THESIS

REPRESENTING BGP AND ROUTING FLOWS IN XML

Submitted by

Jason D. Bartlett

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2013

ABSTRACT

REPRESENTING BGP AND ROUTING FLOWS IN XML

Monitoring routing in the Internet is a significant aspect of network security today. Incorrect information that is introduced into the system can result in problems ranging from a particular service or website becoming temporarily inaccesible, to large blocks of network addresses becoming cut off from the rest of the Internet, to potentially-sensitive user information being redirected to a malicious actor.

Current monitoring projects generate a huge dataset for users for sift through. A single collection point collecting routing data from a dozen routers can archive 1800 update messages every 15 minutes. The largest current monitoring projects have 12-16 collection points, some of which can have several dozen routers feeding data into them, and some of which have been saving data for a decade or more.

These archives are stored in a binary format called MRT that appends metadata about the particular routing session being monitored to the raw data received by a router. They also depend on tools to convert the binary into usable, but rigid, ASCII formats. Ideally, this data could be represented in a standardized ASCII format that both human user and machine application can make use of. Furthermore, such a format ought to be able to be easily extended, whether to represent new features in the underlying data or to transport user-specific annotations, without creating compatibility problems. XML and XSD provide the mechanisms necessary to accomplish this and the framework necessary to do it in such a way that the resulting definitions can become standardized.

This work presents an XSD-based generic format for representing the flow of routing data between arbitrary routers. To provide a concrete realization of this idea, additional schema are defined to describe Border Gateway Protocol messages and several common networking datatypes. All of these schema are defined to provide validation of their underlying data, but are also flexible enough to accommodate extensions within the data and additional datatypes not already included in the schema.

Keywords: BGP, routing monitoring, data formats, XSD, XML, networking, network security

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

Monitoring Internet routing has been important for almost two decades now. Network operators and security researchers take an active interest in the day- to-day changes in the routing tables of the Internet, regardless of whether these changes are benign or malicious in nature.

Monitoring the messages that update the routing tables is one way for network operators to keep abreast of potential threats or errors involving their networks. Monitoring projects deploy special collection software that simulates a router and collects the update messages that it receives, but does not participate in any actual routing. By archiving these messages, operators or researchers can go back and look at the updates to find the point where a specific change took place. This data can then empower the operator to make the appropriate corrective action, if needed. The researcher can use this data to draw inferences about the behaviour of the system.

However, monitoring brings with it some challenges of its own. In any case where large quantities of data are being collected and archived, there needs to be a way to store that data. Also, just as the routing protocols being monitored are standardized, it is important for any *representation* of that data to also be standardized. Creating these standards reduces the likelihood of competing data formats or processing tools introducing errors into analyses. Ideally, such a format would be easy for both machine and user to read. This format should also be able to accommodate new data but still maintain compatibility between instances that have the new data and those that do not. XML [BPSM$^+$08] provides all of these, particularly the ability to describe a standard format as an XML Schema Document (XSD).

The primary protocol used in inter-domain routing in the Internet today is the Border Gateway Protocol [RLH06]. This protocol disseminates some of the paths that data may take through the thousands of Autonomous Systems (ASes) that comprise the Internet. Theoretically, these paths ensure that data arrives at the correct destination. In reality, whether by deliberate attack or inadvertent misconfiguration, incorrect paths can be introduced into the Internet. These incorrect paths can cause massive service outages to users around the world or even route sensitive data to an attacker.

Regardless of the particular routing protocol in use, routing requires messages to pass from one router to another. This is how new members of the network are detected, how broken links are detected and routed around, and how each router determines the ”best” way to forward the data it receives. Without knowing details of which routing protocol is being used, the movement of these messages can be thought of as an example of a network flow. Characterizing this flow could provide new insight into routing dynamics. At a minimum, it would simplify looking up data from message archives because it would allow the data to be indexed by source or destination router.

This work uses XSD to attempt to describe this routing-message flow. The challenge of this is distilling the core components of the flow without relying on knowledge of the routing protocol or network topology. While the definition may be specified this way, any implementation of it would need to know, at a minimum, which protocol is being described. Similarly, since the definition is written in XSD, there must also be a representation of the routing protocol messages in XSD, so that the data can be represented and included within the description of the flow.

BGP, as the de facto standard inter-domain routing protocol, makes an ideal candidate to encode in XSD and use in an example of a routing flow. There are several BGP monitoring projects already running which rely on a rigid binary format to store the data and a tenured, but noncomprehensive, tool to generate useful ASCII data. These projects might find a standardized XML-based alternative to representing BGP data useful. Not only would XML provide a cleaner way to store the existing data in, but having an XML schema would also pave the way for construction of databases that could store the data and allow active querying from users.

Such a standard would need to not only describe as many current BGP standards as possible, but also do it in a way that is easily augmented to reflect new additions to the protocol. In addition to checking the validity of the values in an XML BGP instance, it would also be very useful for the XSD to only validate XML instances of valid BGP messages. This would eliminate the need to develop tools to parse the current binary formats and confirm correctness before translating into XML, and would only allow proper BGP messages in instances created from scratch.

Part of ensuring this validity and correctness requires that the XSD also be able to confirm the validity of the content of the BGP message. One example of this are the network prefixes being withdrawn or announced. Another is making sure that fields reserved for 2-byte AS numbers do not end up with a 4-byte number inside it. These definitions are used throughout BGP, so it would make sense for them to be defined as datatypes within the XSD.

However, these types are not unique to BGP, and this is not the only work that would benefit from having XSD datatypes that describe and validate common formats. However, trying to include every kind of network datatype as part of this work would be folly. Instead, the library will be written in such a way that common types will be included, but in a framework that can be extended for additional types.

The goal of this work is to provide an XML-based description of routing data moving between routers. The description of a routing flow and XSD schema that encodes it accomplishes this by distilling the fewest features necessary to uniquely identify a particular routing protocol's messages moving from one router to another without relying on details of the protocol itself. The BGP XSD was developed to accomplish two things: to validate only XML instances of valid BGP messages, and to be easily extended to accommodate new features in the protocol. Writing this schema required in-depth understanding of BGP standards, particularly the types of messages and permissible message configurations. Two dozen standards are represented in the BGP XSD, and the use of abstract-type substitution within the XSD allows modifications to BGP to be quickly introduced in the XSD. This work required a network-datatypes XSD that provided validation of the included address families and other types as well as providing the framework and templates to extend the schema for additional types. Each of these pieces represent a contribution toward a greater understanding of Internet routing monitoring in general, and particularly BGP monitoring.

The remainder of this paper is structured as follows. Chapter 2 describes current approaches to BGP monitoring and standards development. Chapter 3 provides a high-level description of the components of the new format. Chapter 4 describes the basic network datatype XML library that the other components build on top of. Chapter 5 presents the overview of the XML-based format for BGP messages. Chapter 6 describes the design and XSD representation of a routing flow. Chapter 7 presents the conclusions of the work.

2 Related Work

2.1 BGP Monitoring

There are many projects that monitor BGP traffic. The largest of these are the RouteViews Project at the University of Oregon [Cen] and the RIS archive at RIPE NCC [RIPa]. These projects have been archiving BGP data from peer routers around the world since 2001 and 1999, respectively. These archives provide the entire stream of update messages received by their route collectors broken down by 15-minute interval.

These collectors run the Quagga routing software suite [Sav]. Quagga implements BGP (as well as several other routing protocols), which allows it to simulate a router that can establish peering connections just like any other BGP router. The difference between these Quagga collectors and any other router is that, instead of propagating the routing information it receives or introducing any routing information of its own, all it does is export the messages it receives to its archive.

These collectors are designed to handle multiple peering connections simultaneously. If they simply dumped the raw bytes of the routing messages into the archive, then the archive would be muddled with potentially dozens of parallel streams, with no way to tease out which messages came from which router. To avoid this, the Quagga suite uses the Multi-threaded Routing Toolkit (MRT) format [BKL11], which encapsulates each message it receives within a header that includes metadata about the connection the message was received over. This allows the archiver to separate the parallel streams and allows any user to select only that data he is interested in.

The MRT format is a binary format, ergo, anyone that uses the data would need a tool to parse it. The most common tool that exists to do this is *bgpdump* [RIPb], which can read the MRT binary and generate ASCII for either a human or script to digest. The tool is open-source and hosted at RIPE, the same organization that hosts the RIS archive.

The BGP-based aspects of this work are derived from the BGPmon project at Colorado State University [Gro]. Like the RouteViews and RIS projects, BGPmon uses a simulated BGP peer to collect and archive BGP messages. BGPmon differs somewhat in that it can peer directly with other

BGP routers, but can also handle MRT input, which allows it to use the output from RouteViews' collectors. Also, instead of using MRT for output, it writes the data out in an XML-based format that is the motivation for the standard presented in this work.

2.2 Standards Development

The internet today is built from a cornucopia of hardware and software. If every hardware vendor and software developer were to use their own representation of network data, there would be obscene levels of incompatibility between all of those different implementations. Clearly there need to be common formats in use among all the different implementations so that they can communicate with each other.

These standards are just as necessary for XML-based data representations. XML is designed to be user-friendly in terms of readability, and that introduces the potential for ambiguity when describing otherwise straightforward data. For instance, consider an IPv4 CIDR prefix 1.2.3.0/24. One implmentation may use a $< CIDR >$ element to encapsulate the prefix, while another may use $< IP\_BLOCK >$ to do the same thing. A user could read either of these and understand what the data represents, but an XML parser that is looking for one or the other would not be able to understand the other representation. This would be disastrous in a BGP montitoring system; if half of the XML-formatted BGP updates were skipped, the entire data set would be utterly useless as a tool. It is critical that any representation of a protocol be just as standardized as the protocol itself. With that in mind, it then makes sense for such a standard to be vetted by the same organizations that created the protocols on which the new representation is based.

One of these organizations is the Internet Engineering Task Force (IETF) [IETa]. This organization is a collaborative effort among peers in a wide range of network protocols to pool their collective knowledge and experience to design and maintain open standards. One important aspect of the IETF's process is that it is open; anyone can submit their standard for consideration. These standards can contain many different data representations, so it is important to know whether XSD is an appropriate format to submit a standard to the IETF in. Fortunately this work is not the first that proposes an XSD standard to the IETF. One such standard [EN10] has already been adopted as an RFC, which suggests that the IETF may be an appropriate venue to submit this work to for adoption and standardization.

This process is relatively straightforward and primarily described in RFC 2026 [Bra96]. The interested party can submit their work as an ”Internet Draft,” an informal document that is made available to the community for comments and updates. After this initial review process is passed, a Working Group within the IETF can request that the specification be reclassified as a Proposed Standard. At this point, the specification is given an RFC number and published. While a Proposed Standard (at least 6 months), the community is encouraged to implement the specification and submit any issues they encounter, but to also consider the specification an immature standard subject to change. After sufficient testing by the community and revision by the author, an action may be called to promote the specification to a Draft Standard. Such a standard is considered well-understood and stable enough to deploy in production. Finally, with widespread stability and interoperability between implementations of the specification, it can be promoted to an Internet Standard [IETb].

Another organization that publishes standards documents that are relevant in the internet is the World Wide Web Consortium (W3C) [W3C]. Whereas the IETF is largely concerned with networking protocols that exist in the Network and Transport layers of the networking stack, W3C is mostly concerned with standards for internet applications and the protocols underlying them such as HTTP, CSS, and XML. However, their review process is extremely similar to that of the IETF. A W3C member in a Working Group can submit a Working Draft for initial review and maintenance. After some time as a Working Draft, the specification can become, in turn, a Candidate Recommendation, a Proposed Recommendation, and a W3C Recommendation. These levels correspond in rigor and requirement to the Standard levels in the IETF; in fact W3C specifies that their Recommendations can be submitted to other bodies for adoption as standards according to the other group’s criteria [Con].

While open standards are crucial to the continued development of the internet, they are not the only standards in play. Private companies often have a need for proprietary representations of data. While these proprietary standards do not generally undergo the peer-review that a standard from the IETF or W3C would, they may become considered de facto standards if the company has a significant enough market share. In the context of this work, the primary vendors of interest are the companies that make routing hardware. The largest market shares in this niche belongs to Cisco Systems [cisa].

In 2012, the IETF and W3C signed a collaborative agreement with the Internet Society [Int], IEEE [IEE], and Internet Architecture Board (IAB) [IAB]. This agreement codifies each organization's commitment to an open standards process that ensures Due Process, Broad Consensus, Transparency, Balance, and Openness [Sta]. I investigated each of the IEEE, IAB, and Internet Society and found that, while each group is involved in standards development, none have a standard that is related to this work.

Networking standards from the IEEE are concerned with the computer- engineering aspects of networking, that is, the link layer. As the data and protocols that this work focuses on are all above the link layer, the IEEE is an inappropriate place to look for related work.

The IAB and Internet Society are both involved in standards work as well, but both of their standards pages refer back to IETF and W3C working groups. Therefore any standard that originates from them will be subject to the vetting processes described above.

This work is concerned with describing aspects of network- and transport-layer concepts and data. Standards from the W3C are not relevant because they deal with the application layer. Standards from the IEEE are not relevant because they deal with the link layer. Any standard from the IAB or Internet Society are vetted through the IETF or W3C, so these two organizations are the most likely to have relevant standards.

3 Overview of Contributions

Monitoring projects today collect routing data from routers throughout the internet. This data provides a historical record of changes to the routing tables. Researchers and operators can use this data to understand general trends in the network, for example the growth of the routing table over time, or to gain insight into specific events, such as World IPv6 Day or a massive route hijack.

These monitoring projects generate an enormous amount of data. As with any large data set, then, there is a need for a format to store the data in. Obviously, any such format should present the data in the same way regardless of who is using it or for what purpose. Consequently, the data format should be standardized itself, so that the data can be shared between users without needing to be translated into intermediate formats or alternate versions. It would also be nice if the data were presented in a format that is easily readable by users and also easily parsed by their scripts. This would eliminate the need for tools to translate the data into a usable format, and thereby eliminate a possible source of data corruption. Finally, even though such a data format would be standardized, that does not guarantee that the data itself could not change. This is a common occurrence in routing protocols, where new functionality is frequently introduced. It would be great if the format were flexible enough to accommodate new features without introducing compatibility issues in downstream systems. Such extensibility would also allow a user to add their own application-specific annotations to the data yet still maintain compatibility with any other implementation.

