigabytes per second. One of the main mechanisms to retrieve information about network traffic is flow accounting. Flow accounting is concerned with measuring the IP traffic on a per flow basis. A flow is typically identified as a unidirectional sequence of packets from one source point to one destination point having one or several common IP header fields. The most commonly used fields to identify an IP flow are: source IP address, destination IP address, IP protocol number, source port number, and destination port number. The IP header fields that define an IP flow are also called flow keys. In IP flow accounting during the traffic measurement process, at least the following information is collected for each IP flow: number of packets in the flow, total bytes transferred, start time of the flow, end time of the flow, TCP flags observed, incoming and outgoing router interface.

Today, storage of IP flow records is mostly done by centralized components. Since traffic volumes increase, such centralized solutions can no longer cope completely with the resulting data increase and use sampling as a mechanism to reduce data volumes. Therefore, another strategy for processing such high data volumes in the future is needed. Using Peer-to-Peer (P2P) mechanisms for the development of storage platforms has shown that reliability and robustness of centralized platforms can be highly improved [14]. Storage of IP flow records is very challenging, mainly because a retrieval operation is typically preceded by a complex search operation, thus, distributing the storage (and search operations as well) should greatly improve the performance of a query operation.

The IETF is currently standardizing the IPFIX [10] protocol as well as requirements for IP flow storage. These specifications define different templates to be used for flow records. The work presented in this paper is not constrained by a particular IP flow export protocol. The prototype implemented, however, uses NetFlow version 5 [18] as the export protocol for IP flow records. A change to the IPFIX protocol is foreseen for the near future and will require minimal change in the existing prototype.

The main goal of this work is to investigate, whether P2P mechanisms that proved efficient in other application areas can be used to improve storage scalability and query performance of IP flow storage by distributing these tasks to multiple nodes. Distribution in the context of this paper does not imply (but does not exclude either) geographic distribution. Nodes may be collocated or distributed within the network of an ISP. DIPStorage provides the underlying mechanisms that allow multiple nodes to share their resources for storing IP flow records. In order to achieve this goal, a prototype was

.Manuscript received July 14, 2008. This work was supported in part by the IST Network of Excellence EMANICS funded by the European Union under contract number FP6-2004-IST-026854-NoE and in part by the DaSAHIT project funded by the Swiss National Science Foundation under contract number 200021-118128/1.

C. Morariu is with the Department of Informatics, University of Zurich, CH-8050 Zurich, Switzerland. (Phone: +41 44 635 6751; Fax: +41 44 635 6809; e-mail: morariu@ifi.uzh.ch).

T. Kramis is with the Department of Informatics, University of Zurich, CH-8050 Zurich, Switzerland (email: kramis@ifi.uzh.ch).

B. Stiller is with the Department of Informatics, University of Zurich, and is associated with TIK ETH Zürich, CH-8050 Zurich, Switzerland (email: stiller@ifi.uzh.ch).

designed, implemented, and evaluated. The evaluation shows that DIPStorage meets the desired goal. Due to the fact that the distributed storage platform was based on P2P mechanisms, traditional database (centralized or distributed) systems have not been evaluated.

The remainder of this paper is organized as follows: Section II gives an overview on other work in the field of IP flow accounting and distributed traffic monitoring to differentiate the new approach, as presented in Section III, which provides an overview on the architecture of DIPStorage. Section IV shows key implementation details of the prototype developed. Finally, Section V presents evaluation results and Section VI concludes the paper.

II. Related Work

With the constant increase of traffic observed on network operators' backbone links major research is focused on the field of packet sampling and flow sampling in order to significantly decrease the amount of traces that an operator needs to process. The authors of [2], [3], [5], [24] present packet and flow sampling algorithms that besides reducing the amount of data also keep the error of the sampling estimations within low limits. Those sampling proposals, although alleviating the computational requirements of high-speed packet processing, are not very accurate in some scenarios where complete information is required (such as Intrusion Detection Systems (IDS) or usage-based charging systems). Investigations have been made into detecting how sampling algorithms impact the performance of intrusion detection systems. [1] and [15] show that the sampling rate directly impacts the quality of intrusion detection. The work of [6] outlines that sampling may also decrease the revenue of network operators or it may artificially increase users' bills when sampled data is used for charging.

