THESIS

# HERMES - SCALABLE REAL-TIME BGP BROKER WITH ROUTING STREAMS INTEGRATION

Submitted by

Belyaev Kirill Alexandrovich

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2011

## ABSTRACT

## HERMES - SCALABLE REAL-TIME BGP BROKER WITH ROUTING STREAMS INTEGRATION

BGP is the de facto inter-domain routing protocol of Internet and understanding BGP is critically important for current Internet research and operations. Current Internet research is heavily dependent upon the availability of reliable up-to-date BGP data sources and often evaluated using data drawn from the operational Internet. The BGP real data supports a wide range of efforts ranging from understanding the Internet topology to building more accurate simulations for network protocols.

To study and address the Internet research challenges, accessible BGP data is needed. Fortunately a number of BGP monitoring projects have been deployed for BGP data provision. However experience over a number of years has also indicated some major limitations in the current BGP data collection model with the most dramatic one being the inability to deliver real-time data and incapability to process and analyze this data fast enough in a flexible and efficient manner.

This thesis presents the design and implementation of the new tool for analyzing BGP routing data in real-time - Hermes BGP Broker. Hermes is build upon the solid foundation of the related project - BGPmon [CSU] that is the BGP aggregation and monitoring platform that uses a publish/subscribe overlay network to provide real-time access to vast numbers of peers and clients. All routing events are consolidated into a single XML stream. XML allows to add additional features such as labeling updates to allow easy identification of useful data by clients and other related data structuring. Hermes as the Broker for BGPmon represents the next generation of route monitoring and analysis tools that bring routing data to the level of end-user applications.

The main contribution of this thesis is the design and implementation of a new BGP route analysis platform that can be extensively used both in research and operational communities. Our work on Hermes has delivered the system that is able to analyze continuous XML data stream of BGP updates in real time and select non-duplicate messages that correspond to the specified regular expression pattern. Besides effective filtering mechanism Hermes is capable to scale really well with a large number of concurrent stream subscribers. Its performance under intensive benchmarking has been evaluated and estimated to be suitable for real-world deployment under heavy load with a large number of concurrent clients. The system is also able to distribute the filtering computations among a number of nodes and form Hermes data stream meshes of various topologies.

## DEDICATION

*To Tamiris, for providing constant distractions and a steady stream of better things to do*

*To the loving memory of my GrandParents - Anastasya and Grigoriy Chernov(s) for providing indispensable support and steering in my life*

TABLE OF CONTENTS

# LIST OF TABLES

## LIST OF FIGURES

# Chapter 1

# Introduction

Understanding the mechanisms of global routing is of critical importance for current Internet research and operational network security. Modern Internet organized into a collection of Autonomous Systems (AS) is a highly distributed system where routing between the AS domains carry the vital information required for the successful operation of such a complex digital organism as Internet is nowadays. The interdomain routing between ASs is accomplished via Border Gateway Protocol (BGP) [rfcd] - a global routing protocol with well-known research challenges. Modern Internet research is heavily dependent upon the availability of operational routing data. In order to comprehensively understand and properly analyze the global routing system, one needs to collect BGP data from a wide range of sites with dispersed geographical locations and various ISP tier levels.

As would be discussed in this thesis several well-established BGP collecting projects have been providing data for operational research for quite some time. However experience over a number of years has also indicated some major limitations in the current BGP data collection model with the most dramatic one being the inability to deliver real-time data and incapability to process and analyze this data fast enough in a flexible and efficient manner.

An ideal route monitoring system would scale to a vast number of peer routers and provide BGP data in real-time to an even larger number of clients. For example, one might like to add operational routers from different geographic locations and lower tier ISPs. At the same time, real-time access would enable all interested parties to analyze the data to detect events such as fiber cuts, prefix hijacks, and so forth. And even more crucial feature of the ideal monitoring system would be the elimination of redundancy in routing traffic and provision of non-duplicate stream targeted towards the needs of individual client subscribers.

The ability to monitor and detect the network anomalies such as prefix hijacks is extremely important not only from the technical perspective but also from the business logic standpoint. The consequences of extended service downtime for large web service providers such as Amazon, Yahoo! or others are drastic. eBay suffered outages in 1999 that outraged users and sent the stock down. And for major commerce sites, the problem can have ripple effects. Both Amazon and eBay provide a commercial foundation used by many partners and entrepreneurs. Based on 2008 last quarter's revenue of 4.13 billion US dollars globally, a full-scale global outage would cost Amazon more than 31,000 USD per minute on average. For North America, it would be more than 16,000 USD per minute. (Those figures don't include revenue from other sources such as search or contextual advertisements or Amazon Web Services.) During the hour and half outage of 2008 alone the New York Times reported that ”Amazon, by some estimates, lost more than a million dollars an hour in sales.” [Sha] If the service outage were to be caused by the planned prefix hijack the attacker might have also installed the clones of the web service under attack routing unaware users to its web servers. This would result in the harvesting of user account data causing identity theft and subsequent financial and social impact on the economy.

This thesis presents the design and implementation of the new tool for analyzing BGP routing data in real-time - Hermes BGP Broker. Hermes is build upon the solid foundation of the related project - BGPmon [CSU] that is the BGP combining and monitoring platform that uses a publish/subscribe overlay network to provide real-time access to vast numbers of peers and clients. All routing events are consolidated into a single XML stream. XML allows to add additional features such as labeling updates to allow easy identification of useful data by clients and other related data structuring. Hermes as the Broker for BGPmon represents the next generation of route monitoring and analysis tools that bring routing data to the level of end-user applications.

Our work on Hermes has delivered the system that is able to analyze continuous XML data stream of BGP updates in real time and select non-duplicate messages that correspond to the specified regular expression pattern. Besides effective filtering mechanism Hermes is capable to scale really well with a large number of concurrent stream subscribers. Its performance under intensive benchmarking has been evaluated and estimated to be suitable for real-world deployment under heavy load with a large number of concurrent clients. The system is also able to distribute the filtering computations among a number of nodes and form Hermes data stream meshes of various topologies.

The remainder of this thesis is organized as follows: Chapter 2 provides general background on BGP and its operational characteristics. Chapter 3 covers the previous projects in the area of BGP data collection and analysis and outlines the path behind the motivation for the development of Hermes. Chapter 4 provides the crux of the Hermes itself covering its design and implementation aspects. Chapter 5 introduces the new area in the field - data streams and draws similarities between existing infrastructure and data stream platforms. Chapter 6 shows current deployment status of Hermes and the experimental data obtained from the deployment. Chapter 7 concludes the thesis and discusses the future development roadmap.

# Chapter 2

# BGP Routing

Modern Internet is organized as a collection of Autonomous Systems (AS) [rfca], each of each is under the control of a single administrative entity. The routing between ASs is done via Border Gateway Protocol (BGP) [rfcd] that was designed to provide the interdomain routing facility between the routing domains (ASs). Autonomous Systems are used for the provision of additional model of hierarchical aggregation of the routing information in large internets with the goal of scalability improvement [Hus]. The routing problem itself is therefore divided into two parts: routing within a single autonomous system and routing between ASs themselves. Routing within the AS itself is implemented using intradomain routing protocols such as RIP [rfcc] and OSPF [rfcb] that are essentially distance-vector and link-state protocols respectively. These protocols are usually computing routes based on a defined algorithm such as SPF (Shortest Path First) and are basically deterministic in their behavior.

The routing at the interdomain level is quite different from the model used inside the AS. The interdomain routing provides a way for exchange of the reachability information between different ASs announcements regarding networks that could be reached via a given AS. This task is currently managed with BGP. There has been an earlier interdomain routing protocol the Exterior Gateway Protocol (EGP) [rf9] that had a number of limitations one of the most severe of which was its constrain to the tree-like topology of the Internet. EGP did not allow for the topology to become more general and as Internet evolved was not able to cope with the change. In the treelike structure of the original Internet there was a single backbone and ASs were connected only as parents and children and not as peers. The replacement for EGP that came in the form of

BGP (currently version four) allows the Internet to be an arbitrarily interconnected set of ASs with multiple backbone networks [Gao01].

The most important challenge of interdomain routing imposed by BGP today is the necessity for each AS to determine its own routing policies – a task that is very different from the intradomain deterministic routing model. Thus, while intradomain focuses on a well-defined problem of optimizing the scalar cost of the path, interdomain is focused on finding the best, non-looping and policy-compliant path possibly involving a notion of trust – a much more complex optimization problem that essentially rather belongs to the economic game theory domain than to the classic routing on a graph approach [FJB05][BC01]. BGP does not belong to either distance-vector or link-state protocols but rather is a path-vector protocol that advertises complete paths as an enumerated list of ASs to reach a particular network. BGP does not try to calculate the optimal path costs for a path that crosses multiple ASs but rather advertises only reachability of a particular network prefix. Since in a path-vector protocol such as BGP the entire routing path is exchanged between neighbors the edge (border) routers in each network learn multiple paths to reach a particular destination and store all of them in their routing tables. Based on a set of economic policies that are embraced by a particular AS a border router then selects a single active route. BGP thus allows support of flexible local policies that give each AS control over its incoming and outgoing traffic depending on the business relationships [rfcd][WG03].

The most common relationships in BGP are customer-provider, peer and sibling. In a customer-provider relationship, the customer normally pays the provider for transit service; as such, the provider announces the routes learned from any customer to all neighboring ASs, but the customer normally only advertises the routes learned from its provider to its own customers. In a peer-peer relationship, two ASs find it mutually beneficial to carry traffic between each others customers, often free of charge. Peering agreements often indicate that the routes learned from a peer can only be advertised to customers. Sibling ASes typically belong to the same institution, such as a large ISP, and provide transit service to each other. Upon learning routes for a prefix from multiple neighbors, an AS typically prefers to use customer-learned routes, then siblings, then peers, and finally providers, to maximize revenue [Gao01][WG03]. The BGP router (speaker) is under no obligation to advertise any route to a destination, even if it has one. This is how an AS can implement a policy of not providing transit – by refusing to advertise routes to prefixes that are not contained within the AS, even if it knows how to reach them. BGP is designed to operate on top of reliable TCP connection that does guarantee the reliability of exchanging of the routing updates between the BGP routers therefore eliminating the need to resend the updates. Normally only short keep alive messages are being exchanged between routers in the absence of a topology update information requirement [rfcd].

BGP also provides a mechanism to gracefully close a connection with a peer. In other words, in the event of a disagreement between the peers, be it resultant of configuration, incompatibility, operator intervention, or other circumstances, a NOTIFICATION error message is sent, and the peer connection does not get established or is torn down if it's already established. The benefit of this mechanism is that both peers understand that the connection could not be established or maintained and do not waste resources that would otherwise be required to maintain or blindly reattempt to establish the connection. The graceful close mechanism simply ensures that all outstanding messages, primarily NOTIFICATION error messages, are delivered before the TCP session is closed.

Initially, when a BGP session is established between a set of BGP speakers, all candidate BGP routes are exchanged. After the session has been established and the initial route exchange has occurred, only incremental updates are sent as network information changes. The incremental update approach has shown an enormous improvement in CPU overhead and bandwidth allocation compared with complete periodic updates used by previous protocols, such as EGP [WZP+02].

Routes are advertised between a pair of BGP routers in UPDATE messages. The UPDATE message contains, among other things, a list of <length, prefix > tuples that indicate the list of destinations that can be reached via a BGP speaker [1]. The UPDATE message also contains the path attributes, which include such information as the degree of preference for a particular route

[1]Although it is true for traditional IPv4 networks the mechanism is not quite the same for IPv6 where a designated MP_REACH_NLRI BGP attribute is used to convey IPv6 prefixes and shall be set to 16, when only a global address is present, or 32 if a link-local address is also included in the Next Hop field.

and the list of ASs that the route has traversed.

