SCRIPT: A framework for scalable real-time IP flow record analysis

Morariu, C; Racz, P; Stiller, B
SCRIPT: A Framework for Scalable Real-time IP Flow Record Analysis

Cristian Morariu, Peter Racz, Burkhard Stiller
Communication Systems Group CSG, Department of Informatics IFI, University of Zürich
Binzmühlestrasse 14, CH—8050 Zürich, Switzerland
Email: [morariu|racz|stiller]@ifi.uzh.ch

Abstract — Analysis of IP traffic is highly important, since it determines the starting point of many network management operations, such as intrusion detection, network planning, network monitoring, or accounting and billing. One of the most utilized metering data formats in analysis applications are IP (Internet Protocol) flow records. With the increase of IP traffic, such traffic analysis applications need to cope with a constantly increasing number of flow records. Typically, centralized approaches to IP traffic analysis have scalability problems, which are addressed by replacing existing hardware with more powerful CPUs and faster memory. In contrast, this paper developed and implemented SCRIPT (Scalable Real-time IP Flow Record Analysis), which defines a scalable analysis framework that can be used to distribute flow records to multiple nodes performing traffic analysis in order to balance the overall workload among those nodes. Due to its generic design, the framework developed can be extended and used to distribute other metering data, such as packet headers, payloads, or accounting records.

Index Terms—IP Flow Accounting, Peer-to-Peer, Distributed Analysis

I. INTRODUCTION

IP (Internet Protocol) network traffic flowing through the backbone of a network operator is most of the time metered, and the result of the metering process is sent to an analysis application for different purposes: e.g., to charge clients for their traffic, to identify malicious network activity, to detect network anomalies (e.g., congestion, broken links), or to measure Quality-of-Service (QoS) parameters, such as throughput, delay, or jitter.

During the last decade network traffic flowing in the operators’ backbone networks experienced yearly increases of 50%-100% in volume [14]. A study on the evolution of IP traffic [3] shows that this behavior will most probably increase even further and by 2012 the Internet traffic will be about 75 times the amount as of 2002.

Although the CPU (Central Processing Unit) performance and memory access speeds improved significantly during the same period, the performance increase of these happened at a lower rate (e.g., Dynamic Random Access Memory (DRAM) access speed improves 7-9% every year) compared to the increase of network traffic [14]. Such an unbalanced evolution makes traffic analysis even more difficult every year, as less time is available to process a single metering record. In order to address this problem, a reduction of the amount of data that needs to be processed is achieved by sampling. However, sampling has its own disadvantages. Since some data remains unprocessed, some traffic analysis applications, such as Intrusion Detection Systems (IDS) or charging may provide inaccurate results, as shown in [2] and [13].

Distributed systems are one way of addressing complex problems by splitting a job in multiple tasks and assigning each task to a separate processing node. Distributed analysis approaches (cf. Section II) have already shown that by sharing the workload between several analysis nodes, more analysis data can be processed per unit of time. The major drawbacks of existing distributed solutions include a reduced scalability and an application-oriented approach. The reduced scalability of those approaches does not only refer to design concepts (some use a single node that distributes the data to other existing nodes, hence being a possible bottleneck), but also to approaches that show a scalable design. However, in practice this would not be very user-friendly in an operational environment when the number of nodes increases (e.g., it would require the reconfiguration of each node, or replacement of physical hardware in case hardware traffic splitters are used). The second drawback refers to the fact that each existing distribution solution was designed for the purpose of serving a very specific analysis application only.

Therefore, this paper tackles these problems and addresses both of them in an integrated solution for traffic analysis that is scalable, flexible, and standards-conforming, that can be used for more than one analysis application.

Thus, the SCRIPT (Scalable real-time IP Flow Record Analysis) approach developed defines a framework for building a distribution overlay for IP Flow Information Export (IPFIX) records for the purpose of distributed network traffic analysis. This solution proposed here overcomes those drawbacks observed with other approaches and provides an applicable framework to deploy traffic analysis applications in a distributed environment. In addition, due to the usage of the IPFIX protocol, SCRIPT can be used to distribute any kind of metered data as long as it can be transported in IPFIX payloads.