Different distributed architectures for network monitoring tasks are proposed in literature. In [4] the authors introduce the idea of *trajectory* sampling in the context of IP flow accounting. In their approach each packet is processed either by a single router, either by all the routers on the packet's path. However, this solution does not guarantee that a packet always reaches a router responsible with its capturing. In [13] a distributed packet capturing architecture is proposed based on a high-performance machine that splits the traffic across multiple capture nodes. A similar approach is found in [9] with the main advantage that different tasks of network monitoring are distributed while storage uses several databases each storing the data aggregated at different time scales (*e.g.* 5 minutes, 1 hour, 24 hours intervals).

In the area of IP flow records storage *flow-tools* [7] and *nfdump* [19] are two widely used open-source tools. The main disadvantage of these tools is that they are centralized, thus, suffering of performance drops in case of large volume of IP flow records stored.

Several P2P storage systems such as [14] and [22] are proposed by the research community as well as the industry. These solutions are targeted mainly towards persistent file storage. In the context of IP flow records storage such a solution would produce a large overhead due to the small size of stored objects and a file system approach included. Moreover, these solutions are not optimized to query and aggregate a large number of objects.

III. DIPStorage Architecture

Existing centralized solutions to IP flow collection have several major drawbacks: they lack storage scalability, suffer from a single point of failure, and flow retrieving performance degrades as the number of stored IP flow records increases. P2P systems have proven to be an efficient solution to scalability and reliability [14] problems of centralized storage systems. Since all nodes are *"equal"*, the tasks of a node, which leaves the network, may be taken over by any other node. In order to benefit from advantages of P2P applications, the architecture of DIPStorage is built on P2P concepts. In general, DIPStorage does not require that nodes are geographically distributed, but it provides a platform for sharing resources for storing IP flow records.

The main component of DIPStorage architecture is a Distributed Hash Table (DHT). This DHT is the enabler of DIPStorage. The way it is organized highly influences the way flow records are distributed in the storage network. Each node of the DHT is responsible for storing of a subset of flow records. For each IP flow record received by the storage platform a 64 bit flow ID is calculated by applying a hash function on the IP flow keys. Each node is responsible for the storage of a range of flow IDs. Based on its flow ID each IP flow record is routed through the storage platform to the node responsible for its storage.

Two strategies have been identified for routing an IP flow record within the storage platform: *random storage* strategy and *structured storage* strategy. In the *random storage* strategy the flow ID for each IP flow is randomly generated. By the *random strategy* approach flow IDs are uniformly distributed in the$[0\text{-}2^{64}\text{-}1]$ interval. If this interval is split in equal parts and each of the storage nodes is responsible for one of those intervals, the storage load is distributed evenly between storage nodes. However, the main disadvantage of this approach is that each query for IP flow records needs to be broadcasted to all nodes and processed by all storage nodes.

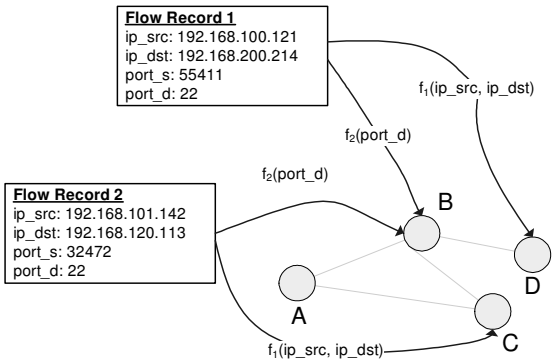

Fig. 1. Flow Storage in Structured Storage Strategy

Therefore, the *structured storage* approach addresses this drawback by grouping IP flow records with similar flow keys *close* with respect to their flow ID. Fig. 1 shows the effect of

how *structured storage* strategies store two IP flow records with similar characteristics. The figure shows two flow records representing two distinct secure shell (ssh) flows having a common destination port number (22), but different source and destination IP addresses. In the example above, two hash functions (f_1 and f_2) are used for storing each flow record twice. By using multiple hash functions for storing a single flow record redundancy is introduced in order to achieve higher fault tolerance. As f_1 is a function of IP source address and IP destination address, hash results of this function for the two flow records in the above example are different. This is shown in the figure by the storage of the two flow records on different nodes (C and D). Since *f2* generates a hash code based only on the destination port number, which is the same for the two IP flows in the example, the resulting hash codes for the two flow records are the same so both flow records will be stored on the same node. Therefore, DIPStorage is based on the *structured storage* strategy.