KEEPALIVE messages are sent periodically between BGP neighbors to ensure that the connection is kept alive. KEEPALIVE packets (19 bytes each) should not cause any strain on the router CPU or link bandwidth, because they consume a minimal amount of bandwidth (one instantaneous 152-bit packet every 60 seconds, or about 2.5 bps per peer for a periodic rate of 60 seconds).

BGP keeps a table version number to keep track of the current instance of the BGP routing table. If the table changes, BGP increments the table version number. A table version that increments rapidly is usually an indication of network instability (although this is quite common in large Internet service provider networks). Because of this, instability introduced by Internet-connected networks anywhere in the world will result in the table version number incrementing on every BGP speaker that has a full view of the Internet routing tables. Route flap dampening and other provisions have been designed to scope the effects of this instability [BBAS03][PAMZ05].

## 2.1 BGP Message Types

To provide the complete background introduction to BGP we should list the main types of messages being exchanged by BGP peers. In a BGP session, the peers mainly exchange the following four types of messages:

- *OPEN:* An open message is sent after the TCP three-way handshake is completed. The BGP OPEN message is used to open a BGP session. The OPEN message contains information about the BGP neighbor initiating the session.

- *KEEPALIVE:* A keepalive message is used to keep the session running when there are no updates. Keepalives are sent between BGP speakers to let each other know they are still there.

- *UPDATE:* An update message carries network reachability information. An update either advertises a prefix or withdraws a previously announced prefix. Specifically, it manly consists of three parts:

  - *Withdrawn Routes:* The IP address block, or prefix, that can not be reached.
  - *Path Attributes:* A set of parameters describing the characteristics of the route, such as AS-Path, multi-exit discriminator (MED), next hop, community etc. Generally, the path attributes are used by routing policies and route selection process.
  - *Network Layer Reachability Information:* The IP address block, or prefix, that can be reached.
  - *ASPath:* A path composed of a sequence of AS numbers in the order of reaching the destination.

- *NOTIFICATION:* A notification message is sent when an error condition is detected. The BGP session is closed immediately after it is sent.

## 2.2 BGP Route Selection

To model the BGP process each BGP router (speaker) has different pools of routes and different policy engines applied to the routes (although in reality only one pool may exist). The model would involve the following components:

- *A pool of routes that the router receives from its peers*
- *An Input Policy Engine that can filter the routes or manipulate their attributes*
- *A decision process that decides which routes the router itself will use*
- *pool of routes that the router itself uses*
- *An Output Policy Engine that can filter the routes or manipulate their attributes*
- *pool of routes that the router advertises to other peers*

A BGP router typically maintains three types of Routing Information Bases (RIBs), *Adj-RIBs-In*, *Loc-RIB*, and *Adj-RIBs-Out*, to store BGP routes. First of all a BGP router maintains an Adj-RIBs-In for each neighbor and stores the routes learned from neighbors into their respective Adj-RIBs-Ins after applying import policies. If there are several alternate routes to the same prefix in these Adj-RIBs-Ins, the best route selection process will be executed. The process consists of an ordered list of attributes across which the routes are compared as shown in the Table 2.1.

Table 2.1: Steps in the best router selection process

| Step | Attribute |
|---|---|
| 1 | Highest Local Preference |
| 2 | Shortest AS Path |
| 3 | Lowest Origin Path |
| 4 | Lowest Multi-Exit Discriminator |
| 5 | EBGP over IBGP |
| 6 | Shortest IGP path |
| 7 | Lowest Router ID |

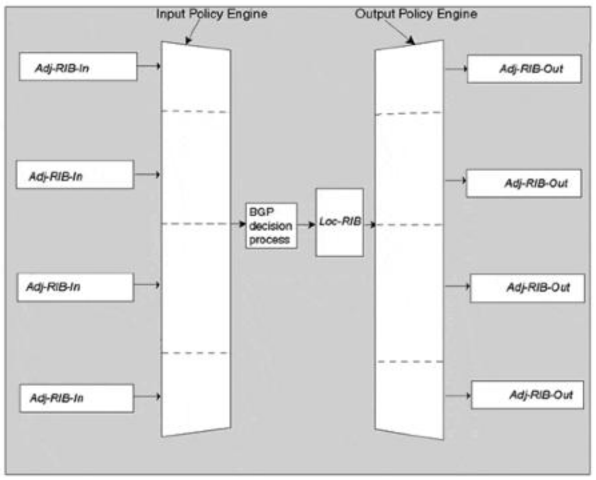

Figure 2.1: Logical Representation of BGP Routing Table

Each attribute in the list is compared across all the routes. If the routes have different values for one attribute, the route with the more desirable attribute will be selected. Otherwise the next attribute in the list will then be compared. The ordered list allows the network operator to influence various stages of the selection process. For example by changing LocalPref, an operator can force a route with a longer AS path to be chosen over a shorter one as the Local Preference (LocalPref) is the first step in the list. After comparison of all the routes, the best one is stored in the Loc-RIB. Then a single best route for each prefix is selected from all Adj-RIBs-Ins into the single Loc-RIB. Finally the BGP router may advertise the best routes in the Loc-RIB to its neighbors. For various reasons, not all best routes are advertised to all neighbors. Export policies which can be specified on a per neighbor basis are needed to decide which routes should be advertised to which neighbors. As a result, typically a BGP router maintains one Adj-RIBs-Out for each neighbor which contains the routes from the Loc-RIB after applying the export policies.

BGP as a protocol presents some basic elements of routing that are flexible enough to allow total control from the administrator's perspective. The power of BGP lies in its attributes and its route-filtering techniques. Attributes are simply parameters that can be modified to affect the BGP decision process. Route filtering can be done on a prefix level or a path level. A combination of filtering and attribute manipulation can achieve the optimal routing behavior. Because traffic follows a road map laid out by routing updates, modifying the routing behavior will eventually modify the traffic trajectories [MZHL05].

# Chapter 3

# Previous Work

The importance of processing BGP data has been assessed by a number of previous projects in the area. The work carried out by these projects is the main focus of this chapter. First section describes Oregon RouteViews and RIPE RIS data collectors, their goals and functional mechanisms of operation. Second section introduces BGPmon - the new tool to combine the data feeds from route collectors.

## 3.1 Oregon RouteViews And RIPE RIS

Attempts to adequately analyze global BGP routing data could be traced to a number of previous projects in the area most notably the Oregon RouteViews project [UO] and the RIPE RIS [RIPa] that have been the first notable attempts to provide the coherent view of BGP routing updates in the Internet. The importance of obtaining real BGP data is of crucial importance to the academic community that is involved in research targeting the global Internet. Real BGP data, i.e. the messages exchanged among Internet BGP routers provide academic researchers with sufficiently enough information to understand the current global routing system, such as Internet topology, routing dynamics, routing table growth and so forth. BGP routing updates mainly from Oregon RouteViews and RIPE RIS project repositories have been extensively used in the majority of research publications on the topic and sometimes provide the bulk of data used in the academic analysis. Real BGP data allows Internet operators to diagnose their networks. It also provides remote views of their networks from different locations around the Internet [ZLMZ05][UM].

The difference between the two generally lies in the geographical data scope coverage of the Internet with RIPE RIS mainly providing data for the European part of the Internet and Oregon RouteViews accumulating data sets that spread out over a wider range of topological locations mainly originating within the US Internet branch. RouteViews has nearly 55 unique US IPv4 peering sessions with major US ISPs while RIPE has around 255 IPv4 unique peers with European ISPs. Both route repositories collect the routing data using so called trace collectors – nodes that peer with commercial ISP networks via BGP sessions. A collector receives BGP messages from its peers, but it does not advertise any prefixes back to them. Periodically, the collector dumps its full routing tables and routing updates received from its peers. Typically a collectors routing table has more than one hundred thousands entries from each peer AS since each peer AS tells the collector how it reaches the entire destination address space [Bro01].

RIPE routing data is heavily used in so called Internet Routing Registry databases. The purpose of the IRR is for operators to coordinate global policy settings. Network operators may register routing policies with the IRR. The databases that form the IRR are manually maintained by operators mostly on a voluntary basis. Information therein may be incorrect, incomplete, or outdated. The RIPE portion of the IRR is actively used by ISPs in Europe to filter route announcements and many European exchange points require operators to register with RIPE. Consequently, RIPE data is considered the most reliable information in the IRR [RIPa].

Oregon RouteViews and RIPE RIS share the similar approaches in monitoring the BGP routing. Simply stated, the monitors (Routing Collectors) in these systems monitor the BGP system by peering with operational routers in order to collect BGP messages. Then these BGP messages are provided to users in order to conduct further analysis. As already stated RouteViews and RIPE RIS have a proven track record in benefiting the research community as well as the operations community and both of them are built upon a centralized, file based infrastructure. Typically the monitor (Route Collector) in both projects can be either a CISCO router or a Linux box with zebra/quagga suite installed. Zebra/quagga is an open source routing software suite that provides implementation of OSPF, RIP and BGP for Unix platforms [RIPb].

BGP updates and Routing Information Base (RIBs) entries are dumped periodically into files

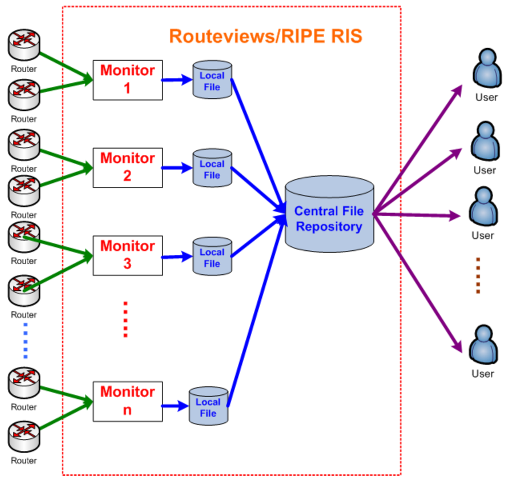

Figure 3.1: Centralized, File-Based Infrastructure

at each Routing Collector and the central file repository periodically pulls the dumped files from all monitors (Collectors) in order to make BGP data accessible to users. The dumped BGP updates and RIBS are in a special MRT compressed format. The MRT format defines a way to export and exchange routing information [iet]. Then users have ability to obtain the compressed BGP data in MRT from the centralized file repository via http or ftp. RouteViews and RIPE RIS are somewhat similar in some aspects such as using a centralized, file based infrastructure, using zebra/quagga and dumping BGP data in MRT format. Both projects have some common issues which were identified over the past several years.

Both RouteViews and RIPE do not actually provide BGP data in real time. The RouteViews archives update messages every 15 minutes and RIBs every 2 hours, while RIPE does this with 5 minutes and 8 hours respectively. Some delay does exist between receiving and publishing data. For instance, about 0.5 hours and 6 hours delay does exist in updates and RIBs respectively in RouteViews' data alone. Limited by the centralized, file-based infrastructure mentioned above, it is quite hard to achieve real time access in these systems [UO][RIPa].

Obviously this impacts the accuracy of academic research and does not quite satisfy the needs of some tools developed for analyzing the BGP updates.

For example, some prefix hijack detection systems such as PHAS [LMP$^+$06] use BGP data from the public BGP monitoring system to detect possible hijacks. The delay in accessing BGP data limits the real-time detection ability of such a system.

