UNIVERSITY OF TWENTE.

Flow-Based Monitoring of GTP Traffic in Cellular Networks

Master of Science Thesis

by

E.H.T.B. Brands

Date:	July 20, 2012
Committee:	dr. ir. Aiko Pras
	Rick Hofstede, M.Sc.
	dr. ir. Georgios Karagiannis
Institution:	University of Twente, Enschede, The Netherlands
Faculty:	Electrical Engineering, Mathematics and Computer Science (EWI)
Chair:	Design and Analysis of Communication Systems (DACS)

Acknowledgements

First of all I would like to thank Paolo Lucente, founder and developer of the PMACCT project [1], for his time and effort he put into implementing my developed extension into the PMACCT source code. Thanks to his help, we managed to set up a successful proof of concept.

Special thanks thanks to my supervisors at KPN: Paul Schilperoort and Rene Soutjesdijk for their guidance during this project. They helped me organize things and provided me the resources needed to make this research a success.

Finally I would like to thank my daily supervisor at the University of Twente, Rick Hofstede, for his guidance and feedback during the whole project. I also would like to thank the other members of my graduation committee, Aiko Pras and Georgios Karagiannis, for their feedback on this report.

Erik Brands
Enschede, July 2012

Abstract

Network monitoring is becoming more important for network operators, due to increased interest in user traffic profiling. Traditionally all these monitoring activities were performed in a packet-based manner, often using Deep Packet Inspection (DPI). Due to the introduction of new laws that forbid the use of DPI, these packet-based techniques are currently at stake. This is one of the reasons why currently a shift is taking place from packet-based to flow-based measurement techniques. This work proposes a flow-based solution for monitoring packet-switched roaming traffic, which is characterized by the GPRS Tunneling Protocol (GTP). The proposed solution is able to monitor both GTP v.0 and GTP v.1 traffic in as well the control plane as the user plane and uses IPFIX for the export of flow information. To demonstrate the feasibility of our solution we set up a proof of concept by developing an extension to the existing flow-based monitoring application PMACCT. The proposed solution was validated in the testcenter of Dutch mobile operator KPN.

Contents

List of Acronyms

APN	Access Point Name
AS	Autonomous System
ASN	Autonomous System Number
CDF	Charging Data Function
DPI	Deep Packet Inspection
DSCP	Differentiated Services Code Point
DTLS	Datagram Transport Layer Security
ISP	Internet Service Provider
GERAN	GSM EDGE Radio Access Network
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GRX	GPRS Roaming Exchange
GSN	GPRS Support Node
GTP	GPRS Tunneling Protocol
IE	Information Element
ISP	Internet Service Provider
IPFIX	IP Flow Information Export
LTE	Long Term Evolution

MCC	Mobile Country Code
MNC	Mobile Network Code
MS	Mobile Station
MVNO	Mobile Virtual Network Operator
NSAPI	Network Layer Service Access Point Identifier
PEN	Private Enterprise Number
PDN	Packet Data Network
PDP	Packet Data Protocol
PLMN	Public Land Mobile Network
QoS	Quality of Service
RAN	Radio Access Network
RAT	Radio Access Technology
RRD	Round Robin Database
SGSN	Serving GPRS Support Node
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SMS	Short Message Service
TFT	Traffic Flow Template
TLS	Transport Layer Security
UTRAN	UMTS Terrestrial Radio Access Network

Chapter 1

Introduction

The area of Internet traffic measurements has advanced greatly in recent years. One of the main reasons is the increasing interest of Internet Service Providers (ISPs) in user traffic profiles. An accurate user profile, can help ISPs in better serving their customers, e.g, by capacity planning. ISPs realize that measuring traffic to and from the customer is essential in understanding the user's behavior.

Besides this increased interest in traffic measurements, we also see a shift taking place from packet-based to flow-based measurements. There is an ongoing discussion about legal issues when performing packet-based monitoring, especially when the payload of packets is inspected. On 8 May 2012, the Dutch government accepted a new telecommunication law, which should provide net-neutrality in the Netherlands [2]. Since this new law also restricts the use of Deep Packet Inspection (DPI), certain packet-based monitoring solutions are at stake. Flow-based monitoring techniques aggregate traffic into flows, which implies only summaries of the actual traffic are exported. This also makes flow-based monitoring techniques much more scalable than packet-based solutions, which will become an important aspect of monitoring, with the forecasted data-growth that comes with the introduction of LTE. These two arguments increase the interest of service providers to look for flow-based solutions to monitor their traffic. This research was performed in collaboration with Dutch service provider KPN [3], in order to research the possibilities of monitoring packet-switched roaming traffic, which is essentially GPRS Tunneling Protocol (GTP) traffic, in a flow-based manner. By the time of writing, there are no flow-based monitoring solutions available that fully support the parsing of GTP traffic. Some flow-based monitoring solutions claim to support the monitoring of GTP traffic, but further analysis showed this monitoring is limited to identifying tunneled data in the user-plane.

The goal of this work is to develop a flow-based solution for monitoring GTP traffic in cellular networks using IP Flow Information Export (IPFIX), which is an IETF proposed standard for exporting information about traffic flows. This solution can serve as an alternative to packet-based solutions, that are currently deployed. In order to achieve this goal the following research questions have been defined:

1. *Which properties of packet-switched roaming traffic are relevant for monitoring by mobile operators?*
2. *How can these properties be measured using IPFIX?*
3. *How can the received Flow Records be analyzed using a data analysis application?*

In order to answer these research questions we started with performing a requirements analysis at KPN. After having defined a clear set of requirements, we analyzed several packet-traces from the live network of KPN to get insight into the structure and semantics of the GTP traffic. Parallel to analyzing these traces, we also researched the 3GPP standards on GTP [4][5][6] to find out how the required information could be extracted from the GTP traffic. After researching serveral RFC's from the IPFIX Working Group [7], we defined how the relevant fields inside the GTP traffic could be exported using IPFIX. We set up a proof of concept by developing an extension to the existing flow-based monitoring application PMACCT [1]. We validated our solution using the proof of concept by setting up a test environment at the KPN test-center. In the end we used the network graphing tool Cacti [8] to demonstrate how the acquired results can be visualized.

The structure of this work is as follows. Chapter 2 provides background information on GPRS and IPFIX. In Chapter 3 we defined the requirements of KPN that are posed on a flow-based solution for monitoring GTP traffic. Chapter 4 describes the existing solutions in the area of monitoring GTP traffic and explains why these solutions do not satisfy our requirements. Chapter 5 shows how the required information can be extracted from the GTP messages. In Chapter 6 the complete IPFIX architecture of the proposed solution is described. Chapter 7 describes the implementation and validation of our solution using the proof of concept. In Chapter 8 we give an example of a data analysis application that can be used to visualize the results. Conclusions are drawn in the final chapter of this work, where also suggestions for future work are provided.

Chapter 2

Background

2.1 GPRS

General Packet Radio Service (GPRS) is defined as the packet bearer service for GSM (2G), UMTS (3G) and WCDMA mobile networks to transmit IP packets to external Packet Data Networks (PDNs), such as the Internet. Although GSM and UMTS networks use different Radio Access Networks (RANs), respectively GERAN and UTRAN, they rely on the same packet-switched core network.This common core network, together with these Radio Access Networks, provides GPRS services. The core network is designed to support several Quality of Service levels to allow efficient transfer of both real-time traffic (e.g. voice, video) and non real-time traffic. Applications based on standard data protocols and SMS are supported, and inter-working with IP networks is defined [9].

2.1.1 The GPRS Core Network

The GPRS core network provides mobility management, session management, and transport for IP packet services. Besides that, it also provides support for additional functionalities such as billing and lawful interception [10]. The GPRS core network consists of a number of network elements, which are interconnected by various logical interfaces. Figure 2.1 gives an overview of the GPRS logical architecture. This figure is a simplified version of Figure 1 in the the 3GPP 29.060 standard [5]. Figure 2.1 will be used as a reference in the remainder of this chapter.

The GPRS core network functionality is logically implemented on two network nodes, namely the Serving GPRS Support Node (SGSN) and the Gateway GPRS Support Node (GGSN). SGSN and GGSN are commonly

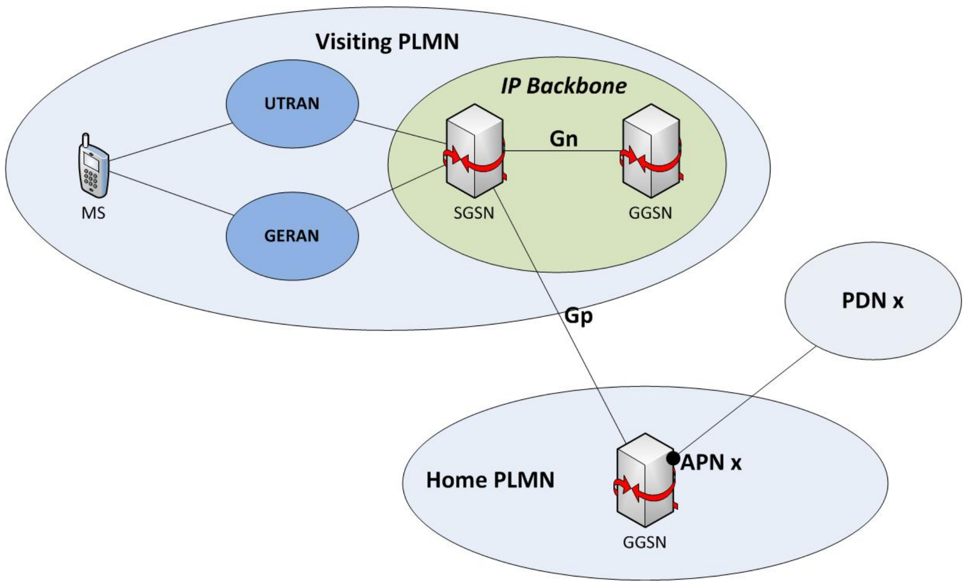


Figure 2.1: Simplified overview of the GPRS logical architecture.

known as GSNs. The interface between the SGSN and the GGSN is called the Gn interface in case both GSNs are in the same Public Land Mobile Network (PLMN). When SGSN and GGSN are in different PLMNs, which is commonly referred to as roaming, the interface between the two network elements is called the Gp interface. The Gp interface provides the functionality of the Gn interface, plus security functionality required for inter-PLMN communication. This security functionality is based on mutual agreements between operators [9]. The difference between Gn and Gp interface is illustrated in Figure 2.1.

The Gateway GPRS Support Node (GGSN) provides interworking with other Packet Data Networks (PDNs), such as the Internet, and is connected to various other core network nodes in the same PLMN via an IP-based backbone network. The GGSN contains the routing information for packet-switched-attached users, which is used to tunnel data packets to the SGSN, the user is currently attached to [9].

The Serving GPRS Support Node (SGSN) is connected to the GERAN or to the UTRAN, as can been seen in Figure 2.1. The SGSN is responsible for the delivery of data packets from and to the Mobile Station within its geographical service area. Its tasks include packet routing, transfer management, mobility management (attach/detach of MS's and location management), logical link management, authentication and charging functions. [10]

2.1.2 GPRS Tunneling Protocol (GTP)

Within the GPRS core network, the GTP is the most important carrier protocol. GTP allows multi-protocol packets to be tunneled between GGSN and SGSN and between SGSN and UTRAN. Besides that, it offers the possibility to tunnel packets between different PLMNs over the Gp interface of the SGSN/GGSN.

GTP consists of a suite of IP-based communication protocols. It includes the GTP control plane (GTP-C), GTP user plane (GTP-U) and GTP' (GTP Prime) protocol. Figure 2.2 shows which GTP protocols apply to which interfaces of the GPRS Core network.

The GTP-U protocol is implemented by SGSNs and GGSNs in the UMTS/GPRS backbone and by Radio Network Controllers (RNCs) in the UTRAN to provide a tunneling mechanism for carrying user data [9]. GTP-U over the Iu interface (i.e. the interface between SGSN and UTRAN) is

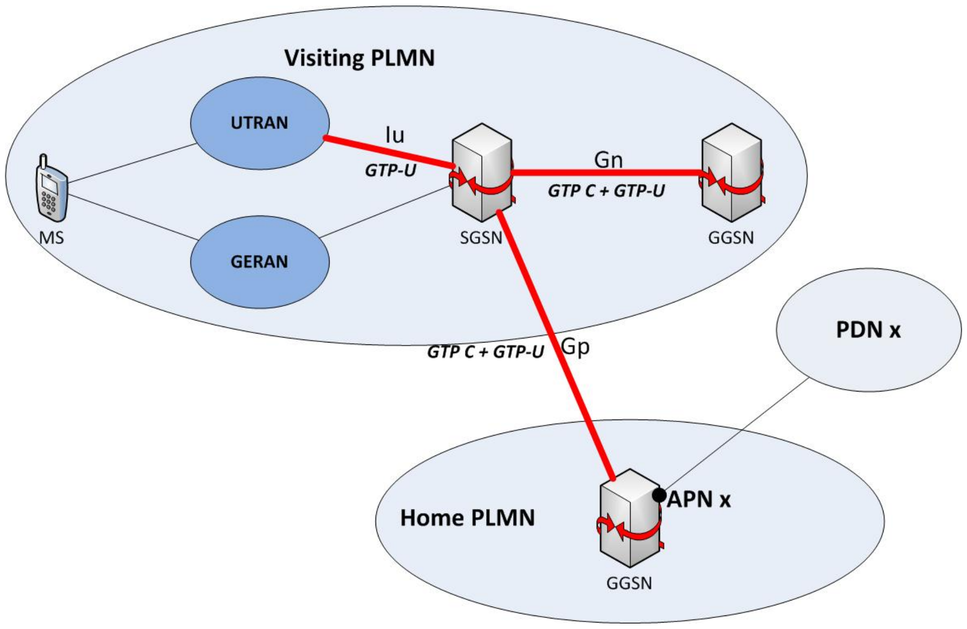


Figure 2.2: Presence of the GPRS Tunneling Protocol (GTP) in the GPRS core network.

not considered in this work.

The GTP-C (Control plane) protocol is implemented by SGSNs and GGSNs in the UMTS/GPRS Backbone. GTP-C can be seen as a tunnel management protocol, which allows the SGSN to provide mobile stations access to Packet Data Networks (PDNs). GTP-C control plane signaling is used to create, modify and delete tunnels.