The eXtensible Markup Language (XML) [BPSM$^+$08] is a format that will satisfy all of these goals. The langauge itself is standardized, so the data can be read by any script or application that understands XML. Additionally, since XML is an ASCII-based langauge, human users will be able to read it as well. It is also a markup langauge, so it is designed to be able to handle user-added information. Any XML parser will simply ignore any data that it is not interested in. Finally, specific uses of the language can be standardized via the development of schema that fully describe the data. There are several different XML schema languages, but one, XML Schema Document (XSD), is a standard created by the same organization that defined XML itself. This makes XSD a natural choice to use for this work.

To be useful as part of a monitoring project, then, this work will provide an XSD document that describes, at a high level, the flow of routing data from one router to another. The challenge of designing such a description is doing it in such a way that routing flows can be represented for any routing protocol. This implies that the routing flow being described cannot depend on knowledge of the routing protocol being used. If such a definition can be distilled, it would provide a versatile, consistent format that could be used by anyone doing work in route monitoring. However, this definition could only exist as an abstract model; we would need additional schema to represent the messages from each routing protocol that we would like to model within this framework.

With this in mind, an XSD encoding of one routing protocol needs to be included as an example. BGP is the de facto standard protocol used for inter-domain routing in the Internet today, so it would be a significant contribution to be able to describe its messages in XSD. As discussed earlier, this data description should not only encode the current range of BGP messages, but also be able to handle new extensions to the protocol. Additionally, it would be nice if the BGP data itself could be validated via validation of its XML representation against the XSD schema.

One side effect of providing this validation will be having to check the validity of specific content within each message, such as the network prefixes being announced or withdrawn, or the allowable values for a 2-byte AS number versus a 4-byte AS number. This can be done in XSD by writing definitions for such network datatypes within a schema. While these types are necessary for the BGP XSD, they are not unique to BGP. In addition, since the BGP XSD should ideally become standardized, it would also make sense for these datatypes to become standardized as well. This way any network- based XSD schema could use a common representation for these types. The XSD that provides these definitions will be useful for any future work that wants to describe network data in XML.

The relationships between these various schema are shown in Figure 3. The figure uses a UML-like convention to show the relationships between the schema. The dotted arrows represent a ”used-by” relationship. The solid line with open arrow indicates a ”contained within” relationship, and the solid line/solid arrow represents an ”is-derived-from” relationship.

Each specification described herein satisfies a different set of goals. The network datatypes

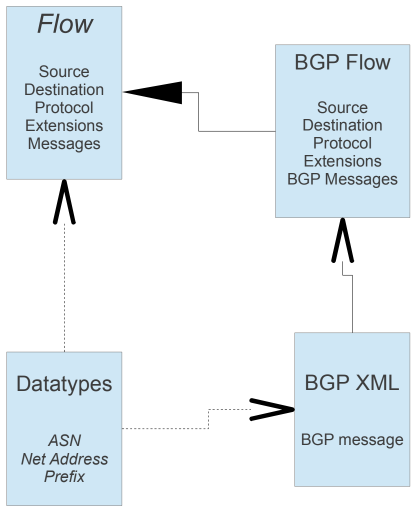

Figure 3.1: A diagram showing the relationship between the libraries defined in this work.

library is designed to validate the content that it describes and provide a framework and templates to add new types with the same rigor of the included types. Both of these are improvements over existing approaches, which either lack some of these types or do not provide validation for them. The BGP XSD is an application of a deep understanding of BGP standards. This goes beyond simply reading the documents and translating the message formats into XSD; it requires an understanding of where each piece goes within a particular BGP message, and under which circumstances it cannot be included at all. This schema, when standardized, provides a completely new approach to archiving BGP data. Not only can the XSD be used to refactor existing MRT archives into XML, but it can also be used to create a relational database to store the data in, which would allow much more efficient analyses of BGP data. Finally, the network flow definition is an attempt to understand the basic facets of routing information moving in the Internet, and describe them in XSD so that this data movement can be described for any routing protocol.

4 Representing Network Datatypes in XML

BGP is just one of many network protocols that use many of the same basic data types, such as network addresses or AS numbers. It would be useful to have definitions of these types in an XSD so that this and future projects can easily describe them.

This could be trivially accomplished by using the base data types already available in XML. For example, one might simply represent an IP address as a string, or an AS number as an integer. While this may be enough for some applications, it would be far too general to be adopted as a true standard.

Data structures and types such as these have commonly-understood written formats and valid values, for example the dotted-decimal notation used for IPv4 addresses $a.b.c.d$ where $a - d$ are decimal values between $0 - 255$. One of the two major contributions of this library is that the datatypes are defined in such a way that validating an XML instance against the XSD also implicitly validates the legality of the data itself.

On the other hand, this library cannot possibly contain full descriptions of every single networking construct out there. There are simply too many protocols for such an undertaking to be practical. This library contains definitions of the most common formats, but it is by no means comprehensive.

The other major contribution of this library is the framework to enable users to add descriptions of new types without having to modify their other schema to achieve validation.

4.1 Approaches Considered

This work is certainly not the first to try to leverage XML for network-related purposes. It is therefore conceivable that such XML constructs already exist and have been published as standards for others to use. It would be a great boon to this work to find and use such a library if it exists.

4.1.1 W3C

My search of the W3C's working groups led me to the POWDER working group. This group's focus is defining standards related to network resource identification. They have an early document [W3C07] that contains XSD descriptions of some networking primitives such as IPv4 addresses. The definition of IPv4 addresses also includes a regular expression, which is desirable for validation purposes.

Unfortunately, this document is not suitable for purposes of the standards being developed in this work. The document is listed as a first working draft, which implies it had not progressed far enough in the W3C's standards process to be useful as a work to build other standards upon. In addition, the document wound up being obsoleted within its own working group for a revision that did not contain the XSD definitions that were relevant to this work.

4.1.2 IETF

Searching the IETF's RFC archive for examples of XSD yielded only a single result, RFC 5935 [EN10]. This standard provides equivalent XSD definitions of SMI primitive dataypes [CMRW99]. IPv4 addresses are one of the data types it encodes, and does so with a regular expression.

This standard is also not appropriate for this work. The XSD encodings defined in this RFC do not include any other primitives that this work seeks to build upon. In addition, this RFC is designed for a particular use. Any base library that this work should build on needs to be much more general than this document.

4.1.3 CISCO Systems

Searching the W3C and IETF's standards libraries failed to turn up an appropriate primitives library. The next step would be to look at the major private-sector companies to see if one of them has a usable standard. CISCO Systems has a schema for common datatypes [Cisb] that appeared promising, but I could not get a copy of the XSD due to it being behind a paywall. Clearly, trying to build an open-source standard on top of a pay-per-use closed standard is not viable.

```
<xsi:complexType name="asn" abstract="true">
   <xsi:simpleContent>
       <xsi:extension base="xsi:unsignedInt">
            <xsi:attribute name="byte_len" type="xsi:unsignedByte" use="required"/>
       </xsi:extension>
   </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="asn4">
   <xsi:simpleContent>
       <xsi:restriction base="net:asn">
           <xsi:attribute name="byte_len" type="xsi:unsignedByte" fixed="4" use="required"/>
       </xsi:restriction>
   </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="asn2">
   <xsi:simpleContent>
       <xsi:restriction base="net:asn">
           <xsi:maxInclusive value="65535"/>
           <xsi:attribute name="byte_len" type="xsi:unsignedByte" fixed="2" use="required"/>
       </xsi:restriction>
   </xsi:simpleContent>
</xsi:complexType>
```

Figure 4.1: XSD representation of 2-byte and 4-byte AS Numbers.

4.2 Network Datatypes in XSD

A search of the major open-source organizations that develop standards related to networking and XML failed to produce a standard which provides a somewhat broad base of network primitives with validation capabilities and extensibility for new types. Therefore, before this work can proceed with designing new XML standards for higher-level protocols and constructs, there must first be a new XSD for networking primitives.

4.2.1 AS Number

Autonomous System (AS) Numbers identify individual organizations within the larger internet, and they are used in a variety of protocols. AS numbers were originally defined to be 2 bytes in length, but have since been extended to be 4 bytes long. While XML provides a single data type, *unsignedInt*, that represents any 4-byte unsigned integer, protocols that use AS numbers often need to know the length of the AS number even if either format is accepted. XSD's abstract typing provides the facilities necessary to accomplish all of this. The resulting types are shown in Figure 4.1.

```
<xsi:simpleType name="port">
    <xsi:restriction base="xsi:unsignedShort"/>
</xsi:simpleType>
```

Figure 4.2: XSD representation of a Port Number.

4.2.2 Port Number

Any service provided over the network is accessed via a port on the host system. Many of these ports are reserved through IANA [IAN]. For example, port 179 is set aside for BGP communications. Regardless, ports are represented by a 2-byte unsigned integer. XSD has an entire hierarchy of integer data types, one of which is exactly a 2-byte unsigned integer [W3C12]. Simply renaming the *unsignedShort* XSD data type as a port number type accomplishes two things. First, it guarantees bounds-checking without unnecessary data facets. Second, it ensures that the user uses the proper data type for a port number without needing to go check XSD documentation for which one to use.

4.2.3 Network Addresses and Prefixes

While IPv4 and v6 are the dominant network addressing schemes in use today, they are not the only ones. In fact, there is an entire registry at IANA that keeps track of these different address families and assigns each of them a unique Address Family Identifier to distinguish them from one another. One of the goals of this standard is to be extensible, so a user should be able to extend this schema for new address families without introducing compatibility problems or having to redesign the concept of an address family within XSD.

Some address families, for instance IP, also have a concept of a network prefix that represents an aggregated portion of the address space. These prefixes are not simply a vaguely-modified network address, and so they must be treated as a unique, albeit similar, data type.

To create the clean interface necessary to be useful in derived schema and extensible for any address families not defined here, we make use of XSD's abstract types. Network addresses being represented in XML should be human-readable, so a string representation of the address is desirable. In addition, every standard address family has an AFI that distinguishes the address or prefix

```
<xsi:complexType name="network_addr" abstract="true">
    <xsi:simpleContent>
        <xsi:extension base="xsi:string">
            <xsi:attribute name="afi" type="xsi:unsignedShort" use="required"/>
            <xsi:anyAttribute/>
        </xsi:extension>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="network_prefix" abstract="true">
    <xsi:simpleContent>
        <xsi:extension base="xsi:string">
            <xsi:attribute name="afi" type="xsi:unsignedShort" use="required"/>
            <xsi:anyAttribute/>
        </xsi:extension>
    </xsi:simpleContent>
</xsi:complexType>
```

Figure 4.3: Abstract types to define a network address or prefix.

being represented, so that should be a mandatory facet of the general type. Some address families have other facets that may be associated with them, for example SAFI in IP. While these should be allowed, they cannot be mandatory. The abstract formats of a network address or prefix in XSD is shown in Figure 4.3.

It is important to note here that, while the content models of these two types are identical, it is necessary to have them both. XSD schema may use either of these types to validate instances of any derived type. If the derived types of network addresses and families had the same abstract type, then the two would be interchangable within XML instance documents, which is undesirable.

While this specification is meant to be extensible, that does not mean that it should not include some common network family types. Such types should not only provide the validation and completeness that the rest of the standard does, but they should also provide an easy template for users to write types for new address families.

4.2.4 IPv4 and IPv6 Address Families

The problem with current approaches to representing IP addresses in XML is that they either do not provide validation at all or they do not include validation for both IPv4 and IPv6 addresses. Since the abstract type uses strings for the address representation, it makes sense then that these any specific address family should be representable via regular expression. The IPv4 regular expression restricts its values to a string of the form $[0-255].[0-255].[0-255].[0-255]$. The IPv6 regular

```
<xsi:complexType name="ipv4_addr">
    <xsi:simpleContent>
        <xsi:restriction base="net:network_addr">
            <xsi:pattern value="(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25
[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])"/>
            <xsi:attribute name="afi" type="xsi:unsignedShort" fixed="1" use="required"/>
            <xsi:attribute name="safi" type="xsi:unsignedByte" use="optional"/>
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="ipv6_addr">
    <xsi:simpleContent>
        <xsi:restriction base="net:network_addr">
            <xsi:pattern value="/((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|
(([0-9A-Fa-f]{1,4}:){6}(:[0-9A-Fa-f]{1,4}|((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.
(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|
(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-Fa-f]{1,4}){1,2})|:((25[0-5]|2[0-4]\d|1\d\d|
[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|
(([0-9A-Fa-f]{1,4}:){4}(((:[0-9A-Fa-f]{1,4}){1,3})|((:[0-9A-Fa-f]{1,4})?:((25
[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]{1,4}){1,4})|((:[0-9A-Fa-f]{1,4}){0,2}:
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){2}(((:[0-9A-Fa-f]{1,4}){1,5})|((:[0-9A-Fa-f]{1,4}){0,3}:
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){1}(((:[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-f]{1,4}){0,4}:
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(:(((:[0-9A-Fa-f]{1,4}){1,7})|((:[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|2[0-4]\d|1\d
\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:)))(%.+)?/"/>
            <xsi:attribute name="afi" type="xsi:unsignedShort" fixed="2" use="required"/>
            <xsi:attribute name="safi" type="xsi:unsignedByte" use="optional"/>
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>
```

Figure 4.4: Derived types for IPv4/v6 addresses that ensure valid content and proper AFI assignment.

expression is significantly more complex because of the many possible canoncial representations of any given IPv6 address. In addition, we want to ensure that these types make sure that any IP address, v4 or v6, is associated with its proper AFI, and has the ability to specify a SAFI. Using regular expressions from [Hat11] gives us the XSD definitions for IPv4 and IPv6 addresses shown in Figure 4.4.