The remainder of the paper is organized as follows. Section II gives an overview on existing distributed approaches to traffic analysis and sampling. Section III presents a selective set of demanding scenarios for distributed traffic analysis. While Section IV outlines the design of SCRIPT, Section V summarizes key implementation details. Finally, Section VI evaluates the prototype implemented, and Section VII draws conclusions and sketches possible future developments.
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 [5], [9], [11], and [26] present packet and flow sampling algorithms that — besides reducing the amount of data — also keep the error of sampling estimations within low limits. Those sampling proposals, although alleviating computational requirements of high-speed packet processing, are not very accurate in certain scenarios, where complete information is required (such as for an Intrusion Detection System or in usage-based charging systems). Investigations have been made in detecting how sampling algorithms impact the performance of IDS: [2] and [20] show that the sampling rate directly impacts the quality of intrusion detection. The work of [12] 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 the literature. [13] introduces the idea of trajectory sampling in the context of IP flow accounting. In their approach each packet is sampled either by no router or by all routers on the packet’s path. However, this solution does not guarantee that a packet always reaches a router responsible for its capturing. [17] proposes a distributed packet capturing architecture based on a high-performance machine that splits the traffic across multiple capture nodes. A similar approach is found in [16] 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, or 24 hour intervals). Another idea for deploying existing traffic analysis applications in a distributed environment is presented in [22]. The authors propose to send the whole traffic to several machines and apply filters on each machine to select the interesting traffic to be processed.

In the context of IP flow accounting, Cisco developed the NetFlow protocol for carrying IP flow records. Following this work, the IETF’s (Internet Engineering Task Force) IPFIX [7] working group specified an updated protocol and data format for the transfer of IP flow records. The IPFIX protocol specification was based on version 9 of the NetFlow protocol. For IP flow record storage flow-tools [15] and nfdump [23] are two widely used open-source tools. The main disadvantage of these tools is that they are centralized, thus, suffering performance drops in case of large volumes of IP flow records to be handled and stored. The IPFIX working group defined the terms exporter and collector as devices that generate and receive respectively IPFIX records.

The IETF Packet Sampling (PSAMP) working group specified a framework [10] for packet sampling and standardized different packet sampling and filtering techniques [27]. The PSAMP framework defines the packet information to be collected for sampled packets and enables the selection of packets according to a set of standardized mechanisms across network elements. The PSAMP protocol [8] supports the transfer of packet information and is based on the IPFIX protocol. Based on per-packet information, characteristics related to individual packets and flows can be determined.

Many traffic analysis applications require investigation of NetFlow or IPFIX records which typically reside in very large repositories. Storage and query of such repositories became a real problem, as the network of a larger operator could easily generate a billion records every hour during peak times. Several Peer-to-Peer (P2P) storage systems such as [18] and [25] are proposed by the research community and the industry for building file sharing networks, such as Kazaa [19], or robust backup storage repositories for files, such as Wuala [25]. These solutions are targeted mainly toward 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.

As a concept, P2P sounds very promising because it allows participants in such a system to organize themselves in order to share resources in order to solve a problem (e.g. storage space in case of file sharing).

In contrast to these existing approaches, SCRIPT does address (a) analysis of IP traffic on high speed network links, (b) distributed processing of traffic data, and (c) generic distribution framework to be used by various analysis applications in an integrated, scalable real-time manner as described below, avoiding many of those drawbacks discussed above.

III. SCENARIOS

In order to motivate the need for a distributed traffic analysis tomorrow a set of selected scenarios are described and respective shortcomings of centralized traffic analysis applied in these scenarios are highlighted. While Scenario 1 assumes an application which stores IPFIX records and provides a query language for those records, Scenario 2 introduces an application of delay measurement based on IPFIX records, Scenario 3 proposes an application of asymmetric-route detection based on correlation of IPFIX records belonging to the two flows of a biflow [24].

A. Data Retention

Different legal regulations require that network operators keep traffic traces for some period of time. Even without such requirements network operators keep traces for a while in order to inspect them and detect possible anomalies in the traffic observed. A centralized approach for storing traffic traces may become a bottleneck by overloading the network link to this central repository or by sending traffic traces at a higher rate than the maximum rate at which the repository can write these traces into persistent storage. Distributing this process does distribute the network and storage load to several nodes. In addition, storing all these data at a single location means that, if the repository shows a failure, all traffic traces become
unavailable and no new trace is being saved. In such a case a distributed system still enables access to all traffic traces except those ones stored on the damaged node. In case of inspecting traffic traces stored a distributed system does help by running this process in parallel on multiple nodes, making results available even faster.