GTP' (GTP prime) can be used for carrying charging data from the Charging Data Function (CDF) of the GSM or UMTS network to the Charging Gateway(s) within a PLMN [9]. However, in this work we focus on GTP-C and GTP-U, GTP' is out of scope.

2.1.3 PDP Contexts

A Packet Data Protocol (PDP) context offers a packet data connection over which the Mobile Station (MS) and the selected Access Point Name (APN) can exchange IP packets. APN is a logical name referring to the PDN and/or to a service that the subscriber wishes to connect to. Depending on the network operator, a single APN can provide access to one or more services, e.g. MMS, Internet or WAP. The APN is hosted at a GGSN, as depicted in Figure 2.1. One GGSN can host multiple APNs. Considering this, the MS needs to be aware of the service it wants to use and the APN hosting that service. A Mobile Station can have multiple simultaneous PDP contexts, one for each active service. For each PDP context a different Quality of Service (QoS) profile may be requested. For example, some PDP contexts may be associated with e-mail that can tolerate lengthy response times. Other applications cannot tolerate delay and demand a very high level of throughput, such as interactive applications. These different requirements are reflected in the QoS profile the MS requests. When establishing a PDP context with an APN, the MS receives an PDP Address, either IPv4 or IPv6 type, that it has to use when communicating over that PDP context. This means that when a MS has established several connections to different APN the MS will have different IP addresses for each of the provided services.

As Mobile Stations develop, users will run multiple services at the same time. These services can have different QoS parameters and can be hosted at different APNs. Especially in case of IMS, the number of simultaneous PDP context per MS will grow, because IMS services are all packet-switched. [11] In case multiple simultaneous PDP contexts are set up from the same MS, two scenarios can be differentiated:

- Multiple primary PDP contexts: In this case two or more PDP contexts are set up independently from each other to different APNs. Every PDP context gets a unique PDP Address, Network Layer Service Access Point Identifier (NSAPI) and set of QoS values. Besides that, every PDP context has a separate Iu interface radio access bearer and a separate GTP tunnel to transfer user plane data. Figure 2.3 illustrates this scenario. Multiple primary PDP contexts can be activated/deactivated separately from each other.

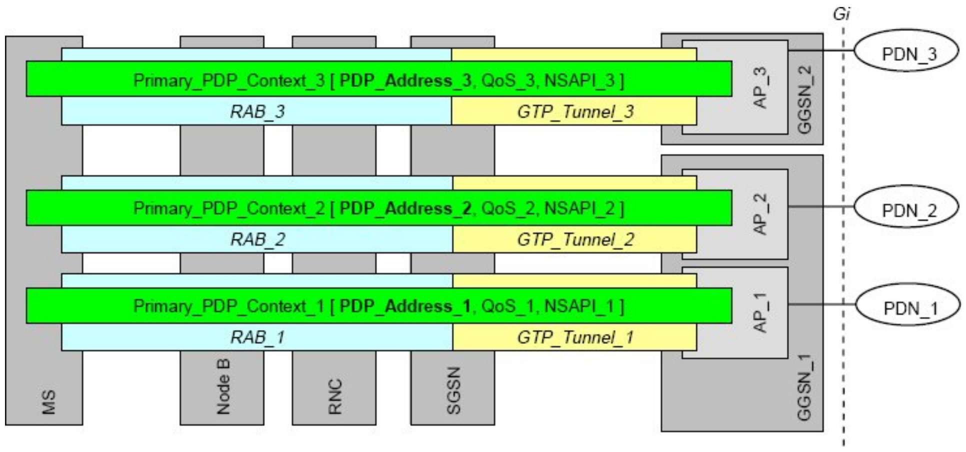

Gi
PDN_3
PDN_2
PDN_1
Primary_PDP_Context_3 [PDP_Address_3, QoS_3, NSAPI_3]
RAB_3
GTP_Tunnel_3
Primary_PDP_Context_2 [PDP_Address_2, QoS_2, NSAPI_2]
RAB_2
GTP_Tunnel_2
Primary_PDP_Context_1 [PDP_Address_1, QoS_1, NSAPI_1]
RAB_1
GTP_Tunnel_1
AP_3
AP_2
AP_1
GGSN_2
GGSN_1
MS
Node B
RNC
SGSN

Figure 2.3: Multiple primary PDP contexts, copied from [11].

- Secondary PDP contexts: In this case there is a primary PDP context, which is always set up first. After that, one or more secondary PDP contexts can be set up. These secondary PDP contexts reuse the PDP address of the primary PDP context and always connect to the same APN as the primary PDP context. The NSAPI is used to differentiate between the different PDP contexts. The benefit of secondary PDP contexts is that each PDP contexts can have its own set of QoS values. Each PDP context, primary or secondary, has its own Iu interface radio access bearer and GTP tunnel to transfer user plane data. The Traffic Flow Template (TFT) is introduced to route downlink user plane data into the correct GTP tunnel and hence into the correct radio access bearer for each context. Without the TFT the GGSN would not be able to know in which GTP tunnel to send the user plane data, because all secondary PDP contexts use the same PDP address. Figure 2.4 gives an example of a scenario with two secondary PDP contexts. Any of the PDP contexts can be deactivated while keeping the associated primary PDP context active. However, when the primary PDP context is deactivated, all secondary PDP contexts will also be deactivated.

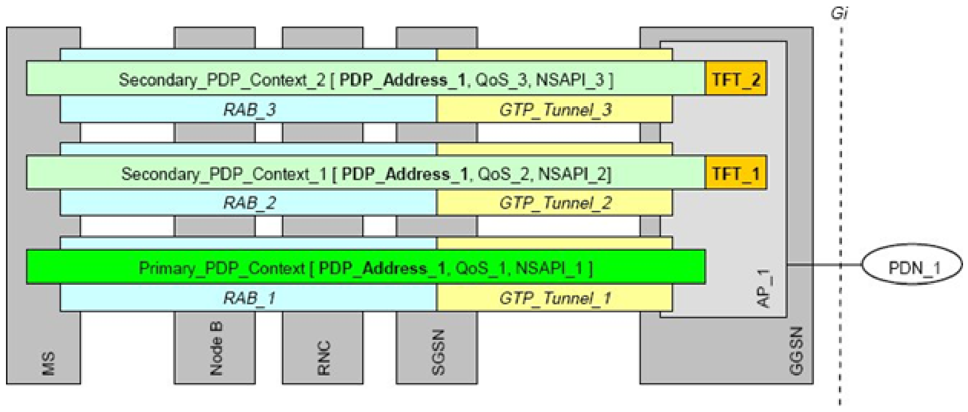

Gi
Secondary_PDP_Context_2 [PDP_Address_1, QoS_3, NSAPI_3]
TFT_2
RAB_3
GTP_Tunnel_3
Secondary_PDP_Context_1 [PDP_Address_1, QoS_2, NSAPI_2]
TFT_1
RAB_2
GTP_Tunnel_2
Primary_PDP_Context [PDP_Address_1, QoS_1, NSAPI_1]
RAB_1
GTP_Tunnel_1
AP_1
PDN_1
MS
Node B
RNC
SGSN
GGSN

Figure 2.4: One primary with two associated secondary PDP contexts, copied from [11].

Setting up a PDP Context starts with the PDP context Activation procedure: Upon receiving an *Activate PDP context Request* message or an *Activate Secondary PDP context Request* message from the MS, the SGSN shall initiate procedures to set up PDP contexts. The first procedure includes subscription checking, APN selection and host configuration, while the latter procedure excludes these functions and reuses PDP context parameters including the PDP address, except the QoS parameters. After the PDP context activation procedure the MS is able to send packets over the established connection. After the PDP context Activation procedure, one or more modification procedures can take place during the lifetime of a PDP context. Modification procedures modify parameters that were negotiated during an activation procedure for one or several PDP contexts. An MS, a GGSN, an SGSN, or an RNC can request or initiate a modification procedure [9]. The PDP context Deactivation procedure is used to change the state of the PDP context from *active* to *inactive*. The initiative to deactivate a PDP context can come from an MS, a GGSN or an SGSN. All PDP contexts with this PDP address are deactivated,which includes every associated secondary PDP context that shares this PDP address.

2.2 IPFIX

The IPFIX protocol is an IETF proposed standard for exporting information about traffic flows. The protocol is the logical successor of Cisco NetFlow v9, upon which it is based [12].

Key to the IPFIX protocol is the definition of a flow. A flow is defined as a set of IP packets passing an Observation Point in the network during a certain time interval [13]. All packets belonging to a particular flow have a set of common properties. Each property is defined as the result of applying a function to the values of:

1. one or more packet header fields (e.g., destination IP address), transport header fields (e.g., destination port number), or application header fields (e.g., RTP header fields).

2. one or more characteristics of the packet itself (e.g., number of MPLS labels).

3. one or more fields derived from packet treatment (e.g., next hop IP address, output interface).

A packet is defined as belonging to a flow if it completely satisfies all the defined properties of the flow [13]. Table 2.1 gives an example of four different flow records. In this example, a packet belongs to the first flow in the table if and only if it has sourceIPv4Address 192.0.2.1, destinationIPv4Address 192.0.2.65.2 and Differentiated Services Code Point (DSCP) 4.

The above definition of flows implies that the Flow Key in IPFIX can be flexibly defined in contrast to the traditional five-tuple Flow Key (sourceIPv4Address, destination-IPv4Address, sourceTransport-Port, destination-TransportPort, protocolIdentifier), which is used in older versions of NetFlow. In the example in Table 2.1 the Flow Key is defined as sourceIPv4Address, destinationIPv4Address and ipDiffServCodePoint, leaving packetDeltaCount as a non-key field.

sourceIPv4Address	destinationIPv4Address	DSCP	packetDeltaCount
192.0.2.1	192.0.2.65.2	4	23423
192.0.2.23	192.0.2.65.67	4	32432
192.0.2.23	192.0.2.65.67	2	32
192.0.2.129	192.0.2.65.67	4	12

Table 2.1: Example Flow Records.

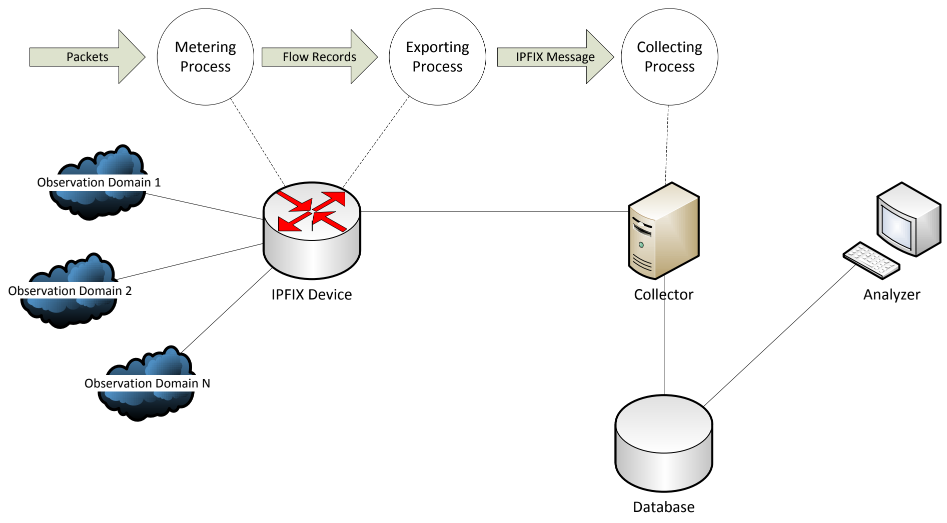


Figure 2.5: Mapping between the functional and physical IPFIX architecture.

Figure 6.1 gives a combined overview of the logical and physical IPFIX architecture. Central in the IPFIX architecture is the IPFIX device which collects IP packets from one or more Observation Domains. Each Observation Domain has a metering process associated with it, that generates and maintains flow records according to the predefined Flow Key. Once expired the metering process will forward the flow-records to an exporting process, which is often located in the same physical device, ofter referred to as IPFIX Device. An exporting process transports the flow records inside an IPFIX message to the collecting process. The collecting process is hosted at a collector and might process or store received flow records [13]. Besides the metering, exporting and collecting process, the IPFIX architecture also defines a fourth type of process: intermediate processes. These processes are used to modify flows (e.g., to aggregate, correlate, or anonymize them) [12]. Intermediate processes are outside the scope of this work and therefore not visualized in Figure 6.1. The flow data received at the collector can be analyzed with any tool that supports IPFIX input.

Data is transferred between Exporter and Collector in *Messages*. These messages can either hold templates or data sets. Templates describe the structure and semantics of the information in the data sets. Templates are always sent to the collector before the corresponding data sets are sent. Oth-

ElementID	12
Name	destinationIPv4Address
Data Type	ipv4Address
Data Type Semantics	current
Description	The IPv4 destination address in the IP packet header.

Table 2.2: Example Information Element (IE).

erwise, the collector would not be able to correctly interpret the data inside the data sets. Information in messages of the IPFIX protocol is modeled in terms of Information Elements (IEs) [14]. An IE represents a named data field with a specific data type and meaning [12]. An Example of an IE is shown in Table 2.2. A template is essentially an ordered list of IEs [12]. Many IEs are standardized by IANA [15]. Besides these IANA-assigned IEs, IPFIX also offers the possibility to self-define IEs, referred to as enterprise-specific IEs. If an application has a need for an IE which is not IANA-assigned and cannot be added to the IANA registry (because it is either commercially sensitive, experimental, or of too limited applicability to justify an IANA registration), it can be allocated in the enterprise-specific space, scoped to an SMI Private Enterprise Number (PEN) [12].

IPFIX messages are sent from exporter to collector in *transport sessions*. As IPFIX is unidirectional, a transport session typically consists of an exporting process initiating a connection to the collecting process, then sending templates followed by data described by those templates [12]. The IPFIX protocol prefers SCTP for transport, but TCP and UDP can be used as well. Besides these three options, transport sessions can also be stored in the IPFIX file format and transported using a protocol like HTTP or SMTP. TLS and DTLS can be used to provide the confidentiality, integrity, and authentication assurances required by the IPFIX protocol, without the need for pre-shared keys [16].