4.2.5 IPv4 and IPv6 CIDR Prefixes

As with IP addresses, CIDR prefixes are also poorly represented in the current approaches. As with the address types above, regular expressions are necessary to ensure legal content within these derived prefix types. One might think that simply bolting on a netmask to the IPv4 and v6 regular expressions above will do the trick. For IPv6, that is sufficient because of how many different ways one can represent a v6 prefix. In IPv4, however, adding the netmask introduces new ways to represent some prefixes. For example, $10.0.0.0/8$ can be equally represented as $10/8$. It is important to not only accommodate these different versions, but also to provide validation for the legality of the mask for the given prefix. Putting all this together, we get the following two types shown in Figure 4.5 for representing IPv4 and IPv6 CIDR prefixes:

4.2.6 802 Address Family

IPv4 and v6 are the predominant address families in the internet today, but those protocols only cover most of one part of the networking protocol stack. Physical devices operate on an entirely different set of protocols and addresses. The most widely-used address family in this area is the IEEE 802 address family, also known as MAC addresses. Since this is a standardized address family, it has an AFI (6). Therefore we can write an XSD type for it just by specifying the regular expressions that capture the whole address format and assigning the AFI attribute accordingly. The resulting format is shown in Figure 4.6.

4.2.7 Dual Representation of Data

One of the design goals of this specification is to provide a combination of human- and machine readability. In most standards, fields are encoded as numeric values. While this is obviously the preferred form for a machine to process, having an XML element to communicate the type of a message that looks like element [1] in Figure 4.7 could be confusing for a human user. XML is an ASCII format; it is designed for users to be able to read.

Instead consider element [2] in Figure 4.7. There is no ambiguity in this representation. Obviously this message type is UPDATE, and the type code is 2. Including the type code as an attribute within this element not only preserves the integrity of the translation, but also would allow an XML parser to simply extract the *value* attribute from the *MESSAGE_TYPE* element and process the remainder of the message accordingly.

Element [2] above also provides an additional benefit. If the MESSAGE_TYPE element were defined in an XML Schema (XSD) document in such a way that the text content and numeric attribute were fixed together, then validating the XML instance would implicitly prove the validity of the original data. This is a powerful boon to anyone that has to sift through tons of binary data.

```
<xsi:complexType name="ipv4_prefix">
    <xsi:simpleContent>
        <xsi:restriction base="net:network_prefix">
            <xsi:pattern value="(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/([0-8])))"/>
            <xsi:pattern value="(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.)
           ([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(\d|1[0-6]))"/>
            <xsi:pattern value="(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){2}
           ([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(\d|1\d|2[0-4]))"/>
            <xsi:pattern value="(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}
           ([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(\d|[1-2]\d|3[0-2]))"/>
            <xsi:attribute name="afi" type="xsi:unsignedShort" fixed="1" use="required"/>
            <xsi:attribute name="safi" type="xsi:unsignedByte" use="optional"/>
            <xsi:attribute name="prefix_len" use="optional">
                <xsi:simpleType>
                    <xsi:restriction base="xsi:unsignedByte">
                        <xsi:maxInclusive value="32"/>
                    </xsi:restriction>
                </xsi:simpleType>
            </xsi:attribute>
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="ipv6_prefix">
    <xsi:simpleContent>
        <xsi:restriction base="net:network_prefix">
            <xsi:pattern value="((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|
(([0-9A-Fa-f]{1,4}:){6}(:[0-9A-Fa-f]{1,4}|
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|
(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-Fa-f]{1,4}){1,2})|:((25[0-5]|2[0-4]\d|1\d\d|
[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|
(([0-9A-Fa-f]{1,4}:){4}(((:[0-9A-Fa-f]{1,4}){1,3})|((:[0-9A-Fa-f]{1,4})?:((25
[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]{1,4}){1,4})|((:[0-9A-Fa-f]{1,4}){0,2}:
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){2}(((:[0-9A-Fa-f]{1,4}){1,5})|((:[0-9A-Fa-f]{1,4}){0,3}:
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){1}(((:[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-f]{1,4}){0,4}:
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(:(((:[0-9A-Fa-f]{1,4}){1,7})|((:[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|2[0-4]\d|1\d
\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:)))(%.+)?(\/(\d|\d\d|1
[0-1]\d|12[0-8]))"/>
            <xsi:attribute name="afi" type="xsi:unsignedShort" fixed="2" use="required"/>
            <xsi:attribute name="safi" type="xsi:unsignedByte" use="optional"/>
            <xsi:attribute name="prefix_len" use="optional">
                <xsi:simpleType>
                    <xsi:restriction base="xsi:unsignedByte">
                        <xsi:maxInclusive value="128"/>
                    </xsi:restriction>
                </xsi:simpleType>
            </xsi:attribute>
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>
```

Figure 4.5: Derived types to represent IPv4/v6 CIDR prefixes.

```
<xsi:complexType name="ieee_802mac">
   <xsi:simpleContent>
       <xsi:restriction base="net:network_addr">
           <!-- The following two regular expressions match any EUI-48/MAC-48
           address.  There are two expressions here to cover addresses that
           use either a colon or a dash as their delimiter, but to avoid
           matching addresses that incorrectly mix the two. -->
           <xsi:pattern value="(([0-9A-Fa-f]{2}:){5}([0-9A-Fa-f]{2}))"/>
           <xsi:pattern value="(([0-9A-Fa-f]{2}-){5}([0-9A-Fa-f]{2}))"/>

           <!-- The following two regular expressions match any EUI-64 address.
           Again, two expressions are needed to avoid matching incorrect
           mixed-delimiter addresses. -->
           <xsi:pattern value="(([0-9A-Fa-f]{2}:){7}([0-9A-Fa-f]{2}))"/>
           <xsi:pattern value="(([0-9A-Fa-f]{2}-){7}([0-9A-Fa-f]{2}))"/>

           <!-- The following regular expression matches an 802 MAC address
           that is represented in dotted-quad notation (ex. 1234.dead.BEEF).-->
           <xsi:pattern value="(([0-9A-Fa-f]{4}\.){2}[0-9A-Fa-f]{4})"/>

           <xsi:attribute name="afi" type="unsignedShort" fixed="6" use="required"/>
       </xsi:restriction>
   </xsi:simpleContent>
</xsi:complexType>
```

Figure 4.6: An XSD format for representing the 802 MAC address family.

```
<MESSAGE_TYPE>2</MESSAGE_TYPE>                              [1]

<MESSAGE_TYPE value="2">UPDATE</MESSAGE_TYPE>               [2]
```

Figure 4.7: Example of representing a specific value in a single format and in dual formats.

```
<xsi:complexType name="dual_repr">
    <xsi:simpleContent>
        <xsi:extension base="xsi:normalizedString">
            <xsi:attribute name="value" type="xsi:anySimpleType" use="required"/>
        </xsi:extension>
    </xsi:simpleContent>
</xsi:complexType>
```

Figure 4.8: An XSD type to annotate a string with any simple data value.

By converting the bits to an XML instance document and then validating that against a well-defined XSD, error detection and incident response could be significantly improved.

Accomplishing this in XSD requires a small bit of type gymnastics, but nothing outrageous. XML is an ASCII format; it is meant to be readable by humans. Therefore in any case where we want to dual-represent some data facet, we want the content of the element to be text, and the attribute to retain its type and value.

The type shown in Figure 4.8 gives us exactly the expressive form we seek; string content annotated with some primitive type code. This type is a great generic type to use, particularly within abstract types such as the *network_addr* type above. Then, within the derived concrete types, this type can be restricted down into its proper, context-specific, instance.

This allows the user to do exactly what is proposed above: ensure that specific human-readable values are bound to their proper machine codes, and cause validation of an instance of data to fail if they do not match. This type is universally useful in any specification where machine codes are being translated into XML.

4.2.8 Writing Extensions for this Library

The types defined in Section 4.2 represent the most common network primitives, but the library itself is by no means comprehensive. Ideally, this library will become standardized, widely deployed, and augmented by operators in disciplines across the protocol stack.

This is not to say that every network protocol should be represented in XML and included in this library. This library is meant to provide a consistent representation of very primitive networking constructs. Higher-level protocol-specific constructs, for example BGP Communities, should be defined within a higher-level library.

What this library provides is the interface and general structure necessary to easily create types for new address families. The general structure of any such type is the same: one or more regular expressions that describe the address family's address format, and an AFI attribute with the identifier assigned to that address family.

See Appendix A for the full text of the XSD.

5 Representing BGP Data

5.1 Current Approaches

5.1.1 MRT

As mentioned in Chapter 2, the two largest BGP monitoring projects out there, RouteViews and RIS, use the MRT format [BKL11] to encapsulate and save the routing updates their collectors receive from peer routers.

The MRT format has several good things going for it. It is standardized and defined in RFC 6396 [BKL11]. It is binary, so it is more space-efficient than an equivalent ASCII format. It includes the metadata necessary to distinguish between many simultaneous peering sessions. Finally, as mentioned above, it is already used at RouteViews and RIPE.

For research purposes, MRT does have some drawbacks. The format itself is complex, and it can be tricky to write end systems to handle all the aspects of the protocol. For example, when an MRT collector announces RIB tables for its connected peers, it first sends an index table that assigns values to each of its connected peers. Subsequent RIB messages are then indexed according to this table, so any downstream system must maintain state for each of these indexed peers. This can get expensive, especially when the downstream system is only interested in a small subset of the peers.

A researcher may also only be interested in some small cross-section of the data, for example their company's owned prefixes or AS numbers. In this case, there is no way for him to directly find or extract the relevant information short of parsing through a flat binary data set. While this task can frequently be automated, it is still computationally wasteful.

Finally, there is no room for a downstream user to add extra metadata to MRT output. Beyond the time and peering information in the MRT header, the user is left with just raw BGP data. It could be useful to annotate a BGP message with additional data, for example geographical location of the peer or collector, or perhaps the canonical name of the owner of AS numbers in the AS Path attribute. MRT would not be able to handle such annotations.

5.1.2 bgpdump

To save developers, researchers, and operators the hassle of having to write a new MRT parser for every new application, RIPE NCC maintains libbgpdump [RIPb], a C library originally written by Dan Ardelean. This library can be included in one's project, or can be run as a standalone application called BGPdump. BGPdump can parse MRT data and produces ASCII output in both human-readable and pipe-delimited machine-readable format.

As with MRT, BGPdump does have its drawbacks. Despite being widely used, the output can be insufficient to extract some information. One example of this is the SAFI attribute of an announced or withdrawn prefix. A peer maintains two routing tables: one for unicast addresses and another for multicast addresses. A BGP update will distinguish between the two cases, but BGPdump does not extract that field. Since multicast announcements are less common than unicast announcements, an operator may mistakenly assume a multicast prefix was being withdrawn from a unicast routing table. Another example is the next hop field in the MP_REACH path attribute. This field is not restricted to a single IPv4 address the same way the original next hop attribute is. RFC 2545 [MD99] permits an operator to send two IPv6 addresses in the next hop field, for example. BGPdump's machine-readable format just drops the second address. This obviously compromises the integrity of the data.

Any tool will have bugs in it, and BGPdump is no different. Unfortunately, the issues in the current and previous versions of BGPdump can significantly affect the integrity of the data in a long-term study. The author's own experience mirrors a bug report on the libbgpdump source page [RIPb] where certain MRT files would be incorrectly parsed, resulting in binary garbage being produced instead of proper data at the end of the output. Without any way to tell how many messages are lost due to this corruption, the data set cannot be considered complete. This particular bug report is over a year old and as of this writing, no progress has been logged on it. The author attempted to use a previous version to see if this corruption could be avoided. It turned out that the earlier version of BGPdump did not produce the corrupt data, but would fail to include the AS Path in BGP announcements, which is a mandatory facet of any BGP announcement. Again, such corruption makes the entire data set suspect.

Despite these shortcomings, BGPdump is widely used in research today. Scripts around the world would break if BGPdump were updated or augmented to include new data.

5.2 The Case for XML

Given the drawbacks to MRT and BGPdump, there is a need for a new way to store and represent BGP data. Such a format should be simple for both operators and scripts to read through. It would also be nice if the format were flexible enough to not only handle any new features in BGP itself, but also be able to accomodate new metadata without negatively impacting other users' scripts.

With these parameters in mind, XML [BPSM+08] jumps out as an appropriate answer to the problem. XML is an ASCII format, so operators can easily read through tags to find the information they want. There are libraries available such as libxml2 [XML] that provide several different interfaces to parse the data to obtain any desired information while ignoring anything else. On top of this, there are bindings for libxml2 in many different languages to give the developer or researcher additional flexibility in designing scripts to handle the data. Finally, XML is a markup language, so adding new information is as simple as inserting new elements or attributes into the structure, which is just another function of the libraries.

BGP is a complex protocol with a lot of possible message configurations. Representing it in XML could give rise to many different implementations, each with a different strategy to represent the same data. Fortunately, XML allows for a schema to be written, which lays out a standard structure for the output that all implementations must adhere to. XML schema can also be tailored so that an implementation can add whatever information it wants without violating the schema. This is a great benefit because new features in BGP can simply be added to the XML output on a rolling basis, with no need to worry about compatibility between implementations that support the new feature and those that do not. On top of that, there are standard rules that can be applied to convert an XML schema into a relational database. By converting the existing archives into such a database and then feeding new data into it, users could then query the data directly. This would be much more efficient than requiring a brute-force search through a large, flat, MRT-based archive.

Additionally, it is not difficult to convert XML output to other formats. A researcher can just write a small script to use the XML libraries to grab the fields she needs, format them to look like BGPdump output, and pipe that output to whatever stable back-end script she is already running. Someone could even go as far as extracting each field from the XML and reconstructing the message in MRT.

This combination of expressive power, readability, and extensibility make XML a suitable language to describe BGP in. The next step is to design such a standard that achieves all of these design goals.

5.3 An XML Format for BGP

The fundamental particle of data transfer in BGP is the message. There are several different types of message, each of which communicates a different set of information from one peer to another. For instance, one message type is used by peers to agree on parameters for a potential peering session. Another is used to exchange routing information. A third is used simply to indicate to one peer that a session is still active, just in case there are few routing updates being exchanged. Regardless of their contents, a BGP session can be thought of as a stream of these messages being passed between peers.