B. Delay Measurements

A delay measurement application measures the time a particular packet spent between two observation points. The PSAMP working group proposed a measurement delay application [27] using the IPFIX protocol for transporting metered data from each observation point. For each incoming IPFIX record, the delay measurement application needs to lookup, if for the respective packet other measurements from other observation domains may be in place. A high packet rate has two effects on this application: there will be more IPFIX records to be kept in main memory, which increases the lookup time, but at the same time there will be less time available to process a single IPFIX record. Distributing this application will both decrease the lookup time by storing less records in the main memory, and increase the available time to process a single record, by splitting records between multiple nodes.

C. Real-time Asymmetric Route Detection

Network operators often want to avoid asymmetric routes in their networks as these are usually caused by network problems such as congestion or misconfiguration. Routes are asymmetric, if a flow does not traverse the same routers in one direction as in the other. To detect asymmetric routes flow records can be used by examining flow records belonging to a flow and its reverse flow, whether the same routers exported these records in one direction as in the other. To be able to do this, records belonging to a flow and its reverse flow have to arrive at a single collector from all possible exporters. Similar to those scenarios above, in case of a centralized solution the central collector has to deal with high IPFIX record rates (received from all exporters) and has less time to process a single record. Distributing this application reduces the load on a single collector and increases the time available to process a single record. It is important to note that the distribution scheme has to ensure that records of a flow and records of its reverse flow arrive at the same collector.

D. Shortcomings of Centralized Solutions

Most of those key disadvantages of centralized solutions observed in all scenarios investigated can be summarized as different bottlenecks due to:

a) Incoming IPFIX data arrives at a rate higher than the maximum write rate of the hard disk or storage device;
b) The network link bandwidth of the centralized collector is not sufficient for aggregated IPFIX streams from all exporters; and
c) In case of real-time processing needs, required at collector’s side, processing time of an IPFIX record will be higher than the inter-arrival time of IPFIX records.

IV. FRAMEWORK DESIGN

The framework consists out of the network architecture, the Central Configuration Repository, the SCRIPT Node, and the handling of IPFIX records under investigation. These major components, their interactions, and key design decisions are outlined below.

A. Network Architecture

The SCRIPT flow record distribution framework is organized as a Kademlia-based P2P overlay [21] as shown in Figure 1. The SCRIPT flow record distribution framework is organized as a Kademlia-based P2P overlay. The SCRIPT flow record distribution framework is organized as a Kademlia-based P2P overlay. Each node has two tasks: (1) to forward incoming IPFIX records to appropriate nodes and (2) to deliver a subset of the incoming IPFIX records to one or more analysis applications running on that node. The Central Configuration Repository (CCR) is involved in the bootstrap process (c.f. Section V.B) as well as in the management of different configuration aspects of the nodes, but it is not involved in forwarding flow records.

As Figure 1 depicts, each router can choose any node of the P2P overlay to forward its IPFIX records. Starting at that node, a flow record routing process will start that will assure (c.f. Section IV.1) that the intended node to process an IPFIX record will receive that record. If a node is the final destination of a particular IPFIX record, it delivers the respective record to one or more analysis applications running on that node.

B. Central Configuration Repository (CCR)

The CCR is responsible for node management, supporting the bootstrap process of nodes, and management of flow templates and their mapping to analysis applications.
1) Node Bootstrap

Each node has a 64-bit identity, as shown in Figure 2, assigned by the CCR during the bootstrap process. The identity consists of a 32 bit node identifier (ID), 16 unused zero bits, and a 16 bit area ID. The CCR — by knowing all participating nodes due to its involvement in the bootstrap process for every node — assigns a node ID following a uniform distribution. This will result in a nearly equal number of flow records received by each node. For example, for the scenario depicted in Figure 1 these eight nodes will see node identifiers starting with the following byte 1F, 3F, 5F, 7F, 9F, BF, DF, and FF.

![Figure 2: SCRIPT Node Identity](image)

The area ID of a node identity is used to organize nodes according to geographic location in order to optimize the overlay routing, or to build logical overlays. The usage of area id is optional and is intended for extensions of the prototype.