Unfortunately not many router vendors have yet adopted IPFIX in their hardware. By the time of writing, from the major network equipment vendors only Nortel and Juniper have implemented IPFIX on some of their routers.

Chapter 3

Requirements Analysis

Developing a flow-based solution for monitoring packet-switched roaming traffic starts with finding out what characteristics of packet-switched roaming traffic are relevant for monitoring. To answer this question a requirements analysis was performed at KPN. Several stakeholders within the company were interviewed about the relevant characteristics of packet-switched roaming traffic. They were asked what information is lacking in current packet-switched roaming monitoring tools and what information they would like to see in a new application. The set of requirements is divided into three categories: general requirements, which apply to the requested solution in general and categories for GTP-C [4][5] and GTP-U [6], which corresponds to the two protocols in the GTP suite.

This chapter starts by describing the set of general requirements, followed by the requirements for GTP-C and GTP-U, respectively. For a general description of GPRS and GTP, we refer to Appendix 2.1.1.

3.1 General

This section describes the set of general requirements, which are not directly related to one of the two GTP protocols, but are posed on the flow-based monitoring solution in general.

The system should...

1. *Identify roaming partners based on Autonomous System Numbers (ASNs) rather than IP addresses.*
   Roaming partners are mobile network operators that defined a certain roaming agreement. The content of all roaming agreements is

standardized by the GSM Association (GSMA) [17]. The benefit of identifying roaming partners based on ASNs instead of IP addresses is that ASNs are less susceptible to changes. Roaming partners sometimes change the IP addresses of their SGSNs/GGSNs or place them in another subnet, while the chance that the ASN of a roaming partner changes is rather small.

2. *Monitor both inbound and outbound roaming traffic.*
   From a service provider point of view inbound roaming are visitors who use your Public Land Mobile Network (PLMN) to connect to their home PLMN abroad. Outbound roaming are your own customers (abroad) that connect via a foreign PLMN to your own PLMN. In practice this implies monitoring both directions of the Gp interface (see Appendix 2.1.1).

3. *Support online monitoring of packet-switched roaming traffic.*
   In case of online measurement, the analysis is performed while the data is captured. While in offline measurements, a data trace is stored and analyzed later.

4. *Monitor GTP v.0 and GTP v.1 traffic.*
   These are currently three version of GTP: v.0 [4], v.1 [5][6] and v.2 [18]. GTP v.0 and GTP v.1 are the most common versions these days. GTP v.2, is used for evolved packet services, such as Long Term Evolution (LTE). By the time of writing, LTE is not yet widely deployed in Western Europe, but many operators, including KPN, are planning to launch an LTE network in the near future. For this reason, the proposed solution should now only support the analysis of GTP v.0 and GTP v.1; the analysis of GTP v.2 is saved for future work. However, the proposed solution should be developed in an extensible manner, so that GTP v.2 support can be easily added later.

3.2 GTP-C

This section describes the system requirements for monitoring the GTP-C [4][5] traffic.

The system should...

1. *Monitor the number of Create/Update/Delete PDP Context request and response messages per roaming partner.*

These messages are the key control plane messages in roaming traffic. They are used to set up, update and delete the PDP Context of the roaming user respectively, (see Appendix 2.1.1). The information needed to satisfy the rest of the requirements in this section is all encapsulated in these messages.

2. *Monitor the number of unique Create PDP Context Request messages per roaming partner, leaving duplicates from the same MS out*
   As a single MS can have many simultaneous PDP contexts and even more attempts to set up a PDP context, it could be valuable to also monitor the number of unique Create PDP context request messages, leaving duplicate attempts from the same MS out.

3. *Identify the value of the Cause field in Create, Update and Delete PDP Context Response messages.*
   The response message which is always sent as a reply to the corresponding create/update/delete PDP context request message, always contains a cause value. This value indicates whether the request was accepted or gives the reason for rejection. The failure and success rate of each of these messages together with the reasons for rejection provide valuable information for a service provider.

4. *Identify the APN the Mobile Station wants to connect to from Create PDP Context Request messages.*
   From a service providers point of view, the APN your outbound roamers connect to could provides insight in the performance of difference APNs. Especially in combination with the information provided by the *cause* field.

5. *Identify the country in which the user is located from Create PDP Context Request messages.*
   From a service providers point of view it could be particularly interesting to monitor the countries in which your outbound roamers are residing.

6. *Identify the roaming partner the user is connected to from Create PDP Context Request messages.*
   The network operator one of your outbound roamers is connected to in combination with various other statistics can provide information about the QoS the roaming partner provides to your customers. This information can be used to test various Service Level Agreements (SLAs) you have agreed on with the other roaming partner.

7. *Identify the Radio Access Network (GERAN/UTRAN) the MS is connected to from Create PDP Context Request messages.*
   As KPN currently only has GSM and UMTS deployed, there is only need to identify the Radio Access Networks (RANs) of GSM and UMTS, GERAN and UTRAN, respectively. Identifying E-UTRAN, the RAN of LTE, will be saved for future work.

8. *Identify the GTP version used by roaming partners from the header of the GTP message*
   The version field, which is present in every GTP message, can provide information about the GTP version used by roaming partners.

3.3 GTP-U

This section describes the requirements for monitoring the GTP-U [4][6] traffic. This is the traffic after a PDP Context is established, i.e. the actual user data.

The system should...

1. *Monitor the amount of GTP-U traffic per roaming partner.*
   This information can particularly be useful for setting up roaming agreements (SLAs).

2. *Monitor the amount of GTP-U traffic per country.*
   In contrast to the amount of traffic per roaming partner, the amount of traffic per country is not directly of interest for setting up SLAs. However, for a service provider it would be interesting to know the amount of GTP-U traffic that is being transferred to and from other countries.

Chapter 4

Existing Solutions

First step after defining the requirements in Chapter 3 is searching for existing solutions that are able to satisfy these requirements. This section describes the state-of-the-art in the area of GTP traffic monitoring, flow-based monitoring in particular. Both packet-based and flow-based solutions will be described and we will explain why these solutions are not able to satisfy the requirements.

4.1 Packet-Based Monitoring Solutions

To the best of our knowledge, all major service providers in The Netherlands use a packet-based application for monitoring their packet-switched roaming traffic. These monitoring applications are specialized in monitoring the Gp (and the Gn) interface of the GPRS core network. The Gp interface is characterized by carrying mostly packet-switched roaming traffic, i.e. GTP traffic (see also Appendix 2.1.1). An application that is capable of monitoring the GTP traffic on these interfaces is DataMon [19]. DataMon extracts various parameters out of the GTP-C and GTP-U messages and displays its results in various graphs and tables. Another similar packet-based solution is Daromo [20], which also monitors GTP traffic on the Gp interface. Both applications use one or more passive measurement probes and a central database to store the captured data. Both solutions are capable in satisfying most of the GTP-C and GTP-U requirements. However, all these solutions are packet-based, while this work aims at a flow-based solution for monitoring GTP traffic. Besides that, both solutions identify roaming partners by IP Addresses, while one of the requirements states roaming partners should be identified by Autonomous System Number (ASN).

4.2 Flow-Based Monitoring Solutions

One of the major benefits of flow monitoring technologies such as NetFlow v5/v9 and IPFIX is that they are implemented on routers (and switches). This implies no extra hardware probes are required, only a (centralized) Collector. The shortcoming of NetFlow v5/v9 is that there is a fixed set of traffic features that can be monitored, such as the traditional five-tuple flow-key (source IP address, destination IP address, source port, destination port and protocolIdentifier), see also Appendix 2.2. IPFIX adds more flexibility to this by introducing IEs. However, as can be seen in the IANA registry [15],there are no standardized IEs defined for GTP. Besides that, not many router vendors have yet adopted IPFIX in their hardware (see Appendix 2.2). This forces us to turn to a software-based solution. The number of available software-based solutions that support flow-based monitoring of GTP traffic is minimal. nProbe [21] is a NetFlow/IPFIX probe that claims to support the monitoring of GTP traffic. nProbe is available as a stand-alone software application or as an embedded system named nBox. Further analysis of nProbe shows that the GTP monitoring capabilities of nProbe are limited to identifying the *Tunnel Identifier* in the GTP header and the ability to identify the tunneled data rather than only the envelope (i.e. the GTP message). Another NetFlow/IPFIX-based monitoring tool is PMACCT [1]. PMACCT also supports the inspection of tunneled traffic, such as GTP. However, just like nProbe, this is limited to identifying the tunneled data.

None of the available flow-based monitoring solutions supports the analysis of GTP-C traffic, which covers most of the requirements in Chapter 3. In order to satisfy these requirements we have to develop a new dedicated flow-based GTP monitoring solution.

Chapter 5

GTP Packet Analysis

In order to fulfill the requirements in Chapter 3 we have to research how the required information can be extracted from the GTP traffic. This Chapter describes the structure of the GTP v.0 [4] and GTP v.1 [5][6] packet, both header and payload, and describes the fields that have to be analyzed in order to fulfill the requirements. For a more general description of GPRS and GTP we refer to Appendix 2.1.1.

Before describing the relevant fields inside the GTP packet, we first address some of the more general requirements in Chapter 3. One may notice that all requirements focus on traffic measurements per roaming partner, not per individual user. This implies that monitoring roaming traffic per individual user is not considered in this work. Traffic measurements per roaming partner means monitoring the streams of traffic from one roaming partner to and from other roaming partners. Looking at the GPRS architecture (as described in Appendix 2.1.1), this means monitoring the Gp interface, i.e. the interface between two Public Land Mobile Networks (PLMNs). Almost all traffic over the Gp interface is GTP traffic. GTP tunnels over the Gp interface always have two endpoints: the SGSN of one roaming partner and the GGSN of another roaming partner. One of the general requirements in Chapter 3 states that roaming partners should be identified by Autonomous System Number (ASN) rather than IP addresses. This is illustrated in Figure 5.1. The figure shows that the IP addresses of the tunnel-endpoints, i.e. the GPRS Support Nodes (GSNs), are mapped to the ASNs where the GSNs are located. As some service providers span multiple Autonomous Systems (ASs), some roaming partners can be identified by multiple ASNs.

Another requirement in Chapter 3 states that both inbound and outbound roaming should be monitored. Since the the Gp interface is the only

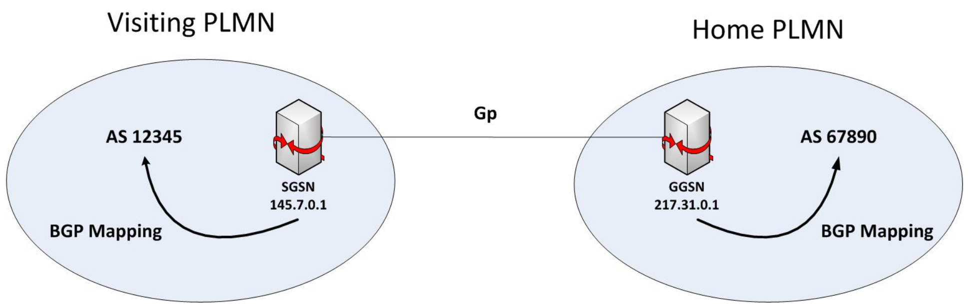


Figure 5.1: Identifying roaming partners by ASN instead of IP address.

interface between PLMNs, this requirement implies monitoring both traffic directions of that interface. In practice this means that one PLMN will be the source AS and another PLMN will be the destination AS. Which PLMN is considered source and which PLMN is considered destination will determine whether inbound or outbound roaming takes place.

5.1 GTP Header

This sub chapter describes how the relevant fields inside the GTP header can be identified. As described in Chapter 3, this work will focus on GTP v.0 [4] and GTP v.1 [5][6], GTP v.2 [18] is out of scope. The basic transfer unit in GTP is called a *message*. As the content and structure of the GTP v.0 and GTP v.1 messages slightly differ, the first step of measuring a GTP message is to determine the GTP version, because this will influence the rest of the GTP packet analysis. Besides that, identifying the GTP version is listed as one of the requirements. Table 5.1 illustrates the difference between the GTP v0 and GTP v1 header. The fields which are of interest are marked bold. Both for GTP v.0 as for GTP v.1 holds that the GTP header is the same for both GTP-C and GTP-U messages.

One of the major differences between the GTP v.0 and GTP v.1 header is that the GTP v.0 header has a fixed length, while the GTP v.1 header has a variable length. The length of the GTP v.0 header is always 20 bytes, while the GTP v.1 header has a minimum length of 8 bytes. The length of the header is crucial, because it determines the payload offset. There are three flags that are used to signal the presence of optional fields: the PN flag, the S flag and the E flag. These flags indicate the presence of the *N-PDU Number* field, the *Sequence Number* field and the *Next Extension Header*

	Bits							
octets	8	7	6	5	4	3	2	1
1	**Version**			**PT**	Spare ‘1 1 1’			SNN
2	**Message Type**							
3-4	**Length**							
5-6	Sequence Number							
7-8	Flow Label							
9	SNDCP-N-PDULLC Number							
10	Spare ‘1 1 1 1 1 1 1 1’							
11	Spare ‘1 1 1 1 1 1 1 1’							
12	Spare ‘1 1 1 1 1 1 1 1’							
13-20	TID							

	Bits							
octets	8	7	6	5	4	3	2	1
1	**Version**			**PT**	*	**E**	**S**	**N**
2	**Message Type**							
3	**Length (1st Octet)**							
4	**Length (2nd Octet)**							
5	Tunnel Endpoint Identifier (1st Octet)							
5	Tunnel Endpoint Identifier (2nd Octet)							
5	Tunnel Endpoint Identifier (3rd Octet)							
5	Tunnel Endpoint Identifier (4th Octet)							
9	Sequence Number (1st Octet) 1) 4)							
10	Sequence Number (2nd Octet) 1) 4)							
11	N-PDU Number 2) 4)							
12	Next Extension Header Type 3) 4)							

Table 5.1: GTP v.0 Header, copied from [4] (left) and GTP v.1 Header, copied from [5] (right).

Field field, respectively [5]. None of these fields itself is of our interest, because they contain no information needed to satisfy the requirements. Only their presence is important, because they influence the payload offset.

In the remainder of this chapter the GTP message will be discussed in general, meaning the descriptions are valid for both GTP v.0 messages and a GTP v.1 messages, unless stated otherwise.