Another tool entitled Cyclops [COZ08] is a network audit tool for service providers and enterprise networks, providing a mechanism to compare the observed behavior of the network and its intended behavior. Cyclops is able to detect several forms of route hijack attacks, i.e. when Internet routes are maliciously diverted from their original state but its usefulness could be severely affected by the outdated archived data that is the main source of its data manipulation routines.

In addition to non-real time properties of batch data files both RouteViews and RIPE RIS systems have rather primitive access approaches. Users cannot selectively receive data from these monitoring systems and have to retrieve the entire archive file via http or ftp even when they only need a small portion of the data.

## 3.2 BGPmon

The above mentioned limitations of the original attempts to provide usable and flexible access to BGP updates have been overcome by the new BGP monitoring system – BGPmon [CSU] that uses a publish-subscribe overlay network to provide real-time access to vast numbers of peers and clients. The major difference from previous work is the integration of XML into the model and the transformation of BGP update messages from text files into XML format. Each BGP message collected from peering connection is now transformed into XML BGP message format [cg]. XML permitted to add additional features such as updates labeling for easy data identification and more importantly enable the use of XML parsing libraries to parse and manipulate the BGP messages at a fine granularity level. BGPmon is able to provide a constant stream of XML-formatted BGP messages to every client connecting to its service port.

In addition to providing the data in real time BGPmon is able to scale and adjust to vast majority of peers providing routing data via publish-subscribe mechanism that consists of brokers, publishers and subscribers communicating via an overlay network. The so called brokers form the overlay network allowing publishers to send update streams to the overlay network and allowing subscribers to receive update streams from this network. The BGPmon model of data access proposes that publishers and subscribers should interact only with brokers and not with each other

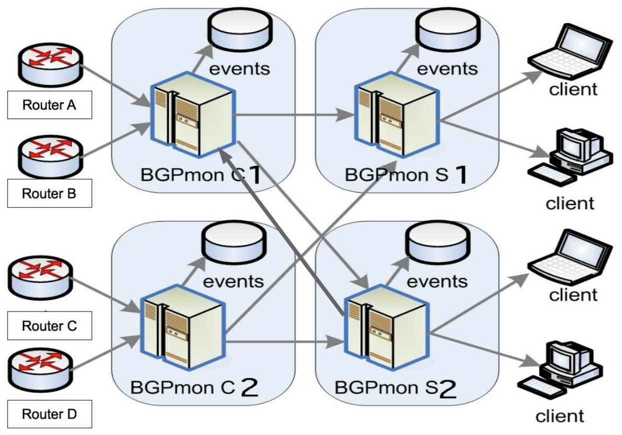

Figure 3.2: BGPmon mesh

allowing the overlay network to insulate publishers and subscribers from each other [YOB+09].

Although the provision of XML-formatted BGP messages in real-time and the ability to construct a scalable data distribution network is a a major step forward in comparison to the batch file archives this still does not quite meet the goals of enabling coherent fine-grained analysis of the BGP routing events at the level of specific end user application or designated problem case. The end point receiving the vast stream of BGP events from BGPmon is still not benefiting from data and is forced to ensure additional filtering mechanisms to make data useful for its specific application. The problem of receiving a vast stream of data is further exacerbated by the possibility of forming a loop in the data stream distribution that would cause the unnecessary and costly increase in the traffic volume that is being transmitted through the BGPmon mesh network (as one in figure 3.2 ). The actual BGP routers that are peered with BGPmon send periodic updates of their routing table entries to it without understanding of how these BGP messages would be further handled by the BGPmon mesh. If the BGPmon nodes are interconnected in such a way that XML messages are send to each other without a proper duplicate suppression mechanism a client would be receiving a massive data stream twice saturating its bandwidth consumption. And that is even when he is not able to consume and analyze the original stream in the first place.

Figure 3.2 illustrates how data flows in a BGPmon mesh. When a route changes at router A, A pushes an update to the BGPmon collector at C1. C1 then converts the update to XML and pushes it to S1 and S2. Now S2 would push the update received from C1 back to C1 (since it has a link back to C1) that in turn would push it to S1 and S2 again if no proper duplicate suppression mechanism is in place. S2 also pushes the update to its connected clients who are performing the pull operation to retrieve the stream. In the figure no actual loop would form because current BGPmon instances do have the duplicate suppression mechanism in place. Routers perform push operation on request to pull from BGPmon collectors which in turn do push in request to pull from the end clients.

The roadmap to the level where a client is able to receive data he actually is looking for is described in the next section of this thesis outlining the Hermes real-time route analysis system implementation and deployment.

# Chapter 4

# Hermes – Design And Implementation

Based on the background presented in chapter 3 it is well understood that the current academic research conducted on the BGP routing and AS level Internet topology has the need for a new update distribution mechanism since this research is heavily dependent upon the availability of accurate and up-to-date routing data sources. The failure of such sources to provide data that meets the requirements of the intended research objectives could severely impact the academic accuracy of the obtained results and undermine the long-term validity of claimed outcomes. The main route repositories outlined above being RIPE RIS and Oregon RouteViews respectively form the main platforms for providing routing updates for further study and analysis. Both projects provide quite limited batch file oriented service with no support for real time data retrieval.

This inability to deliver routing data in real time has huge impact on the effectiveness of some relevant Internet probing tasks such as hijack detection, AS-level failures detection and other emerging demands. In essence this discrepancy in timing leads to the fact that analysis is conducted on samples that are already outdated and replaced by new data by the time these samples are retrieved limiting the degree of responsiveness and ability to predict short-term trends in the topology.

The attempt to overcome some of these shortcomings has been made by the BGPmon project [CSU] that proposed to convert routing data into extensible XML messages and stream those in real-time to the connecting clients forming the publish-subscribe overlay network. This approach has provided real-time service but did not quite solve the problem of end points receiving the alleged routing updates in XML format. The problem is that end points in the service could be represented not only by routing devices or their software implementations but could amply involve higher level applications such as monitoring systems and network policy regulation engines. These types of networking applications would not generally be able to directly peer with the BGPmon instances and receive real-time XML BGP updates without proper adaptation layer integrated into them. Most of the clients of such type would not be able to coherently interpret and process massive streams of routing updates without an external broker between them and the BGPmon mesh network. The data although streamed in real-time is mostly meaningless and useless for these applications without proper refinery. Even worse – they might not be specifically engineered for massive network-level IO operations involved in routing updates that need to be carried out on a constant basis involving not a single network peer but potentially a number of such peering points.

Besides this no simple and effective approach for efficient merging and propagation of data in real-time between application nodes has been proposed for further data mining and analysis.

We claim that a new stratum of service layer between BGPmon and the end-point clients could be introduced towards specifically targeting the needs and requirements of these networking applications residing at a higher level of logical hierarchy. The routing middleware service system is therefore presented in the following section being the attempt to address some of the outlined limitations of the current route monitoring approaches.

## 4.1 Overview

As already mentioned BGPmon provides necessary infrastructure for the so called BGP brokers - clients connecting to the BGPmon instance for data retrieval [YOB$^+$09]. One of possible BGP broker implementations is the project code-named Hermes - experimental real-time BGP broker and route analysis system presented in this thesis paper. Hermes named after ”the great messenger of the gods in Greek mythology and additionally as a guide to the Underworld” represents the second stage of development for the BGPmon route aggregation architecture providing potential overlay network infrastructure to meet the demands of the Internet scale with additional service extensions. Unlike BGPmon that simply collects route data, converts it to XML format and streams it to the connected client Hermes is capable of providing a set of new service extensions such as filtering and aggregating routing data based on the specified criterion allowing clients connecting to Hermes instance to receive non-duplicate targeted data stream in real-time. One of the major incentives for potential Hermes deployment in the production ISP network infrastructure is the need for BGP prefix hijack detection practically in real time in the very early stage of hijacking. This could be easily accomplished by any Hermes instance that is connected to a major BGPmon peer possessing the streaming access to the aggregated routing feed. Hermes instructed to monitor the stream for a particular prefix originating from a particular source would immediately dump the messages that match the pattern into its underlying storage facility in real time making it immediately accessible to researchers or system personnel for further analysis. This example could be further elaborated towards any types of potential network attack scenarios or statistical estimations of BGP traffic provided that Hermes possesses enough expressive power in its parsing and regular expressions engine to construct the desired case expression.

## 4.2 Start-Up Options

Hermes presently has a limited set of command-line options provided upon start-up that direct its mode of operation. Presently no configuration file is supported since all the required functionality directives could be expressed via arguments to the executable. The following arguments are currently supported:

- -f (mandatory: name for text data file): Indicates the name of the data file where the processed and matching BGP messages should be dumped to
- -h (optional: hostname of the publisher for XML data stream): Indicates the hostname of the XML data publisher to connect to in case of a client (subscriber) mode of operation. If not specified the connection is attempted to the default BGPmon publisher at Colorado State University routing exchange point
- -p (optional: port of the XML data publisher): Indicates the XML publisher port to connect to in case of a client (subscriber) mode of operation when -h is specified

- -s (optional: publisher (server) mode flag): Instructs Hermes to go into publisher (server) mode of operation with the default server port hardcoded to port 50008 to listen for incoming clients (subscribers) connections.
- -l (optional: log flag): instructs Hermes to log its operation into the the standard /var/log/hermes.log location by default. If this option is not specified or permissions to write into /var/log/ are insufficient logging is directed to /dev/null
- -d (optional: name of DSMS data file to write to): Indicates the name of the DSMS data file – instructs Hermes to format each matching message into the DSMS data format and write to a separate disk file
- -e (mandatory: pattern matching expression): Indicates the actual BGP filtering expression that follows in ” ” (double quotes).

## 4.3 Language Specification

Current Hermes implementation employs XML parsing and specific pattern-matching that provides standard logical operator constructs to combine the BGP attributes into a pattern set that is applied to the BGP message packed in the XML format on the fly as it is received from the network. The whole pattern-matching expression starts with the -e (expression) command line option and is enclosed within the double quotes (” ”). The example expression could be modeled as:

-e ”(MULTI_EXIT_DISC <10 & SRC_AS = 6447 & type = MESSAGE) | (ORIGIN = EGP & value = 0)”

The expression could be considered either:

- simple – involving only one subexpression and one type of logical operators
- complex – involving more then one subexpression and two types of logical operators - OR and AND (like the expression above)

The complex expression could be constructed out of a number (two and more) of simple subexpressions connected via a logical | OR operator and a set of parentheses.

Sample tables for simple and complex expressions are shown next.

