Design and Implementation of a Distributed Platform for Sharing IP Flow Records

Cristian Morariu, Peter Racz, and Burkhard Stiller

Department of Informatics IFI, University of Zurich, Switzerland
{morariu,racz,stiller}@ifi.uzh.ch

Abstract. Experiments using real traffic traces are of key importance in many network management research fields, such as traffic characterization, intrusion detection, and accounting. Access to such traces is often restricted due to privacy issues; research institutions typically have to sign non-disclosure agreements before accessing such traces from a network operator. Having such restrictions, researchers rarely have more than one source of traffic traces on which to run and validate their experiments.

Therefore, this paper develops a Distributed Platform for Sharing IP Flows (DipSIF) based on NetFlow records between multiple institutions. It is assumed that NetFlow traces collected by each participant are archived on separate storage hosts within their premises and then made available to others using a server that acts as a gateway to the storage. Due to privacy reasons the platform presented here uses a prefix-preserving, cryptography-based, and consistent anonymization algorithm in order to comply to different regulations determining the exchange of traffic traces data.

1 Introduction and Problem Statement

Internet Protocol (IP) traffic traces are widely used by researchers for different purposes. On one hand, traces are used to observe traffic characteristics to improve traffic analysis tools. On the other hand, they are used to evaluate new intrusion or traffic anomaly detection algorithms based on real traffic traces. One important drawback when designing new algorithms for the analysis of IP traffic is the lack of access to many real traffic traces based on which the new algorithms can be evaluated. Often traffic traces are used, which are available locally (such as those captured within a lab network or within a subnetwork of a university), but there is rarely access to traces collected by other parties. Problems that researchers face when accessing remote traffic traces are of both technical and legal nature. Technical problems relate mainly to the lack of common ways to access different repositories as well as the lack of a way to pre-process traffic traces before being downloaded, which often leads to downloading huge traces while only a small part of those traces is used for analysis. The legal problems that make access to flow traces difficult mainly relate to privacy and data protection laws. Although the authors of this paper agree that legal problems are very important to be solved in order to successfully deploy a NetFlow traces sharing platform, this paper only investigates technical problems that make sharing of NetFlow traces difficult.

C. Bartolini and L.P. Gaspary (Eds.): DSOM 2009, LNCS 5841, pp. 1–14, 2009.

This paper presents the Distributed Platform for Sharing IP Flows (DipSIF) that was developed in one of the activities of the European project EMANICS [7]. DipSIF can be used to share NetFlow records collected at multiple sites in order to make them available to researchers at different sites. In order to achieve this sharing, the platform offers well-defined interfaces allowing researchers to access in a similar way NetFlow records stored locally as well as remotely from any host connected to the Internet. In order to fulfil privacy protection and to comply to regulations for traffic trace exchange, the platform uses a prefix-preserving, cryptography-based, and consistent anonymization algorithm. The platform is fully specified, it has been implemented as a prototype as described below, and tested.

After discussing related work in Sect. 2, the design of the IP flow records sharing platform is presented in Sect. 3. While Sect. 4 outlines implementation details, Sect. 5 provides a functional evaluation of the system. Finally, Sect. 6 summarizes the paper and discusses possible future work.

2 Related Work

The two most important aspects that determine the design of DipSIF are (a) the traffic trace collection and storing as well as (b) the anonymization for sharing.

2.1 Flow Record Collection and Storage

As defined in [5] a flow record contains information about a specific flow that was observed at an Observation Point. A flow record contains measured properties of the flow (*e.g.*, the total number of packets and bytes transferred in the flow) and usually characteristic properties of the flow (*e.g.*, source and destination IP addresses). Typically an IP flow record is identified by at least the following unique keys: *source IP address, destination IP address, source port, destination port, and protocol type*. Different solutions, tools, and protocols exist to collect and store IP flow records.