The *Protocol Type (PT)* field inside the GTP header is used as a protocol discriminator between ordinary GTP (value “1”) and GTP’ (value “0”). The GTP’ (*prime*) protocol is used to transfer charging data to the Charging Gateway Function in the GPRS core network. GTP’ is outside the scope of this work. Therefore the *Protocol Type* field should only be used to filter out the GTP’ messages.

Another important field in the GTP header is *MessageType*. This field identifies the type of GTP message and is present in every GTP message. For the GTP-C traffic we are interested in six message types, as shown in Table 5.2. These message types are also explicitly mentioned in the requirements of Chapter 3. For a description of the messages, see Appendix 2.1.1. Besides counting the number of occurrences of these messages, these messages also provide all the necessary information to satisfy the other GTP-C requirements. This will be further described in the next sub-chapter.

For GTP-U there is only one message type in which we are interested,

Hex.code	MessageType	GTP-C	GTP-U
0x10	Create PDP Context Request	X	
0x11	Create PDP Context Response	X	
0x12	Update PDP Context Request	X	
0x13	Update PDP Context Response	X	
0x14	Delete PDP Context Request	X	
0x15	Delete PDP Context Response	X	
0xFF	G-PDU		X

Table 5.2: Relevant GTP-C and GTP-U message types.

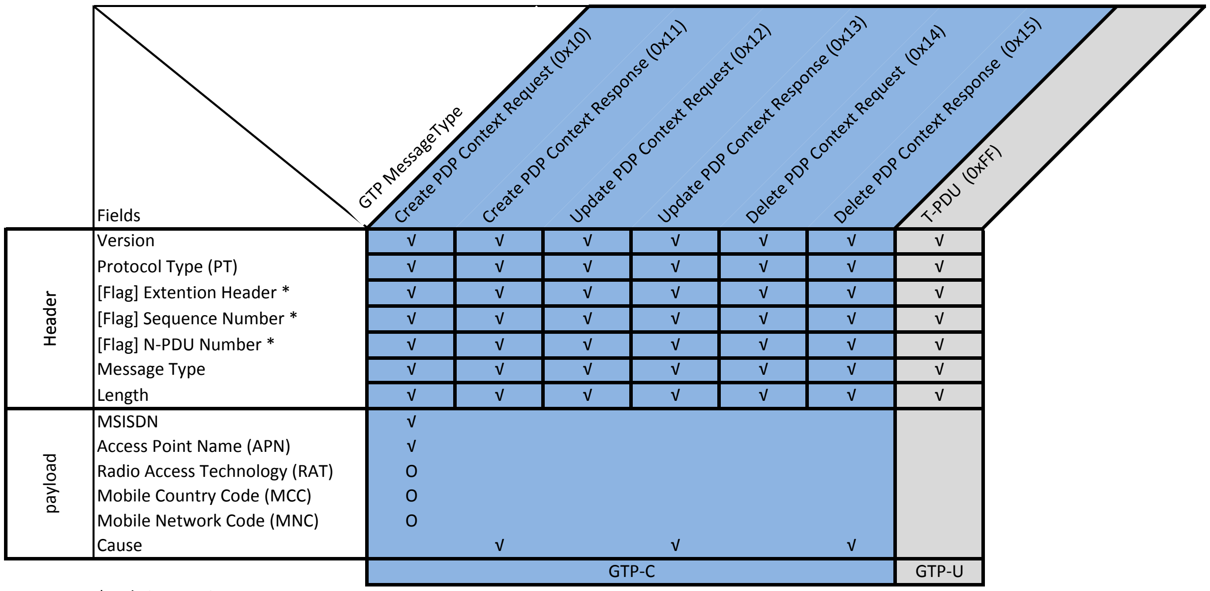
namely *GPD-U*. GPD-U messages contain message type 255 and consists of a GTP header and a *T-PDU* as payload. The T-PDU is the original packet, e.g., an IP datagram, from the MS or a network node in an external Packet Data Network (PDN). A T-PDU is the original payload that is tunneled in the GTP-U tunnel [9].

In this work *MessageType* is also used to differentiate between GTP-C and GTP-U. 3GPP 29.060 [9] shows that almost all GTP message types are used by GTP-C, only a few are used by GTP-U. As described above, the only GTP-U message type we are interested in is G-PDU. This makes it is easy for us to differentiate by GTP *MessageType*. Another option would have been to differentiate by UDP port number, since this number is different for GTP v.0 and GTP v.1. GTP v.0 uses port number 3386, while GTP v.1 uses 2123 for GTP-C and 2152 for GTP-U. Why *MessageType* is used and not the UDP port number, will be explained in Chapter 6.

Figure 5.2 provides an overview of relevant header fields as discussed in this section and their presence inside the relevant GTP message types listed in Table 5.2. Figure 5.2 shows all listed header fields are present in every GTP message type, because of the fact these header fields are all mandatory. In the next section the fields in the payload of the GTP message will be discussed. These fields are not all mandatory for every GTP message type.

5.2 GTP Payload

This section will describe the relevant fields inside the GTP payload, that are needed to satisfy the requirements as stated in Chapter 3. Note that all relevant fields in the GTP payload are in GTP-C messages, as the payload of a GTP-U messages carries the actual user data (T-PDUs), as described



	Fields \ GTP MessageType	Create PDP Context Request (0x10)	Create PDP Context Response (0x11)	Update PDP Context Request (0x12)	Update PDP Context Response (0x13)	Delete PDP Context Request (0x14)	Delete PDP Context Response (0x15)	T-PDU (0xFF)
Header	Version	√	√	√	√	√	√	√
	Protocol Type (PT)	√	√	√	√	√	√	√
	[Flag] Extention Header *	√	√	√	√	√	√	√
	[Flag] Sequence Number *	√	√	√	√	√	√	√
	[Flag] N-PDU Number *	√	√	√	√	√	√	√
	Message Type	√	√	√	√	√	√	√
	Length	√	√	√	√	√	√	√
payload	MSISDN	√						
	Access Point Name (APN)	√						
	Radio Access Technology (RAT)	O						
	Mobile Country Code (MCC)	O						
	Mobile Network Code (MNC)	O						
	Cause		√		√		√	
		GTP-C						GTP-U

* only in case GTP v.1
√ = Always Present
O = Optional

Figure 5.2: Overview the relevant GTP fields and their presence inside the various GTP messages types.

in the previous section.

Figure 5.2 gives an overview of all relevant GTP fields inside the GTP header and GTP-C payload. The major difference with the header fields, that are described in the previous section, is that not all fields inside the GTP payload are mandatory; some are optional. Besides that, the payload fields are present in only some of the GTP message types. Figure 5.2 shows which fields are always present, i.e. mandatory, by marking them with "V" and which fields are optional, by marking them with "O". If a field is not marked at all for a specific GTP message type, than that field is either not present or not of interest for answering the requirements.

One of the requirements in the GTP-C section of Chapter 3 states that the *cause* value should be identified in a Create, Update and Delete PDP Context Response message, respectively. This *cause* field indicates whether a corresponding request is accepted or gives a reason for rejection. The *cause* field is always present for these type of messages and is located in the payload of the GTP-C packet.

As described in Appendix 2.1.1 a MS can set up multiple parallel PDP Contexts: one primary and multiple associated secondary PDP Contexts. To satisfy the requirement that states that the system should be able to monitor the number of unique Create PDP Context Request messages per roaming partner, we should pick a value should that uniquely identifies a MS. For this, *MSISDN* is used, which is more commonly known as the mobile telephone number. An alternative would be to use *IMSI*, which is the unique identifier of the SIM inside the MS. The benefit of using MSISDN over IMSI is that the MSISDN is present in every primary Create PDP Context Request message, while IMSI is optional.

To identify the Access Point Name (APN) a MS connects to, the *APN* field in the Create PDP Context Request message should be analyzed. This field is present in every primary Create PDP Context Request message. In secondary PDP Context request messages this field is left out, because secondary PDP Contexts reuse the APN of the associated primary PDP Context.

The *Radio Access Technology (RAT)* field inside the Create PDP Context Request message indicates the network the MS is connected to. This can be either GERAN (GSM) or UTRAN (UMTS). According to 3GPP 29.060 [5] this field is optional, which means some SGSNs/GGSNs will not include this field.

The Mobile Country Code (MCC) and Mobile Network Code (MNC) indicate the location of the SGSN the user is connected to: the country and the mobile network, respectively. For example, for KPN these values are 204 (MCC) and 08 (MNC). Both MCC and MNC fields are two bytes long and can occur in two different fields inside the GTP-C payload: in the *User Location Information* field and/or in the *Routing Area Indentity (RAI)* field. According to 3GPP 29.060 [5] both fields are not mandatory, so they may not occur in every GTP message. Usually either the *User Location Information* field or the *Routing Area Indentity (RAI)* field is present in a GTP message. However, in 3% of the observed traffic neither of these two fields were present, which implies that MCC and MNC values are unknown for these messages. For example, SGSNs of KPN Mobile The Netherlands B.V. never include the *Routing Area Indentity (RAI)* field, but always include the *User Location Information* field. It does not make any difference from which field the MCC and MNC are extracted, because both fields should contain the same information.

For most of the requirements in Chapter 3 it is sufficient to count the number of GTP messages, which corresponds to the number of observed packets, as every packet can carry at most one GTP message. For the requirements regarding the size of the GTP-U traffic, we need to count the number of bytes of the GTP message.

Chapter 6

Flow-Based Architecture

In Chapter 5 we determined the relevant fields inside the GTP packet. In this chapter we will define how this information can be measured and exported in a flow-based manner. The different processes in the IPFIX architecture, as displayed in Figure 6.1, will be used as guidance in this Chapter. First the IPFIX Metering Process and its associated Flow Key are described. After that, the IPFIX Exporting Process is described. The last section of this chapter describes the IPFIX Collecting Process. For a general description of IPFIX, see Chapter 2.2.

6.1 Metering Process

First step in setting up a flow-based solution is defining the format of the Flows in the Metering Process, i.e. defining which packets belong to the same Flow. Packets belonging to a particular Flow have a set of common properties. This set of common properties is referred to as the Flow Key. In other words, key fields contibute to the uniqueness of the flow, whlie non-key fields do not. In Chapter 5 we defined the fields that need to be extracted from the GTP packet in order to satisfy the requirements in Chapter 3. As described in Chapter 5 not all these fields need to be exported; the fields *Length* and *Protocol Type (PT)* fields, together with the *PN*, *S* and *E* flags are solely used to filter out irrelevant packets. The remaining fields, which are defined in Chapter 5, need to be exported. For these fields we have to define which are key and which are non-key. This is a very crucial step, because if certain fields are incorrectly marked as non-key, certain information may get lost due to the aggregation, while marking every field as key may result in a big and inefficient flow cache in the IPFIX Device. For each relevant field we will determine if the field is key or non-key, keeping in mind the consequences of this decision for the requirements. *Source ASN* and *Destination ASN* should be definitely marked as key, because these represent the endpoints of the connections we want to measure. *gtpVersion* should also be marked as key, because a roaming partner can use two versions of GTP simultaneously, e.g., when a roaming partner has both an UMTS and an LTE network. If *gtpVersion* would be marked as non-key, we would not be able to differentiate between different versions of GTP traffic, because different versions of GTP traffic would account to the same flow. The header field *Message Type* should also be marked as key. If *Message Type* was non-key, it would be impossible to count the number of GTP messages per message type, since all message types would account to the same flow. The same reason holds for the *Cause* field; this field should be marked as key, because if it is marked as non-key it will be impossible to count the number of messages per cause value. One of the requirements states that the system should be able to count the number of unique PDP Context Create Messages per roaming partner. This means *MSISDN*, which we use as unique identifier for a Mobile Station (MS), should be marked as key, otherwise the system would only be able to count the total number of PDP Context Request messages per roaming partner and not filter out duplicates from the same

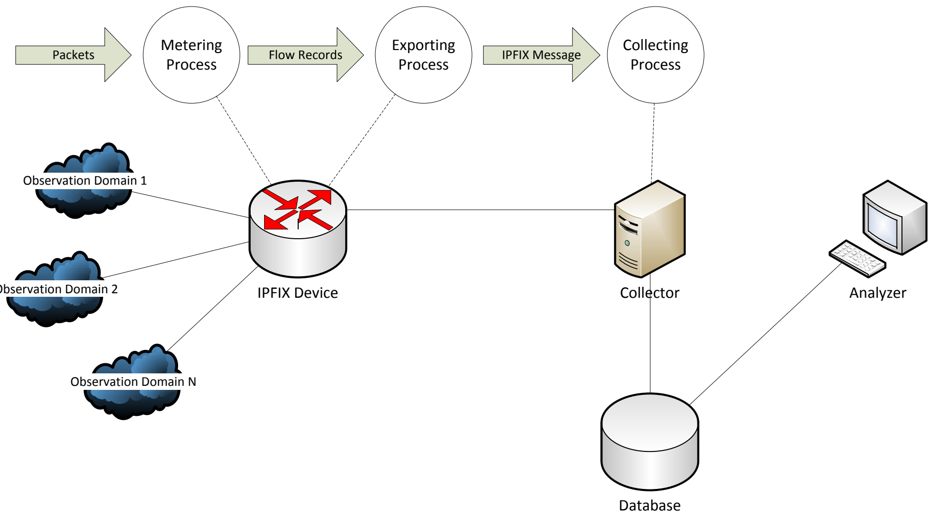


Figure 6.1: Mapping between the functional and physical IPFIX architecture.

MS. *APN* should be marked as a key field in order for the system to count the number of bytes per APN. Same reason holds for *RAN*; this field should be marked as key in order to count the number of bytes/packets per Radio Access Network. *MNC* and *MCC* should be marked key, because otherwise the system will not be able to count the number of bytes/packets per mobile network and county, respectively. In first instance, MNC might appear to be a one-on-one mapping to the ASN, which would mean it does not contribute to the uniqueness of the key and thus could be marked as non-key. However, this is not always the case, e.g, Mobile Virtual Network Operators (MVNOs) can use ASN of their host operator, but have their own MNC.

Together with packet and byte counters, the following Key and Non-Key fields are defined.