It is important to maintain the discrete nature of messages in any format that would archive BGP data. This is critical for research and security purposes. Having a complete stream of saved BGP messages allows an operator or researcher to replay those messages into a test router or simply examine them in the order they were received. This can show exactly how the routing tables were changing during the period of interest.

Clearly, this new XML-based representation of BGP should be defined in such a way that the top-level XML element will represent a single BGP message.

5.3.1 Supported BGP RFC's

BGP is a complex protocol that has undergone dozens of modifications and updates throughout its lifetime. While the basics of the protocol are described in RFC 4271 [RLH06], there are many other standards that add new and updated functionality to that base protocol. The specification defined in Appendix B includes support for the following RFC's:

- *RFC 4271* - This is the base specification for BGP-4, the current version of the protocol. [RLH06]
- *RFC 2918* - Defines the Route Refresh message type. [Che00]
- *RFC 4456* - Defines Route Reflection, which includes the ORIGINATOR_ID and CLUSTER_LIST path attributes. [BCC06]
- *RFC 1863* - Defines two BGP path attributes. Now classified as Historic. [Has95]
- *RFC 4760* - Defines the MP_REACH_NLRI and MP_UNREACH_NLRI path attributes. These attributes enable BGP to carry routing information for address families besides IPv4. [BCKR07]
- *RFC 4360* - Defines the EXTENDED_COMMUNITIES path attribute. [SS06]
- *RFC 1997* - Defines the COMMUNITIES path attribute. [CTL96]
- *RFC 3471* - Defines the TRAFFIC_ENGINEERING path attribute. [Ber03]
- *RFC 5543* - Defines additional payloads for the TRAFFIC_ENGINEERING path attribute. [OBFR09]
- *RFC 5512, 5566, 5640* - Define the TUNNEL_ENCAPSULATION path attribute and several different payloads for it. [MR09, BWR09, FMP09]
- *RFC 5701* - Defines the IPV6-SPECIFIC_EXTENDED_COMMUNITY path attribute. [Rek09]
- *RFC 6368* - Defines the ATTR_SET path attribute. [MRP$^+$11]
- *RFC 4486* - Adds new error subcodes for the CEASE NOTIFICATION message. [CG06]
- *RFC 6608* - Adds new error subcodes for the FINITE_STATE_MACHINE error NOTIFICATION message. [DCS12]

- *RFC 3392* - Defines Capabilities Advertisement in OPEN messages. [SC09]
- *RFC 2545* - Describes scenarios to include two IPv6 addresses in the next hop field of the MP_REACH_NLRI path attribute. [MD99]
- *RFC 4893* - Provides 4-byte AS Number support in BGP. [VC07]
- *RFC 5065* - Defines BGP Confederations, which includes the AS_CONFED_SEQUENCE and AS_CONFED_SET AS Path Segment types. [TMS07]
- *RFC 6286* - Allows the BGP Identifier to be represented as a 4-byte unsigned integer instead of an IPv4 address. [CY11]

5.3.2 Describing BGP in XSD

As mentioned above, BGP is a complicated protocol that is constantly being updated and extended with new functionality. The specification in this work covers all of the standards that are currently seen in the wild, but another primary design goal of this new specification is to be able to accomodate new features in BGP without damaging compatibility with earlier implementations.

XSD provides a couple of different strategies for handling this problem, both of which involve the use of abstract types. These types are similar to the concept of abstract classes in object-oriented programming. An abstract type is defined that describes the generic form of some group of related types. This abstract type may then be used in any other schema as a sort of 'placeholder'. An instance document cannot instantiate an element of the abstract type, but can use an element of *any type derived from the abstract type*, and the instance will validate against the schema that employs the abstract type. Within the schema, there need to be the following three things to make this whole scheme work:

- *an Abstract Type* - Describes the general form of the specific type we're after, for example a Path Attribute from an *UPDATE* message.
- *a base element* - An element of the abstract type.
- *Derived Types* - Concrete types that restrict the abstract type to a particular form.

```
<xsi:complexType name="path_attribute" abstract="true">
    <xsi:sequence>
        <xsi:element name="OPTIONAL" minOccurs="0"/>
        <xsi:element name="TRANSITIVE" minOccurs="0"/>
        <xsi:element name="PARTIAL" minOccurs="0"/>
        <xsi:element name="EXTENDED" minOccurs="0"/>
        <xsi:element name="TYPE" type="xfb:dual_repr"/>
        <xsi:any minOccurs="0" maxOccurs="unbounded"/>
    </xsi:sequence>
    <xsi:attribute name="length" type="xsi:unsignedShort" use="required"/>
</xsi:complexType>
```

Figure 5.1: An XSD abstract type that contains the basic elements of a BGP Path Attribute.

```
<xsi:element name="UPDATE">
    <xsi:complexType>
        <xsi:sequence>
            <xsi:element name="WITHDRAWN" type="net:ipv4_addr" minOccurs="0" maxOccurs="unbounded"/>
            <xsi:sequence minOccurs="0">
                <xsi:element name="PATH_ATTRIBUTE" type="xfb:path_attribute" maxOccurs="unbounded"/>
                <xsi:element name="NLRI" type="net:ipv4_addr" maxOccurs="unbounded"/>
            </xsi:sequence>
        </xsi:sequence>
        <xsi:attribute name="withdrawn_len" type="xsi:unsignedShort" use="required"/>
        <xsi:attribute name="path_attr_len" type="xsi:unsignedShort" use="required"/>
    </xsi:complexType>
</xsi:element>
```

Figure 5.2: An XSD representation of a BGP UPDATE message.

For a specific example, consider the Path Attributes mentioned above. According to RFC 4271 [RLH06], there is a general format that all BGP path attributes must follow. Extracting these fields produces the abstract type shown in Figure 5.1.

As you can see, each of the four attribute flags are present within the abstract type, but none of them are required to be there. This is because each path attribute has a different subset of these flags, so the derived types *will* require some of these elements to be there. Note also the *length* attribute on this type. This field is defined as a 2-byte unsigned integer, and must always be present, so it is defined and made to be required in the abstract type so that it must be present in every derivation of this type. Finally, the wildcard *any* element permits additional elements to be included, which is how the vast array of path attribute data is to be accommodated.

The next piece that must be in place for this substitution scheme to work is the base element that can take on any derived-from-the-abstract type. Since this example is for path attributes, consider the definition of an UPDATE BGP message given in Figure 5.2:

Note specifically the *PATH_ATTRIBUTE* element. It is of type *path_attribute*, and can appear

```
<xsi:complexType name="origin_path_attr">
    <xsi:complexContent>
        <xsi:restriction base="xfb:path_attribute">
            <xsi:sequence>
                <xsi:element name="TRANSITIVE" minOccurs="1"/>
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="xfb:dual_repr">
                                <xsi:enumeration value="ORIGIN"/>
                                <xsi:attribute name="value"
type="xsi:unsignedByte" fixed="1" use="required"/>
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="ORIGIN" type="xfb:bgp_origin_type"/>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
use="required"/>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<xsi:complexType name="next_hop_attr">
    <xsi:complexContent>
        <xsi:restriction base="xfb:path_attribute">
            <xsi:sequence>
                <xsi:element name="TRANSITIVE" minOccurs="1"/>
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="xfb:dual_repr">
                                <xsi:enumeration value="NEXT_HOP"/>
                                <xsi:attribute name="value"
type="xsi:unsignedByte" fixed="3" use="required"/>
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="NEXT_HOP" type="net:ipv4_addr"/>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte" use="required"/>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>
```

Figure 5.3: XSD definitions of the BGP ORIGIN and NEXT_HOP Path Attributes.

multiple times. This is consistent with the BGP definition of the path attributes in an UPDATE message. This leads to the final piece of the puzzle: the derived types that represent the path attributes themselves. Each attribute must be a restriction on the abstract type, so that the definition of each attribute must be less general than the abstract type. The definitions of the ORIGIN and NEXT_HOP attributes are shown in Figure 5.3 as examples.

The first definition in Figure 5.3 describes the *ORIGIN* path attribute. Note first that the only attribute flag element present is the Transitive element. This shows that the path attribute is not

Optional and cannot be Partial or Extended. The *TYPE* element is another example of the dual representation pattern, which makes sure that any path attribute purporting to be of type ORIGIN has a type code of '1'. Finally, the *ORIGIN* element itself is defined to be another type from within the schema. It turns out that there are several predefined values for the ORIGIN field, each of which has a human-readable definition associated with it. As a result, the *bgp_origin_type* is actually an abstract type with derived types just like the ORIGIN path attribute type itself!

The *NEXT_HOP* type is a bit more straightforward. It too is mandatory transitive, and has an associated type code and definition. Its payload is defined to be an IPv4 address, so for that we can use the definition of that address family from the Network Datatypes library defined in Section 4.

This strategy of using abstract types is how this specification provides the necessary facilities for extension. Just as how a user would extend the Network Datatypes library to handle new address families, a user of this library can write an XML description for a new path attribute using the existing definitions as a template and XML instances that use the new path attribute will validate against the schema without needing to rewrite the definition of a path attribute.

See Appendix B for the full XSD.

5.3.3 Toward a BGP Database

As mentioned, one of the biggest drawbacks of the current archiving strategy is that the data is simply stored in flat directories. These files are saved at 15-minute intervals, and are only distinguished by which collector saved the messages.

It is rare that a user would be interested in the BGP archive in its entirety. Usually the user has a specific cross-section of the data that he is interested in, for example the messages containing his organization's prefixes or all updates from a particular peer router. To extract this data, the user then has to crawl through the entire archive and extract the relevant messages before any meaningful analysis can take place. This is both time-consuming and computationally wasteful.

A more efficient way to present the data to the user would be to use a relational database. Such a database could be designed to simply save update messages, which would allow a user to extract specific messages and perform longitunidal analysis based on prefixes, source, or any other set of criteria contained within a BGP message. Such a database would drastically reduce the overhead of searching the archive for specific messages, which could in turn provide greater agility for operators that are trying to analyze significant routing events such as route hijacks.

The database could also be designed to track prefix dynamics over time. In such a database, peers and prefixes are represented as a many-to-many relationship. Then, each combination of peer and prefix have specific records that correspond to when a given prefix is announced or withdrawn from a given peer's routing table. A database set up like this would provide the ability to perform novel analyses of prefix dynamics, for example which prefixes are the most or least stable or how prefix reachability differs between peers over time.

Any BGP database will grow in only a few different dimensions. Obviously there must be a table to represent the peer routers and the collectors they peer with. RouteViews and RIPE use about one- and three-dozen collectors, respectively, and maintain on the order of a hundred peers each. These numbers remain in the order of tens over time. Another table will be needed to store prefixes. In the IPv4 routing tables this number is in the order of 100,000's and the IPv6 tables are nearing the order of 10,000's.

Obviously there will be additional tables to store other information, for example AS numbers and timestamps. The database will grow in these dimensions as well. If the database is designed to represent prefix dynamics, then it will grow in the dynamics of the prefixes themselves, which is an unknown factor. This is distinct from the number of update messages because some updates are redundant and others do not add new records to the database. The author has experimented with such a database to study IPv6 data, and found that one month's worth of RouteViews' archive data produced a database of roughly ten million records. This is nontrivial, but hardly prohibitive or even within the realm of what currently constitutes "Big Data." Since the number of peers and collectors are relatively stable, the database can be expected to grow in $Num_{prefixes} * Num_{peers} * Flap_{avg}$ where $Flap_{avg}$ represents the average instability in all prefixes across all peers.

Using the BGP XSD to create a database also provides a way to find errors in the control-plane data. While a legal BGP message may produce undesirable results, e.g. a route hijack or a customer announcing a full table to its provider, these are data-plane errors. As already discussed, the database provides a significant improvement to the operator who needs to diagnose such a problem. However, the schema and database can also aid in finding and saving incorrect control-plane messages as well. Because the binary messages have to be converted into XML, they can be validated against the schema. In many cases, messages that do not pass this validation would just be thrown out. By adding an additional table to track incorrect, corrupted, or malformed messages, users can account for possible holes in their data set or see if bad messages were the cause of some network problem.

6 Describing a Routing Flow in XML

The standard described in Section 5.3 can represent any BGP message being sent between any pair of BGP peers. However, there is a great deal of additional data that would be lost by simply archiving massive numbers of these XML messages. For instance, BGP messages do not carry metadata to identify the peers in a flow, nor is there any representation of time within a message. These metadata are crucial to understanding the dynamics of the protocol after the fact.

Consider the researcher that is examining details of a significant internet-routing event, i.e. a massive prefix hijack or an announcement of a large number of new prefixes, for instance as on the World IPv6 Days in 2011 and 2012. Without the ability to distinguish which messages belong to which peering flows and the order in which each message was received, the dataset loses its diagnostic utility. It would be nice if, in addition to an XML representation of the data itself, we had a corresponding standard to represent the flow between two routers.

This utility would not be exclusively beneficial to BGP users either. As already mentioned, BGP is just one routing protocol among many. Distance-vector protcols such as RIP [Mal98] and link-state protocols such as OSPF [Moy98] are also commonly used, and researchers or operators that use these protocols would also benefit from a way to encode this idea of a routing flow for these other protocols. This presents a challenge: how to distill the common elements from each of these approaches and codify them in a basic XSD standard while maintaining enough flexbility so that the unique facets of each protocol can be fully expressed.

6.1 Defining a Routing Flow

To create a standard representation of a routing flow, the first step is to define what exactly a routing flow is. Consider an arbitrary internet- style network topology that contains at least two connected routers, and has at least one connected router that is running a routing protocol. Since one router is running a routing protocol, it is probably sending routing messages. Since it is able to send routing messages, there must be another router receiving them. There might be intervening hardware somewhere along the path, but that is irrelevant. The fact that one router is sending messages and another is receiving those same messages is enough to think of this routing flow as a pipe between the two.

A logical next question, then, is how one would describe the pipe? Since we have no knowledge of any other data on the network going from the source router to the destination router, the pipe has to be defined in terms of the routing protocol itself. In any standardized routing protocol, the protocol has a name, and commonly has a version number which may indicate incompatibility with any other version.