NetFlow [4] — embedded in most Cisco routers — includes a NetFlow exporter component, which summarizes the traffic traversing routers in NetFlow records and exports those to remote collectors using the NetFlow protocol. Today different versions of the NetFlow protocol exist. Version 1 — the export format released initially — is rarely used today. Version 5 enhanced the initial specification and it is the most commonly used version today. Version 7 added support for data export from aggregation caches. The latest version is Version 9 and it specifies a flexible and extensible format, which provides the versatility needed for supporting new fields and record types. This format accommodates support of new technologies and introduces a template-based flow record transfer. Templates provide an efficient record transfer and allow record format extensions without concurrent changes to the basic record format.

The *IP Information Export (IPFIX)* protocol [5] is a flow record transfer protocol standardized by the Internet Engineering Task Force (IETF). The IPFIX protocol is based on the NetFlow v9 protocol.

Cflowd [3] is a flow analysis tool that can be used to analyze Cisco NetFlow records. The package allows data collection as well as analysis in support of trends estimation, traffic tracking, accounting, and billing.

Flow-Tools [11] is a powerful combination of a network flow collector and a flow analyzer. The flow collector can support single, distributed, and multiple servers for several NetFlow versions. The *flow-capture* module is used to collect NetFlow records (only User Datagram Protocol, no Stream Control Transmission Protocol) from network devices and store them in files in raw format. Then, either *flow-print* or *flow-cat* decodes the files for analyzing purposes.

Nfdump [10] is a package providing tools to collect, process, and store network flow data. The nfdump toolkit also provides a converter for transforming flow-tools NetFlow records into nfdump formatted files. Nfdump stores flow records in time-sliced files with a timestamp attached to the filename. Afterwards, particular files can be either merged or divided again in order to perform further analysis on it.

Argus [1] is a real-time flow monitor, which is able to track and report network transactions it observes from packets collected on a network interface in promiscuous mode. The major difference to Cisco NetFlow records is that Argus flow records consider each traffic flow as a bidirectional sequence of packets that typically contain two sub-flows, one for each direction. There are two types of Argus records: the *Management Audit Record* and the *Flow Activity Record*, where the former provides information about Argus itself, and the latter provides information about network flows.

DatCat [6] is a project that aimed at building a traffic traces repository including NetFlow data. Traces can be published by network operators using a web portal, while researchers can browse the existing repository and retrieve traces they need. One drawback of *DatCat* addressed in this paper is that all flow processing needs to be done on the client side. If the traffic trace is big, even if a researcher needs only a small part of the flow records (*e.g.*, flow records with port 22 as a destination) the whole trace would have to be downloaded.

2.2 Flow Record Anonymization

Anonymization is the process of hiding the identity of a particular client of a network by inspecting IP packets sent. Anonymization of traffic traces cannot be applied for all applications. Some applications, *e.g.*, accounting and traffic engineering, require the original traffic in order to assign costs to a particular user or to detect where and why a congestion appeared in the network. Several anonymization algorithms are known, but there is no standardized method for anonymization of traffic traces. The following anonymization techniques are encountered in different tools:

- *Truncation*: This defines the basic type of IP address anonymization. The user chooses the number n of least significant bits to truncate from an IP address. *E.g.*, truncating n=8 bits would replace an IP address with the corresponding class C network address.

- *Random permutations*: With this method, a random permutation on the set of possible IP addresses is applied to translate IP addresses. A 32-bit block cipher represents a subset of permutations on the IP address space. The use of random hash tables implements a truly random permutation [9]. Thus, it is possible to generate any permutation during anonymization, not just one from a subset of the possible ones.
- *Prefix-preserving pseudonymization*: This special class of permutations has a unique structure-preserving property: Two anonymized IP addresses match on a prefix of *n* bits, if and only if the unanonymized addresses match on a prefix of *n* bits as well. This is achieved by a user-supplied passphrase, which generates a cipher key that in turn determines the permutation to be applied. This allows for an anonymization with a consistent mapping between multiple anonymizers sharing a common key.

Based on these mechanisms several anonymization tools are used in practice. *TCPdpriv* [12] implements a table-based, prefix-preserving translation of IP addresses. In order to maintain the consistency of the anonymization, it stores a set of <*raw, anonymized*> binding pairs of IP addresses. Whenever a new IP address needs to be anonymized, it will be compared with all raw IP addresses within stored binding pairs for the longest prefix match. For remaining bits, where no prefix is stored inside the binding table, a pseudorandom function is applied. TCPdpriv removes sensitive information by operating only on packet headers The TCP and UDP payload is simply removed, while the entire IP payload is discarded for protocols other than TCP or UDP. The tool provides multiple levels of anonymization, from leaving fields unchanged up to performing a more strict anonymization.