Table 4.1: Simple Expression

| Example | Description |
|---|---|
| ”expr1” | 1: should NOT have parentheses |
| ”” | 2: empty expr |
| ”type = UPDATE” | 1 tuple |
| ”ORIGIN = IGP” | 1 tuple |
| ”type = UPDATE & SRC_AS = 6447” | 2 tuples |
| ”MULTI_EXIT_DISC = 100 & SRC_PORT = 4321” | 2 tuples |
| ”type = UPDATE \| DST_AS = 3200” | 2 tuples |
| ”type = MESSAGE \| type = STATUS” | 2 tuples |
| ”type = UPDATE \| type = MESSAGE \| type = KEEPALIVE” | 3 tuples |
| ”MULTI_EXIT_DISC = 100 \| SRC_AS = 6447 \| SRC_PORT = 4321” | 3 tuples |

Table 4.2: Complex expression

| Example: LCO is \| | Description |
|---|---|
| ”(subexpr1) LCO (subexpr2)” | 2 subexprs |
| ”(type = STATUS) \| (type = UPDATE)” | 2 subexprs |
| ”(subexpr1) LCO (subexpr2) LCO (subexpr3)” | 3 subexprs |
| ”(type = STATUS) \| (type = UPDATE) \| (type = KEEPALIVE)” | 3 subexprs |

Due to complexity of constructing regular expressions using -e command-line option we should follow the following guidelines:

- simple expression should NOT include ( ) parentheses: otherwise it would not be evaluated properly
- simple expression has only ONE type of logical operator - it is either OR | or AND &
- simple expression could have one or more tuples - see simple expression table
- complex expression must have more then one set of ( ) parentheses - see complex expression table
- | OR operator is the only operator used to chain subexpressions in complex expression involving multiple ( ) parentheses: (e1) | (e2)

- | OR operator could NOT be used inside the ( ) parentheses
- & AND operator could NOT be used to chain subexpressions in complex expression involving multiple ( ) parentheses
- & AND operator must be used inside the ( ) parentheses in complex expressions
- | and & could not be used within a single expression enclosed in ( ) parentheses or within a single simple expression

Here we provide the sample use cases of valid and invalid expressions:

- -e ”type = STATUS | type = UPDATE” - valid simple expression
- -e ”type = STATUS | type = UPDATE | type = KEEPALIVE” - valid simple expression
- -e ”type = UPDATE & SRC_AS = 6447” - valid simple expression
- -e ”type = UPDATE” - valid simple expression
- -e ”(type = STATUS) | (type = UPDATE)” - valid complex expression
- -e ”(type = STATUS) | (type = UPDATE) | (type = KEEPALIVE)” - valid complex expression
- -e ”(type = UPDATE)” - invalid complex expression - only 1 set of ( ) parentheses
- -e ”(type = STATUS | type = UPDATE)” - invalid complex expression - only 1 set of ( ) parentheses
- -e ”(type = UPDATE & SRC_AS = 6447)” - invalid complex expression - only 1 set of ( ) parentheses
- -e ”(type = UPDATE) & (SRC_AS = 6447)” - invalid complex expression - & should not be used to chain the ( ) parentheses

- -e ”(type = STATUS) | (type = UPDATE & MULTI_EXIT_DISC = 100) | (MULTI_EXIT_DISC = 10) | (type = KEEPALIVE & DST_AS >3200)” - valid complex expression
- -e ”(type = STATUS) | (type = UPDATE & MULTI_EXIT_DISC = 100 & SRC_AS = 6447 & SRC_PORT = 4321 & ORIGIN = EGP) | (MULTI_EXIT_DISC = 10 & SRC_AS = 6447 & SRC_PORT = 4321 & ORIGIN = EGP) | (type = KEEPALIVE & DST_AS >3200)” - valid complex expression

To summarize the following two points should be taken into consideration when constructing complex expression:

- | operator is used OUTSIDE of ( ) parentheses: ”(exp1) | (exp2) | (exp3)”
- & operator is used INSIDE of ( ) parentheses: ”(tuple1 & tuple2) | (tuple3 & tuple4 & tuple5) | (tuple6 & tuple7)”

The Hermes Language operators are presented in the Table 4.3 with sample use cases.

Table 4.3: Hermes Language operators

| Operator | Description | Example use |
|---|---|---|
| = | equality operator | ORIGIN = EGP |
| ! | not-equal operator | SRC_AS ! 6447 |
| < | relational less than operator | MULTI_EXIT_DISC <10 |
| > | relational greater than operator | MULTI_EXIT_DISC >10 |
| & | logical AND | ORIGIN = EGP & value = 0 |
| \| | logical OR | ORIGIN = EGP \| value = 1 |
| () | parentheses | (ORIGIN = EGP & value = 0) \| (type = MESSAGE) |

Parentheses are used for complex expression handling separating subexpressions within a single complex expression as shown in the last column of the table. The () parentheses are chained through logical OR operator.

Hermes provides a set of operators designed specifically for processing network prefixes including CIDR ranges. The operators follow the PREFIX attribute with the subsequent network range value. These are ordinary English letters having special meaning when used within the expression.

Table 4.4: Prefix operators

| Operator | Description | Example use | Semantics |
|---|---|---|---|
| e | exact prefix match operator | PREFIX e 211.64.0.0/8 | matching the networks with the exactly defined network prefix range. |
| l | less specific prefix match operator | PREFIX l 211.64.0.0/8 | matching the networks with less specific network prefix range. |
| m | more specific prefix match operator | PREFIX m 211.64.0.0/8 | matching the networks with more specific network prefix range. |

Equality and negations operators (= and !) could be used in expressions involving both string and integer values and change the semantics of operation depending on operand type.

## 4.4 Implementation Details

Hermes BGP broker is implemented in pure C due to the need for adequate flexibility and efficiency in handling massive data streams and is code-compatible with BGPmon developed in C as well. The backend engine behind its current XML parsing and filtering capability lies in the deployment of Libxml2 library which is the XML C parser and toolkit developed for the Gnome project. The library is specifically targeted to work in C environments and provides extensible collection of XML manipulation functions. The incoming XML stream is transformed into the tree where each leaf node represents a particular BGP attribute of the BGP Message packet thus making it very easy to locate the value of any BGP attribute through recursive tree traversal. Current Hermes development code is multithreaded due to the requirement of supporting multiple clients connecting to the instance simultaneously for data retrieval. Multithreading is implemented through the standard PThreads library facilities.

Due to the number of constantly connected client instances Hermes utilizes the existing BGPmon queue management mechanism to provide adaptable data streaming policy per client. As for now Hermes uses a single circular Filtered Queue that is populated with incoming matched BGP messages from the BGPmon peer instance. The current model is based on the concept of one writer and many readers thus employing a variant of publish/subscribe mechanism used in BGPmon as well. Writer is the main Hermes thread that receives continuous stream of route data and performs pattern matching mechanism to filter designated data. Data (BGP message in the XML format) matching the pattern is inserted into the above mentioned Filtered Queue. Readers are the threads that are dispatched on demand upon the incoming client connection and read the Queue thus removing the read entries and performing Queue emptying (consumption). Due to the code base imported from BGPmon the Queue management mechanism in Hermes is capable of adjusting slow performing readers using the concept of ideal reader offering a way to efficiently control queue capacity without overflowing it with non-acquired data entries.

The BGPmon queue module imported into Hermes is build on a circular array and each item in this array contains a generic pointer which points to the real message. This makes the queue module generic enough to hold any types of messages. It also keeps track of all the readers and writers. The queue module implements a Readers/Writers pattern where multiple threads may access the same queue simultaneously, some reading and some writing. Each message written by a writer is available to all the readers and a message can be deleted from the queue only after all the readers have read it [YOB+09]. A general layout of the BGPmon queue structure imported into Hermes is provided in Figure 4.1.

Another module that has been imported from BGPmon is the Clients control module that is used to manage the Hermes clients. The initial Hermes design has its own thread pool with a limited number of Reader threads (64 by default) but since the Queue module is well integrated with the Clients control module it was decided to bring both of these modules from BGPmon into Hermes to preserve the consistency. Clients control module consists of a single server thread and multiple client threads - the actual Readers. In Hermes the integrated Clients control thread listens on TCP port 50008 (hardcoded in site_defaults.h) and spawns one client thread for each connecting client. Each client thread reads the messages from the Filtered Queue and sends them to the client via a TCP connection.

It has been observed that the BGPmon meshes used as the source for the BGP routing data could possibly form a loop delivering the same BGP message twice to the receiver. In order to detect

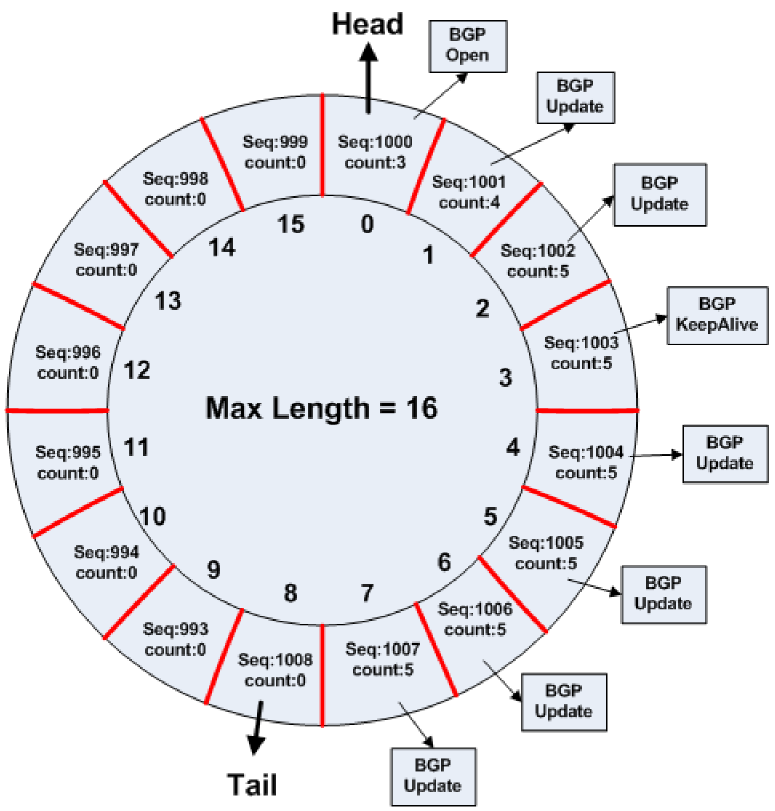

Figure 4.1: An Example of a Queue

such cases and remove duplicate messages a special loop detection mechanism entitled Duplicate Suppression has been integrated into Hermes. The mechanism uses the Duplicate Suppression cache that is based upon the simple open-addressed hash table (HT) that hashes the combination of specific tuples within each BGP message:

- the attributes of BGPmon_SEQ tag comprising id - unique BGPmon creator ID, seq_num - sequence number of the message
- two additional attributes from the TIME tag element: timestamp and datetime of the message

The hash is constructed out of these four attributes and inserted into the corresponding slot in the Duplicate Suppression hash table cache. When a new BGP message arrives at the Writer (the main thread of control) it extracts the combination of these tuples and performs the insertion into the cache. In case of a collision the collision is recorded and the incoming BGP message is discarded if collision factor reaches the predefined threshold - it does not enter the Filtered

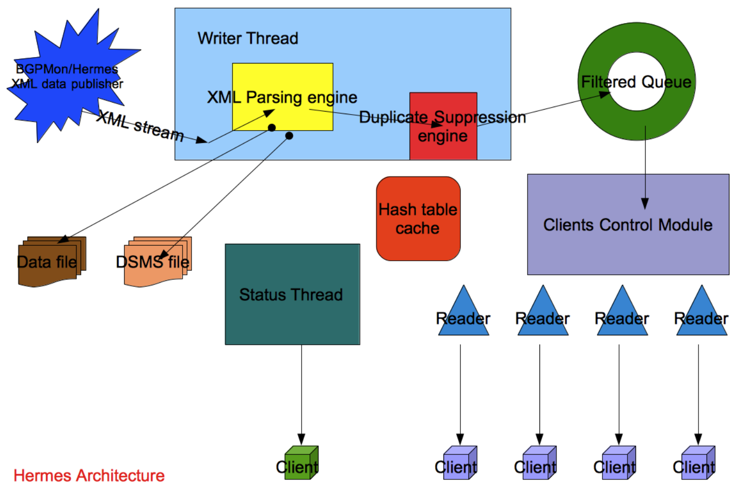

Figure 4.2: Hermes architecture

Queue to be read by the Reader threads. When the collision factor reaches a predefined threshold (set to a fraction of the cache capacity) the cache is cleared and Duplicate Suppression mechanism restarts. The integration of hash cache allows fast and efficient lookup to be performed for detecting the duplicates in BGP message delivery and prevents clients from receiving the looping routing information. Although the technique does not provide full loop avoidance solution in constant time and enables duplicate detection only until the threshold is reached it still does provide an effective way to eliminate duplicate messages in the short term. Duplicate Suppression is handled by the Writer Thread in main.