A. Tank-based View

For storing data DIPStorage uses attributes from IP flow records, which include IP addresses and ports from both source and destination. As observed in the example shown in Fig. 1, the chosen hash function and the flow keys to which it is applied highly influence the way the IP flow records with similar flow keys are distributed in the storage network.

DIPStorage establishes the idea of a *tank-based view*. A Tank is a subset of peers which form a group that stores IP flow records under a specific set of rules. These rules include a hash function and the corresponding flow keys to which it is applied.

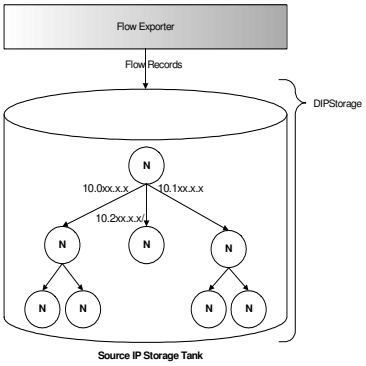

Fig. 2. Tank-based View

Drawing from the ideas of JXTA (Juxtapose) [12], a single tank forms a group of interest that actually stores incoming data based on a specific attribute. A single tank is able to completely handle incoming data and queries on it's own without requiring other tanks to be working. A single tank is being constructed using a *tank indexer* (*TI*) to manage the nodes in the respective tank. Although all nodes are equal within a tank group, they assume data storage responsibilities in a treelike fashion. Depending on available nodes the tank may have multiple levels, where each level refines the granularity of the data stored in the subtree. The more nodes or respectively peers are available, the more fine-grained the data routing can be. In case the storage tank uses the source IP address of the IP flow record as its storage attribute, routing is done by splitting the IP range by the number of nodes available on each level. In the IP case, the first level splits the first part of the IP address, the second the second part and so on. Whenever a node receives a flow record it calculates its flow ID and decides, whether it is responsible for that flow record. If not, then it calculates — based on the flow ID and the known children — the next hop where to send the flow record to. Since different tanks in DIPStorage have different storage rules, search queries can be efficiently routed through the storage network to find data according to an attribute (for example the IP source address). More specifically, nodes within a storage tank can be queried directly, because their parent tank indexer knows exactly where data may be stored and therefore redirects the query to a specific subtree. In case of multiple levels involved, every parent of a subtree sends the query down to its matching children. It waits for all answers, aggregates them, and sends them to its own parent. In order to avoid deadlocks, a time-out prevents a node to wait indefinitely long for an answer from one of its children. If the time-out is reached, the parent node can initiate any administrative tasks needed to rejoin the missing node or to recover the data lost. By introducing levels of responsibilities, the system is able to handle failovers efficiently. If a parent node fails, the second level of responsibility is delegated toward child nodes, which shall elect a replacement for the missing node. All child nodes during the election process need to answer incoming queries all together.

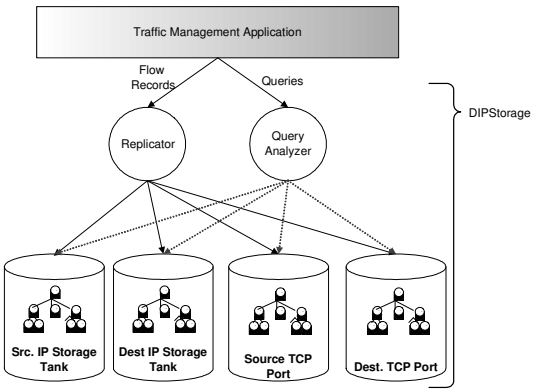

Fig. 3. Multi-Tank View

B. Multi-Tank Organization

Since a single data tank stores IP flow records based on a particular IP flow attribute each query that does not contain that attribute cannot be optimized. This happens because the query needs to be forwarded to all nodes in all sub-trees as the information required to route the query is missing. In order to address this issue, DIPStorage uses several data tanks for storing the IP flow records under different rules. Fig. 3 shows how four data tanks are used by DIPStorage to store the IP flow records based on: source IP address, destination IP address, source port number, destination port number. Based on the query received, the *query analyzer* decides which data tank is best optimized for answering the query, and then forwards the query to that tank.