CryptoPAn [14] is a cryptography-based network trace anonymization tool with a prefix-preserving property. In CryptoPAn the mapping from original IP addresses to anonymized IP addresses is one-to-one. To anonymize traces, CryptoPAn uses a secret key that is used across multiple traces. This allows multiple traces to be anonymized in a consistent way over time and across locations. That is, the same IP address in different traces is anonymized to the same address, even though traces might be anonymized separately at different time and/or at different locations. CryptoPAn is used in many different tools such as CANINE [9] or nfdump.

AAPI (Anonymization Application Programming Interface) [8] is a C-based anonymization library. This tool provides a fast on-line anonymization support as well as composition, decomposition, and filtering of NetFlow version 5 and 9 traces previously saved in the tcpdump [13] format. AAPI uses randomizing, replacing, and prefix-preserving anonymization techniques.

CANINE [9] addresses several aspects. It supports different anonymization methods on different fields of a flow record; IP address truncation, random permutations, and prefix-preserving pseudonymization; bilateral classification, black marker anonymization of port numbers; protocol number replacement; timestamp annihilation, random timeshift, timestamp enumeration, and byte count replacement. CANINE also acts as a powerful converter between different versions of NetFlow records.

2.3 Comparison and Discussion

TCPdpriv shows two major drawbacks: it only provides online anonymization and it cannot anonymize parallel traces within a distributed environment in a consistent way. CryptoPAn does not have these restrictions and it allows for a consistent anonymization between several anonymizers based on the shared cipher key. Furthermore, it is fast to anonymize flow records online. AAPI and CANINE provide more than IP address anonymization and allow hiding other IP fields as well. Besides several techniques for veiling different flow record fields, both support cryptography-based, prefix-preserving IP anonymization.

In case operators decide to anonymize their data, researchers do not have a tool at their disposal to easily find these traces in the Internet, thus, they often have to spend time to look for new public traces. Often researchers use well known traffic traces over and over, and often algorithms which could be used to detect certain network problems (such as worms) could only be evaluated based on real traffic traces long after the problem appeared. DipSIF addresses these issues by providing a platform with which any NetFlow data owner can share data, while still being able to apply authorization policies and anonymize data before being sent. DipSIF integrates the widely used NetFlow data storage tools into existing repositories.

3 Design

In order to enable such a sharing between different institutions, the design of DipSIF focused on an architecture with access control and online traffic anonymization. Based on requirements determined, the resulting architecture is fully decentralized.

3.1 Requirements

Based on the investigation of distributed trace handling the following key requirements have been identified for DipSIF:

- Retrieval of remotely stored NetFlow data: The major requirement is to supply a researcher with an Application Programming Interface (API) for querying flow records from any remote NetFlow repository of participating parties.
- Access control: The platform shall offer an authentication mechanism and fine granular access control to NetFlow data stored. Each storage provider shall be in the position to decide independently, which users or groups of users will have access to flow traces collected.
- Online anonymization: Any request for NetFlow records shall result in anonymized records being sent to the requester. The anonymization shall be done online and at least mutate IP addresses stored within NetFlow records in a prefix-preserving manner.
- Encryption: Communication channels between all distributed components are typically not trustworthy, all communication shall be cryptographically secured.

3.2 Architecture

Exploiting those requirements a three-tiered architecture (cf. Fig. 1) has been designed. The architecture consists of a retrieval service running on separate storage nodes (1st tier), an authentication and forwarding service on a set of server nodes (2nd tier), and a client library that offers an API (3rd tier) for accessing arbitrary NetFlow datasets from any of those repositories available. The secure communication is achieved by using Secure Socket Layer (SSL), offering client and server authentication, data integrity, and a cryptographically secured channel for communication between client and server. A query including optional filters and other control messages from clients results in anonymized NetFlow records being returned to the requester.