A separate Status Thread in Hermes enables the retrieval of some general information about the running Hermes instance. The Status Thread listens on port 50009 and dumps the following info about the instance to the telnet connection:

- total number of messages processed
- number of messages matching the pattern
- ratio of above
- uptime info

- hash collision value
- current number of items in the Queue
- number of connected clients
- clients' IP addresses

## 4.5 Client/Server Architecture

Client-server model is the key factor of the Hermes functionality and its ability to process the routing data. As indicated above Hermes is multithreaded with main data collection and pattern matching Writer Thread and a number of Reader Threads called on demand upon the incoming connection from a client. It employs the threading architecture similar to the BGPmon one although presently much simpler in terms of thread layering. Present development model assumes the need for one Writer due to the fact that current Hermes instance is connecting to exactly one BGPmon peer instance (or a peer Hermes instance) that serves as the sole data provider aggregating a massive data set from a large network of stratum one route data providers or peer BGPmon meshes hidden from Hermes instance. Thus one BGPmon peer instance provides the necessary aggregated data stream for a single Hermes instance to operate on.

The unique feature of every Hermes instance is its ability to instantiate itself in one of two possible operational modes: either in the server mode or in the client mode depending on the specified command-line option flag. In fact Hermes is even able to operate in dual-mode being both a client and the server all at once. The server mode of operation launches the Hermes instance as a server listening for incoming client connections to arrive on the specified port. The client mode of operation forces Hermes to become a client connecting to the specified Hermes server instance (or BGPmon) to receive route data stream. Each mode assumes that specific data pattern construct should be specified at the command-line to perform incoming stream matching.

Since each Hermes instance is designed to connect to a single BGPmon peer then no direct way of aggregating two or more BGPmon streams into a single Hermes exists in the current implementation. Nevertheless the interested operator has the option of running the local BGPmon instance

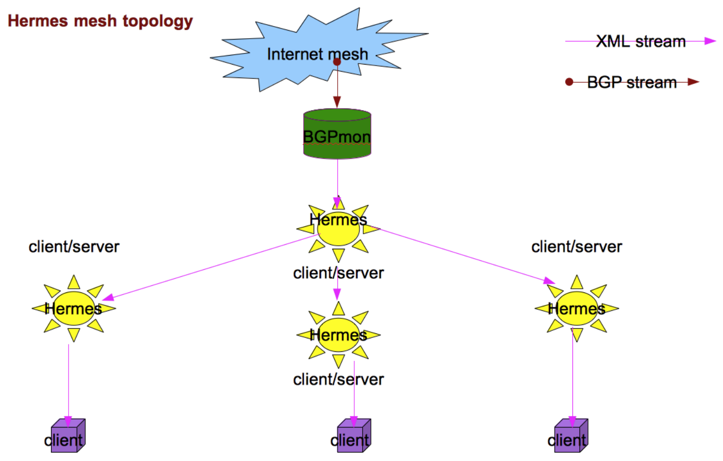

Figure 4.3: Hermes mesh topology

that plays the role of a frontend and connects to a number of other BGPmon peers and aggregates their traffic in a single XML stream that would be subsequently delivered to the direct Hermes peer. This saves Hermes from complex multi-stream aggregation (for now) involving duplicates identification and depicts interesting aggregation model that could be potentially used by operators seeking to diversify the BGP updates sources for further filtering and analysis.

## 4.6 Moving To Distributed Computations And Delivery

The chaining of server and client instances of Hermes allows potential deployment of the overlay Hermes network which nodes are piping the refined data stream over the Internet to other peer nodes for local consumption or further forwarding to the interested subscriber thus implementing a network pipe mechanism analogous to a UNIX local pipe scheme. The network pipe mechanism would make it possible to distribute the computation of the routing analysis task among a collection of geographically dispersed Hermes instances where XML route data is refined and filtered en route as it traverses a chain of data refinement points. The distributed mechanism would be possible in case of a Hermes instance operating in a dual mode of operation being simultaneously a server serving arriving requests for data from client Hermes nodes and a client to other Hermes instance or BGPmon peer that receives the aggregated base data stream. It is worth noting that the described model is not mandatory one allowing local data filtering and consumption at any client Hermes node without further forwarding along the hypothetical chain. The chain could actually be modeled as the tree (or a graph to some extent) with root element being the BGPmon instance as the base data provider and further elements being Hermes nodes operating either in server or client mode or both.

The sample network pipe mechanism is provided in the figure below. Here the base Hermes instance at the root of the tree acts as the main provider of aggregated routing data for the Hermes leaf nodes down below. The root Hermes is filtering its incoming stream from BGPmon peer and outputs to its clients only the BGP messages matching the regular expressions pattern of "(MULTI_EXIT_DISC <10 & SRC_AS = 6447) | (ORIGIN = EGP)". Other hermes refinery points receiving this uniform stream then conduct further refinery based on their individual policies. For example one of them is carrying out the policy of locating messages that match the pattern of "PREFIX m 211.64.0.0/8 & ORIGIN ! IGP & SRS_AS = 6447". This captured data is inserted into the Filtered Queue and made accessible for further manipulation by another Hermes instance further away in the chain. This one is then adhering to a very simple policy of capturing any routing messages that exactly match the network in the range of 211.64.10.0/24.

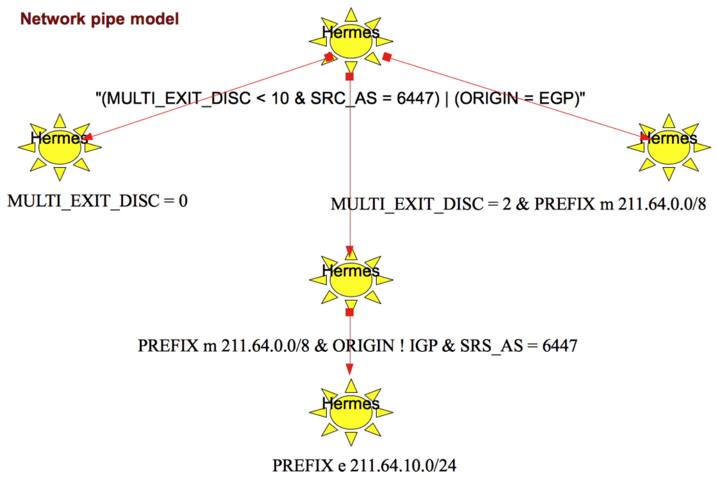
Network pipe model
Hermes
"(MULTI_EXIT_DISC < 10 & SRC_AS = 6447) | (ORIGIN = EGP)"
Hermes
Hermes
MULTI_EXIT_DISC = 0
MULTI_EXIT_DISC = 2 & PREFIX m 211.64.0.0/8
Hermes
PREFIX m 211.64.0.0/8 & ORIGIN ! IGP & SRS_AS = 6447
Hermes
PREFIX e 211.64.10.0/24

Figure 4.4: Network pipe model

The network pipe model theoretically allows the facility of simulating nested subexpressions within the expression through the distribution of the subexpressions on different Hermes nodes. This could be a useful feature since nested subexpressions are not currently supported in Hermes due to the potential complexity of evaluation order. This would allow for instance to split the global expression ((expression1) | (expression2 | ((expression3) | (expression4)))) into several computations performed separately on different nodes and then process the collected data. At the same time this approach might not be a feasible solution and might probably be not even required since most of the expressions for reasonably well-defined data collection matches could be specified using ordinary expressions enclosed in one level parentheses connected with logical OR. In fact nested subexpressions avoidance reduces the possibility of constructing ambiguous and hard to follow pattern-matching expressions but theoretically still could be constructed via Hermes chaining if so desired.

# Chapter 5

# Data Streams

This chapter presents the necessary background and provides motivation to move to new data transmission facilities – data streams. It has been outlined in the previous chapters that Hermes as a route filtering and aggregation instrument could serve as a potential middleware broker between higher level networking applications and the raw XML update stream. It has been observed that most of the networking applications residing at the higher level of operational logic do not currently support direct access to the massive streams of routing updates and often do not even have properly engineered built-in support for intensive network IO operations. These types of applications might serve as the direct clients (subscribers) for the Hermes route aggregation and filtering facility for retrieving desired datasets.

Some types of applications of this kind do actually serve as direct complements for the Hermes route distribution facility. Specifically there is an emerging set of networking applications called Data Stream Management Systems (DSMS) that might serve as a direct extension for the Hermes storage facility. The difference between a file system or Database Management System (DBMS) and a DSMS is simple: current file systems or DBMSs expect all data to be managed within some form of persistent data set; in a DSMS, the concept of a data stream, possibly unbounded, is as important as a conventional stored data set. By nature, a stored data set is appropriate when significant portions of the data are queried again and again, and updates are small and/or relatively infrequent. In contrast, a data stream is appropriate when the data is changing constantly (often exclusively through insertions of new elements), and it is either unnecessary or impractical to operate on large portions of the data multiple times [BW01] [CBB$^{+}$03].

DSMS systems (most notably Stanford STREAM Project [ABB$^+$04], Berkeley Telegraph [UCB] and Aurora Project [Uni]) have been designed to overcome some of the shortcomings of traditional database management systems that are best equipped to run one-time queries over finite stored data sets. However, many modern applications most notably network monitoring and sensor networks generally require long-running continuous queries over continuous unbounded streams of data [BW01]. These types of environments including the routing updates stream processed by the Hermes BGP broker mainly exhibit properties where streams may be rapid, stream characteristics and query loads may vary over time and system resources may be limited. In fact since Hermes is directly providing access to the publish-subscribe facility of BGPmon meshes we can map a pub-sub system to a DSMS by considering publications as streams and subscriptions as continuous queries [ABB$^+$04].

The definition of a query in this context should be presented as the execution of database-level operational logical constructs in the form of SQL statements - Sequential Query Language – a relational query language used in the Database Systems to manipulate the data records. Since we are focusing on the data streams carried over the network the subsequent uses of query executions further in the text are referring to the extension of SQL over data streams called Continuous Query Language (CQL)  the relational query language based on SQL but developed specifically for manipulating the continuous nature of data streams as append-only relations [ABW06]. Syntactically CQL is a relatively minor extension to SQL. The more elaborate description of CQL and the architecture of DSMS systems is beyond the scope of this text mainly due to the fact that we are dealing only with possibility of transforming routing updates dumped by Hermes into the DSMS appropriate format and further processing of the data sets is highly dependent upon individual DSMS implementation and middleware application in use.

We claim that integration of Hermes and SDBMS might provide much more effective utilization of the routing updates processed by Hermes and could in fact serve as the underlying basis for the support of so called interdomain routing streams [GM03]. Interdomain routing streams such as BGP sessions that are transformed into XML messages via BGPmon facility could be thought of as append-only relations once timestamps are added to each message in a stream. Many queries useful in BGP route monitoring can then be expressed as continuous queries over this data stream.