However, a protocol's name and version number may not be enough to fully describe the parameters by which messages are being exchanged. Within any particular version of a routing protocol, new functionality can be added. Some examples of this are Multiprotocol support in BGP-4, or IPv6 extensions to RIPng. These extra features are not necessarily present in every single channel of routing messages, even between the same two routers. Consequently, each unique routing flow is also chararcterized by the specific additional functionality, if any, of that particular flow. Again, the specifics of any of these extra features are irrelevant; this description has no need to know what ”4-byte AS number support” means, it just needs to know that it is active.

This is still not quite enough to fully describe the routing flow. Any two peers could, theoretically, have multiple simultanous routing flows using the same protocol with the same capabilities. Even worse, the routers could be using the same flow to send messages in both directions, as in BGP. Without knowing which messages belong to which flow, monitoring the flows becomes pointless. Ergo, the final distinguishing feature of a routing flow has to be the messages themselves.

There are a couple of implications that follow here. Including the routing messages as a facet of the flow implies that there are messages to begin with, so defining a routing flow between routers that never see the same routing messages would be invalid. Likewise, a connection-based protocol such as BGP cannot be represented as a single routing flow because any distinction between sender and receiver requires knowledge of the protocol, and not having that means that one cannot differentiate between messages being sent one way from those being sent the other. In cases like these, the actual BGP peering session can be represented as a pair of routing flows, one for each direction of message exchange.

One additional, subtle nuance in this definition is the requirement that the messages that the receiver gets are *identical* to the ones the sender sent. In many protocols, routing messages get modified by intermediate routers. Without knowing how the protocol changes the messages, we cannot say with certainty that the messages that the destination is receiving are the same messages that the source was sending, so we cannot say that there is a routing flow between these two routers. Again, decomposing the issue provides a way to represent the system; as the set of routing flows between each successive pair of routers along a long route.

This concept of a routing flow achieves the goals laid out above. Anyone that wants to represent the movement of routing messages can do so, without any knowledge of the inner workings of the routers involved or the protocols themselves. All she needs to know is the endpoints of the pipe, the dimensions of the pipe, and the contents of the pipe. All that remains is a convenient way to describe these data facets.

6.2 Describing Routing Flow Components

Following on from the previous chapters, XML provides a nice vehicle to represent a routing flow in. Because a routing flow is a descriptive entity, it fits very well within an XML-based representation.

6.2.1 Peer

Following the defintion above, the first pieces of a routing flow are the two peer routers. These routers, obviously, must be part of a network. In any modern network, they therefore must have network addresses, which have an AFI associated with them. We can represent them in XML, then, as instances of the *network_addr* abstract type that was defined in Section 4.

In addition to an address family, routing protocols often use TCP (as in BGP) or UDP (RIP) to send their update messages over. Therefore, these protocols make use of port numbers. While many of these ports are assigned by IANA, participating routers are not bound to use them, or in the case of router-emulating software like Quagga, might not be allowed to use them. Consequently, it makes sense to include the port numbers that each peer accepts routing messages on.

In a similar vein, every router in today's internet belongs to some AS. As a result, border routers exchange updates across AS boundaries. While AS numbers are not terribly important when all the routers belong to the same AS, it is crucial to have them in inter-domain routing. Again, without knowing anything about the details of a particular routing flow's protocol, the AS number must also be included.

6.2.2 Protocol and Extensions

Now that the participants in the flow have been defined, the next step is to include details of the particular routing protocol being used. As discussed in Section 6.1, a routing protocol can generally be identified by its name and version number, if present.

However, that is not enough to fully dimensionalize a routing flow. Each routing protocol has some set of extensions, and generally these need to be consistent between routers or the destination may not accept the source's messages.

Encoding these extensions requires a small violation of the design goal of total protocol ignorance. At a minimum, we must know the extension's name and include that in the routing flow's description. In practical usage, it might also be nice if we could include the machine-readable token that denotes the extension as well, but that should never be required. Nor should any specific parameters of the extension in a particular routing flow.

6.2.3 Routing Messages

Finally, the routing flow itself would be nothing but a pointless block of configured router memory without the messages being sent. Since this work presents a routing flow as an XML-based construct, clearly the routing messages themselves should be encoded in XML. Better still would be messages encoded in XML that conforms to a schema, but aside from BGP messages, that is beyond the scope of this work.

6.3 A Routing Flow XSD

The basic particle that we would like to represent in XML is a single routing flow. This flow can take on one of multiple definitions, based on the specific protocol being used. Once again,

```
<xsi:element name="SESSION" type="routing_session"/>
```

Figure 6.1: The root element in the XSD definition of a routing flow.

```
<xsi:complexType name="peer">
    <xsi:sequence>
        <xsi:element name="ADDRESS" type="net:network_addr"/>
        <xsi:element name="PORT" type="net:port"/>
        <xsi:element name="AS" type="net:asn"/>
    </xsi:sequence>
    <xsi:attribute name="name" type="xsi:normalizedString"/>
</xsi:complexType>
```

Figure 6.2: Definition of a peer router in a routing flow. The source and destination routers are instances of this type.

XSD abstract-type substitution saves the day, and helps define the root-level element in Figure 6.1

Section 6.2 describes the basic elements that will be present in any particular routing flow. All that is needed is the abstract type that encapsulates all of these pieces into an extensible form.

The first thing that needs to be translated is the definition of the peer. As previously discussed, a peer in a routing flow is identified by three pieces of information: its network address, the port number it accepts messages on, and the AS number that it belongs to. Conveniently, the network datatypes library from Chapter 4 has types for all three of these, so creating an encapsulating type with elements for each one is straightforward.

Experience shows that routers often have multiple addresses that correspond to multiple interfaces. This is even expected in the border routers of any non-stub AS. For a user, trying to decode which set of addresses correspond to which router can be confusing and lead to analysis errors. Therefore, allowing the user to add a canonical name to the peer would be helpful.

Putting these components together gives us the definition of a peer shown in Figure 6.2:

The next piece that needs to be described is the routing protocol being used in the flow. Obviously the name of the protocol needs to be included. Version numbers are common, but not always used, so the facet used for that has to be optional.

Additionally, a description of any extensions being used in the flow need to be included. As discussed in Section 6.2, the only thing that is absolutely necessary to represent a particular extension is its name. However, as also mentioned, these extensions are generally represented in the

```
<xsi:complexType name="protocol_type">
    <xsi:sequence>
        <xsi:element name="NAME" type="xsi:normalizedString"/>
        <xsi:element name="CAPABILITY" type="net:dual_repr" minOccurs="0" maxOccurs="unbounded"/>
    </xsi:sequence>
    <xsi:attribute name="version" type="xsi:unsignedByte"/>
</xsi:complexType>
```

Figure 6.3: Definition of a routing protocol w.r.t. a routing flow.

```
<xsi:complexType name="bgp_session">
    <xsi:complexContent>
        <xsi:restriction base="routing_session">
            <xsi:sequence>
                <xsi:element name="SOURCE_PEER" type="peer_type"/>
                <xsi:element name="DEST_PEER" type="peer_type"/>
                <xsi:element name="PROTOCOL">
                    <xsi:complexType>
                        <xsi:complexContent>
                            <xsi:restriction base="protocol_type">
                                <xsi:sequence>
                                    <xsi:element name="NAME"
type="xsi:normalizedString" fixed="BGP"/>
                                    <xsi:element name="CAPABILITY"
type="net:dual_repr" minOccurs="0" maxOccurs="unbounded"/>
                                </xsi:sequence>
                                <xsi:attribute name="version"
type="xsi:unsignedByte" default="4"/>
                            </xsi:restriction>
                        </xsi:complexContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element ref="xfb:BGP_MESSAGE" maxOccurs="unbounded"/>
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>
```

Figure 6.4: XSD definition of a BGP routing flow.

protocol by some machine code, a binary string or integer. It might be useful in actual use of this construct to include such a code. The network datatypes library actually includes a complex type that will accomplish this (*dual_repr*), so this is easily defined. Putting these together gives us the representation of a routing protocol given in Figure 6.3

Finally, the description of a routing flow needs to contain the messages sent from the source to the destination. The XML description of any particular routing protocol's messages, aside from BGP messages, are beyond the scope of this work.

However, since there is a BGP XSD available, we can use its definition of a BGP message to arrive at a definition of a BGP routing flow. This definition is shown in Figure 6.4.

The full text of the routing flow XSD and an example of a BGP routing flow are available in

Appendix C and Appendix D, respectively.

7 Conclusions

Monitoring routing dynamics is an important enterprise for operators and researchers. Therefore it is important to monitor and archive the routing messages to understand the dynamics of the network as well as provide insight when things go awry. This can, and does, create huge datasets. As with any such dataset, the data needs to be represented in a consistent, standardized format.

This new data representation should be human-readable, machine-readable, and extensible, but also be able to validate content, so that data errors can be detected without forcing a user to parse binary. XML and XSD provide us with the framework to accomplish all of this.

Monitoring is not limited to one particular protocol. As a result, trying to describe routing information flows in XSD cannot rely on knowledge of any one protocol, even though the protocol is a characteristic of the flow being described. This led to the development of an abstract XSD schema that provides the basis necessary to describe any routing flow, dependent on an associated schema to describe the messages of a particular protocol.

BGP is the de facto standard protocol used to route data between organizations in the internet. However, the current data format and tools used at the largest BGP archiving projects are difficult to use and can affect the integrity of the data. Therefore, we designed an XSD representation of BGP messages that provides legality and correctness validation of XML representations of BGP data, and can be easily extended for new additions to the protocol. This specification will be submitted to the appropriate organizations for consideration as a standard.

A side effect of encoding BGP into XSD is that several common networking constructs required definition as datatypes within XSD. As these are common constructs that are used in a wide range of places, they were developed into a separate schema that will also be submitted for consideration as a standard library.

By abstracting routing in this way, this work provides a template for anyone working on routing monitoring to represent whatever protocol they want in a readable, extensible, and portable manner, using a well-defined but also easily extended datatypes library. For anyone involved in BGP monitoring, this work also provides a very robust data format to represent their data in. This format can not only be used to check correctness of its source data, it can easily be extended to accommodate new features in the protocol without breaking already-deployed systems. This could provide a new level of clarity in fields beyond just monitoring.

References

[BCC06] T. Bates, E. Chen, and R. Chandra. *BGP Route Reflection: An Alternative to Full Mesg Internal BGP (IBGP)*. IETF, Apr 2006. RFC 4456.

[BCKR07] T. Bates, R. Chandra, D. Katz, and Y. Rekhter. *Multiprotocol Extensions for BGP-4*. IETF, Jan 2007. RFC 4760.

[Ber03] L. Berger. *Generalized Multi-Protocol Label Switching (GMPLS Signaling Functional Description*. IETF, Jan 2003. RFC 3471.

[BKL11] L. Blunk, M. Karir, and C. Labovitz. *Multi-Threaded Routing Toolkit (MRT) Routing Information Export Format*. IETF, Oct 2011. RFC 6396.

[BPSM$^+$08] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Langauge (XML) 1.0 (Fifth Ed.)*. W3C, Nov 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[Bra96] S. Bradner. *The Internet Standards Process – Revision 3*. IETF, Oct 1996. RFC 2026/BCP 9.

[BWR09] L. Berger, R. White, and E. Rosen. *BGP IPsec Tunnel Encapsulation Attribute*. IETF, Jun 2009. RFC 5566.

[Cen] Advanced Network Technology Center. University of oregon route views project. http://www.routeviews.org.

[CG06] E. Chen and V. Gillet. *Subcodes for BGP Cease Notification Message*. IETF, Apr 2006. RFC 4486.

[Che00] E. Chen. *Route Refresh Capability for BGP-4*. IETF, Sep 2000. RFC 2918.

[cisa] Cisco systems, inc. http://www.cisco.com/.

[Cisb] Cisco Networks. *Cisco XML Schemas*. http://www.cisco.com/en/US/docs/ios_xr_sw/iosxr_r4.1/xml/programming/guide/xl41sche.html.

[CMRW99] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. *Structure of Management Information Version 2 (SMIv2)*. IETF, Apr 1999. RFC 2578.

[Con] World Wide Web Consortium. W3c process document. http://www.w3.org/2005/10/Process-20051014/tr.

[CTL96] R. Chandrasekeran, P. Traina, and T. Li. *BGP Communities Attribute*. IETF, Aug 1996. RFC 1997.

[CY11] E. Chen and J. Yuan. *Autonomous-System-Wide Unique BGP Identifier for BGP-4*. IETF, Jun 2011. RFC 6286.

[DCS12] J. Dong, M. Chen, and A. Suryanarayana. *Subcodes for BGP Finite State Machine Error*. IETF, May 2012. RFC 6608.

[EN10] M. Ellison and B. Natale. *Expressing SNMP SMI Datatypes in XML Schema Definition Language*. IETF, August 2010. RFC 5935.

[FMP09] C. Filsfils, P. Mohapatra, and C. Pignataro. *Load-Balancing for Mesh Softwires*. IETF, Aug 2009. RFC 5640.

[Gro] Network Security Group. Bgp monitoring system. http://bgpmon.netsec.colostate.edu.

[Has95] D. Haskin. *A BGP/IDRP Route Server alternative to a full mesh routing*. IETF, Oct 1995. RFC 1863.

[Hat11] Mark Hatton. Regular expressions for ip addresses, cidr ranges and hostnames, Mar 2011. http://blog.markhatton.co.uk/2011/03/15/regular-expressions- for-ip-addresses-cidr-ranges-and-hostnames/.

[IAB] Internet architecture board. http://www.iab.org.

[IAN] Internet assigned numbers authority. http://www.iana.org/.

[IEE] Institute of electrical and electronics engineers. http://www.ieee.org/index.html.

[IETa] The internet engineering task force (ietf). http://www.ietf.org.

[IETb] IETF. The ietf process: an informal guide. http://www.ietf.org/about/process-docs.html.

[Int] Internet society. www.internetsociety.org.

[Mal98] G. Malkin. *RIP Version 2*. IETF, Nov 1998. RFC 4822.

[MD99] P. Marques and F. Dupont. *Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing*. IETF, Mar 1999. RFC 2545.