Key Fields:
Source ASN, Destination ASN, Version, MessageType, Cause, MSISDN, APN, RAT, MNC, MCC

Non-Key Fields:
Total Bytes Transferred, Total Packets Transferred

In Chapter 5 we described that we also use the *MessageType* field for differentiating between GTP-C and GTP-U traffic. In case we would differentiate on UDP port-number, this would mean an extra field, *udpPortNumber*, should be added to the key. As described earlier, we choose to keep the flow-key as small as possible, because we want the Flow Cache to be fast and efficient. Due to this reason, we use *MessageType* and not the UDP port number for differentiating between GTP-C and GTP-U traffic.

As almost all fields, except the packet and byte counters, are Key fields, we suggest using one single Flow Key. An alternative would be to use multiple smaller Keys, e.g., one per requirement. However, this would mean a lot of redundant information will be transmitted, because certain fields, like Source ASN and Destination ASN, would occur in every Flow Key. A more efficient way is to use one Flow Key and one Data Template. Once received, these Flows can then be sub-aggregated by the Collector, which means the Collector creates multiple views of the same Flows. The consequence of using a single Flow Key is that in some Flow Records not all key fields will be recorded with a meaningful value. For example, flows with MessagType 0xff (GTP-U traffic) will never contain a value for Cause, MSISDN, APN, RAT, MNC or MCC, because these fields are not present in GTP messages of that message type. For these cases we suggest recoding a default value of 0 in the corresponding key-field. Taking zero as default is a safe choice, because according to the GTP standard [5] non of the fields we marked as key, can ever contain a meaningful value of 0. The only exception is the Version field, since all GTP v.0 messages will have the version field set to 0. For this we suggest using the maximum value for a single byte (255), since this value will never occur in normal traffic. Note that setting a default value for Version and other fields in the GTP header should not even be necessary, because these fields occur in every GTP message.

The Metering Process maintains a database of all the Flow Records, often referred to as a flow cache. This flow maintenance includes creating new Flow Records and updating existing ones. Once a Flow is expired the Metering Process will forward the Flow Records to the Exporting Process. For long-running Flows, the Metering Process should expire the Flow on a regular basis or based on some expiration policy. This periodicity or expiration policy should be configurable at the Metering Process [13].

6.2 Exporting Process

The previous section described how observed packets are classified into flows based on a number of selection criteria. In the IPFIX architecture [13] this is referred to as the Metering Process. As displayed in Figure 6.1, the next step in the IPFIX architecture is the Exporting Process, which is responsible for sending the Flow Records to a Collector. This Exporting Process runs on the same device as the Metering Process, often referred to as the IPFIX Device. As described in Chapter 2.2, the transport of Flow Records from the IPFIX Device to the Collector is done by using IPFIX *Messages*. This section describes the IPFIX Exporting Process and its associated Templates and Information Elements (IEs).

IPFIX Messages can either hold Templates or Data Sets. Templates describe the structure and semantics of the information in the Data Sets. Information inside a Message is modeled in terms of Information Elements (IEs). In order to export our Data Sets to a Collector, we first need to specify a Data Template. Considering all data in the flow cache is stored according to one single Flow Key, which we defined in the previous section, we also need one single Data Template that describes how the key and non-key fields are encapsulated into Data Sets. For this we need to map every field, key and non-key, to an Information Element. This mapping, often referred to as IE encoding, can either be done in the Metering Process or the Exporting Process [22]. Currently there are no IEs defined for GTP in the IANA registry [15]. The only two key fields that can be mapped to an

ElementID	Name	Data Type	Data Type Semantics	Description
1	octetDeltaCount	unsigned64	deltaCounter	The number of octets since the previous report (if any) in incoming packets for this Flow at the Observation Point. The number of octets includes IP header(s) and IP payload.
2	packetDeltaCount	unsigned64	deltaCounter	The number of incoming packets since the previous report (if any) for this Flow at the Observation Point.
16	bgpSourceAsNumber	unsigned32	identifier	The autonomous system (AS) number of the source IP address. If AS path information for this Flow is only available as an unordered AS set (and not as an ordered AS sequence), then the value of this Information Element is 0.
17	bgpDestinationAsNumber	unsigned32	identifier	The autonomous system (AS) number of the destination IP address. If AS path information for this Flow is only available as an unordered AS set (and not as an ordered AS sequence), then the value of this Information Element is 0.

Table 6.1: IANA assigned IEs used for IE encoding.

IANA-assigned IE are Source ASN and Destination ASN. For this we can use the IANA-assigned IEs *bgpSourceAsNumber* and *bgpDestinationAsNumber*, respectively. For the non-key fields Total Bytes Transferred and Total Packets Transferred, we can use *octetDeltaCount* and *packetDeltaCount*, respectively. Table 6.1 gives an overview of the structure and semantics of these IANA-assigned IEs. The information in the table is taken from the IANA registry [15].

For the remaining fields inside the flow cache, there are no IANA-assigned IEs, because they are all specific to GTP. That means every remaining field in the flow cache should be mapped to an self-defined enterprise-specific IE. For this we use the mechanism for encoding IE Type Information, proposed in RFC 5610 [23]. Table 6.1 provides an overview of the Type Information of the self-defined enterprise-specific IEs. Please note, the IE Type Information, i.e. the colums of the table, is the same for the enterprise-specific

ElementID	Name	Data Type	Data Type Semantics	Description	Range
10	gtpVersion	unsigned8	identifier	The GTP version field inside the GTP header.	0-2
11	gtpMessageType	unsigned8	Identifier	The message type field in the GTP header indicating the type of GTP message.	0-255
12	gtpCause	unsigned8	identifier	The cause field inside the GTP payload indicating whether a request is accepted or the reason for rejection.	0-255
13	gtpRAT	unsigned8	identifier	The Radio Access Technology (RAT) field in the GTP payload.	0-5
14	gtpAPN	string	default	A Unicode string containing a human readable representation of the Access Point Name field inside the GTP payload.	
15	gtpMSISDN	unsigned64	identifier	The MSISDN field in the GTP payload indicating the MSISDN of the Mobile Station.	
16	gtpMNC	unsigned16	identifier	The MNC field inside the GTP payload indicating the Mobile Network Code of the SGSN where the Mobile Station is registered.	
17	gtpMCC	unsigned16	identifier	The MCC field inside the GTP payload indicating the Mobile Country Code of the SGSN where the Mobile Station is registered.	

Table 6.2: Self-defined enterprise-specific IEs.

IEs as for the IANA-assigned IEs displayed in Table 6.1. Most of the Type Information in Table 6.2 is self-explanatory, for the remaining fields we will provide a short motivation. The *Data Type* field corresponds to the data type of the corresponding field in the GTP message. Almost all IEs are set to data type Unsigned$x$ because the corresponding fields in the GTP message are non-negative Integers, for which a data storage of x bits is required. The data type of *gtpAPN* is *String*, because the value of this field will always be a Unicode String. The *Data Type Semantics* field defines the semantics of the, previously described, *Data Type* field. In our case all data types represent an certain *Identifier* inside the GTP packet, except the IE *gtpAPN*, which has data type semantics *default*, because this is mandatory for String data types [23]. The last column in Table 6.2 specifies the *Range* for those IEs which can only hold a specific set of values. For the corresponding IEs in the table, these ranges are specified in the various 3GPP standards for GTP.

In order to export the enterprise-specific IEs in Table 6.2, these IEs should be scoped to a Private Enterprise Number (PEN). To export type information about an IANA-assigned IE, there is no need to export a PEN or the PEN can be set to zero [23]. However, when exporting type information about an enterprise-specific IE, the PEN should be exported. This PEN should be registered by IANA [24]. We use one of the Private Enterprise Numbers of the University of Twente, namely 785. Since this PEN has also been used for other work, other enterprise-specific IEs are also scoped to this PEN. That is why our first enterprise-specific IE, *gtpVersion*, starts with 10 instead of 1.

After having defined all the individual IEs in Table 6.2 we can define the Data Template that is needed to inform the Collector about the information that will be sent in the Data Sets. The Data Template is shown in Table 6.3. The format of the table corresponds to the byte structure of the Data Template, meaning every row contains four bytes, as displayed on top of the table. The *Set ID* is set to 2, which is standard for Data Templates. The *Length* indicates the total length of the Template in bytes. *Template ID* is set to 256, which is the first Set ID that can be used to identify templates that describe data sets. All Data Sets that are described by this Template have the same value for *Set ID* to indicate the data records inside the Data Set are described by this Template. A Template is essentially an ordered list of IEs. The *Field Count* describes the number of IEs in the Template. What follows is the actual list of IEs. First the IANA-assigned IEs, which have the Enterprise bit set to zero, followed by the enterprise-specific IEs, which have the Enterprise bit set to one. For each IE in the Data Template, from left to right the Enterprise bit, name, ElementID and field length are defined. For the enterprise-specific IEs also the PEN to which they are scoped, is defined. All field lengths in the template are relatively small, except the length of the *gtpAPN* field. The IPFIX Template mechanism is optimized for fixed-length Information Elements [16]. However, some fields like *gtpAPN* can vary considerably in length. For these variable length fields IPFIX offers the variable-length IE [16]. This means the *Field Length* in the Data Template is recorded as 65535. This reserved length value notifies the Collecting Process, that the actual length of this IE will be recorded in the content of this IE inside the Data Set.

However, a Collector receiving Data Sets described by this Template,

Bits 0..15			Bits 16..31
Set ID = 2			Length = 88
Template ID = 256			Field Count = 12
0	octetDeltaCount	ID = 1	Field Length = 4
0	packetDeltaCount	ID = 2	Field Length = 4
0	bgpSourceAsNumber	ID = 16	Field Length = 2
0	bgpDestinationAsNumber	ID = 17	Field Length = 2
1	gtpVersion	ID = 10	Field Length = 1
PEN = University of Twente - TIOS (785)			
1	gtpMessageType	ID = 11	Field Length = 1
PEN = University of Twente - TIOS (785)			
1	gtpCause	ID =12	Field Length = 1
PEN = University of Twente - TIOS (785)			
1	gtpRAT	ID = 13	Field Length = 1
PEN = University of Twente - TIOS (785)			
1	gtpAPN	ID = 14	Field Length = 65535
PEN = University of Twente - TIOS (785)			
1	gtpMSISDN	ID = 15	Field Length = 18
PEN = University of Twente - TIOS (785)			
1	gtpMNC	ID = 16	Field Length = 2
PEN = University of Twente - TIOS (785)			
1	gtpMCC	ID = 17	Field Length = 2
PEN = University of Twente - TIOS (785)			

Table 6.3: IPFIX Data Template.

can only treat the enterprise-specific IEs in this Template as opaque octets, because the Collector still only knows the name of these IEs, not their meaning. For the IANA-assigned IEs we can assume an IPFIX Collector knows hows to interpret them, but for the enterprise-specific IEs the Collector certainly does not know how to interpret them. In order to solve this problem RFC 5610 [23] defines an Information Element Type Options Template [23], which can be sent to the Collector, followed by IE type records for each enterprise-specific IE in the Data Template. The function of the IE Type Options Template is to inform the Collector about the type information in the type records by using IANA-assigned IEs. The relation between IE Type Options Template and type records is comparable to the relation between Data Template and Data Sets. Figure 6.2 shows a sequence diagram of the different IPFIX *messages* sent between Exporting Process and Collecting

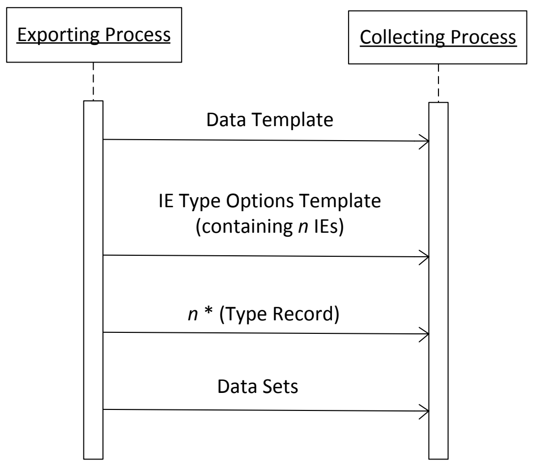


Figure 6.2: Sequence of IPFIX messages between Exporting Process and Collecting Process.

Process. As can be seen in the figure, the IE Type Options Template and associated type records are sent after the Data Template, but before the Data Sets.

Table 6.4 gives an overview of the structure of the Information Element Type Options Template. The *Set ID* is set to 3, which is the standard value for Options Templates. The *Length* indicates the total length of the Template in bytes. The *Template ID* is set to 257, which is one more than the Template ID of the Data Template, see Table 6.3. Next field in the Template is the Field Count, which specifies the number of IEs in the Template. The *Scope Field Count* defines the number of *Scope Fields* inside the Template, which are normal Fields except that they are interpreted as scope by the Collector. The scope uniquely identifies the reported IEs in the data records [16]. In this case the combination *privateEnterpriseNumber* and *informationElementID* defines the scope for our IEs. What follows is a list of IEs followed by their length. The IEs are chosen in a manner that all headers in Table 6.2 can be referenced by this Options Template. For all these IEs the Enterprise bit is set to zero, because they are all IANA-assigned (otherwise the Collector would still not know how to interpret these IEs). Next to the name of every IE the *ElementID* is listed. This ElementID is the ID as it is registered by IANA. The *Field Length* which is listed after each IE indicates the number of bytes that are needed in the type records. This value is chosen

Bits 0..15	Bits 16..31		
Set ID =3	Length = 42		
Template ID = 257	Field Count = 7		
Scope Field Count = 2	0	privateEnterpriseNumber	ID = 346
Field Length = 4	0	informationElementID	ID = 303
Field Length = 2	0	informationElementDataType	ID = 339
Field Length = 1	0	informationElementSemantics	ID = 344
Field Length = 1	0	informationElementRangeBegin	ID = 342
Field Length = 8	0	informationElementRangeEnd	ID = 343
Field Length = 8	0	informationElementName	ID = 341
Field Length = 65535	0	informationElementDescription	ID = 340
Field Length = 65535			

Table 6.4: IPFIX Information Element Type Options Template.

in a manner that the field length is enough to hold the maximum possible value of the IE. This field length is calculated by looking at the *Data Type* in the IANA registry. For example, the IE *informationElementID* has *Data Type* unsigned16, which means that two bytes are needed to store this value. For the IEs *informationElementName* and *informationElementDescription* variable lenght encoding is used, which we used earlier for the *gtpAPN* field in the Data Template. This is visualized in the Template by recording the Field Length as 65535.