A traffic management application deals with the generation

of flow records and their delivery to DIPStorage as well as with creation of queries of IP flow records. For example, a router with NetFlow export capabilities could be used to generate IP flow records. The flow replicator is responsible for distributing incoming data evenly across the tanks. Every tank stores each IP flow record according to its internal distribution rules. Subsequent queries for data can be handled by the appropriate data tank. In case one of the tanks fails to provide the requested data, other tanks can still potentially answer the query, although in that case the overall query time may be higher. The *query analyzer* has the task to decide which data tank a query is routed to such that the query is answered efficiently. Multiple tank strategies also provide redundancy. In case any node or a whole tank fails, other tanks have the ability to get the lost data and send it for duplication in the affected tank.

C. Load Balancing

One of the reasons for distributing IP flow records to different nodes is to decrease the query time in case of large IP flow records databases. Preferably the data should be equally distributed among participating nodes, so that response times for any two queries are similar. As the flow storage responsibility is based on a hash value of the IP flow record, the load of a node can be increased or decreased by changing the range of hash values for which that node is responsible. A load balancing process is started whenever a node is detected to store considerably more flow records then an overall average. As nodes may be added or removed arbitrarily from DIPStorage, a load balance mechanism also allows for redistribution of work in these situations.

D. Redundancy

One key concept of DIPStorage is the storage of each IP flow record in several data tanks. Such an approach has a twofold advantage: on one hand it achieves a better fault tolerance by redundant storage of each IP flow record, while on the other hand it improves the query efficiency by optimizing each data tank for a particular type of queries.

Since each flow record is stored more than once, if a storage node disappears from the storage network, its data is not lost from the system, but only from its data tank. Once a node is detected as “down“, its data tank requests one of the other data tanks the missing data. For doing that, the data tank that detects a missing node first calculates the range of IP flow records for which the missing node was responsible. Afterwards the data tank performs a rebalancing of the load, so that the missing range is reassigned to other nodes in the data tank. Once the load balancing is done, the data tank asks one of the other tanks for all flow records within the missing range. Using this technique, an IP flow record is lost only if the respective responsible nodes in every tank are lost in a short time interval.

The second advantage of having redundant data, which is organized under different rules, is to optimize different types of queries. For example, a query such as “*get all IP flow records which have the source IP address 10.100.100.1*“ can be answered very fast by the data tank 1 (cf. Fig. 3), because the query can be routed directly to the node that stores flow records with that source IP addresses. However, a query such as ”*get all flow records with the destination port 22*“ asked to data tank 1 would have the effect of being broadcast to all nodes in the data tank. This second query can be answered more efficiently by data tank 4, since it stores IP flow records based on destination port numbers. Such a behavior would be implemented in a traditional databases approach by using *index* files. In the presented scenario, however, such index files would be too expensive to keep due to the time required for the update operation.

IV. Implementation

The implementation of DIPStorage follows a layered approach as shown in Fig. 4. The P2P layer of DIPStorage provides the ability to handle a P2P network on top of Pastry technology, and further more, establishes a second overlay for building a hybrid super peer network. The P2P layer is implemented on top of Freepastry [8], which is a JAVA-based framework, for building Pastry networks. On top of Pastry DIPStorage uses an add-on called Scribe [20], which is also a part of the Freepastry framework.

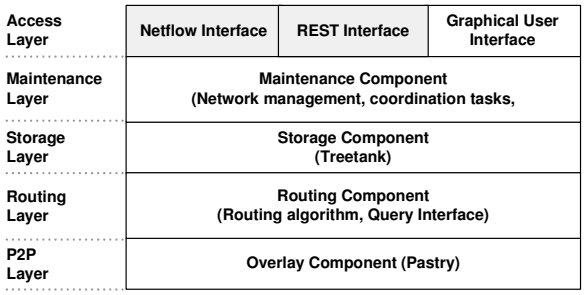

Fig. 4. DIPStorage Layers

The P2P layer also contains all messages that need to be sent from one node to another using either Pastry directly or the multicast mechanisms of Scribe. Therefore, a message either implements the *Message* Class from Pastry, or the *ScribeContent* Class from Scribe. If a message travels through Scribe and Pastry, it primarily implements *ScribeContent* and is then packed into a Pastry Message when sent directly through Pastry. Most messages either form a request for an arbitrary task or a response to it. The routing layer handles two tasks. It routes incoming data to the appropriate storage client and queries the system efficiently. The routing of IP flow records is based on IP flow attributes. Each node knows where to route an IP flow record or query based on flow keys.