Since most of the underlying datasets obtained by Hermes from BGPmon meshes are collected from Oregon RouteViews and RIPE RIS repositories it is worth noting that typically the RouteViews alone collects from five to six million updates per day. This is approximately the same order of magnitude of updates that might be collected inside a large ISP with BGP streams arriving from 25 large metropolitan regions [GM03]. Since networks need to be running all the time much of this data is collected continuously on different time scales and results in very large and fast-growing databases. For example packet traces collected in the Sprint IP backbone alone amount to 600 Gigabytes of data per day [BSW01].

The announcements and withdrawals of prefixes, together with AS PATH BGP attributes and message timestamps could be thought of as an append-only relation and many queries useful in BGP monitoring can then be expressed as continuous queries over this data stream. The BGP table of every router contains best routes to all of the destinations it has learned. Given a continuous BGP stream, starting at some time t in the past, sent from neighboring router R, we can construct that portion of Rs BGP table that has changed since time t. If a BGP session reset has occurred since t, then we can reconstruct the entire BGP table (since a router must send its entire table when a session is reestablished after a reset). This table can be thought of as a materialized view of the BGP update stream. An example of one simple query over this view could be the number of routes in the table [GM03].

For the sake of completeness it is worth stating that data streaming approach to BGP routing streams could be directly applied to all kinds of ISP network operations as well. Data collected to enable traffic management applications in an ISP includes: network packet and flow traces, active measurements of packet delay, loss and throughput, Simple Network Management Protocol (SNMP) data maintained by various network elements (e.g., routers, switches). Different kinds of processing must be performed on the collected data to enable the deployment of sophisticated traffic management applications. The network topology is maintained by joining SNMP data and/or configuration data from different network elements. Statistics of link and router utilization are maintained by aggregating packet traces or from SNMP data. Packet losses, per-hop and end-to-end delays, and network throughput are measured either by joining packet traces collected from multiple points in the network, or by using a dedicated system that generates network traffic to measure these parameters online. Due to the sheer volume of data and complex processing, most current network traffic management applications process the collected data offline. The data is typically loaded into a centralized file system or data warehouse and processed by specialized toolkits [BSW01]. It follows the same pattern with Hermes as well where routing data is currently dumped into the file system storage.

It is well understood that enabling online data processing for tracking changes in network topology and traffic distribution would enable congestion cause detection, adaptive intra-domain and inter-domain routing policies, resource allocation mechanisms for guaranteed application-level quality of service (QoS), admission-control and traffic-policing, and even detecting denial-of-service attacks. And more specifically the integration of DSMS with Hermes would make BGP data available to this set of higher level networking applications for further processing and analysis.

## 5.1 Architecture For Hermes And Stanford STREAM DSMS

We have chosen to integrate Hermes with Stanford STREAM DSMS [ABB$^+$04] for a number of reasons. First of all STREAM Project represents a general-purpose DSMS system targeted towards basic streaming needs and thus is mainly well suited for operating upon routing updates. Secondly we have committed significant modifications to the experimental code base to enable STREAM to process and provide streaming data to a number of subscribers which was not the case with the vanilla distribution that has been limited to self-looping only. Thirdly STREAM already has a Java Stream client component application designed for querying and delivering data upstream to the actual DSMS system which removes the burden of developing additional application dedicated for the Hermes to DSMS streaming channel establishment.

The provision of BGP dataset from Hermes to DSMS is utilized through the dsms conversion routine in Hermes that transforms the XML encoded BGP message into the tuple-oriented text input file suitable for Java DSMS client application. Matching messages are written into the DSMS input file that in turn is repeatedly read by the Java Stream client that streams the data in the converted format directly to the central DSMS server for further storage and access by other Java Stream clients. Therefore the integration of the conversion mechanism inside the Hermes machinery itself with the premise that it already performs XML processing eliminates the need for an external conversion application to prepare the dsms data for data mining.

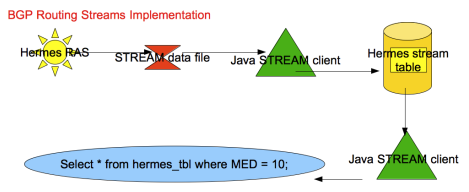

Figure 5.1: BGP routing streams

The choice of placing the DSMS either on the same node in the network where the Hermes is residing or forcing Java Stream client to deliver the data to the remote DSMS node in the network is highly dependent upon the intended use model. Nevertheless it is highly recommended to place the DSMS server (such as Stanford STREAM) on the same node with the operational Hermes installation. With this model the original Java Stream client would be residing on the same node with Hermes and DSMS establishing local (localhost) connectivity to the DSMS not involving the network. This would be a relatively reliable way of ensuring the guaranteed delivery of original data from Hermes text file facility to the actual DSMS table buffers. The motivation behind this approach lies in the fact that both Java Stream Client and DSMS itself (in the case of Stanford STREAM implementation) are not particularly robust in terms of intensive network IO operations. Any network anomalies such as congestion or QoS policies might severely affect the delivery of data to the central DSMS repository table with an assumption that nodes are separated by a number of intermediary routers or switching devices. At the same time if nodes are connected by a cross-cable connection DSMS and Hermes could reside on different nodes since the probability of cross-node congestion or related failures is minimal provided that both nodes are dedicated to running the designated applications only.

Other Java Stream clients could be optionally distributed all over the network and access the dynamically changing BGP routing table at the central DSMS server by expressing the CQL queries over the network. Thus we can summarize that original Hermes updates file would be streamed through the Java Stream client facilities to the RAS stream/relation on the central DSMS server that makes data globally accessible to other users of the DSMS system. This would allow the CQL machinery to be applied to the actual real-time BGP route data bringing flexibility of existing business logic applications to the routing domain.

For example we could easily query and select specific aggregate information regarding individual BGP attributes such as MED, AS PATH, ORIGIN and utilize SUM, AVG, MEAN or related SQL functions on the designated attributes if needed provided that DSMS has the necessary support for these types of relational operators applied to streams [ABB+04] [BW01].

## 5.2 Total Overall Architecture: BGPmon + Hermes + DSMS

The total overall architectural model of deployment for the proposed integration of Hermes with DSMS facilities does not directly affect the role of BGPmon in the envisaged data distribution topology. BGPmon would remain the primary source of raw XML routing input without any realization of what would be happening to the streamed updates. At the same time BGPmon meshes would still play a crucial role in accumulating and provisioning of routing information to the Hermes refinery points where data would be filtered according to individual site policies and needs and then optionally converted to the DSMS readable format for the delivery to the higher level logic networking applications.

The deployment architecture is shown in the figure where actual BGP routers on the Internet are providing BGP updates to a number of BGPmon meshes that aggregate them and make available for a single Hermes instance to start its analysis and filtering. The filtered data is then converted into the DSMS-readable format that is streamed into the DSMS stream table on the DSMS server for further real-time retrieval by CQL-enabled clients. The example shows a simple CQL query that is being performed on the MED BGP attribute to select all BGP messages with MED = 10.

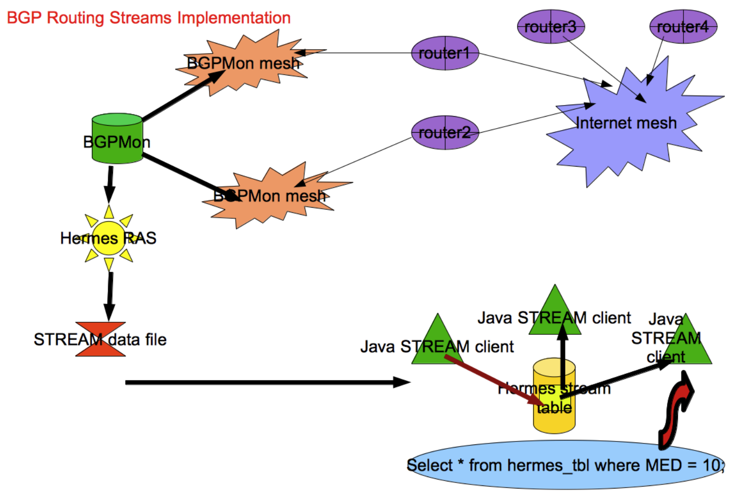

Figure 5.2: BGP routing streams - full topology

The proposed model would directly benefit not only ISPs that need the real-time easily accessible data but also the Internet research community that would be empowered with flexible way of conducting the analysis of rapidly changing Internet topology and making long-term assessments of fluctuating behavior of the Internet.

# Chapter 6

# Evaluation

This chapter provides the evaluation data gathered from performing various benchmark and functionality testing of the Hermes BGP Broker. We are testing the following aspects of the running system:

- message processing speed
- pattern matching functionality
- pattern matching functionality integration with DSMS
- network pipe model of distributed stream refinery

Message processing speed is chosen as a performance measuring factor which is the crucial indicator of Hermes ability to receive a vast XML stream and be able to provide it to the subscribers at maximum speed. We anticipate that this message processing speed would play a crucial role in the environment where a single Hermes instance would be serving the needs of a large number of subscribers with the requirement of providing the maximum message processing speed to avoid client starving scenario.

Pattern matching functionality is at the core of the Hermes BGP Broker and is essentially the most important aspect of the system. Therefore pattern matching on the live XML stream together with DSMS integration testing and network piping of the XML stream among the distributed nodes are the major factors that show the actual usefulness of the system to the end user.

## 6.1 Methodology

To test the message processing performance of Hermes we decided to bypass the network latency and send the XML stream at machine local IO speed between two endpoints residing on the same machine. This ensured that the socket interface would not be overloaded by external factors such as network performance etc. A special pipe server was written in C that reads the BGP messages from the file and sends them to the connected Hermes at the machine IO speed. A 1088 MB BGP messages file with 521 755 BGP messages of varying size and type was used for performing the load benchmark test. The Hermes connected to the Pipe server was essentially recreating the same file used by the Pipe server for streaming. Both executables have been running on the same machine. The total processing time has been calculated based on the difference between initial file write time and last file write time expressed in minutes.

We have conducted the benchmarking tests on CSU CS lab machines with random load executing only Pipe server and Hermes client/server on our behalf.

We have tested the hermes performance under the various subscribers loads varying from 0 to 64 with step size of 8 clients. The rationale behind 8 client granularity lies in the observation that trying to add lesser number of clients per shift would not impact the Hermes performance at a level significant enough to be noticeable in the tests.

## 6.2 Results

We have run the following command executables:

- hermesPipe - listens on port 5000 and plays bgpdata.txt file to the hermes client at local IO speed
- hermesd -s -h yam -p 5000 -f out.txt -e ”” - connects to hermesPipe server, receives XML stream and writes it to the out.txt file with no specific pattern matching directive ( -e ”” replication mode). -s option puts hermesd into server mode to serve subscribers

- hermesd -h yam -p 50008 -f /dev/null -e ”” - connects to hermesd running on host yam at port 50008 and subscribes to the XML stream

The tests have been conducted on the CSU CS lab HP machines with 8 core Intel(R) Xeon(R) CPU E5450 @ 3.00GHz with 16GB of RAM. The clients were hermesd instances running on separate machine(s) in the same network with average RTT during benchmarking load staying around 25 ms .

The results shown in figures 6.1 and 6.2 produce nearly linear graphs without any spikes that stay constant regardless of a number of connected hermes client instances tested up to 64 stream subscribers:

- with number of clients ranging from 0 to 64 the reading speed stays nearly constant at about 805 messages/sec
- total processing time is constant staying at about 11 minutes (rounded) for 521 755 (1088 MB) BGP messages processed

At the same time we should note that the memory consumption of the heavily loaded hermes server instance could be quite large reaching up to 780 MB of RAM on the test instance serving 64 hermes clients. CPU utilization for HermesPipe – Hermes chain ranged from 35 to 55 percents during every round of benchmarking.