After sending the IE Type Options Template, the associated type records that describe the enterprise-specific IEs need to be sent. As can be seen in Figure 6.2, one type record is sent for every IE in the IE Type Options Template. Table 6.5 gives an example of a type record. In this table we used the IE *gtpVersion* as example. Please note, that all IEs in the IE Type Options Template of Table 6.4 also appear in the type record. The field length of the fields in the type record corresponds to the field length specified in the Template. The field lengths for variable-length IEs *informationElementName* and *informationElementDescription* are in this example set to an example value, because their actual length depends on the number of bytes the Unicode string values will take. Note that the actual length of these two variable-length IEs appear in the first octet of the IE content, as displayed in Table 6.5. Table 6.5 gives the type record for the IE *gtpVersion*, similar type records have to be sent for all other enterprise-specific IE in the Data Template.

Bits 0..15	Bits 16..31	
Set ID = 257	Length = 44	
PEN = University of Twente - TIOS (785)		
ID = 10	Data Type = 0x01	Data Type Sem.= 0x04
RangeBegin = 0		
RangeEnd = 2		
Length = 7	Name = gtpVersion	
Length = 11	Description = GTP version field inside the GTP header	

Table 6.5: Example Type Record for the enterprise-specific IE *gtpVersion*.

6.3 Collecting Process

When receiving the Data Template and subsequently the IE Type Options Template and the associated type records, the Collector has enough information to interpret the Flow Records in the Data Sets. However, in practice we see real IPFIX Collectors are rarely deployed. Most IPFIX Collectors that are currently available are not able to interpret enterprise-specific IEs, only IANA-assigned IEs are supported. These Collectors often do not support the learning of IEs via IE Type Options Templates. In practice this means the structure and semantics of enterprise-specific IEs are often hard-coded in the Collector. Examples of known IPFIX Collectors are Vermont [25], nTop [26], ipfixcol [27], ripfix [28] and Scrutinizer [29]. We did not verify whether these Collectors fully supports the requested functionality, i.e. IE Type Options Templates and variable length IEs, since the Collecting Process is not part of the focus of this work. Besides that, the choice for a Collector really depends on the requested data analysis.

For the storage of Flow Records at the Collector, we propose a storage mechanism similar to the flow cache in the Metering Process. The storage format is variable; it can be a memory table, a database or even flat-files. The choice is implementation specific and mainly depends on the requirements that are posed on the data analysis.

Chapter 7

Implementation and Validation

In the previous chapter we proposed a flow-based solution for monitoring GTP traffic using IPFIX. In this chapter we will demonstrate the feasibility of that solution by setting up a proof of concept. The KPN testcenter will be used as a test environment for our proof of concept. The proof of concept will be validated using a captured packet-trace.

7.1 Proof of Concept

After defining an IPFIX solution for monitoring of GTP traffic in Chapter 6, we will now set up a proof of concept to demonstrate the feasibility of this solution. As described in Chapter 4 there are currently no flow-based monitoring tools available that fully support the monitoring of GTP-C and GTP-U traffic. Existing flow-based monitoring tools, like nProbe and PMACCT, are only able to analyze the traffic inside the tunneled packet, often referred to as looking inside the envelope. This means that in order to test our proposed solution we either have to develop a new flow-based monitoring tool or develop a software extension to an existing flow-based monitoring tool. We choose the latter, because a lot of functionality in existing tools can be re-used. PMACCT [1] was picked as the tool for which we developed a software extension, because it offers BGP peering functionality, which is required for mapping IP addresses to Autonomous System Numbers (ASNs). Besides that, KPN suggested PMACCT, due to good experiences in the past.

The functionality of the Metering Process and Exporting Process was implemented nearly fully compatible with the specifications of the proposed solution, as specified in Chapter 6. The Metering Process, including the associated Flow Cache, was implemented in the PMACCT deamon, *pmacctd*, as shown in Figure 7.1. In order to ensure only GTP traffic is being accounted we provided *pmacctd* with a pcap filter. In this way all packets that do not contain a GTP message will be filtered out. The format of the pcap filter is listed below. By applying this pcap filter, *pmacctd* only analyzes traffic that uses the transport ports defined for GTP, as described in Chapter 5. By adding *vlan* to the pcap filter, we also account GTP traffic that is carried by a IEEE 802.1Q VLAN packet.

```
port 2123 or port 2152 or port 3386 or (vlan and (
    port 2123 or port 2152 or port 3386))
```

The Exporting Process was implemented in the Netflow/IPFIX probe, *nfprobe*, which is a plugin to *pmacctd*. The only functionality that was left out of the Exporting Process was the IE Type Information Options Template and its associated type records. Instead, we implemented this functionality into the Collecting Process.

The PMACCT Collector, *nfacctd*, is used for the collecting of our Flow Records. As mentioned earlier, the export of the IE Type Options Template and associated type records was left out. This was done, because nfacctd currently lacks the ability to handle IE Type Options Templates. Instead of

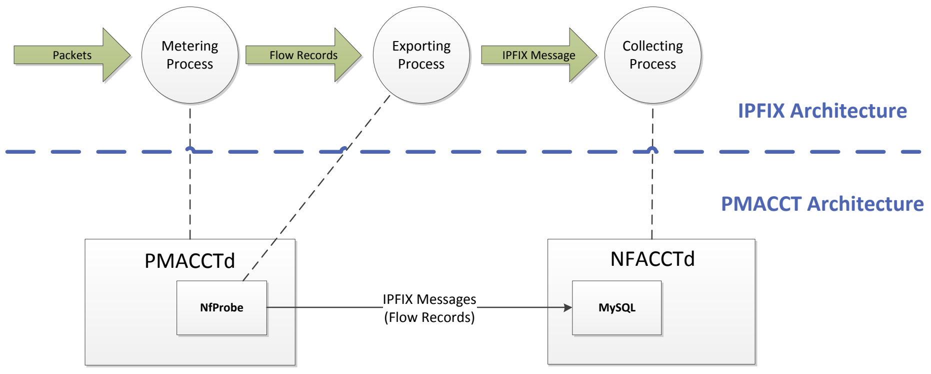


Figure 7.1: Mapping between the IPFIX architecture and the PMACCT architecture.

this, the structure and semantics of the enterprise-specific IEs, as described in Table 6.2, were hard-coded in the PMACCT Collector. In this way the Collector still knows how to handle data sets containing these enterprise-specific IEs. We did implement the variable length encoding functionality for IEs, as described in Chapter 6. So when the PMACCT Collector receives a Data Template containing an IE with Field Length recorded as 65535, it knows the actual length of this IE is recorded in the first octet of corresponding IE in the Data Set. Due to time constraints we decided to focus on aspects that directly influence the performance of the proposed solution, like the variable-length IEs, and leave the functionality for the Type Information Options Template for future work. For the storage of Flow Records a MySQL database is used, which is implemented in the *MySQL* plugin of *pmacctd* (see Figure 7.1). We chose for MySQL, because it is free and relatively easy to implement compared to other relational databases. Besides that, it was recommended by PMACCT as a suitable storage format.

The source code of the developed extension is stored in a CVS repository and can be donwloaded from the following location:

```
CVSROOT=:pserver:anonymous@cvs.pmacct.net:2403/home/repo-0.14-gtp

Password: pmacct
```

Later this year the source code will be integrated into the mainstream code of PMACCT.

7.2 Test Environment

In order to validate the proposed solution we tested our proof of concept at the KPN testcenter. A simplified overview of the test environment is displayed in Figure 7.2. In practice, the IPFIX Device and the Collector will most likely be implemented in different physical devices, because multiple IPFIX Devices will usually provide input to a single Collector. Since we just want to validate the correctness of our solution, the entire solution is build on a single server, which we will refer to this server as the "PMACCT server". This means both *pmacctd* and *nfacctd* run on the the same physical device, as shown in Figure 7.1. The transport of Flow Records from *pmacctd* to *nfacctd* will go via the loopback address of the PMACCT server. Table 7.1 gives a summary of the architecture of the PMACCT server.

In order to get the correct data, the point of measurement is crucial. As we are interested in packet-switched roaming traffic, the *BorderGateWay*

System:	HP ProLiant DL380 G5
CPU:	Intel Xeon 3000 Mhz Dual Core
Memory:	6 Gb ECC
Operating System:	Red Hat Enterprise Linux Server release 5.8

Table 7.1: PMACCT Server - System Specifications.

router of the GPRS core network seems the most logical point of measurement, since all inbound and outbound roaming traffic of KPN will pass through this router. The BorderGateWay router is the located on the edge of a GPRS core network and forms the connection between the home PLMN and other PLMNs. Since KPN does not directly peer with other roaming partners, all international packet-switched roaming traffic goes via a third-party GRX operator. A GPRS Roaming Exchange (GRX) operator acts as a hub for GPRS connections between roaming partners, removing the need for dedicated connections with every individual roaming partner. So for KPN

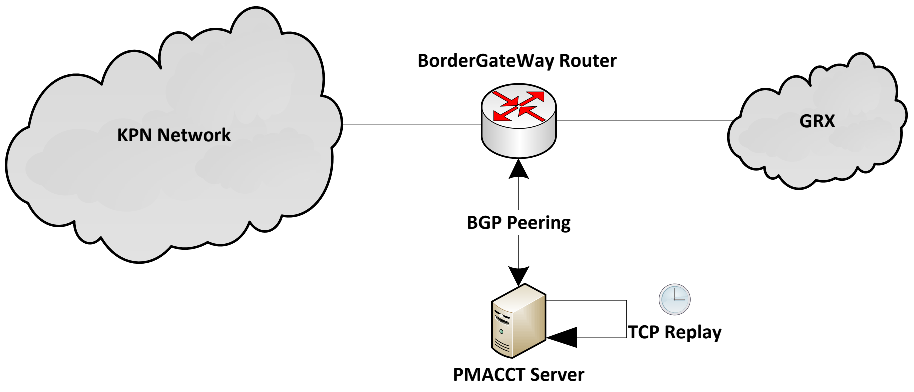


Figure 7.2: Experiment setup of the proof of concept at KPN testcenter.

the BorderGateWay router is the logical link between the home PLMN and the GRX operator, which makes it the best point of measurement for our solution. However, since our test environment is located at the KPN test-center and not in the live network of KPN, there is no live packet-switched roaming traffic passing through the BorderGateWay router. One possibility would be to generate GTP traffic using a roaming simulator. However, the drawback of using a roaming simulator is that it is only be able to simulate one roaming partner. Since we want to prove the correctness of our solution, it is important that the measured traffic is as realistic as possible. Due to this reason, we chose to a make a 100 MB packet-trace (approximately 260.000 packets) of the BorderGateWay router in the live network of KPN. This packet-trace contains packet-switched roaming traffic to and from various other mobile network operators KPN has roaming agreements with. By offering this packet-trace in a continuous loop to *pmacctd* we can simulate a continuous stream of GTP traffic. For this, we use the tool Tcpreplay [30], as shown in Figure 7.2. Tcpreplay echoes this packet-trace at a pre-configured speed on the Ethernet interface of the PMACCT server where *pmacctd* is listening. Another benefit of using Tcpreplay over a roaming simulator is that we have exact knowledge of the data that is being inputted to PMACCT, which makes validation easier.

Now we have simulated a steam of traffic, we still need to map the IP Adresses of the various roaming partners to Autonomous System Numbers (ASNs). Again the BorderGateWay routers seems to be a good solution, because it contains BGP information of all roaming partners KPN has roaming agreements with. However, since our test environment is located at KPN testcenter without real roaming partners, this BGP information is unavailable. By loading a copy of the BGP table of the BorderGateWay router in the live network of KPN (where the packet-trace was captured) into the BorderGateWay router in the testcenter, we acquired the necessary BGP information. By setting up a BGP peering between the BorderGateWay router and the PMACCT server, this BGP information will also be available at the PMACCT server, including BGP updates received at the BorderGateWay router. This BGP information can be used by *pmacctd* to perform the mapping of IP Addresses to ASNs. One drawback of using this static BGP table in the BorderGateWay router is that there will be no BGP updates. This may lead to future errors in mapping the IP addresses to ASNs when another packet-trace is used that contains unknown IP addresses. However, in a normal scenario, like the live network of KPN, these situations will not occur, because thanks to the BGP peering, the BGP table will be managed in a dynamic way.

7.3 Validation

In this section we will validate our proposed solution using the proof of concept, described in Chapter 7.1. The purpose of this validation is to prove that the Flow Records stored at the Collector contain the same information as the data that was monitored. That means on one hand no data should get lost, but on the other hand no extra data should be created. We will perform our validation using the packet-trace that was also used as input for Tcpreplay [30], as described in the previous section. We analyze the packet-trace using the tool Wireshark [31]. The Flow Records at the Collector are analyzed using MySQL client, because they are stored in a MySQL database. For the validation process we will take the results reported by Wireshark as a benchmark. In case the results reported by the Collector are approximately the same as the results reported by Wireshark, than we will assume these results are correct. This in turn would imply our validation is succeeded. The set of parameters that we measure corresponds to the relevant fields inside the GTP packet, which we defined in Chapter 5. Since these fields contain the information needed to satisfy the requirements of Chapter 3, we indirectly validate a number of these requirements.

Table 7.2 gives an overview of the measured parameters and their values in both the packet-trace, the Collector and the difference between these two. Ideally the values in the packet-trace and the values in the Collector are exactly equal. Since we decided to take the value of the packet-trace as a benchmark, the difference is denoted as the value in the Collector minus the value in the packet-trace. The difference is listed both as an absolute value and as a percentage.

The first parameter we measured is the total number of IP packets. Table 7.2 shows the value in the packet-trace is 2% larger than the value in the Collector. This is correct, because the PCAP filter, described in Chapter 7.1, filters out all IP packets that do not contain a GTP message. Note that the packet-trace is captured on the Gp interface of the GPRS core network of KPN, which almost solely transfers GTP traffic. Next parameter in Table 7.2 shows that the number of GTP messages in the Collector is 0,1% smaller than the value in the packet-trace. This means only 0,1% of the GTP messages is not accounted by our proof of concept. This small portion of non-accounted GTP messages plus the packets that do not contain a GTP message explains the difference between the total number of packets in the packet-trace and the total number of packets in the Collector (i.e. the first parameter). Further analysis of the packet-trace shows the non-GTP packets are IP packets that mainly contain Diameter, ICMP or DNS content.