The storage layer includes the main storage component of DIPStorage. The main intention was to use an existing system that is optimized for storage and query efficiency. One of the bigger open-source database systems available, MySQL [17], introduces drawbacks. First, it requires a node to have a running MySQL installation. Second, it does not enable easily replication. Furthermore, the scheme of the database has to be designed in advance and embedding it requires the usage of SQL language (neglecting the existence of persistence frameworks). This is why DIPStorage uses a JAVA-based XML Database called TreeTank [21], which enables DIPStorage to provide easy replication and standardized querying through XPath [23] queries. The XML nature of that

database enables DIPStorage to either store the data in a treelike structure on physical nodes, when enough nodes are available, or to build a XML based tree structure on one node. Additionally, traditional NetFlow collector engines, such as nfDump or flow-tools, may be used as storage components. In such a case DIPStorage would serve as an enabler for a distributed operation of those tools.

Basic tasks of the maintenance layer are the self organization of the underlying P2P network. The first node to join DIPStorage starts acting as a *Bootstrap Server*. Nodes within DIPStorage do not need necessarily to know the IP address of a *Bootstrap Server*, but can get relevant service information through the MulticastDNS [16], which is implemented in DIPStorage using JmDNS [11]. Starting with the second node, each participant generates an identifier and connects to a *Bootstrap Server*. There may be multiple nodes acting as a *Bootstrap Server* in order to load-balance the workload or for resilience against failovers. One of the duties of a *Bootstrap Server* is the assignment of tasks to new nodes. As soon as a sufficient number of nodes are ready to form one main tank, the *Tank Coordinator* will enable DIPStorage to receive and query data by adding middleware nodes.

V. Evaluation

In order to evaluate DIPStorage, a functional evaluation as well as a performance evaluation was performed. For doing the tests, a set of 6 to 15 nodes have been used in different DIPStorage configurations. During all the tests nodes were connected via a single Local Area Network. The functional evaluation shall show that the prototype can build a storage network, can perform P2P node management within the storage network, and can route IP flow records and queries within the data tanks. Therefore, the main goal of DIPStorage mentioned in the introduction of the paper has been achieved. Using P2P mechanisms the DIPStorage prototype may be used to build a scalable storage platform for IP flow records. The current version of the prototype, however, does not replicate IP flow records, and only uses a single data tank for storage. The addition of this functionality is foreseen for a next version of the prototype.

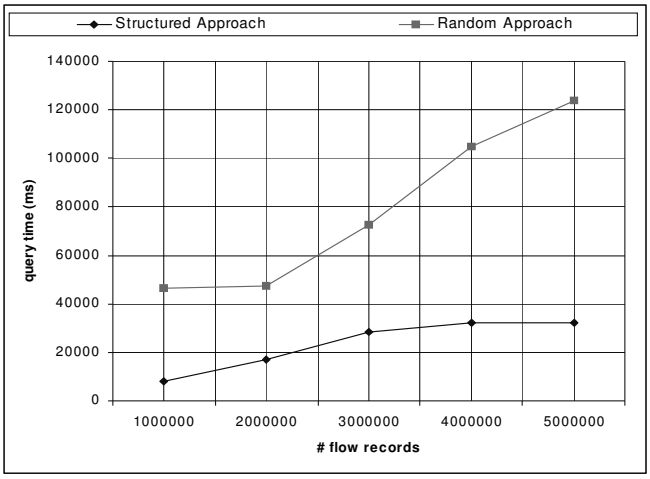

Fig. 5. Distributed Storage Strategies Comparison

The second part of the evaluation investigated how much query performance improvement DIPStorage achieves in comparison to a centralized solution. During the performance evaluation two different distributed storage strategies (random approach and structured approach) have been evaluated in order to see which one performs better under load. For these tests the storage network was not idle, but was receiving flow records at a constant rate.