[Moy98] J. Moy. *OSPF Version 2*. IETF, Apr 1998. RFC 2328.

[MR09] P. Mohapatra and E. Rosen. *The BGP Encapsulation Subsequent Address Family Identifier (SAFI) and the BGP Tunnel Encapsulation Attribute*. IETF, Apr 2009. RFC 5512.

[MRP$^+$11] P. Marques, R. Raszuk, K. Patel, K. Kumaki, and T. Yamagata. *Internal BGP as the Provider/Customer Edge Protocol for BGP/MPLS IP Virtual Private Networks (VPNs)*. IETF, Sep 2011. RFC 6368.

[OBFR09] H. Ould-Brahim, D. Fedyk, and Y. Rekhter. *BGP Traffic Engineering Attribute*. IETF, May 2009. RFC 5543.

[Rek09] Y. Rekhter. *IPv6 Address Specific BGP Extended Community Attribute*. IETF, Nov 2009. RFC 5701.

[RIPa] Ripe network coordination centre. http://www.ripe.net.

[RIPb] RIPE NCC. *libBGPdump*. https://bitbucket.org/ripencc/bgpdump/wiki/Home.

[RLH06] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. IETF, Jan 2006. RFC 4271.

[Sav] Savannah Project. *Quagga Routing Suite*. http://savannah.nongnu.org/projects/quagga.

[SC09] J. Scudder and R. Chandra. *Capabilities Advertisement with BGP-4*. IETF, Feb 2009. RFC 3392.

[SS06] Y. Rekhter S. Sangli, D. Tappan. *BGP Extended Communities Attribute*. IETF, Feb 2006. RFC 4360.

[Sta] Open Stand. The modern standards paradigm. http://open-stand.org/principles/.

[TMS07] P. Traina, D. McPherson, and J. Scudder. *Autonomous System Conferderations for BGP*. IETF, Aug 2007. RFC 5065.

[VC07] Q. Vohra and E. Chen. *BGP Support for Four-octet AS Number Space*. IETF, May 2007. RFC 4893.

[W3C] World wide web consortium (w3c). http://www.w3.org.

[W3C07] W3C. *Protocol for Web Description Resources (POWDER): Web Description Resources Datatypes (WDRD)*, Sep 2007. http://www.w3.org/TR/powder-xsd/.

[W3C12] W3C. *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*, April 2012.

[XML] XMLsoft. *The XML C parser and toolkit of Gnome*. http://www.xmlsoft.org.

Appendices

A Network Primitives XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:ietf:params:xml:ns:network_base"
xmlns:net="urn:ietf:params:xml:ns:network_base"
elementFormDefault="qualified">

    <xsi:annotation>
        <xsi:documentation>
            net_base.xsd
            Written by Jason Bartlett
            Colorado State University
            30 August 2012

            This XSD provides a library of types to represent several primitive
            networking constructs.  It contains definitions for representing
            Autonomous System Numbers, port numbers, IEEE 802 MAC addresses,
            IPv4 and IPv6 addresses, and IPv4/v6 Classless Inter-Domain Routing
            (CIDR) prefixes.  The library also provides validation for these types.

            In addition, the library can be extended to define and validate
            other types of address families.  Addresses and prefixes are
            represented via abstract types.
            To extend this schema to support additional Address Family
            Identifiers (AFI), one need only define types that restrict the
            abstract types.
        </xsi:documentation>
    </xsi:annotation>

    <!-- The 'asn' abstract type provides a representation of an Autonomous System
    Number (ASN).  The derived types 'asn2' and 'asn4' provide bound-checking
    and fix the length of the ASN, in bytes, as an attribute. -->
<xsi:complexType name="asn" abstract="true">
   <xsi:simpleContent>
       <xsi:extension base="xsi:unsignedInt">
            <xsi:attribute name="byte_len" type="xsi:unsignedByte" use="required"/>
       </xsi:extension>
   </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="asn4">
   <xsi:simpleContent>
       <xsi:restriction base="net:asn">
           <xsi:attribute name="byte_len" type="xsi:unsignedByte" fixed="4" use="required"/>
       </xsi:restriction>
   </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="asn2">
   <xsi:simpleContent>
       <xsi:restriction base="net:asn">
           <xsi:maxInclusive value="65535"/>
           <xsi:attribute name="byte_len" type="xsi:unsignedByte" fixed="2" use="required"/>
       </xsi:restriction>
   </xsi:simpleContent>
</xsi:complexType>

<!-- A simple type to validate port numbers.  Will not allow values
    outside of the defined range. -->
    <xsi:simpleType name="port">
        <xsi:restriction base="xsi:unsignedShort"/>
    </xsi:simpleType>

    <!-- The network_addr type is an abstract type which provides a clean
    interface for other schema to use for elements that represent address
    families.  An instance may not use this type explicitly, but a schema that
    uses it will validate any instance of a derived type. -->
    <xsi:complexType name="network_addr" abstract="true">
        <xsi:simpleContent>
            <xsi:extension base="xsi:string">
                <xsi:attribute name="afi" type="xsi:unsignedShort" use="required"/>
                <xsi:anyAttribute/>
            </xsi:extension>
        </xsi:simpleContent>
    </xsi:complexType>

    <!-- The network_prefix type is an abstract type which provides a clean
    interface for other schema to use for elements that represent a prefix for
    some address family.  An instance may not use this type explicitly, but
    a schema that uses it will validate any instance of a derived type. -->
    <xsi:complexType name="network_prefix" abstract="true">
        <xsi:simpleContent>
            <xsi:extension base="xsi:string">
                <xsi:attribute name="afi" type="xsi:unsignedShort" use="required"/>
                <xsi:anyAttribute/>
            </xsi:extension>
        </xsi:simpleContent>
    </xsi:complexType>

    <!-- The ipv4_addr type is derived from the abstract network_addr type
    above.  It provides validation for the contained address via a regular
    expression, and also requires IPv4 addresses to be associated with their
    assigned Address Family Identifier, which is 1.  An instance may also
    specify a Subsequent Address Family Identifier, however, this value is not
    validated. -->
    <xsi:complexType name="ipv4_addr">
        <xsi:annotation>
            <xsi:documentation>
                Source: http://blog.markhatton.co.uk/2011/03/15/
                regular-expressions-for-ip-addresses-cidr-ranges-and-hostnames/
            </xsi:documentation>
        </xsi:annotation>
        <xsi:simpleContent>
            <xsi:restriction base="net:network_addr">
                <xsi:pattern value="(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}
            ([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])"/>
                <xsi:attribute name="afi" type="xsi:unsignedShort" fixed="1" use="required"/>
                <xsi:attribute name="safi" type="xsi:unsignedByte" use="optional"/>
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <!-- The ipv6_addr type is derived from the network_addr abstract type
    above.  It provides validation for any correct canonical representation of
    an IPv6 address, and includes the AFI for IPv6 addresses (2). -->
    <xsi:complexType name="ipv6_addr">
        <xsi:annotation>
            <xsi:documentation>
                Source: http://blog.markhatton.co.uk/2011/03/15/
                regular-expressions-for-ip-addresses-cidr-ranges-and-hostnames/
            </xsi:documentation>
        </xsi:annotation>
        <xsi:simpleContent>
            <xsi:restriction base="net:network_addr">
                <xsi:pattern value="/((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|
(([0-9A-Fa-f]{1,4}:){6}(:[0-9A-Fa-f]{1,4}|((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.
(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|
(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-Fa-f]{1,4}){1,2})|:((25[0-5]|2[0-4]\d|1\d\d|
[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|
(([0-9A-Fa-f]{1,4}:){4}(((:[0-9A-Fa-f]{1,4}){1,3})|((:[0-9A-Fa-f]{1,4})?:((25
[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]{1,4}){1,4})|((:[0-9A-Fa-f]{1,4}){0,2}:
```

```
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){2}(((:[0-9A-Fa-f]{1,4}){1,5})|((:[0-9A-Fa-f]{1,4}){0,3}:
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){1}(((:[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-f]{1,4}){0,4}:
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(:(((:[0-9A-Fa-f]{1,4}){1,7})|((:[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|2[0-4]\d|1\d
\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:)))(%.+)?/"/>
                <xsi:attribute name="afi" type="xsi:unsignedShort" fixed="2" use="required"/>
                <xsi:attribute name="safi" type="xsi:unsignedByte" use="optional"/>
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <!-- The ipv4_prefix type provides a representation for an IPv4 Classless
    Inter-Domain Routing (CIDR) prefix.  It provides validation for the content
    as well as for the specification of IPv4's AFI.  This type also defines an
    optional attribute for the length of the prefix.  This value is restricted
    to valid values for an IPv4 prefix, but is not checked against the content
    of the element.  When the two disagree, the user SHOULD use the value
    contained within the element. -->
    <xsi:complexType name="ipv4_prefix">
        <xsi:annotation>
            <xsi:documentation>
                Source: http://blog.markhatton.co.uk/2011/03/15/
                regular-expressions-for-ip-addresses-cidr-ranges-and-hostnames/
            </xsi:documentation>
        </xsi:annotation>
        <xsi:simpleContent>
            <xsi:restriction base="net:network_prefix">
                <xsi:pattern value="(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/([0-8])))"/>
                <xsi:pattern value="(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.)
               ([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(\d|1[0-6]))"/>
                <xsi:pattern value="(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){2}
               ([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(\d|1\d|2[0-4]))"/>
                <xsi:pattern value="(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}
               ([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(\d|[1-2]\d|3[0-2]))"/>
                <xsi:attribute name="afi" type="xsi:unsignedShort" fixed="1" use="required"/>
                <xsi:attribute name="safi" type="xsi:unsignedByte" use="optional"/>
                <xsi:attribute name="prefix_len" use="optional">
                    <xsi:simpleType>
                        <xsi:restriction base="xsi:unsignedByte">
                            <xsi:maxInclusive value="32"/>
                        </xsi:restriction>
                    </xsi:simpleType>
                </xsi:attribute>
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <!-- The ipv6_prefix type provides a representation for an IPv6 Classless
    Inter-Domain Routing (CIDR) prefix.  It provides validation for the content
    as well as for the specification of IPv6's AFI.  This type also defines an
    optional attribute for the length of the prefix.  This value is restricted
    to valid values for an IPv6 prefix, but is not checked against the content
    of the element.  When the two disagree, the user SHOULD use the value
    contained within the element. -->
    <xsi:complexType name="ipv6_prefix">
        <xsi:annotation>
            <xsi:documentation>
                Source: http://blog.markhatton.co.uk/2011/03/15/
                regular-expressions-for-ip-addresses-cidr-ranges-and-hostnames/
            </xsi:documentation>
        </xsi:annotation>
        <xsi:simpleContent>
            <xsi:restriction base="net:network_prefix">
                <xsi:pattern value="((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|
(([0-9A-Fa-f]{1,4}:){6}(:[0-9A-Fa-f]{1,4}|
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|
(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-Fa-f]{1,4}){1,2})|:((25[0-5]|2[0-4]\d|1\d\d|
[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|
(([0-9A-Fa-f]{1,4}:){4}(((:[0-9A-Fa-f]{1,4}){1,3})|((:[0-9A-Fa-f]{1,4})?:((25
[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]{1,4}){1,4})|((:[0-9A-Fa-f]{1,4}){0,2}:
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){2}(((:[0-9A-Fa-f]{1,4}){1,5})|((:[0-9A-Fa-f]{1,4}){0,3}:
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(([0-9A-Fa-f]{1,4}:){1}(((:[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-f]{1,4}){0,4}:
((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|
(:(((:[0-9A-Fa-f]{1,4}){1,7})|((:[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|2[0-4]\d|1\d
\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:)))(%.+)?(\/(\d|\d\d|1[0-1]\d|12[0-8]))"/>
                <xsi:attribute name="afi" type="xsi:unsignedShort" fixed="2" use="required"/>
                <xsi:attribute name="safi" type="xsi:unsignedByte" use="optional"/>
                <xsi:attribute name="prefix_len" use="optional">
                    <xsi:simpleType>
                        <xsi:restriction base="xsi:unsignedByte">
                            <xsi:maxInclusive value="128"/>
                        </xsi:restriction>
                    </xsi:simpleType>
                </xsi:attribute>
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

   <!-- The ieee802_mac type contains regular expressions that will provide
   validation for IEEE 802-style MAC addresses.  It will match any EUI-48,
   MAC-48, or EUI-64 address, as well as an address in the dotted-quad notation
   used by some network hardware companies. -->
   <xsi:complexType name="ieee_802mac">
       <xsi:annotation>
           <xsi:documentation>
               Source: http://stackoverflow.com/questions/4260467/
what-is-a-regular-expression-for-a-mac-address
               Source: http://manoharbhattarai.wordpress.com/2012/02/17/
regex-to-match-mac-address/
           </xsi:documentation>
       </xsi:annotation>
       <xsi:simpleContent>
           <xsi:restriction base="net:network_addr">
               <!-- The following two regular expressions match any EUI-48/MAC-48
               address.  There are two expressions here to cover addresses that
               use either a colon or a dash as their delimiter, but to avoid
               matching addresses that incorrectly mix the two. -->
               <xsi:pattern value="(([0-9A-Fa-f]{2}:){5}([0-9A-Fa-f]{2}))"/>
               <xsi:pattern value="(([0-9A-Fa-f]{2}-){5}([0-9A-Fa-f]{2}))"/>

               <!-- The following two regular expressions match any EUI-64 address.
               Again, two expressions are needed to avoid matching incorrect
               mixed-delimiter addresses. -->
               <xsi:pattern value="(([0-9A-Fa-f]{2}:){7}([0-9A-Fa-f]{2}))"/>
               <xsi:pattern value="(([0-9A-Fa-f]{2}-){7}([0-9A-Fa-f]{2}))"/>

               <!-- The following regular expression matches an 802 MAC address
               that is represented in dotted-quad notation (ex. 1234.dead.BEEF).-->
               <xsi:pattern value="(([0-9A-Fa-f]{4}\.){2}[0-9A-Fa-f]{4})"/>

               <xsi:attribute name="afi" type="unsignedShort" fixed="6" use="required"/>
           </xsi:restriction>
       </xsi:simpleContent>
   </xsi:complexType>

    <!-- This utility type provides a way to represent data as a string as well
    as some machine-readable simple type. -->
    <xsi:complexType name="dual_repr">
        <xsi:simpleContent>
            <xsi:extension base="xsi:normalizedString">
                <xsi:attribute name="value" type="xsi:anySimpleType" use="required"/>
            </xsi:extension>
        </xsi:simpleContent>
    </xsi:complexType>

</schema>
```

B XSD Format for BGP

```
<?xml version="1.0" encoding="UTF-8"?>
<xsi:schema targetNamespace="urn:ietf:params:xml:ns:xfb"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema"
    xmlns:net="urn:ietf:params:xml:ns:network_base"
    xmlns="urn:ietf:params:xml:ns:xfb"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xsi:import id="net" namespace="urn:ietf:params:xml:ns:network_base"
        schemaLocation="network_base.xsd" />

    <xsi:annotation>
        <xsi:documentation>
            An XML Format for BGP (XFB), version 0.6.