2) Peer Awareness

The CCR also monitors all nodes and periodically checks, if nodes are alive. Whenever a node is detected as being unavailable, the CCR informs all other nodes about the change. Thus, the unavailable node will be removed from the overlay and no flow records will be sent to it any more. An additional functionality of the CCR is the distribution of application-specific messages to applications running on specific nodes, or to all application instances on all nodes. For example, such messages are queries sent by a network administrator to the flow storage application. The query is received by the CCR and sent to all participating nodes subsequently.

3) Template Management

The CCR stores flow templates and their mapping to analysis applications. One problem identified when dealing with IPFIX records exported by different exporters was that the same template definition received different template identifiers on those records exported by different exporters. In order to address this problem, SCRIPT uses the concept of a Global Template ID (GTID). Each SCRIPT node maintains a mapping between the pair (exporter ID, template ID) and GTID. At the entry point in the SCRIPT network, the template ID is changed to GTID for each IPFIX record. Thus, two IPFIX records having the same template definition and exported by different exporters will always have the same GTID, although the template IDs that these exporters used may have been different. Each node can detect, whether the value in a template ID field is a GTID by looking at the first bit of that value. If the first bit is “1”, the value represents a GTID, otherwise it is a template ID set by an exporter, so it needs to be changed.

C. SCRIPT Node

The SCRIPT node represents the key component of the architecture (cf. Figure 1) and represents a computing device participating in the distributed traffic analysis network. In Figure 1 nodes N1-N8 are SCRIPT nodes. A SCRIPT node is responsible for flow record routing and for delivery of IPFIX records to analysis applications.

1) Flow Record Routing

Forwarding of IPFIX records in SCRIPT is done using a routing function. Analysis applications may have different requirements with respect to how IPFIX records are routed. For example, a delay measurement application requires that all data exported for the same packet at different observation points is forwarded to the same node, while a traffic matrix calculation application may require that all records corresponding to the same (source, destination) pair are forwarded to the same node.

Therefore, the routing function is a hash function applied to some of those fields of a flow record: \( \text{Hash}((\text{record fields})) \), where \( f() \) is a function that enables operations on the record fields before calculating the hash value. For example, \( f() \) can be a logical AND operation on the source and destination address. The result of the routing function applied is a 32-bit identifier based on which the node, responsible for processing of that record, can be found. Based on this 32 bit identifier, the next hop of the IPFIX record is calculated using the Kademlia protocol [21]. If a next hop cannot be found, the IPFIX record is processed locally. The routing identifier is included in every flow record in a 64 bit field called routing hash ID as shown in Figure 3. Besides the routing identifier, the routing hash ID field contains 8 bits that are used to create temporary routing hash IDs (cf. Section IV.D). 8 bits are unused, while the last 16 bits may be set to an area identifier, which will cause the flow record of being routed only to SCRIPT nodes in a specific geographical area (for example due to privacy issues).

![Figure 3: Routing Hash ID](image)

2) Support for Analysis Applications

In order to deploy an analysis application in SCRIPT, an application ID (AppID) is chosen for it and the template (or templates) for the IPFIX records that will feed this application need to be known in advance. Respective templates are configured in the CCR and are mapped to the AppID chosen. In addition, for each newly defined template, a routing function needs to be specified. Whenever an IPFIX record is received by a node, the routing function specified for the respective template is used. If the record has to be processed locally, based on the template of the record, the AppID for those applications that use that template are retrieved and a copy of the record is delivered to each of those analysis applications.

3) Node Architecture and Functionality

The SCRIPT node architecture (cf. Figure 4) consists out of three main blocks: Management, Routing, and Flow Processing.
The Management block consists out of a Control Messaging component, which handles all communications of a node, a P2P Management component, which handles joining and leaving of nodes, and a Controller Unit, which orchestrates the operation of all components of a SCRIPT node. In addition, it defines an Application Programming Interface (API), which allows applications to be built on top of SCRIPT.

The Routing block includes an IPFIX Collector, which handles the receipt of incoming IPFIX records, a Flow Records Router that is responsible for finding the next hop of an IPFIX record, and an IPFIX Exporter component that is used to send IPFIX records to other nodes.