The test consisted out of a set of queries running in parallel. The results presented in Fig. 5 show that the structured storage strategy outperforms the random storage. The main reason is that in the structured approach the query is routed directly to the responsible storage node, which makes a lookup in its local database, while in the random storage the query is received by all nodes and all of them have to perform a local lookup. In case of multiple different queries running in parallel, each query is processed by a different node when using the structured approach. When random strategy is used, all nodes need to process all queries.

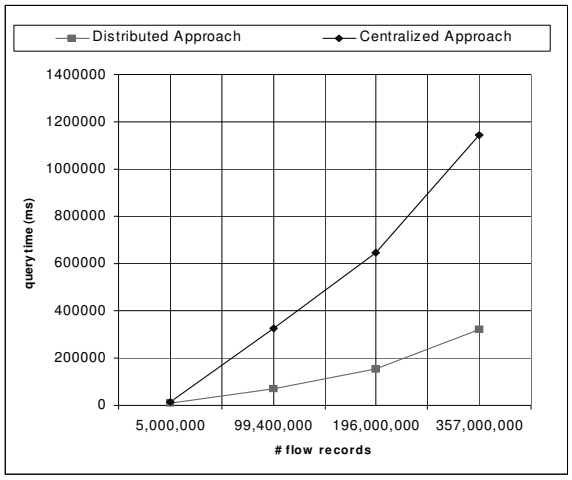

Fig. 6. Centralized Approach vs. Distributed Approach

The second performance test compares the performance of the structured distributed storage with the performance of a centralized storage system. The centralized storage system consists out of a single storage component that stores all flow records received locally. The storage component for the centralized storage system was identical to the one used by the nodes in the distributed approach. This comparison measured the performance increase of work distribution. Results are shown in Fig. 6. For 357 million records, the increasing ratio between query time in the centralized approach and the distributed approach shown in the second test demonstrates that with more flow records added to the storage network the single node performance will deteriorate faster compared to the distributed structured approach. This is because a single node needs to query all the IP flow records, while in the distributed approach each node needs to query a smaller number of flow records. The ratio between the single node and the distributed structured version increases as more nodes are added to the storage network as in the latter case each node has to store less data.

It is interesting to see, how the DIPStorage approach handles load. A centralized component may handle loads not very well, because storage and query tasks, as well as administrative tasks

need to be handled by one single component. In the distributed approach, these tasks are split onto multiple peers. The evaluation of DIPStorage includes a differentiation on the performance between a storage network under load and an idle storage network with respect to query performance. As Fig. 7 shows, the difference in performance between these two cases considered is minimal. This means that the distributed approach used in DIPStorage can handle a high load in a fairly efficient way, which gives confidence in the system to grow with future needs.

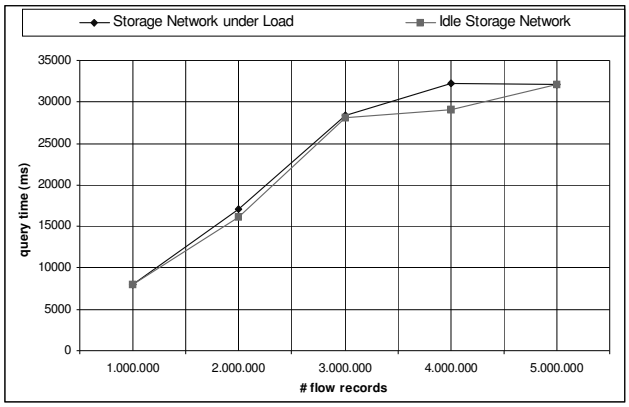

Fig. 7. Load Resilience in Distributed Approach

VI. Concluding Remarks

With the continuing increase of available data rates on backbone network links, the amount of IP flow records network providers have to handle increases as well. Centralized IP flow records storage systems, although easy to maintain, since all the data is stored in a single place, do not scale easily with respect to the storage size they can achieve, as well as with respect to the query performance they need. Often providers need to sample traffic in order to reduce the amount of data they have to handle. A distributed storage platform for IP flow records — as the one presented here — allows for a provider to increase the amount of storage space by adding new storage nodes. Moreover, the performance of IP flow record processing can be improved also by using more computational resources during the retrieval of IP flow records from the storage space.

Therefore, the evaluation of the prototype implemented shows that a P2P-based distributed storage system for IP flow records is feasible. This prototypical implementation proves that storage distribution highly reduces the query time for IP flow records and can be applied in real systems.