            This XSD provides a format to translate BGP-4 messages into XML.
            It currently supports the following RFCs:

            RFC 4271 - This is the base specification for BGP-4, the current
            version of the protocol.
            RFC 3392 - Defines Capabilities Advertisement in OPEN messages.
            RFC 2918 - Defines the Route Refresh capability.
            RFC 4893 - Defines the 4-byte AS Number support capability.
            RFC 4456 - Defines Route Reflection, which includes the ORIGINATOR_ID
            and CLUSTER_LIST path attributes.
            RFC 1863 - Defines the ADVERTISER and RCID_PATH path attributes.
            RFC 1997 - Defines the COMMUNITY path attribute.
            RFC 4360 - Defines the EXTENDED COMMUNITY path attribute.
            RFC 5668 - Defines the 4-Octet AS-Specific Extended Community.
            RFC 4760 - Defines the MP_REACH_NLRI and MP_UNREACH_NLRI path
            attributes.
            RFC 3471 - Defines the TRAFFIC_ENGINEERING path attribute.
            RFC 5543 - Defines values used within the TRAFFIC ENGINEERING
            path attribute.
            RFC 5512, 5566, 5640 - Define the TUNNEL_ENCAPSULATION path
            attribute and additional sub-TLV values for it.
            RFC 5701 - Defines the IPV6-SPECIFIC_EXTENDED_COMMUNITY path
            attribute.
            RFC 4486 - Adds error subcodes for the CEASE Notification message.
            RFC 5065 - Defines BGP Confederations, which includes the
            AS_CONFED_SEQUENCE and AS_CONFED_SET AS Path Segment types.
            RFC 2545 - Allows for more than one address in the NEXT_HOP field
            of the MP_REACH_NLRI path attribute.
            RFC 6608 - Adds error subcodes for the FSM error NOTIFICATION
            message.
            RFC 6286 - Allows the BGP Identifier to be represented as a 4-byte
            unsigned integer instead of an IPv4 address.
            RFC 6368 - Adds the ATTR_SET path attribute.
        </xsi:documentation>
    </xsi:annotation>

    <xsi:element name="BGP_MESSAGE" type="message_repr">
        <xsi:annotation>
            <xsi:documentation>
                The BGP_MESSAGE element is the root-level element of this
                specification. It is defined in such a way that multiple
                definitions of a BGP message (any restriction of the
                message_repr type) can be this element’s content. This
                document defines the ’ascii_msg’ and ’octet_msg’ types.
            </xsi:documentation>
        </xsi:annotation>
    </xsi:element>

    <!-- Section 1: Top-level definitions of the various components of a BGP
        message. These definitions come from RFC 4271. -->
```

```
<!-- The definition of a BGP message header. All BGP messages carry this
    piece. The MARKER element contains hexadecimal data, which should be 0xFF
    repeated 16 times. The TYPE element indicates the specific flavour of the
    rest of the message. Finally, the length attribute records the length in
    bytes of the entire ORIGINAL message, including the header. -->
<xsi:element name="BGP_HEADER">
    <xsi:complexType>
        <xsi:sequence>
            <xsi:element name="MARKER" type="xsi:hexBinary"
                fixed="FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF" />
            <xsi:element name="TYPE" type="bgp_msg_type" />
        </xsi:sequence>
        <xsi:attribute name="length" type="xsi:unsignedShort"
            use="required" />
    </xsi:complexType>
</xsi:element>

<!-- The definition of a BGP OPEN message. This message type has elements
    in it that correspond to the various session options that may be negotiated
    between two BGP speakers. These include version number, source ASN, hold
    time, and zero or more parameters. -->
<xsi:element name="OPEN">
    <xsi:complexType>
        <xsi:sequence>
            <xsi:element name="VERSION" type="xsi:unsignedByte" />
            <xsi:element name="SRC_ASN" type="net:asn2" />
            <xsi:element name="HOLD_TIME" type="xsi:unsignedShort" />
            <xsi:element name="BGP_IDENTIFIER" type="bgp_id_type" />
            <xsi:element name="PARAMETER" type="open_parameter"
                minOccurs="0" maxOccurs="unbounded" />
        </xsi:sequence>
        <xsi:attribute name="opt_parm_len" type="xsi:unsignedByte"
            use="required" />
    </xsi:complexType>
</xsi:element>

<!-- The definition of a BGP UPDATE message. This message type represents
    actual messages that communicate changes in a BGP speaker’s routing table.
    As per RFC 4271, WITHDRAWN and announced (NLRI) prefixes MUST be IPv4 prefixes.
    Any given UPDATE message contains zero or more path attributes. Each of these
    are represented in PATH_ATTRIBUTE elements. -->
<xsi:element name="UPDATE">
    <xsi:complexType>
        <xsi:sequence>
            <xsi:element name="WITHDRAWN" type="net:ipv4_addr"
                minOccurs="0" maxOccurs="unbounded" />
            <xsi:sequence minOccurs="0">
                <xsi:element name="PATH_ATTRIBUTE" type="path_attribute"
                    maxOccurs="unbounded" />
                <xsi:element name="NLRI" type="net:ipv4_prefix"
                    maxOccurs="unbounded" />
            </xsi:sequence>
        </xsi:sequence>
        <xsi:attribute name="withdrawn_len" type="xsi:unsignedShort"
            use="required" />
        <xsi:attribute name="path_attr_len" type="xsi:unsignedShort"
            use="required" />
    </xsi:complexType>
</xsi:element>

<!-- The definition of a BGP NOTIFICATION message. This message type represents
    error conditions that can occur within a BGP peering session. Each NOTIFICATION
    MUST have an error code associated with it. Each of these error codes MAY
    also have a subcode associated with it. Finally, based on the error code
    and subcode, a NOTIFICATION MAY also have a DATA element that contains additional
    information. The contents of such a DATA element MUST be defined within the
    scope of the error code/subcode pair. -->
```

```
<xsi:element name="NOTIFICATION">
    <xsi:complexType>
        <xsi:sequence>
            <xsi:element name="ERROR_CODE" type="notification_error" />
            <xsi:element name="ERROR_SUBCODE" type="notification_error_sub"
                minOccurs="0" />
            <xsi:element name="DATA" type="xsi:string" minOccurs="0" />
        </xsi:sequence>
    </xsi:complexType>
</xsi:element>

<!-- There is no additional definition for a BGP KEEPALIVE message. As per
    RFC 4271, a KEEPALIVE is simply a BGP message that contains only the header. -->

<!-- The definition of a BGP ROUTE-REFRESH message, as given in RFC 2918.
    A BGP speaker may actively request an update stream from a peer by issuing
    one of these messages. -->
<xsi:element name="ROUTE_REFRESH">
    <xsi:complexType>
        <xsi:sequence>
            <xsi:element name="AFI" type="xsi:unsignedShort" />
            <xsi:element name="SAFI" type="xsi:unsignedByte" />
        </xsi:sequence>
    </xsi:complexType>
</xsi:element>

<!-- Any unknown data that is encountered SHOULD be represented in a binary
    format which preserves the bytes as read off the wire. -->
<xsi:element name="UNKNOWN" type="rawBits" />

<!-- SECTION 2: INTERNAL DATA TYPES -->

<!-- This type is defined to save raw binary data. It gives the additional
    ability to annotate such data with attributes, for instance a length field
    or some idea of what the data is supposed to represent. -->
<xsi:complexType name="rawBits">
    <xsi:simpleContent>
        <xsi:extension base="xsi:hexBinary">
            <xsi:anyAttribute processContents="lax" />
        </xsi:extension>
    </xsi:simpleContent>
</xsi:complexType>

<!-- This type is included to support RFC 6286, which relaxes the RFC 4271
    requirement for a BGP Identifier to be a valid IPv4 address associated with
    a BGP speaker, and allows instead for the field to be a 32-bit, unsigned
    integer. This type enables either format to be used. -->
<xsi:simpleType name="bgp_id_type">
    <xsi:union>
        <xsi:simpleType>
            <xsi:restriction base="xsi:normalizedString">
                <xsi:pattern
                    value="(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}
        ([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])" />
            </xsi:restriction>
        </xsi:simpleType>
        <xsi:simpleType>
            <xsi:restriction base="xsi:unsignedInt" />
        </xsi:simpleType>
    </xsi:union>
</xsi:simpleType>

<!-- This is the XSD format for representing any BGP message in ASCII XML.
    Any such message MUST have a header and one of the specific message types,
    UNLESS the message is of type KEEPALIVE, in which case there MUST NOT be
    an additional element for a specific message type. -->
<xsi:complexType name="ascii_bgp">
    <xsi:complexContent>
```

```
        <xsi:restriction base="message_repr">
            <xsi:sequence>
                <xsi:element ref="BGP_HEADER" />
                <xsi:choice minOccurs="0">
                    <xsi:element ref="OPEN" />
                    <xsi:element ref="UPDATE" />
                    <xsi:element ref="NOTIFICATION" />
                    <xsi:element ref="ROUTE_REFRESH" />
                    <xsi:element ref="UNKNOWN" />
                </xsi:choice>
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- This is an additional way to represent an entire BGP message as raw
    binary data. It MUST only contain a single element that holds the binary
    data. No validation will be performed on such data. -->
<xsi:complexType name="octet_bgp">
    <xsi:complexContent>
        <xsi:restriction base="message_repr">
            <xsi:all>
                <xsi:element name="OCTETS" type="rawBits" />
            </xsi:all>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- SECTION 3: ABSTRACT TYPES. This section defines the abstract types
    that are used to enable future extensions to this work. These types MAY be
    used within derived XML Schema documents to validate against ANY complex
    type derived from the abstract type. These types MAY NOT be implemented within
    an XML instance. -->

<!-- This abstract type is defined so that a BGP message MAY be represented
    in an XML format not defined within this document. -->
<xsi:complexType name="message_repr" abstract="true">
    <xsi:annotation>
        <xsi:documentation>
            This abstract type is defined to allow for additional XML-based
            formats for a BGP message. This specification defines the
            "ascii_bgp" and "octet_bgp" derivations. Any additional
            derivations are the responsibility of that document’s author.
        </xsi:documentation>
    </xsi:annotation>
    <xsi:sequence>
        <xsi:any maxOccurs="unbounded" />
    </xsi:sequence>
    <xsi:anyAttribute />
</xsi:complexType>

<!-- This abstract type is defined to enumerate the different possible values
    for the TYPE element within the BGP_HEADER element defined in Section 1. -->
<xsi:complexType name="bgp_msg_type" abstract="true">
    <xsi:simpleContent>
        <xsi:restriction base="net:dual_repr" />
    </xsi:simpleContent>
</xsi:complexType>

<!-- This abstract type defines the general form of the parameters used
    in the BGP OPEN message. -->
<xsi:complexType name="open_parameter" abstract="true">
    <xsi:sequence>
        <xsi:element name="TYPE" type="xsi:unsignedByte" />
        <xsi:any maxOccurs="unbounded" />
    </xsi:sequence>
    <xsi:attribute name="length" type="xsi:unsignedByte"
        use="required" />
```

```
</xsi:complexType>

<!-- This abstract type defines the general form of a BGP Path Attribute.
    It includes definitions for the attribute flags and attribute type values. -->
<xsi:complexType name="path_attribute" abstract="true">
    <xsi:sequence>
        <xsi:element name="OPTIONAL" minOccurs="0" />
        <xsi:element name="TRANSITIVE" minOccurs="0" />
        <xsi:element name="PARTIAL" minOccurs="0" />
        <xsi:element name="EXTENDED" minOccurs="0" />
        <xsi:element name="TYPE" type="net:dual_repr" />
        <xsi:any minOccurs="0" maxOccurs="unbounded" />
    </xsi:sequence>
    <xsi:attribute name="length" type="xsi:unsignedShort"
        use="required" />
</xsi:complexType>

<!-- This type is the base for the different values of the ORIGIN path attribute. -->
<xsi:complexType name="bgp_origin_type" abstract="true">
    <xsi:simpleContent>
        <xsi:restriction base="net:dual_repr" />
    </xsi:simpleContent>
</xsi:complexType>

<!-- This abstract type defines the general form of an Extended Community
    as defined in RFC 4360. -->
<xsi:complexType name="extended_community" abstract="true">
    <xsi:sequence>
        <xsi:choice>
            <xsi:element name="FCFS" />
            <xsi:element name="STANDARDS" />
            <xsi:element name="EXPERIMENTAL" />
        </xsi:choice>
        <xsi:element name="TRANSITIVE" minOccurs="0" />
        <xsi:element name="TYPE" type="net:dual_repr" />
        <xsi:any minOccurs="0" maxOccurs="unbounded" />
    </xsi:sequence>
</xsi:complexType>

<!-- This type defines the format of the IPv6-Specific Extended Communities
    given in RFC 5701. -->
<xsi:complexType name="ipv6_extended_community"
    abstract="true">
    <xsi:sequence>
        <xsi:element name="TRANSITIVE" minOccurs="0" />
        <xsi:element name="TYPE" type="net:dual_repr" />
        <xsi:element name="GLOBAL_ADMINISTRATOR" type="net:ipv6_addr" />
        <xsi:any minOccurs="0" maxOccurs="unbounded" />
    </xsi:sequence>
</xsi:complexType>