Once an IPFIX packet is received by the IPFIX Collector component, respective IPFIX records are decapsulated and passed to the Identification component. For each record, the Identification component checks, if the template ID represents a GTID. If so, the record is passed directly to the Routing and Filtering component. If the template ID is not a GTID the Identification component checks, if a mapping of (template ID, exporter) pair to a GTID exists. If there is no such mapping, a request for such a mapping to the CCR is made using the Control Messaging component. If such a mapping does not exist on the CCR either, the IPFIX record is dropped as well as all future records having that template ID, until an IPFIX record with the template definition for that template ID is received. When such a new template definition is received, it is forwarded to the CCR which returns a new GTID to be used for it and a routing function to be used with that template. Additionally, the CCR stores the new template ID and GTID in its mapping table. The final task of the Identification component, in case of IPFIX records with template IDs set by exporters, is to change these IDs with the corresponding GTID and set an internal flag ($F_{TC}$) for that record, specifying that this change was just performed locally.

Once an IPFIX record arrives at the Routing and Filtering component, the FTC flag is checked. If it is set, a new 64 bit field is added to the IPFIX record, representing a routing identifier ($R_{ID}$) and containing a value calculated by applying the corresponding routing function to that IPFIX record. This identifier will be used by all further SCRIPT nodes to route the IPFIX record. If $F_{TC}$ is not set, the $R_{ID}$ is not calculated, but read from the IPFIX record.

Based on $R_{ID}$ and the P2P routing information, the next hop node is selected. If no next hop is found, this record is delivered to the local Flow Processing block. If a better candidate than the local node is found, the IPFIX record is marked to be delivered to that node and is put in the outgoing Queue by the Dispatching component. The IPFIX Exporter periodically reads all Queues and sends records to the next hop nodes.

The Flow Processing block includes a Pre-Processing Unit (PPU), which dispatches each record that has to be locally processed to analysis applications expecting that record. When an IPFIX record arrives, the Flow Identity Unit (FIU) identifies these applications, which require the respective record, based on the template ID of the record, and the FIU passes the record to the Flow Processor, which notifies those applications by sending a copy of the new record. The Temporary Flow DB is a special application (cf. Section D below).

Finally, the external SCRIPT application receives flow records from the Controller Unit via the SCRIPT API.

**D. Temporary Handling of IPFIX Records**

Due to a number of reasons (such as loss of connectivity, overload, or network congestion) a SCRIPT node may become unavailable for some time. Such situations may be detected by nodes connected to the node experiencing problems. In such cases, IPFIX records that have to be routed to an unavailable node are temporarily stored by other nodes, which try after some time to deliver those records to the initial intended recipient. In order not to overload a single node with all extra work-load the following mechanism is applied: when a node has to forward a record to a node, which is known as being temporarily unavailable, the first 8 bits of the routing hash ID are changed and the record is re-routed using the new temporary routing hash ID. Due to the change of routing information, the node responsible for processing that record is also changed. The format of the routing hash ID contains all information required to identify if a record has the original routing hash ID or a temporary one. It also contains the information required to reconstruct the original routing hash ID when required. As soon as the IPFIX record arrives at the node responsible, showing the temporary routing hash ID, the record is placed in the Temporary Flow DB. After a time interval, the node reconstructs the original routing hash ID and injects the record in the routing process, which will deliver the record to the originally responsible node (in case problems of that temporarily unavailable node have been solved) or to another node for temporary storage. Once a temporarily stored record is reinjected in the
routing process, it is deleted from the temporary storage.

E. Design Trade-offs

Several design trade-offs in those steps described above have been made, which impact the performance or scalability of SCRIPT. One major trade-off in design is the use of a centralized element. A central element could reduce performance and could decrease the reliability of the solution. However, the decision to use a centralized element for some tasks was made due to the fact that using this approach a lower latency can be achieved compared to the same tasks being implemented fully in a distributed manner. The load on the CCR is expected to be small, since it is used only for management operations, such as identity provisioning, template management, or peer configuration. A node only contacts the CCR, when it is started (to receive an identity), when it receives an IPFIX record with an unknown template ID (to receive the template definition, its GTID, and routing function), and when it receives a new template definition (to map the newly observed template to an existing GTID).

Another design trade-off was concerned with the responsibility of the peer awareness task. In the current prototype (cf. Section V), the CCR periodically checks the node’s availability and informs other nodes, when a node becomes unavailable. Designing peer awareness centrally allows for much faster reactions in case of a node being disconnected. A deployment of the solution presented here will not see more than several hundred nodes, thus, such a monitoring task can be performed easily by a single entity due to a reduced number of messages that the CCR has to process.