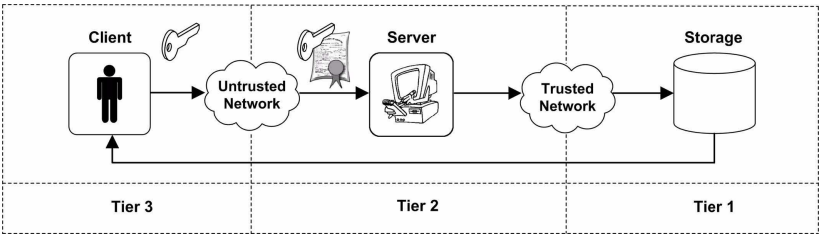

Fig. 1. Three tier architecture

While the first (and third) tier clearly show a server (and client) functionality, the second tier features a dual behavior: it acts as a client, when requesting NetFlow records from the local storage component, and at the same time it acts as a server, when handling requests received from remote clients. Fig. 2 illustrates the architectural design of the sharing platform. It can be seen in this figure how information is exchanged between a client, a server, and a storage node. The figure also shows how the different components are deployed and how they interact.

The end-user using an API can build applications for NetFlow record retrieval according to his needs. The API provides a user with several methods that are used to control the dataset that will be requested (*e.g.,* filters to be applied to reduce the dataset, transmission rate, etc.). In order to access any functionality of the API the client needs to pass an authentication process beforehand. After a successful authentication the client is enabled to request flow data. In a first step a request message containing the type of the desired dataset needs to be created, which is passed to the server by invoking an appropriate remote procedure defined in the API.

When the server receives the client's request message, it checks the preconfigured authorization and anonymization policy using the *Access Control* module. The authorization result is logged and if the user is allowed to have access to the NetFlow repository the request is forwarded together with the anonymization policy to the storage component. The storage service searches the appropriate dataset in the repository, anonymizes the dataset according to the received anonymization policy, and finally applies desired filters, which are passed within the request message. The resulting dataset is returned by the replaying module (in the prototype implemented by nfdump tools) directly to the client as a stream of NetFlow packets.

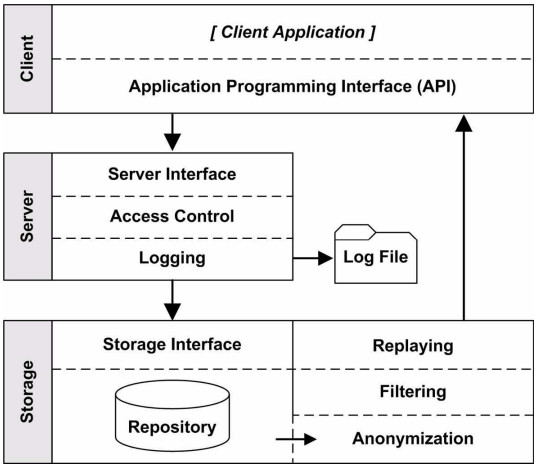

Fig. 2. Architecture

3.3 Client Component

The client component serves as an entry point to DipSIF. It includes a library and its API, which can be used to communicate with other components. The library enables a user — possessing a valid certificate — to request flow traces collected from remote storage repositories. Requests of NetFlow records can be made using the API, which hides the actual retrieval process. Users shall only care about *what* (*i.e.* filters) data and from *where* (*i.e.* server URL) they require data, but not *how* it is obtained.

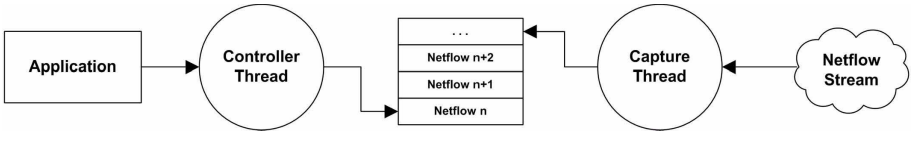

Fig. 3. A controllable capturing unit

The client library can buffer NetFlow records sent by the storage node's replay component and simultaneously deliver them to the application, built on top of the client library, at a controlled rate. Therefore, incoming NetFlow records are received and decoded by the *Capture Thread* that adds them to a temporary FIFO list, as illustrated in Fig. 3. Simultaneously, the *Controller Thread* pops out buffered NetFlow records according to user specified events. In addition to this mechanism, the request message includes a *delay* attribute for the purpose of adjusting the rate of NetFlow records sent by the storage component. This helps avoiding buffer overflows on the receiver side. Before sending a request to a selected server, the client application binds a receiver to an unused local port and attaches this information to the request message. The port number is used to tell the replaying module of the storage component, where to return the requested data to.