Based on these load benchmarks we could clearly see that the performance of Hermes is highly dependent upon the underlying hardware and OS limits and load from other CPU intensive jobs. More then that based on the benchmark plots we could conclude that the performance is mostly constrained by the OS and hardware limits with Hermes not imposing any significant processing delays due to its internal architecture properties.

It is very important to state the fact that the major speed limiting factor during the benchmarking was the need to write the received messages to the file system essentially creating exact copy of the 1088 MB data file read by the hermesPipe test server. Because of the IO limitations the reading speed was constrained within the IO write abilities of the machine since we have been reading and writing each received message back to disk. When the actual pattern was specified in the –e option

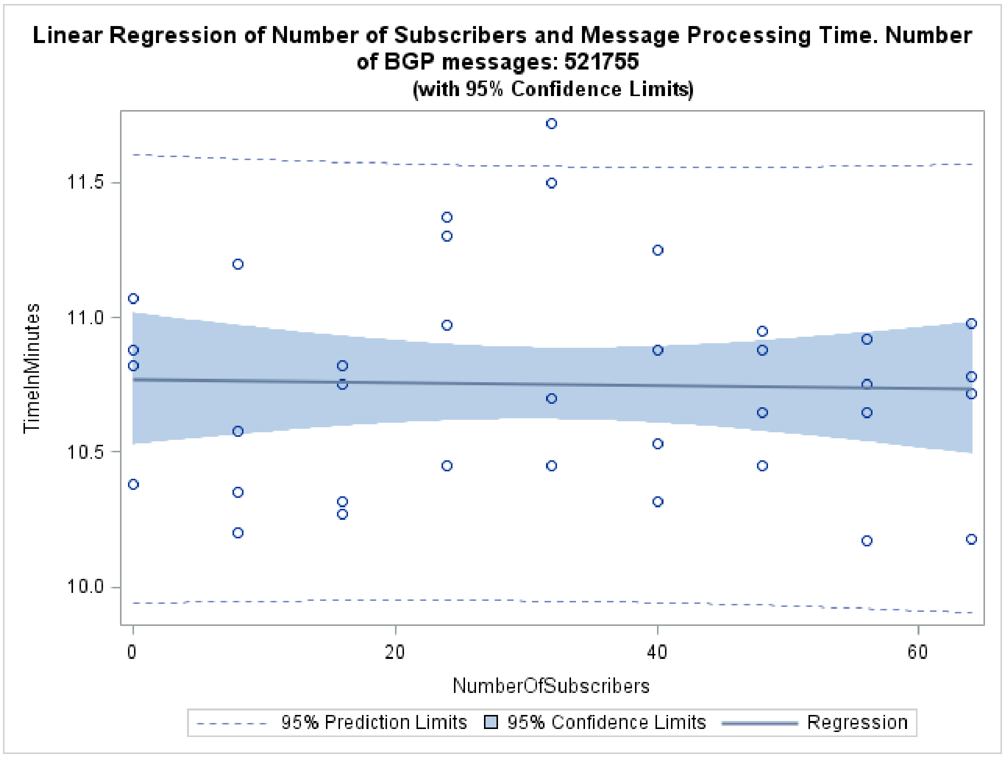

Figure 6.1: Hermes total message processing time statistics.

to limit the writing of only specific messages the reading speed from the actual XML stream went dramatically up reaching on average about 1740 messages/second with average total processing time of 5 minutes (rounded) per 521755 messages. (we have tested with -e ”MULTI_EXIT_DISC = 100 & SRC_AS > 6440” pattern that actually did not match any message in the stream thus removing the disk write IO penalty from benchmarking). It is also worth to state that the CPU load went from 45 percent during original tests with full file writing up to 99 percent with pattern matching turned on showing two fold increase in processing utilization. This supports the argument that intensive IO operations per each BGP message received in original tests were not allowing Hermes to increase CPU utilization over 50 percent threshold while drop in IO during the pattern matching test allowed it to utilize CPU core up to 100 percent and therefore double the message processing speed.

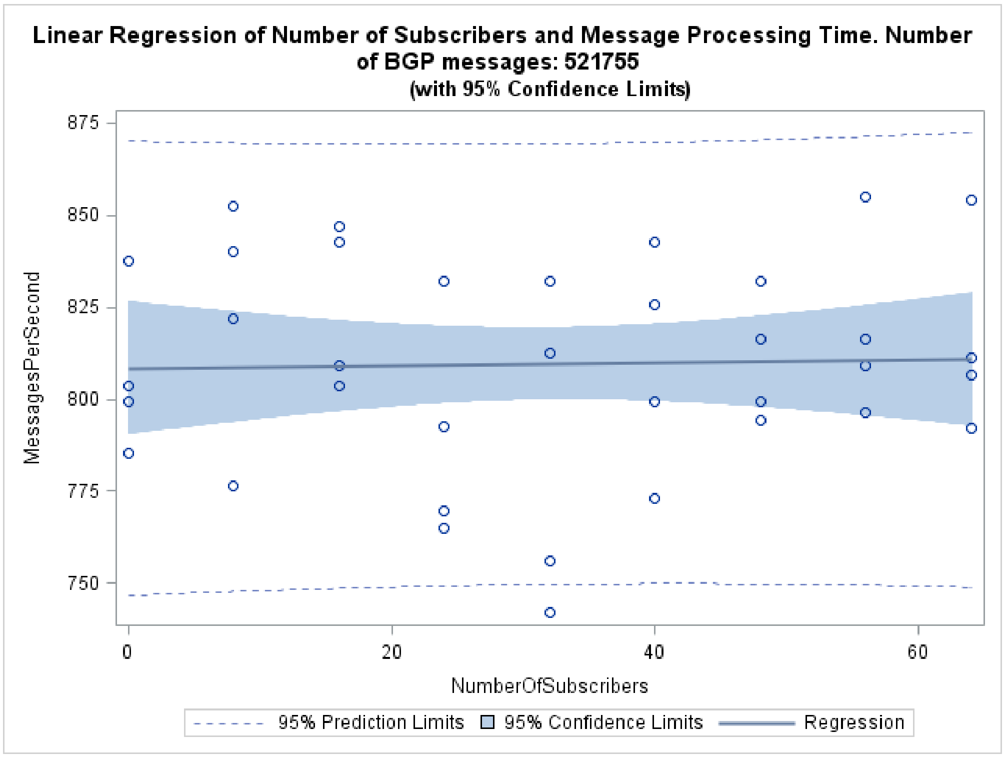

Figure 6.2: Hermes total message processing speed statistics.

## 6.3 Prefix Analysis

As part of the pattern matching functionality we have conducted tests of three different prefixes belonging to three different organizations to assess the percentage of BGP updates announced with these network prefixes. The data has been reported via the Hermes Status Thread.

Table 6.1: PREFIX announcement ratio

| Operator | PREFIX | Matching Messages | Percentage Ratio (BGP messages analyzed (rounded) per single PREFIX test: 530 000) |
|---|---|---|---|
| AT&T Bell Laboratories | 12.0.0.0/8 | 38 | 0.007148 |
| Level 3 Communications, Inc | 8.0.0.0/8 | 27 | 0.005060 |
| RIPE NCC | 195.0.0.0/8 | 396 | 0.073030 |

The following expressions have been used to make the experiments:

- type = UPDATE & PREFIX m 12.0.0.0/8

- type = UPDATE & PREFIX m 8.0.0.0/8
- type = UPDATE & PREFIX m 195.0.0.0/8

Based on these tests we could clearly see that at the time of experiment the RIPE NCC has at least a ratio of 10/1 in number of updates generated in comparison to AT&T and Level 3 traffic.

## 6.4 DSMS Queries And Results

Hermes and Stanford STREAM DSMS integration has been tested with the -d option instructing Hermes to extract several elements from each BGP message and write the tuples into the DSMS file one row per message (each row consisting of tuples - elements) . The file has been read by the separate Java DSMS client and rows streamed to the Stanford STREAM DSMS server into the registered stream table. We have then registered a number of CQL queries on this stream to test the querying functionality on real BGP data. We have instructed Hermes to select only Update messages from the XML stream. The following elements have been extracted to form a row:

- SRC_ADDR, SRC_AS, DST_ADDR, DST_AS, ORIGIN, MULTI_EXIT_DISC, PREFIX, OCTETS

A sample row generated with -d option has the following format with each tuple separated by comma:

- 128.223.51.102,6447,208.51.134.246,3549,IGP,2504,195.95.224.0/23,FFFFFFFFFFFFFFFFFFFFFFFF

Each row has a terminating OCTETS tuple to signify the end of BGP message (the end of row). The following stream schema has been used to create the stream table at the DSMS server:

- (SRC_ADDR char(32), SRC_AS (int), DST_ADDR char(32), DST_AS (int), ORIGIN char(32), MULTI_EXIT_DISC (int), PREFIX char(32), OCTETS char(32) )

We have issued the following CQL queries on the contents of the hermes stream table executed from the remote Java client interface residing on a separate machine on the network with various tuple streaming rates (from 1 tuple/second to 10 000 tuples/sec):

- select * from hermes;
- select * from hermes where med >1000;
- select * from hermes where med = 0;
- select * from hermes where med = 100;
- select * from hermes where origin = ”INCOMPLETE”;
- select * from hermes where origin = ”IGP”;
- select * from hermes where prefix = ”195.%”;
- select * from hermes where origin = ”IGP” and prefix = ”12.%”;
- select * from hermes where med > 0 and origin = ”IGP”;

The queries have been running continuously for several hours producing the expected output. As already mentioned the queries have been registered on the DSMS Java client running on a machine different from the one where the main DSMS server was residing. Thus the output of the queries has been delivered from DSMS STREAM server table buffers over the network to the end client signifying the success of enabling the delivery of real-time BGP message content from BGP route collectors to BGPmon, Hermes and finally to the level of user-level application logic.

## 6.5 Network Pipe Results

We have also conducted the general test of the network pipe model where a chain of hermes servers is filtering the XML stream in the distributed manner. We aimed to refine the stream starting from the generally wide refining criterion with most of the messages remaining in refined stream all the way to the relatively narrow range of messages where the stream is consisting of a very few messages. Three servers have been chained together in the following fashion:

- hermesd -s -f bgpdata.txt -d dsms.dat -e ”type = UPDATE”: - the front server (running on host yam) is connected to the original BGPmon XML stream and selects only UPDATE messages

- hermesd -s -h yam -p 50008 -f bgpdata1.txt -e ”MULTI_EXIT_DISC = 0”: - the middle server (running on host tomato) is connected to front server (on host yam) and out of all UPDATE messages selects only messages with MULTI_EXIT_DISC assigned to 0

- hermesd -s -h tomato -p 50008 -f bgpdata2.txt -e ”PREFIX m 195.0.0.0/8”: - the last server in the chain (running on host carrot) is connected to middle server (on host tomato) and out of all UPDATE messages with MULTI_EXIT_DISC assigned to 0 selects only messages with PREFIX having more specific network addresses in the 195.0.0.0/8 network range

Each server is instructed to save the matching messages in the file so that actual verification of refining validity could be conducted on each server separately. The sample results (obtained via a Status Thread on each server) are indicated in the table below with each column representing the ratio of matching messages to the amount of all messages received at each particular server:

Table 6.2: network pipe

| UPDATE messages ratio | MULTI_EXIT_DISC = 0 ratio | PREFIX m 195.0.0.0/8 ratio |
|---|---|---|
| 48.129086 | 11.975453 | 2.631579 |

Based on this data we could see that the amount of BGP UPDATE messages in the original XML stream received from the BGPmon main server constitutes roughly 48 percents out of the total traffic. From these UPDATE messages only roughly 12 percent have MED equal to 0. And finally out of this 12 percent only approximately 2.5 percent of messages have network range of 195.0.0.0/8 in their PREFIX announcements. This final refined stream of messages was only 38 messages wide (data obtained from the Status Thread on the last server) representing only a negligibly tiny fraction of the total traffic (33639 messages received from BGPmon at the front server) at the time of testing. The analysis of each server data file showed exact match of the specified pattern and the corresponding contents of BGP messages in each file. There were no messages detected in the individual files that did not satisfy the refining pattern specified at each respective server instance.