Finally, robustness of SCRIPT can be improved straightforward by introducing a secondary CCR, which mirrors the configuration and operation of the primary CCR and which can take over in case the primary CCR becomes unavailable.

V. IMPLEMENTATION

The SCRIPT distribution framework is implemented in C++ as a P2P overlay based on the Kademia protocol. Each SCRIPT node holds 32 buckets of other SCRIPT node addresses, for routing purposes. Each bucket can contain up to 20 addresses.

A. Application Support

Applications can be built on top of the SCRIPT framework by using an API provided by the framework. An application is started by registering it with the Controller Unit, by calling the method registerApplication(templateId, application). The template ID passed in the registration call is used to identify those flow records, which will be passed to the application. If the application needs to receive more than one template, a separate registration call is required for each template ID.

An application needs to extend the class LocalProcessor and implement the methods notify(char *msg) and process(sc_flowRecord *fr). The notify method allows application-specific messages (such as configuration options, or queries) to be sent to an application during runtime. The process method is called, whenever a flow record with a template ID required by the application is received. A copy of a record is received using the process method.

B. Bootstrap Process

When starting a node, the IP address and port number of an existing SCRIPT node is needed for the bootstrap process. The first step, already described in Section V.B, is the retrieval of an identity from the CCR. The CCR maintains a list of already assigned identities and always tries to assign an identity in order to keep as much as possible a uniform distribution of assigned identities. The algorithm used by the CCR is to find the largest interval between two consecutive identities \( p \) and \( q \), and choose the value \( [(p+q)/2] \) as a new identity.

During the bootstrap process a set of other existing SCRIPT nodes (IP addresses, port numbers, and node identities) is received from the bootstrap node and are used as an initial routing table. This information is exchanged using the Control Messaging component over UDP (User Datagram Protocol) messages. Whenever a node learns about another node in the network, two IPFIX sessions are created, one in each direction, between the IPFIX exporter of one node and the IPFIX collector of the other node, for exchanging IPFIX records. At the same time the new node is added to the appropriate k-bucket. These operations only take place when a new node connects to the network, so they only create a limited load.

C. IPFIX Collector

The IPFIX Collector component is implemented for UDP and SCTP (Stream Control Transmission Protocol) which are two of the tree transport protocols proposed by IETF for IPFIX. The SCTP version, besides using the protocol preferred by IETF, allows better peer-awareness by using SCTP notifications when the status of an association between two nodes changes. When using SCTP, each time a node leaves SCRIPT all the nodes to which that node has an IPFIX session are immediately notified about the leave, so they can update their routing rules.

D. Routing

The routing function is used to send IPFIX records with similar characteristics to the same SCRIPT node. A routing function works as follows: \( L \) bits of the IPFIX record, starting at offset \( O \) are AND-ed with an \( L \)-bit mask \( V \). The value resulting is passed through a hash function (the implemented prototype uses the BOB [1] hashing function) in order to receive the routing ID. A routing function is expressed as a set of its three elements: \( (L, V, O) \). Using such a combination of a mask and a hash function allows for a flexible manner to manipulate, which IPFIX records will map to the same routing hash ID.

After an IPFIX record is received by a node and its routing hash ID is calculated (or retrieved), a better candidate for processing that routing hash ID is searched in the P2P routing table. Once such a candidate node is found, the IPFIX record is...
appended to a queue of IPFIX records waiting to be sent to that node. A separate process continuously checks the queue status for each SCRIPT node and if the number of records reaches a predefined threshold an IPFIX packet with some of those records is sent to the respective SCRIPT node.

E. Embedded Environment

Cisco has recently introduced the Application Extension Platform (AXP) [4], which allows applications to run within a router. The implementation of SCRIPT has been compiled using the AXP Source Development Kit (SDK) and tested having two of these nodes running within two AXP cards. At the time of implementation and testing of the prototype in AXP, the SCTP protocol was not fully supported by these AXP cards yet, thus, the UDP alternative was applied as a transport protocol for IPFIX, when one or more nodes run on AXP cards.

VI. EVALUATION

In order to assess the applicability of the SCRIPT approach proposed a set of functional as well as performance analysis steps of the SCRIPT architecture and the implemented prototype was performed.