The request message includes the following attributes:

- User identity (*i.e.* accepted and valid certificate)
- Desired server (*e.g.*, emanicslab1.csg.uzh.ch)
- Return address (*i.e.* IP:port)
- Requested data (*i.e.* time-period)
- Optional filters (*e.g.*, port- and/or protocol-based)
- Delay option (*e.g.*, delay each record by 10 microseconds)

Upon the receipt of a request message the server component replies with the result of the request authorization.

3.4 Server Component

The server component serves as a gateway for the client-to-storage communication. It performs client authentication and authorization and controls the access to storage repositories. Each server component maintains its own credential repository (*truststore*) that manages a list of X.509 certificates of users to which access is granted. Requesters need to authenticate by using their corresponding private key stored in their *keystore* before their query is forwarded to the storage node. A *Public Key Infrastructure* (PKI) additionally shares public keys in order to allow for an encryption of the communication channel using SSL.

All servers use a predefined port for accepting incoming queries. The server interface allows client applications to communicate with server instances by calling remote methods defined by the interface using Remote Method Invocation (RMI). The server interface allows for retrieving the server's actual state and for sending request messages to storage components. Users with a valid and authorized certificate can request to order and filter the resulting dataset by means of the following options, or a combination thereof:

- Time (*i.e.* start-/end-time of flow records)
- Port numbers (*e.g.*, traffic with destination port larger 80)
- Protocol types (*e.g.*, TCP — Transport Control Protocol, UDP — User Datagram Protocol, ICMP — Internet Control Messaging Protocol)

In order to provide access control, every client interacting with the server needs to authenticate by means of a certificate. Only authenticated users will receive necessary privileges to retrieve flow records of any of the remote storage repositories. Server administrators can review a client's certificate and decide, whether the certificate is added to the server's truststore (cf. Fig. 4). The certificate can be issued by a local certification authority as well.

The prototype implementation of the server supports two groups of users, administrators and certified clients. The first group, which includes administrators, is allowed to maintain server and storage components, grants registration requests from new users by accepting their certificates to the server's truststore, manages user accounts, and assigns anonymization settings to individual users and groups. The second group includes the clients to which access to NetFlow data is allowed. Certified

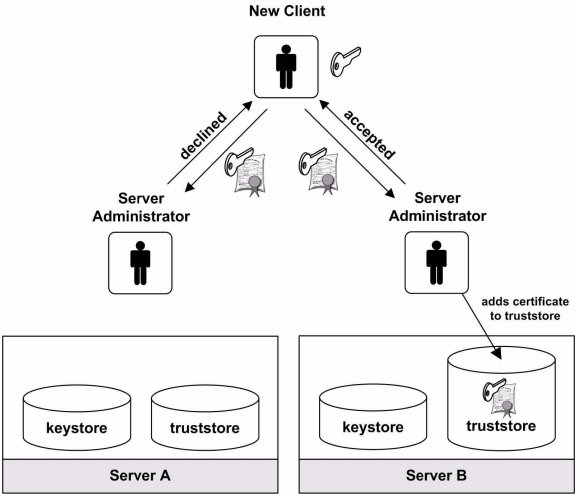

Fig. 4. Registration process

clients must have a valid and accepted certificate and can request flow data. Such requests can be made by applications residing on the client using an API that is described in Sect. 4. Clients are not authorized to modify any flow data and to request any unanonymized flow data. As the server component acts as an intermediary to all flow requests, additional authorization mechanisms could be envisioned, such as: access for a given group of users only to a subset of flow records (for example only to flow records belonging to a subnet), or access to flow records generated within a specific time frame.