The next parameter we measured was *gtpVersion*. As can be seen in Table 7.2, the number of packets containing the version field is both for the packet-trace and for the Collector equal to the number of packets that are reported as GTP. That is correct, because the version field is part of the GTP header and thus present in every GTP message. This also applies to *message type*, which is also a GTP header field. When looking at the number of GTP v.0 and GTP v.1 messages, Table 7.2 shows the values in the packet-trace and in the Collector are approximately equal. The slight difference is caused by the small portion of GTP messages that are not accounted, as described earlier. This is also applies to all other parameters in Table 7.2.

Summarizing our results, Table 7.2 shows that for all measured parameters the value in the packet-trace and the value in the Collector are approximately equal. Besides that, the results prove to be consistent if we repeat this validation procedure a number of times. Based on these findings we can conclude the parameters listed in Table 7.2 are measured correctly by our proof of concept. The slight differences between the values in the packet-

Measured statistics	**Packet-Trace**	**Collector**	**Difference**	
IP packets	257.660	252.507	-5153	(-2,0%)
GTP messages	252.883	252.507	-376	(-0,1%)
Messages that contain gtpVersion	252.883	252.507	-376	(-0,1%)
Messages with gtpVersion == 0	1680	1674	-6	(-0,4%)
Messages with gtpVersion == 1	251.203	250.833	-370	(-0,1%)
Messages that contain gtpMessageType	252.883	252.507	376	(-0,1%)
Messages with gtpMsgType == 0x10	402	401	-1	(-0,2%)
Messages with gtpMsgType == 0xFF	220.991	220.661	-330	(-0,1%)
Messages that contain gtpCause	1259	1256	-3	(-0,2%)
Messages with gtpCause == “0xC0”	78	78	0	(0%)
Messages that contain gtpMSISDN	402	400	-2	(-0,5%)
Messages with gtpMSISDN == “+316591002xxx”	12	12	0	(0%)
Messages that contain gtpAPN	402	401	-1	(-0,2%)
Messages with gtpAPN == “iphonekpn.nl”	16	16	0	(0%)
Messages that contain gtpRAT	396	395	-1	(-0,3%)
Messages with gtpRAT= 2	258	257	-1	(-0,4%)
Messages that contain gtpMCC	397	396	-1	(-0,3%)
Messages that contain gtpMNC	397	396	-1	(-0,3%)

Table 7.2: Comparison on different parameters between a packet-trace and the Collector’s Flow Table.

trace and the values in Collector, are all related to the small percentage (0,1%) of GTP messages that are not accounted by our proof of concept. Since this portion is so small, it is unknown which of the GTP messages causes this difference.

Chapter 8

Data Analysis

Data analysis can be regarded as the frond-end operation of traffic monitoring, while the focus of this work is on traffic measurement, which can be regarded as the back-end operation. To illustrate the applicability of a data analysis application, we will provide an example of how the Flow Records received at the Collector can be analyzed, instead of providing a full scale data analysis that satisfies every requirement in Chapter 3. For this example the network graphing tool Cacti [8] is used. Besides Cacti, there are other solutions available. Every data analysis applications has its own advantages and disadvantages. Cacti, for example, is not able to generate histograms. A histogram would be suitable to provide an overview of the ASs that generate the most traffic, i.e. the top talkers. Excel would be able to generate this kind of graphs, but is unable to perform online monitoring. Which data analysis application to choose depends on the requirements the end-user poses on the data analysis. We chose Cacti, because it offers an online monitoring functionality by using polling scripts, but more important: PMACCT [1] recommended Cacti as a flexible data analysis application, that runs well on top of PMACCT.

This chapter will describe the steps needed to set up graphs in Cacti and will discuss some of the example graphs we created. Note that the scripts, methods and templates described in this chapter are specifically designed for Cacti. The scripts can probably also be used as input for other data analysis applications, because they are regular Shell scripts. The methods and templates are cacti-specific. Other data analysis applications might use a similar approach, but a one-on-one copy will probably result in errors.

8.1 Cacti Configuration

All graphs in Cacti have to be matched to a certain object, which is called a *Device*. Since our solution monitors steams of traffic between two roaming partners, identified by source AS and destination AS, we suggest mapping every AS to a Device. In every observed flow either the source AS or the destination AS is the the operator that performs the monitoring. The other AS is the roaming partner with whom traffic is being exchanged. As common in BGP peering, the operator that performs the monitoring, i.e. the AS where the traffic is being observed, is being mapped to a zero. The other AS is the AS of the roaming partner. This means in every observed flow either source AS or destination AS will be zero, leaving the other AS to be the AS of the roaming partner. In Cacti we will map the AS of the roaming partner to a Device. This means we will create a different Device for every roaming partner that is being peered with.

For every Device, i.e. every roaming partner, a number of statistics can be monitored. In order to generate these statistics, the corresponding information has to be extracted from the Flow Records at the Collector. As described in Chapter 7, our proof of concept uses a MySQL database to store the received Flow Records. By performing SQL queries on this database, we can poll the corresponding flow data. One of the requirements in Chapter 3 states the system should provide online monitoring. This implies the flow data at the Collector should be polled in a nearly continuous manner and not by means of a once-per-day export. However, unless using a streaming query language, the data is never polled in a continuous manner. The reason for this is that alsmost all data analysis applications use some kind of polling interval, which automatically implies the data is being polled in batches. As long as the intervals are chosen small enough the data analysis approaches online monitoring. We use shell scripts containing SQL queries to poll the required flow data out of the MySQL database. Cacti polls this data every five minutes, which is the same interval as other RRD-based applications, like NfSen [32] use. One shell script can contain multiple queries. In Cacti these shell scripts are referred to as *Data Input Methods*.

After defining the monitored objects, i.e. *Devices*, and the scripts, i.e. *Data Input Methods*, to poll the data, next step is to define *Data Templates*. As mentioned before: Cacti uses shell scripts containing SQL queries to poll the data from the MySQL database at the Collector. The data that is being polled from this database is raw data. *Data Templates* are used to describe the context of this data and how to store it in the Round Robin Database (RRD), used by Cacti. For example, Cacti can store the current value of PDP Context Request messages for a certain roaming partner by polling this value from the MySQL database every five minutes or Cacti can just store the difference with the value of the previous polling cycle. Data in Cacti is being averaged over time. That means the graph that describes the results over the last month has a lower granularity than the graph that describes the data over the current day. Data Templates define the period for which the data should be maintained, e.g. only daily analysis or also weekly, monthly or yearly analysis. Data Templates are generic and can be applied to every device we want to monitor.

Last step in visualizing our results, is defining *Graph Templates*. Graph Templates define the structure and appearance of the graphs we want to create. For example, whether a graph is a line, a stack or something else; the meaning of the axis and a legend if required. Graph Templates use the data described by the Data Templates as input for making the graphs.

8.2 Example Graphs

This section will describe a number of example graphs created in Cacti. The corresponding *Data Input Methods*, *Data Templates* and *Graph Templates* of these figures can be found in Appendix A.

For the validation of our proposed solution, described in Chapter 7, we used Tcpreplay [30] to simulate GTP traffic. We could re-use the same packet-trace that we used for validation to simulate a constant stream of traffic, because Tcpreplay offers the *loop-option*, which allows the packet-trace to be played in a constant loop. However, this would result in graphs that show the exact same pattern over and over again. To bring more variation into the output data, we captured 10 packet-traces of 100 MB in the live network of KPN, which consist of roughly 550.000 packets each, and let Tcpreplay play them in a random order. Besides that, we provide Tcpreplay with a random speed-multiplier (between 0,01 and 1,01), so every packet-trace is played at a random speed. Note that a multiplier of 1 equals the original speed of which the packet-trace was captured with.

Figure 8.1 shows the total number of bytes being transferred to and from AS 64640 (network operator *Movistar*, Spain) over the last 24 hours. The traffic is shown from the perspective of KPN, that means “total bytes in” refers to the traffic from Movistar to KPN and “total bytes out” refers to the traffic from KPN to Movistar. As can be seen in the figure, the graph resets to zero at midnight. This is because the MySQL database at the Collector creates a new table every day. The graph shows the traffic over the last

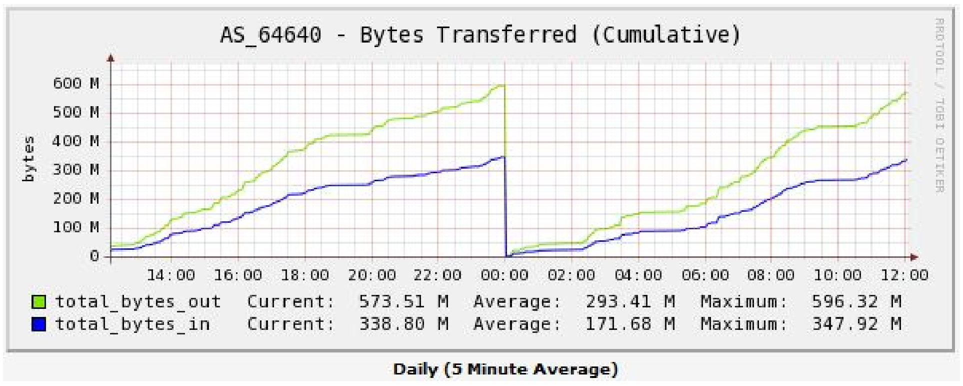
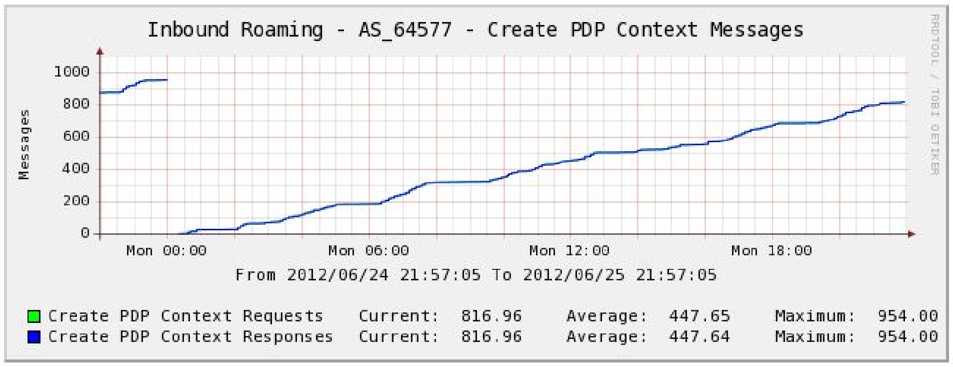


Figure 8.1: Bytes transferred between KPN and Movistar, Spain over the last 24 hours.

24 hours. As specified in the Data Template, Cacti also generates similar graphs for the last hour, week, month and year. As data is averaged over time, the granularity of these graphs reduces as the timespan grows larger.

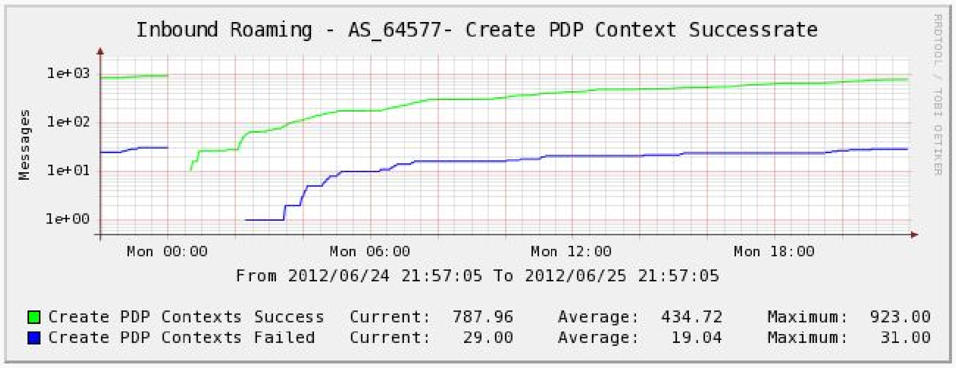
Figure 8.2a shows the number of Create PDP Context Request messages and the number of Create PDP Contexts Response messages for AS 64577 (network operator *Telenor*, Norway). The traffic is shown from the perspective of KPN and is scoped to inbound roaming. That means the figure shows the number of PDP Context Request messages created by customers of Telenor that reside in the Netherlands, together with the corresponding response messages sent from Telenor back to KPN. As can been seen in Figure 8.2a, the number of request messages equals the number of response messages. This is an ideal situation, because all request messages are answered with a response. That is also the reason that there is only one line drawn in the figure. Note that in case the number of request and response messages would slightly differ, this would still be nothing to worry about, keeping in mind the traffic that is being graphed comes from a captured portion of traffic in the live network of KPN.

Figure 8.2b is a further analysis of the situation displayed in Figure 8.2a. Figure 8.2a displayed the number of Create PDP Context Request and Response messages for inbound roaming customers from the network operator Telenor. In Figure 8.2b we see which portion of the Create PDP Context Response messages confirms a successful PDP Context setup and which portion reports an unsuccessful setup. As can be seen in the legend below the figure, the number of successful PDP Context Create messages



```
🟩 Create PDP Context Requests    Current:  816.96     Average:  447.65     Maximum:  954.00
🟦 Create PDP Context Responses   Current:  816.96     Average:  447.64     Maximum:  954.00
```

(a) Create PDP Context Request/Response messages.



```
🟩 Create PDP Contexts Success   Current:  787.96     Average:  434.72     Maximum:  923.00
🟦 Create PDP Contexts Failed    Current:   29.00     Average:   19.04     Maximum:   31.00
```

(b) Successful/unsuccessful Create PDP Context Response messages.

Figure 8.2: Create PDP Contexts Request/Response messages between KPN and Telenor over the last 24 hours for inbound roaming.

is much larger than the number of messages that reports an unsuccessful setup. That is the reason why we chose for a logarithmic scaling on the Y-axis. Whether a response message is successful or unsuccessful is implied by the *cause* value inside the GTP message. We could even further analyze this situation by creating a third figure that displays the reason, i.e. the cause value, of why the PDP Context setup is unsuccessful. However, that figure is not included.