Since most of the functionality on DIPStorage is in place, future work will be concentrated on improving the performance of the storage and query system as well as on the extension to support more complex applications, such as detection of route asymmetries based on IP flow records.

Finally, the conclusion is drawn that DIPStorage can help network operators to build a scalable storage and analysis platform for IP flow records with low costs. Moreover, being able to store more IP flow records and achieving better performance querying them, new application scenarios using IP flow records can be implemented on top of DIPStorage.

References

[1] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, A. Lakhina, *Impact of Packet Sampling on Anomaly Detection Metrics,* 6th ACM SIGCOMM Conference on Internet Measurements, Rio de Janeiro, Brazil, October 25-17, 2006, pp 159-164.

[2] K. C. Claffy, G. C. Polyzos, H. W. Braun: *Application of Sampling Methodologies to Network Traffic Characterization,* ACM SIGCOMM'93, San Francisco, California, U.S.A., September 13-17, 1993, pp 194-203.

[3] W. Cochran: *Sampling Techniques,* Wiley, 1987.

[4] N. G. Duffield, M. Grossglauser: *Trajectory Sampling for Direct Traffic Observation,* IEEE/ACM Transactions on Networking, Vol. 9, No. 3, 2001, pp 280-292.

[5] N.G. Duffield, C. Lund, M. Thorup: *Learn More, Sample Less: Control of Volume and Variance in Network Measurement,* IEEE Transactions of Information Theory, Vol. 51, 2005, pp 1756-1775.

[6] N. Duffield, C. Lund, M. Thorup: *Charging from Sampled Network Usage,* 1st ACM SIGCOMM Workshop on Internet Measurements, San Francisco, California, U.S.A., November 1-2, 2001, pp 245-256.

[7] Flow-Tools Homepage: http://www.splintered.net/sw/flow-tools/, June 2008.

[8] FreePastry Homepage: http://www.freepastry.org/, April 2008.

[9] S.H. Han, M.S. Kim, H.T Ju, J. W. K. Hong: *The Architecture of NG-MON: A Passive Network Monitoring System for High-Speed IP Networks,* 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'02), Montreal, Canada, October 21-23, 2002, pp 16-27.

[10] IEEE IPFIX Working Group Homepage: http://www.ietf.org/html.charters/ipfix-charter.html, April 2008.

[11] JmDNS Homepage: http://streamer.rit.edu/~jeffs/JmDNS/, April 2008.

[12] JXTA Homepage: https://jxta.dev.java.net/, April 2008.

[13] Y. Kitatsuji, K. Yamazaki: *A Distributed Real-time Tool for IP-flow Measurement,* International Symposium on Applications and the Internet (SAINTW'04), Tokyo, Japan, January 26-30, 2004, pp 91-98.

[14] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao: OceanStore: *An Architecture for Global-Scale Persistent Storage,* 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), Cambridge, Massachusetts, USA, November 12-15, 2000, pp 190-201.

[15] J. Mai, C. Chuah, A. Sridharan, H. Z. T. Ye: *Is Sampled Data Sufficient for Anomaly Detection?,* 6th ACM SIGCOMM Conference on Internet Measurements, Rio de Janeiro, Brazil, October 25-17, 2006, pp 165-176.

[16] MulticastDNS Homepage: http://www.multicastdns.org/, April 2008.

[17] MySQL Homepage: http://www.mysql.com/, April 2008.

[18] NetFlow Homepage: http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html, April 2008.

[19] NFDump Project: http://nfdump.sourceforge.net/, June 2008.

[20] SCRIBE Homepage: http://research.microsoft.com/~antr/scribe/, April 2008.

[21] Verteilte Systeme, Universität Konstanz Homepage: http://www.inf.uni-konstanz.de/disy/, April 2008.

[22] Wuala Homepage: http://wua.la/, April 2008.

[23] XPath Homepage: http:// www.w3.org/TR/xpath/, April 2008.

[24] T. Zseby: *Statistical Sampling for Non-Intrusive Measurements in IP Networks,* Ph. D. Thesis, Technische Universität Berlin, Germany, Universitätsbibliothek (Diss.-Stelle), Fakultät IV - Elektrotechnik und Informatik, 2005.