For the purpose of monitoring the platform's performance and controlling the data exchange, any incoming request and outgoing data stream is logged to a file by the server at the moment of delivery. The information included in the log file consists of the request timestamp, the requestor's identity (username, usergroup, and IP address), anonymization options, the time period of requested flow records, filter types applied, and the processing time.

3.5 Storage Component

The storage component is responsible for the storage of NetFlow records. The DipSIF architecture does not define the way on how flow records are collected — any tool can be used to fill the storage component with flow records. NetFlow records can be stored in databases or directly in binary files in their raw format. Thus, the design of DipSIF is independent of the storage type.

The storage interface of the storage component offers methods allowing for the searching and retrieving sets of flow records from a repository independent of the flow record's type and independent of the storage system (database or a raw files). The interface is capable of handling multiple heterogeneous storage repositories attached to the server component as illustrated in Fig. 5. Due to the fact that NetFlow

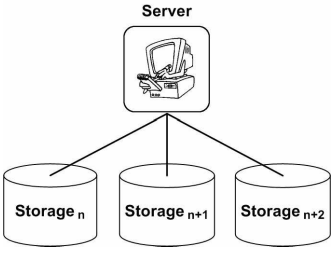

Fig. 5. Multiple heterogeneous storage repositories attached to a server component

records come in many different, incompatible formats, the storage platform allows for handling more than one specific type of flow record. In this case the storage component translates all flow records into the NetFlow version 5 format. The decision of using NetFlow version 5 for transferring NetFlow records from remote repositories to clients that requested those records is due to the fact that most of the investigated repositories, use NetFlow version 5. Changing to NetFlow version 9, or IPFIX requires little change in the replaying unit of the storage component.

For anonymizing IP packets, the CryptoPan [14] prefix-preserving IP address anonymization algorithm is applied on a dataset before being forwarded to a requester. The key for anonymization is created upon the authentication process by the server component and sent to the client once the authentication is successful. The flow data is stored in its original state and is not being anonymized until it leaves the storage node. During anonymization, only IP addresses are anonymized while other NetFlow fields are kept unchanged. The reason why those fields are kept unanonymized is that for many research approaches realistic network traces are needed. In order to increase the privacy protection, the platform rebuilds the cipher key that is used as a base for the anonymization procedure. For every incoming request, the server generates a session key that will be used to anonymize NetFlow records within that session. For higher flexibility, and more control on the NetFlow provider side, a future version of the prototype could use more anonymization options (such as anonymization of port numbers) which could be used depending on the application requirements for which the flow records will be used (*e.g.*, by default the port numbers would be anonymized, but if the application requires that the port numbers are kept in their original form, a special authorization attribute could allow that).

4 Implementation

Based on the design above a prototype was implemented in Java using the Java Development Kit 5.0 (JDK5). An outline of the implementation architecture is shown in Fig. 6. As it can be seen in the figure, communication between the client and the server as well as between the server and the storage component uses Remote Method Invocation (RMI) calls. All communication between the client component and the server uses an SSL connection. The backward communication between the storage component and the client uses the NetFlow version 5 protocol. The prototype uses the

nfdump format to store flow records in files. Since CryptoPAn [14] is an integral part of the nfdump toolchain and its capabilities serves ideally for the given requirements, it is used as the anonymizing tool.

An application developer, who wants to use the sharing platform needs to use the *nfshare* library, extend the class *FlowHandler* found in its API and override the methods *init()* and *handle()*. The *init* method is used to send requests to remote repositories. An example of this method is shown below:

```
private void init() throws Exception {
   Request request = new Request(
   "emanicslab1.csg.uzh.ch",  // the server URL
   null,                      // your IP (auto)
   9992,                      // receiver port
   "2008/06/20-2008/06/22",   // time window
   "src port > 1024 and tcp", // filter options
   1000,                      // delay
   5                          // limit
   );

      FlowController fc = new FlowController();
      fc.subscribe(this);
      fc.start(request);
}
```