The graphs discussed above are the result of various SQL queries launched on the MySQL database of Flow Records at the Collector. Thanks to the flexibility MySQL offers (see Chapter 6) almost an unlimited number of statistics can be generated from the received Flow Records.

Chapter 9

Conclusions and Future Work

In this work we proposed a flow-based solution for monitoring GTP traffic in cellular networks. This solution can be used by service providers as an alternative to the currently deployed packet-based monitoring solutions. All measured parameters in the proposed solution are scoped to the perspective of the roaming partner, not to the individual end-user. In this way the privacy of the end-user is protected. Besides that, only summaries of the measured traffic are exported, because the solution is flow-based. With the new telecommunications law [2] in mind, these two arguments make our solution more privacy aware than most of the currently deployed packet-based solutions. Besides that, flow-based monitoring techniques in general are much more scalable than packet-based solutions.

The requirements analysis performed at KPN resulted in three categories of requirements for monitoring packet-switched roaming traffic in cellular networks: a category of general requirements and categories for GTP-C and GTP-U, which corresponds to the two protocols in the GTP suite. All requirements imposed by KPN aim at measuring statistics per roaming partner, not per individual end-user. In Chapter 5 we described how this required information can be extracted from the various fields inside the GTP packet. We can conclude most of the required information is located in the header and especially the payload of the GTP-C traffic. Conform research question two, this work focuses on measuring this information in a flow-based manner. Because of that, we fully specified the Metering Process and Exporting Process of the IPFIX architecture and provided recommendations for the Collecting Process. We set up a proof of concept to demonstrate the feasibility of our solution by developing an extension to the existing flow-based monitoring application PMACCT. We proved the correctness of our solution by completing almost every test in our validation with expected results. All relevant fields inside the GTP message, which were defined in Chapter 5, were measured correctly, except the number of GTP v.0 packets. The reason for this was that non-GTP packets were also accounted as GTP v.0 packets. We answered our last research question by demonstrating how the received Flow Records can be analyzed using a data analysis application. We chose a MySQL database to store the received Flow Records at the Collector, because of the flexibility MySQL offers compared to NoSQL databases and flat files. For the data analysis we chose for the network graphing tool Cacti. Cacti proved to be a suitable application for visualizing our results, but was unable to produce histograms, which would have been suitable to provide an overview of the ASs that generate the most traffic, i.e. the top talkers. We can conclude that the choice for a data analysis application depends mainly on the requirements the end-user poses on the analysis.

As future work, we will now describe a number of open issues and suggest some possible enhancements to our proposed solution. As described in Chapter 7.2, the IE Type Options Templates and associated type records were left out in the Proof of Concept, due to time constraints. We would suggest adding these templates and type records to make the proof of concept fully complaint with the IPFIX architecture, just as how it was specified in our proposed solution (Chapter 6). The set of fields we monitor in the header and payload of the GTP message is rather small; as can be seen in the GTP standard [5], the payload of the GTP-C message contains a lot more information. Imposed by the requirements in Chapter 3, we only monitor information scoped to the perspective of the roaming partner. A possible enhancement would be to also monitor other relevant fields inside the GTP message: fields that are more scoped to the perspective of individual users. However, when doing this, it is important to take privacy legislations into account. As described in Chapter 5 we only provide support for monitoring GTP-C and GTP-U traffic. There is a third protocol in the GTP suite, GTP' (*prime*), that is used for carrying charging data inside the GPRS core network. Adding support for GTP' is saved for future work. By the time of writing a lot of Dutch mobile operators, including KPN, are in the initial phase of launching a commercial Long Term Evolution (LTE) network. Packet-switched roaming traffic in LTE is carried by the GTP v.2 protocol. A possible enhancement would be to include GTP v.2 support, so the proposed solution could also be deployed in LTE networks.

Bibliography

[1] P. Lucente, “pmacct.” `http://www.pmacct.net`. Accessed on July 20, 2012.

[2] NRC, “Nieuwe Telecomwet aangenomen door Eerste Kamer.” `http://www.nrc.nl/nieuws/2012/05/08/wet-netneutraliteit-aangenomen-door-eerste-kamer/`. Accessed on July 20, 2012.

[3] KPN, “Company Profile.” `http://www.kpn.com/corporate/aboutkpn/company-profile.htm`. Accessed on July 20, 2012.

[4] 3GPP, “General Packet Radio Service (GPRS); GPRS Tunnelling Protocol GPT) across the Gn and Gp Interface,” TS 09.60, 3rd Generation Partnership Project (3GPP), Jan. 2003. `http://www.3gpp.org/ftp/Specs/html-info/0960.htm`.

[5] 3GPP, “General Packet Radio Service (GPRS); GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface,” TS 29.060, 3rd Generation Partnership Project (3GPP), Dec. 2011. `http://www.3gpp.org/ftp/Specs/html-info/29060.htm`.

[6] 3GPP, “GPRS Tunnelling Protocol for User Plane (GTPv1-U),” TS 29.281, 3rd Generation Partnership Project (3GPP), Sept. 2008. `http://www.3gpp.org/ftp/Specs/html-info/29281.htm`.

[7] IPFIX Working Group, “Ipfix Status Pages.” `http://tools.ietf.org/wg/ipfix/`. Accessed on July 20, 2012.

[8] Cacti, “Cacti: the complete RRDtool-based graphing solution.” `http://www.cacti.net/`. Accessed on July 20, 2012.

[9] 3GPP, “General Packet Radio Service (GPRS); Service description; Stage 2,” TS 23.060, 3rd Generation Partnership Project (3GPP), Sept. 2008. `http://www.3gpp.org/ftp/Specs/html-info/23060.htm`.

[10] X. Peng, W. Yingyou, Z. Dazhe, and Z. Hong, "GTP Security in 3G Core Network," in *Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on*, vol. 1, pp. 15 –19, april 2010.

[11] Z. Ghadialy, "A look at PDP Context in UMTS networks." http://www.3g4g.co.uk/Tutorial/ZG/zg_pdp, November 2007. Accessed on July 20, 2012.

[12] B. Trammell and E. Boschi, "An Introduction to IP Flow Information Export (IPFIX)," *Communications Magazine, IEEE*, vol. 49, pp. 89 –95, april 2011.

[13] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek, "Architecture for IP Flow Information Export." RFC 5470 (Informational), Mar. 2009. Updated by RFC 6183.

[14] J. Quittek, S. Bryant, B. Claise, P. Aitken, and J. Meyer, "Information Model for IP Flow Information Export." RFC 5102 (Proposed Standard), Jan. 2008. Updated by RFC 6313.

[15] IANA, "IP Flow Information Export (IPFIX) Entities." http://www.iana.org/assignments/ipfix. Accessed on July 20, 2012.

[16] B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information." RFC 5101 (Proposed Standard), Jan. 2008.

[17] GSM Association, "GSM Association." http://www.gsma.com/. Accessed on July 20, 2012.

[18] 3GPP, "General Packet Radio Service (GPRS); Evolved GPRS Tunnelling Protocol (eGTP) for EPS," TS 29.274, 3rd Generation Partnership Project (3GPP), July 2008. http://www.3gpp.org/ftp/Specs/html-info/29274.htm.

[19] Commsquare, "DataMon." http://www.commsquare.com/products/datamon. Accessed on July 20, 2012.

[20] Gemik, "Daromo, Data Roaming Monitor." http://93.185.141.50/gemik/contents/content5.php. Accessed on July 20, 2012.

[21] ntop, "nProbe v6, An Extensible NetFlow v5/v9/IPFIX GPL Probe for IPv4/v6." http://www.ntop.org/products/nprobe. Accessed on July 20, 2012.

[22] E. Boschi, L. Mark, J. Quittek, M. Stiemerling, and P. Aitken, “IP Flow Information Export (IPFIX) Implementation Guidelines.” RFC 5153 (Informational), Apr. 2008.

[23] E. Boschi, B. Trammell, L. Mark, and T. Zseby, “Exporting Type Information for IP Flow Information Export (IPFIX) Information Elements.” RFC 5610 (Proposed Standard), July 2009.

[24] IANA, “Private Enterprise Numbers.” `http://www.iana.org/assignments/enterprise-numbers`. Accessed on July 20, 2012.

[25] FAU Erlangen and TU München, “Vermont - VERsatile MONitoring Toolkit.” `https://github.com/constcast/vermont/wiki`. Accessed on July 20, 2012.

[26] ntop, “ntop - Traffic analysis with NetFlow and sFlow support.” `http://www.ntop.org/products/ntop/`. Accessed on July 20, 2012.

[27] CESNET, “IPFIXcol.” `http://wp.liberouter.org/?page_id=825`. Accessed on July 20, 2012.

[28] B. Trammell, “ripfix - IPFIX for Ruby.” `http://ripfix.rubyforge.org/`. Accessed on July 20, 2012.

[29] Plixer, “Scrutinizer Flow Analyzer.” `http://www.plixer.com/Scrutinizer-Netflow-Sflow/scrutinizer-flow-analyzer.html`. Accessed on July 20, 2012.

[30] Tcpreplay, “Tcpreplay.” `http://tcpreplay.synfin.net/`. Accessed on July 20, 2012.

[31] Wireshark, “Wireshark.” `http://www.wireshark.org/`. Accessed on July 20, 2012.

[32] NfSen, “NfSen - NetFlow Sensor.” `http://nfsen.sourceforge.net/`. Accessed on July 20, 2012.

Appendix A

Cacti Templates

This Appendix will provide the Cacti [8] templates and associated Shell scripts needed to reproduce figures 8.1, 8.2a and 8.2b. We decided to put the code for the Cacti templates online, instead of posting it in this report. The reason for this is that the code of the templates is XML based. If we would paste the code here in the report, some lines will probably be too long to to fit inside the page margins and will be wrapped around to the next line. Then when someone will copy-paste the code into Cacti, this will result in errors, due to the inserted line breaks. To avoid these kind of errors and also because it is probably easier to copy-paste something from the Internet than from a report, we decided to put the code for the Cacti templates online.

The Cacti Templates needed to reproduce Figure 8.1, 8.2a and 8.2b can be downloaded from the following locations:

- Figure 8.1: Bytes transferred (cumulative)
  Online available at: `http://pastebin.com/J9kB1xXn`

- Figure 8.2a: Create PDP Context Request/Response messages
  Online available at: `http://pastebin.com/HK8a2ybu`

- Figure 8.2b: Successful/unsuccessful Create PDP Context Response messages
  Online available at: `http://pastebin.com/shWuE2MW`

The code can be imported in Cacti via the *Import Templates* option, located in the sidebar of Cacti. Importing the code will result in the corresponding *Data Input Methods*, *Data Templates* and *Graph Templates* of these figures. The version of Cacti used is 0.8.7i. Note that you could experience problems when importing these templates in older versions of Cacti.

The *Data Input Methods* inside the Cacti code refer to Shell scripts that are used by Cacti to poll the information from the MySQL database of Flow Records at the Collector. These Shell scripts are listed below.

pmacct_bytes_as.sh

```
as=$1;
datum=`date +%Y%m%d`

out=$( mysql -uroot -ppmacct -D pmacct -N -s -r -e "
   SELECT sum(bytes) from pmacctd_acct_gtp_$datum
   where as_src='0' and as_dst='$1' group by as_dst"
   );

in=$( mysql -uroot -ppmacct -D pmacct -N -s -r -e "
   SELECT sum(bytes) from pmacctd_acct_gtp_$datum
   where as_dst='0' and as_src='$1' group by as_src"
   );

printf "bytes_out:%s bytes_in:%s total_bytes_out:%s
   total_bytes_in:%s" $out $in $out $in;
```

pmacct_msgtypes_as.sh

```
datum=`date +%Y%m%d`

c_req=$( mysql -uroot -ppmacct -D pmacct -N -s -r -e
    "SELECT SUM(packets) FROM
   pmacctd_acct_gtp_$datum WHERE as_src=$1 and
   as_dst=0 and gtp_msg_type=10 GROUP BY as_src");

c_res=$( mysql -uroot -ppmacct -D pmacct -N -s -r -e
    "SELECT SUM(packets) FROM
   pmacctd_acct_gtp_$datum WHERE as_src=0 and as_dst
   =$1 and gtp_msg_type=11 GROUP BY as_src");

u_req=$( mysql -uroot -ppmacct -D pmacct -N -s -r -e
    "SELECT SUM(packets) FROM
   pmacctd_acct_gtp_$datum WHERE as_src=$1 and
   as_dst=0 and gtp_msg_type=12 GROUP BY as_src");
```

```
u_res=$( mysql -uroot -ppmacct -D pmacct -N -s -r -e
    "SELECT SUM(packets) FROM
   pmacctd_acct_gtp_$datum WHERE as_src=0 and as_dst
   =$1 and gtp_msg_type=13 GROUP BY as_src");

d_req=$( mysql -uroot -ppmacct -D pmacct -N -s -r -e
    "SELECT SUM(packets) FROM
   pmacctd_acct_gtp_$datum WHERE as_src=$1 and
   as_dst=0 and gtp_msg_type=14 GROUP BY as_src");

d_res=$( mysql -uroot -ppmacct -D pmacct -N -s -r -e
    "SELECT SUM(packets) FROM
   pmacctd_acct_gtp_$datum WHERE as_src=0 and as_dst
   =$1 and gtp_msg_type=15 GROUP BY as_src");

printf "creq:%s cres:%s ureq:%s ures:%s dreq:%s dres
   :%s" $c_req $c_res $u_req $u_res $d_req $d_res;
```

pmacct_createpdp_succesrate_as.sh

```
datum=`date +%Y%m%d`

succes=$( mysql -uroot -ppmacct -D pmacct -N -s -r -
   e "SELECT SUM(packets) FROM
   pmacctd_acct_gtp_$datum WHERE as_src=0 and as_dst
   =$1 and gtp_msg_type=11 and gtp_cause=80 GROUP BY
    as_src");

failed=$( mysql -uroot -ppmacct -D pmacct -N -s -r -
   e "SELECT SUM(packets) FROM
   pmacctd_acct_gtp_$datum WHERE as_src=0 and as_dst
   =$1 and gtp_msg_type=11 and gtp_cause!=80 GROUP
   BY as_src");

printf "cres_succes:%s cres_failed:%s" $succes
   $failed;
```