The main purpose of SCRIPT is, to recall its major benefit, to distribute IPFIX records to several machines according to rules required by an analysis application. This is achieved by organizing participating nodes in a P2P overlay and by using the P2P overlay information for distributing the IPFIX records. Using the API provided, applications can define routing functions according to their dedicated requirements. The same API allows for building analysis applications on top of SCRIPT and for receiving IPFIX records delivered by the SCRIPT framework.

![Flow Storage Performance](image)

**Figure 5: Flow Storage performance**

Thus, the performance of the SCRIPT prototype is complex to be assessed, especially in comparison with other tools, since no such generic frameworks for distributed traffic data analysis exist. Therefore, the performance evaluation includes an evaluation of IPFIX records storage in a traditional, centralized collector, compared to the performance of a distributed collector built on top of SCRIPT. The tests were made using similar PCs with 3.6 GHz Intel processors, each having 4 GB memory. All tests have been performed in a switched Local Area Network, each PC having a 1Gbps network card. On the centralized collector the maximum rate of flow records that could be saved was 250,000 flows per second. Using SCRIPT running on 8 similar PCs in parallel a rate of 600,000 flows per second was achieved. In this evaluation, one stream of 150,000 flows per second was sent to 4 of the 8 nodes. Using only 4 nodes with SCRIPT the maximum flow rate that could be achieved in this prototype was 269,000 flows per second. These results are summarized in Figure 5.

During this evaluation it was observed that a single SCRIPT node can not process (in this case store in files) as many flow records as a similar centralized application running on the same node. The reason for this is that when running SCRIPT, some of the resources of a node are spent for calculating hash values and for the routing process, thus leaving less resources for the analysis application.

![Record distribution](image)

**Figure 6: Record distribution**

A second evaluation was performed to check, if SCRIPT distributes flow records equally between participating nodes. These results are shown in Figure 6 and outline that the average rate of flow records during a 60 seconds test using 8 SCRIPT nodes is at about 62,000. As it can be observed in Figure 6, the maximum flow rate was 65,780 flows per second, while the minimum rate was at 60,535, resulting in a maximal deviation of 5% from the theoretical mean value.

VII. SUMMARY AND CONCLUSIONS

IP flow records are frequently used in network management and traffic analysis, but classical flow collection and analysis architectures with centralized collectors have limitations regarding the scalability and performance in high-speed networks. The SCRIPT framework for IP traffic analysis introduced here addresses this problem by distributing flow records and analysis workload to multiple nodes. SCRIPT nodes build a Kademlia-based overlay to route and distribute flow records. If the overall load increases, new SCRIPT nodes can be added to the overlay on demand, requiring no manual configuration effort in the operation.

The SCRIPT framework also distributes the workload of analysis applications, since each SCRIPT node can run a part of the analysis task. Analysis applications, like delay measurement or asymmetric route detection, access the SCRIPT functionality over a well-defined API and the system can be extended with new applications. The SCRIPT framework uses
a flexible routing function that can be specified according to demands of each analysis application separately. It builds on standard protocols and supports IPFIX and NetFlow-based data transfer that supports not only IP flow records but also per-packet information (e.g., packet header).

The SCRIPT framework has been implemented as a prototype and evaluated both on standard PC hardware as well as on Cisco AXP cards. The performance evaluations show that SCRIPT can increase the total number of flow records processed compared to a centralized solution and it scales with the total number of flow records exported in a network. The overhead introduced per SCRIPT node for flow record routing and relaying is low and the Central Configuration Repository (CCR) does not determine a bottleneck, since it is responsible only for management tasks, it does not participate in the flow record transfer, and SCRIPT nodes contact it rarely. As the evaluation reveals, the framework distributes flow records nearly equally among all nodes in the SCRIPT overlay, resulting in a fair balance of workload among all nodes.

Future work includes further performance evaluation both on standard PC hardware as well as on Cisco AXP cards. Additionally, analysis applications will be developed on top of SCRIPT and evaluated in terms of their performance.

Acknowledgements

This work was supported in part by the Cisco University Research Program Fund, Grant-No. 2008-02735, in part by the DaSAHIT project funded by the Swiss National Science Foundations, Contract No. 200021-118128/1, and the IST Network of Excellence EMANICS funded by the European Union, Contract No. FP6-2004-IST-026854-NoE.

The authors would like to express many thanks to Ralf Wolter, Benoit Claise, and David Hausheer for their valuable support of this work and many inspiring discussions.

References