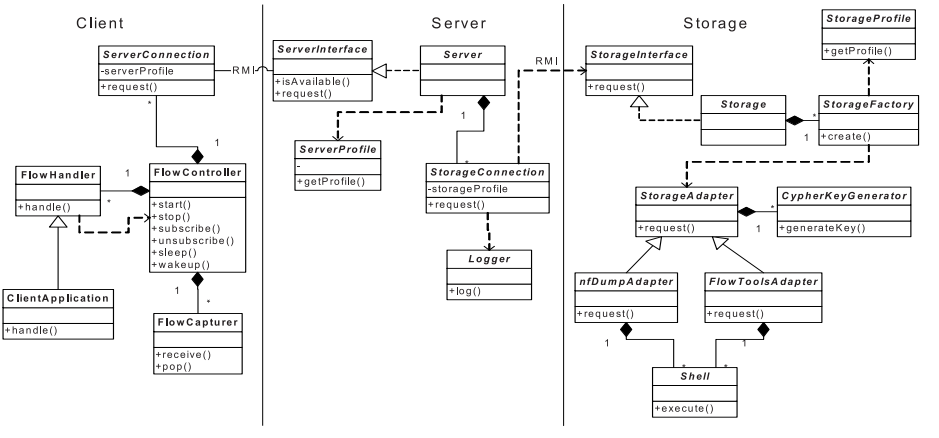

Fig. 6. Implementation Architecture

The *FlowController* object is responsible for handling one request and delivering the data received to the application. The *handle()* method is a callback used by the client library to inform the application whenever a new flow record is received. Its signature is:

```
protected void handle(Flowrecord record)
```

The new flow record is passed to the application by a *FlowController* object as a *FlowRecord* object. An example application component is shown in Fig. 6 as the *ClientApplication* class.

The prototype includes storage modules for nfDump and FlowTools. In order to support other NetFlow storage formats the *StorageAdapter* interface needs to be implemented and specialized for those specific formats. The *Shell* class allows for the execution of Linux shell commands, a requirement for some storage toolkits that do not offer an API to access flow data, but only binary programs for handling that data. The shell commands are predefined in the storage component and a user cannot execute any shell commands directly.

5 Evaluation

Based on the introduced and important functional requirements of the flow record sharing platform (cf. Sect. 3.1), the design was done (cf. Sect. 3.2 to Sect. 3.5), and the implementation (Sect. 4) was undertaken. Thus, a functional evaluation is possible, while the running prototype shows the feasibility:

- *Retrieval of stored NetFlow data:* The DipSIF offers an API allowing for the query of NetFlow records from remote repositories. The library offers several instruments to control the kind of requested data as well as the received data stream. A set of available filter options allows for sorting requested NetFlow records by specific attributes, such as IP protocol or port numbers. The stream received is controlled by delaying individual NetFlow records, by limiting the amount of data, and by interrupting it for a given time period.
- *Access control:* Authentication is implemented using a PKI while authorization uses access lists. Each storage provider is in the position to decide independently to which users access to collected flow traces is granted. Any client needs to hand over a signed certificate to the administrator in charge of the desired repository in order to be able to access the corresponding NetFlow data.
- *Online anonymization:* Any NetFlow record is anonymized before being sent to the requester at the moment it leaves the storage node. The prototype implemented uses the CryptoPAn library to anonymize NetFlow records. The anonymization is done in a prefix-preserving manner.
- *Encryption:* The communication between the client and server component is encrypted using an SSL connection. However, the NetFlow transfer between the storage component and the client is left unprotected but IPSec might be used to protect the record transfer. Another way to improve security during the transmission of NetFlow records would be the use of SCTP (Stream Control Transmission Protocol) and a transport layer security protocol ontop of it.

The modular design as well as a storage interfaces which is independent of the storage technology allow implementation of add-on modules for different storage formats besides the nfdump format which was used in the prototype.

The performance of the DipSIF platform is given by the performance of its individual components. As server and storage components may serve multiple requests at the same time, these components could become bottlenecks when a large number of requests come in.While the server component only has to perform minimal process-

ing for each incoming request (verify the user credentials, check its authorization policies) and thus is less probable of becoming overloaded, the storage component may become overloaded when a large number of requests need to be handled at the same time. The current prototype handles each incoming request independently of other, so when multiple requests arrive at the storage component, each of these requests triggers a lookup in the nfdump files which increases the disk access time.