# Chapter 7

# Conclusion

In this thesis we have presented Hermes - a scalable real-time BGP Broker that is capable of performing extensive analysis of the XML routing stream. Hermes being the next generation of route analysis platforms builds upon the achievements of the existing BGPmon combining infrastructure and carries it well beyond simple accumulation of BGP data.

Thesis has provided the detailed coverage of the Hermes BGP Broker as a new stratum of service layer between BGPmon and the end-point clients. Start-up options and language specification have been explicitly covered outlining the proper way of constructing regular expressions for the stream filtering. The implementation details have been described and internal architecture exposed. We have also provided the coverage of the client-server architecture and ability of Hermes to scale in the distributed environment employing the network pipe model.

Besides this thesis touched upon the emerging field of data streams and Data Stream Management Systems (DSMS) and the integration of Hermes with the DSMS platforms. Proposals such as interdomain routing streams have been discussed and actual architecture for Hermes and Stanford STREAM DSMS has been laid out.

We have provided extensive evaluation of the actual Hermes functionality and performance statistics. Thesis presented the results of actual real-world data streaming tests where the end user was able to receive BGP data in real-time via expressing CQL queries to the DSMS platform. We demonstrated the working model of integrating Hermes with DSMS and constructed the operational data delivery model where real BGP updates are delivered from actual routers all the way to the end-user applications via Hermes adaptation layer and data streams integration. In regard to network pipe we showed that network pipe model using Hermes nodes does work successfully.

## 7.1 Test stream generation

A proportion of Hermes users could be interested in testing the desired pattern on local data before connecting to a live stream. The testing of the validity for the filtering expression that is presented to Hermes is a feature that could be obtained through the use of the Pipe server shipped with the Hermes main codebase. The Pipe server extensively used during the evaluation process could stream test messages from a local text file to a local Hermes instance where a filter installed is tested whether it effectively processes the stream and selects only messages conforming to it. This technique allows to test the expression before it is applied to the actual live data stream to which the instance is subscribing to. The file could consist of a large number of special test messages specifically designed to test individual attributes of the expression.

## 7.2 Future Work

Overall based on the information given in this thesis we could draw to conclusion that Hermes at its present stage of development represents a significant contribution to the attempt of engineering and constructing the new generation of route analysis tools. Nevertheless the following development roadmap could be tentatively anticipated for the future:

- ability to aggregate data streams from multiple sources: this implies the deployment of multiple Writers and would allow to diversify the data feed and deliver wider scope of information to the clients. It is worth noting that mechanisms such as duplicate suppression module are already integrated into the existing system to prevent dissemination of the duplicate messages from several sources

- addition of supplementary data manipulation functions: possible additional aggregates such as count(), min(), avg(), max() and related constructs could be added to the language to facilitate the future data analysis needs

- ability to process several pattern transactions at once: this crucial feature would enable individual users to register and execute user-defined patterns and be able to receive processed data separately from other users. This implies the deployment of separate per-user Queues and due to obvious resources overhead in the number of Queues only a small fixed amount of transactions could be defined to be executed simultaneously per single instance. At the same time even this limited multi-transaction functionality would turn Hermes into a very significant stream processing product.

- ability to stop/start (schedule) the pattern transactions: this additional feature would enable the resource reservation mechanism in the system with potential to cancel costly transactions consuming CPU/memory resources in the multi-transactional environment and apply user-bound execution policies per transaction

- service discovery mechanism: the potentially necessary addition to the feature set that would permit the discovery of Hermes mesh topology outlining the individual filters (queries) installed on each node in the chain.

Multi-transaction capability and ability to aggregate data from several XML streams are undoubtedly the critical add-on components that could bring Hermes into the next level of streaming tools. At the same time we should observe that both features could be easily obtained via the chaining of instances with each other without any extra effort of developing highly complex additional infrastructure with heavy multithreading.

The integration of Hermes instances with the existing Service Discovery Service (SDS) such as Ninja [CZH+99] is a necessary feature in the presence of the Hermes mesh where a subscribing instance should know to what node in the chain it needs to be subscribed to in order to receive the desired data. Hermes should acquire a separate module to continuously listen for SDS server announcements on the global multicast channel in order to determine the appropriate SDS server for its service descriptions. The service description (in our case just the currently active query and supplementary instance info) would be constructed in the form of the XML message and registered with the SDS for discovery propagation.

The similarities between data streams and Hermes XML stream processing capabilities bring the platform at the crossroads between fields of pure networking and database management opening new potentials in research and development of the product. Already projects like XQuery [CCI08] primarily conceived as a database query language in the tradition of SQL designed to query collections of XML data bear strong resemblance to the core of the Hermes functional capabilities. Addition of SQL like constructs to the Hermes language would allow it to be a fully-fledged XML data stream processing system on its own with highly efficient performance characteristics and scalability advantages under heavy subscribers' load.

Finally we could conclude that Hermes as the XML parsing platform with good network performance infrastructure could be easily adopted to other types of structured XML data streams provided that the necessary data collectors foundation is present. Essentially any types of well-formed XML message streams could be utilized making Hermes a potentially lucrative platform for deploying in the fields such as financial transactions, event processing, sensor networks, click stream analysis and other emerging areas.

# REFERENCES

[ABB$^+$04] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R Motwani, U. Srivastava, and J. Widom. Stream: The stanford data stream management system. Technical Report 2004-20, Stanford InfoLab, 2004.

[ABW06] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15:121–142, June 2006.

[BBAS03] A. Bremler-Barr, Y. Afek, and S. Schwarz. Improved bgp convergence via ghost flushing. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*. IEEE, 2003.

[BC01] A. Broido and K. Claffy. Complexity of global routing policies. In *Proceedings of the IMA Special Workshop: Mathematical Opportunities in Large–Scale Network Dynamics*, 2001.

[Bro01] A. Broido. kc claffy. Analysis of RouteViews BGP data: policy atoms. In *Proceedings of Network-related data management (NRDM) workshop, Santa Barbara*, 2001.

[BSW01] Shivnath Babu, Lakshminarayan Subramanian, and Jennifer Widom. A data stream management system for network traffic management. In *Workshop on Network-Related Data Management (NRDM 2001)*, May 2001.

[BW01] Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30:109–120, September 2001.

[CBB$^+$03] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Cetintemel, Ying Xing, and Stan Zdonik. Scalable Distributed Stream Processing. In *CIDR 2003 - First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, January 2003.

[CCI08] Marc Van Cappellen, Wouter Cordewiner, and Carlo Innocenti. Data aggregation, heterogeneous data sources and streaming processing: How can xquery help? *IEEE Data Eng. Bull.*, 31(4):57–64, 2008.

[cg] cheng grow. Bgp routing information in xml format. http://tools.ietf.org/html/draft-cheng-grow-bgp-xml-00.

[COZ08] Ying-Ju Chi, Ricardo Oliveira, and Lixia Zhang. Cyclops: The Internet AS-level Observatory. In *ACM SIGCOMM Computer Communication Review*, 2008.

[CSU] CSU. Bgp monitoring system. http://bgpmon.netsec.colostate.edu/index.html.

[CZH+99] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An architecture for a secure service discovery service. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 24–35, New York, NY, USA, 1999. ACM.

[FJB05] Nick Feamster, Ramesh Johari, and Hari Balakrishnan. Implications of autonomy for the expressiveness of policy routing. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 25–36, New York, NY, USA, 2005. ACM.

[Gao01] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Trans. Netw.*, 9(6):733–745, 2001.

[GM03] T. Griffin and M. Mao. Interdomain Routing Streams. In *Proc. Workshop on Management and Processing of Data Streams (MPDS'03)*, San Diego, CA, June 2003.

[Hus] Geoff Huston. Geoff huston's as number report. http://www.potaroo.net/tools/asn16/.

[iet] ietf. Mrt routing information export format. http://www.ietf.org/internet-drafts/draft-ietf-grow-mrt-07.txt.

[LMP+06] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang. PHAS: A prefix hijack alert system. In *Proc. USENIX Security Symposium*, 2006.

[MZHL05] X. Meng, B. Zhang, G. Huston, and S. Lu. IPv4 address allocation and the BGP routing table evolution. *ACM SIGCOMM Computer Communication Review*, 35(1):71–80, 2005.

[PAMZ05] Dan Pei, Matt Azuma, Dan Massey, and Lixia Zhang. Bgp-rcn: improving bgp convergence through root cause notification. *Comput. Netw. ISDN Syst.*, 48(2):175–194, 205.

[rf9] rf904. Exterior gateway protocol formal specification. http://tools.ietf.org/html/rfc904.

[rfca] rfc1930. Guidelines for creation, selection, and registration of an autonomous system (as). http://tools.ietf.org/html/rfc1930.

[rfcb] rfc2328. Ospf version 2. http://www.ietf.org/rfc/rfc2328.txt.

[rfcc] rfc2453. Rip version 2. http://www.ietf.org/rfc/rfc2453.txt.

[rfcd] rfc4271. A border gateway protocol 4 (bgp-4). http://www.ietf.org/rfc/rfc4271.txt.

[RIPa] RIPE. Ripe (rseaux ip europens) routing information service. http://www.ripe.net/projects/ris/.

[RIPb] RIPE. Verification of zebra as a bgp measurement instrument. http://www.ripe.net/ripe/meetings/ripe-46/presentations/ripe46-routing-zebra-bgp.pdf.

[Sha] Stephen Shankland. Amazon suffers u.s. outage on friday. http://news.cnet.com.

[UCB] UCB. Telegraph - adaptive dataflow for querying streams, the deep web, and beyond. http://telegraph.cs.berkeley.edu/index.html.

[UM] UM. Netviews. http://netlab.cs.memphis.edu/projects_netviews.html.

[Uni] Brown University. The aurora project. http://www.cs.brown.edu/research/aurora/.

[UO] UO. University of oregon route views project. http://www.routeviews.org/.

[WG03] F. Wang and L. Gao. On inferring and characterizing Internet routing policies. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 15–26. ACM New York, NY, USA, 2003.

[WZP+02] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S.F. Wu, and L. Zhang. Observation and analysis of BGP behavior under stress. *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 183–195, 2002.

[YOB+09] He Yan, Ricardo Oliveira, Kevin Burnett, Dave Matthews, Lixia Zhang, and Dan Massey. Bgpmon: A real-time, scalable, extensible monitoring system. *Conference For Homeland Security, Cybersecurity Applications & Technology*, 0:212–223, 2009.

[ZLMZ05] Beichuan Zhang, Raymond Liu, Daniel Massey, and Lixia Zhang. Collecting the Internet As-level Topology. *SIGCOMM Comput. Commun. Rev.*, 35(1):53–61, 2005.