6 Summary and Future Work

Network traces determine one of the wider testing data for researchers working in the field of traffic analysis. Access to such traces is often a problem, either because of administrative reasons, which prevent network operators to share their traffic data, or either because of the lack of a search tool for existing traffic traces in the Internet.

The work presented here proposed the DipSIF architecture as a sharing platform for NetFlow records. On one hand, it allows owners of traffic traces to share their data with researchers and provides them with access control mechanisms to control who will have access to those traces. It also provides anonymization capabilities so that no IP address involved in the real traffic is revealed. On the other hand, it allows researchers and protocol designers with an easy and standard access to repositories residing remotely and provides tools for controlling the data that is actually retrieved. Such a control will help to decrease data transfers, because researchers can specify, which data from a dataset is important for their research and only download these data instead of downloading a whole traffic trace, which includes — besides the interested traffic data — those data, which are unnecessary for the current research question tackled.

The prototype presented can be improved further by adding additional functionality. For example the authorization process can be extended in order to allow an administrator to limit explicit flow records to which an external user has access (*e.g.,* external access could be allowed only to web traffic). Another useful extension is the possibility to perform processing on the remote storage in order to return just an aggregated value, rather than a large number of flow records. The UDP protocol used in the current implementation to transfer NetFlow records could be replaced with the SCTP protocol, which would add more security, reliability, congestion control awareness. As some traffic analysis tools use NetFlow records stored in files as input data, an option to transfer NetFlow records from the storage repository back to the requesting clients over HTTPS could be implemented.

Acknowledgment

This work has been performed partially in the framework of the EU IST NoE EMANICS (FP6-2004-IST-026854). The authors are indebted to Nicolas Baumgardt, who implemented the prototype in his Student Thesis [2]. They also want to thank the reviewers who provided helpful feedback during the paper review process.

References

1. Argus Homepage, http://www.qosient.com/argus/ (last access, April 2009)
2. Baumgardt, N.: Design and Setup of a Distributed Storage Repository for NetFlow Records; Student Thesis. CSG@IFI, University of Zürich, Switzerland (March 2008)
3. cflowd Homepage, http://www.sdsc.edu/~woodka/cflowd.html (last access April 2009)
4. Cisco NetFlow Homepage: http://www.cisco.com/en/US/products/ps6601/products_iosprotocol_group_home.html (last access April 2009)
5. Claise, B. (ed.): Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. IETF RFC 5101 (January 2008)
6. DatCat, Intenet Measurement Data Catalog, http://www.datcat.org/m (last access August 2009)
7. EMANICS Project Homepage, http://www.emanics.org/ (last access May 2009)
8. Koukis, D., Antonatos, S., Antoniades, D., Trimintzios, P., Markatos, E.P.: A Generic Anonymization Framework for Network Traffic. In: IEEE International Conference on Communications (ICC 2006), Istanbul, Turkey (June 2006)
9. Li, Y., Slagell, A., Luo, K., Yurcik, W.: CANINE: A Combined Conversion and Anonymization Tool for Processing NetFlows for Security. In: International Conference on Telecommunication Systems, Modeling and Analysis, Dallas, Texas, USA (November 2005)
10. nfdump Homepage, http://nfdump.sourceforge.net/ (last access April 2009)
11. Plonka, D.: FlowScan: A Network Traffic Flow Reporting and Visualization Tool. In: 14th USENIX Conference on System Administration, New Orleans, Louisiana, USA, December 2000, pp. 305–318 (2000)
12. TCPDpriv Homepage, http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html (last access April 2009)
13. Tcpdump Homepage, http://www.tcpdump.org/ (last access May 2009)
14. Xu, J., Fan, J., Ammar, M., Moon, S.B.: On the Design and Performance of Prefix-preserving IP Traffic Trace Anonymization. In: 1st ACM SIGCOMM Workshop on Internet Measurement (IMW 2001), San Francisco, California, USA, November 2001, pp. 263–266 (2001)