<!-- This abstract type uses a TLV format to represent the Tunnel Encapsulation
    path attribute. Because of the differences in sub-TLVs that are available,
    each derived type MUST define each sub-TLV that may be used within it. -->
<xsi:complexType name="tunnel_encap_tlv" abstract="true">
    <xsi:sequence>
        <xsi:element name="TUNNEL_TYPE" type="net:dual_repr" />
        <xsi:any minOccurs="0" maxOccurs="unbounded" />
    </xsi:sequence>
    <xsi:attribute name="length" type="xsi:unsignedShort"
        use="required" />
</xsi:complexType>

<!-- A general definition for the error codes defined for the NOTIFICATION
    message. This type also provides the association between the error message
    and the integer error code. -->
<xsi:complexType name="notification_error" abstract="true">
    <xsi:simpleContent>
```

```
        <xsi:restriction base="net:dual_repr" />
    </xsi:simpleContent>
</xsi:complexType>

<!-- A general definition for the error subcodes defined for the NOTIFICATION
    message. As with the error codes, an implementation MUST provide the correct
    error code value for the particular error subtype. -->
<xsi:complexType name="notification_error_sub"
    abstract="true">
    <xsi:simpleContent>
        <xsi:restriction base="net:dual_repr" />
    </xsi:simpleContent>
</xsi:complexType>

<!-- SECTION 4: BGP HEADER MESSAGE-TYPE DEFINITIONS These types define all
    current values for the TYPE element of the BGP header. -->

<xsi:complexType name="bgp_open_msg_id">
    <xsi:simpleContent>
        <xsi:restriction base="bgp_msg_type">
            <xsi:enumeration value="OPEN" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="1" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bgp_update_msg_id">
    <xsi:simpleContent>
        <xsi:restriction base="bgp_msg_type">
            <xsi:enumeration value="UPDATE" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="2" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bgp_notification_msg_id">
    <xsi:simpleContent>
        <xsi:restriction base="bgp_msg_type">
            <xsi:enumeration value="NOTIFICATION" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="3" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bgp_keepalive_msg_id">
    <xsi:simpleContent>
        <xsi:restriction base="bgp_msg_type">
            <xsi:enumeration value="KEEPALIVE" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="4" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bgp_rr_msg_id">
    <xsi:simpleContent>
        <xsi:restriction base="bgp_msg_type">
            <xsi:enumeration value="ROUTE_REFRESH" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="5" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<!-- SECTION 5: OPEN PARAMETER DEFINITIONS These are the currently-defined
```

```
    parameters for the OPEN message. -->

<!-- Describes the Authentication parameter defined in RFC 1771. -->
<xsi:complexType name="authentication_parameter">
    <xsi:annotation>
        <xsi:documentation>
            Authentication was defined in the RFC 1771 definition of BGP-4,
            but it has since been deprecated.
        </xsi:documentation>
    </xsi:annotation>
    <xsi:complexContent>
        <xsi:restriction base="open_parameter">
            <xsi:sequence>
                <xsi:element name="TYPE" type="xsi:unsignedByte" fixed="1" />
                <xsi:element name="AUTHENTICATION" minOccurs="1"
                    maxOccurs="1">
                    <xsi:complexType>
                        <xsi:sequence>
                            <xsi:element name="CODE" type="xsi:unsignedByte" />
                            <xsi:element name="DATA" type="xsi:hexBinary" />
                        </xsi:sequence>
                    </xsi:complexType>
                </xsi:element>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- This type describes the Capabilities OPEN parameter as given in RFC
    5492. It allows for multiple CAPABILITY elements, each of which represents
    a single capability TLV. -->
<xsi:complexType name="capabilities_parameter">
    <xsi:annotation>
        <xsi:documentation>
            Capabilities announcement is defined in RFC 5492.
        </xsi:documentation>
    </xsi:annotation>
    <xsi:complexContent>
        <xsi:restriction base="open_parameter">
            <xsi:sequence>
                <xsi:element name="TYPE" type="xsi:unsignedByte" fixed="2" />
                <xsi:element name="CAPABILITY" maxOccurs="unbounded">
                    <xsi:complexType>
                        <xsi:sequence>
                            <xsi:element name="CODE" type="xsi:unsignedByte" />
                            <xsi:any minOccurs="0" maxOccurs="unbounded" />
                        </xsi:sequence>
                        <xsi:attribute name="length" type="xsi:unsignedByte"
                            use="required" />
                    </xsi:complexType>
                </xsi:element>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- This type allows for any other OPEN parameter to be represented in
    an instance document and still validate. The capability parameter MUST be
    preserved in hexBinary within the UNKNOWN element. -->
<xsi:complexType name="unknown_parameter">
    <xsi:complexContent>
        <xsi:restriction base="open_parameter">
            <xsi:sequence>
                <xsi:element name="TYPE" type="xsi:unsignedByte" />
                <xsi:element ref="UNKNOWN" />
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- SECTION 6: UPDATE MESSAGE PATH ATTRIBUTES This section defines types
    for BGP Path Attributes. Supported RFC’s are listed in the top schema documentation. -->

<!-- Describes the Origin path attribute. This is a well-known mandatory
    attribute, and has several defined values, which are defined below. -->
<xsi:complexType name="origin_path_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="ORIGIN" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="1" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="ORIGIN" type="bgp_origin_type" />
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- The following three derived types provide the validation necessary
    between the integer value and the human-readable string for the defined origin
    types. -->
<xsi:complexType name="bgp_igp_origin">
    <xsi:simpleContent>
        <xsi:restriction base="bgp_origin_type">
            <xsi:enumeration value="IGP" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="0" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bgp_egp_origin">
    <xsi:simpleContent>
        <xsi:restriction base="bgp_origin_type">
            <xsi:enumeration value="EGP" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="1" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bgp_incomplete_origin">
    <xsi:simpleContent>
        <xsi:restriction base="bgp_origin_type">
            <xsi:enumeration value="INCOMPLETE" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="2" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
```

```
</xsi:complexType>

<!-- Describes the AS Path attribute. This is a mandatory well-known attribute
    of varying length. It is comprised of one or more AS Path Segments, the definitions
    of which are below. According to RFC 4893, this attribute MAY contain 4-byte
    AS numbers -->
<xsi:complexType name="as_path_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="EXTENDED" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="AS_PATH" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="2" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:choice maxOccurs="unbounded">
                    <xsi:element name="AS_SEQUENCE" type="as_seq_type" />
                    <xsi:element name="AS_SET" type="as_set_type" />
                    <xsi:element name="AS_CONFED_SEQUENCE" type="as_seq_type" />
                    <xsi:element name="AS_CONFED_SET" type="as_set_type" />
                </xsi:choice>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedShort"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- These two types distinguish an AS Sequence and AS Set. The AS Set type
    is represented as an unordered set, while the sequence is required to be
    in some order. -->
<xsi:complexType name="as_seq_type">
    <xsi:sequence>
        <xsi:element name="AS" type="net:asn" maxOccurs="unbounded" />
    </xsi:sequence>
    <xsi:attribute name="length" type="xsi:unsignedByte"
        use="required" />
</xsi:complexType>

<xsi:complexType name="as_set_type">
    <xsi:choice maxOccurs="unbounded">
        <xsi:element name="AS" type="net:asn" maxOccurs="unbounded" />
    </xsi:choice>
    <xsi:attribute name="length" type="xsi:unsignedByte"
        use="required" />
</xsi:complexType>

<!-- Describes the Next Hop attribute. This is a well-known mandatory path
    attribute, and contains an IPv4 address. -->
<xsi:complexType name="next_hop_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="NEXT_HOP" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
```

```
                                    fixed="3" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="NEXT_HOP" type="net:ipv4_addr" />
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the Multi-Exit Discriminator path attribute. This attribute
    is optional and non-transitive, and is just an integer. -->
<xsi:complexType name="med_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="MULTI_EXIT_DISC" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="4" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="MED" type="xsi:unsignedInt" />
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the Local Preference path attribute. This attribute is mandatory
    and transitive. It is just an integer value. -->
<xsi:complexType name="local_pref_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="LOCAL_PREF" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="5" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="LOCAL_PREF" type="xsi:unsignedInt" />
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the Atomic Aggregate path attribute. It is a transitive attribute,
    and is unique in that it has no content. -->
<xsi:complexType name="atomic_aggr_attr">
```

```
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="ATOMIC_AGGREGATE" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="6" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                fixed="0" use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the Aggregator path attribute. This attribute is optional
    and transitive, and contains an AS Number and a BGP Identifier. -->
<xsi:complexType name="aggregator_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="PARTIAL" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="AGGREGATOR" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="7" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="AGGREGATOR">
                    <xsi:complexType>
                        <xsi:sequence>
                            <xsi:element name="AS" type="net:asn" />
                            <xsi:element name="BGP_IDENTIFIER" type="bgp_id_type" />
                        </xsi:sequence>
                    </xsi:complexType>
                </xsi:element>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the Communities attribute as defined in RFC 1997. This path
    attribute is optional and transitive. The RFC defines several well-known
    communities. This attribute MAY contain any one of those or a sequence of
    Community values, each of which is composed of a 2-byte AS number and an
    additional value. -->
<xsi:complexType name="communities_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="PARTIAL" minOccurs="0" />
                <xsi:element name="EXTENDED" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="COMMUNITIES" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="8" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:choice>
                    <xsi:element name="NO_EXPORT" type="xsi:hexBinary"
                        fixed="FFFFFF01" />
                    <xsi:element name="NO_ADVERTISE" type="xsi:hexBinary"
                        fixed="FFFFFF02" />
                    <xsi:element name="NO_EXPORT_SUBCONFED" type="xsi:hexBinary"
                        fixed="FFFFFF03" />
                    <xsi:element name="COMMUNITY" maxOccurs="unbounded">
                        <xsi:complexType>
                            <xsi:sequence>
                                <xsi:element name="AS" type="net:asn2" />
                                <xsi:element name="VALUE" type="xsi:unsignedShort" />
                            </xsi:sequence>
                        </xsi:complexType>
                    </xsi:element>
                </xsi:choice>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedShort"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the ORIGINATOR_ID attribute defined in RFC 4456. This attribute
    is an optional, non-transitive attribute that contains a BGP Identifier. -->
<xsi:complexType name="originator_id_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="ORIGINATOR_ID" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="9" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="ORIGINATOR_ID" type="bgp_id_type" />
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                fixed="4" use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the CLUSTER_LIST attribute defined in RFC 4456. It is an
    optional, non-transitive attribute that carries one or more Cluster identifiers. -->
<xsi:complexType name="cluster_list_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
```

```
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="CLUSTER_LIST" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="10" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="CLUSTER_ID" type="xsi:unsignedInt"
                    maxOccurs="unbounded" />
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- The ADVERTISER attribute is defined in RFC 1863, and made Historic
    by RFC 4223. It is included for compatibility, but SHOULD NOT be used in
    new instance documents. -->
<xsi:complexType name="advertiser_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="ADVERTISER" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="12" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="ADVERTISER" type="net:network_addr"
                    maxOccurs="unbounded" />
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the MP_REACH_NLRI path attribute as defined in RFC 4760.
    It is an optional, transitive attribute that carries path information for
    any address family with a defined AFI. and MP_UNREACH_NLRI are defined in
    RFC 4760. -->
<xsi:complexType name="mp_reach_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="EXTENDED" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="MP_REACH_NLRI" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="14" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="MP_REACH_NLRI">
                    <xsi:complexType>
                        <xsi:sequence>
                            <xsi:element name="AFI" type="xsi:unsignedShort" />
                            <xsi:element name="SAFI" type="xsi:unsignedByte" />
                            <xsi:element name="NEXT_HOP_LEN" type="xsi:unsignedByte" />
                            <xsi:element name="NEXT_HOP" type="net:network_addr"
                                maxOccurs="unbounded" />
                            <xsi:element name="MP_NLRI" type="net:network_prefix"
                                maxOccurs="unbounded" />
                        </xsi:sequence>
                    </xsi:complexType>
                </xsi:element>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedShort"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the MP_UNREACH_NLRI path attribute as defined in RFC 4760.
    It is an optional, non-transitive attribute that gives an AFI, SAFI, and
    specific routes that are being withdrawn by a BGP speaker. -->
<xsi:complexType name="mp_unreach_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="EXTENDED" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="MP_UNREACH_NLRI" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="15" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="MP_UNREACH_NLRI">
                    <xsi:complexType>
                        <xsi:sequence>
                            <xsi:element name="AFI" type="xsi:unsignedShort" />
                            <xsi:element name="SAFI" type="xsi:unsignedByte" />
                            <xsi:element name="WITHDRAWN" type="net:network_prefix"
                                maxOccurs="unbounded" />
                        </xsi:sequence>
                    </xsi:complexType>
                </xsi:element>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedShort"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the Extended Communities path attribute as defined in RFC
    4360. It is optional and transitive, and carries one or more Extended Communities. -->
<xsi:complexType name="ext_community_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="PARTIAL" minOccurs="0" />
                <xsi:element name="EXTENDED" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="EXTENDED_COMMUNITIES" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="16" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="COMMUNITY" type="extended_community"
                    maxOccurs="unbounded" />
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Desribes the AS4 Path Attribute as defined in RFC 4893. Unlike the
    AS Path attribute above, this attribute MUST contain 4-byte AS numbers, and
    MAY NOT contain AS_CONFED_SET/SEQUENCE path segments. -->
<xsi:complexType name="as4_path_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="PARTIAL" minOccurs="0" />
                <xsi:element name="EXTENDED" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="AS4_PATH" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="17" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:choice maxOccurs="unbounded">
                    <xsi:element name="AS_SEQUENCE" type="as4_seq_type" />
                    <xsi:element name="AS_SET" type="as4_set_type" />
                </xsi:choice>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedShort"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- These two types define either a sequence or set of 4-byte AS Numbers. -->
<xsi:complexType name="as4_seq_type">
    <xsi:sequence>
        <xsi:element name="AS" type="net:asn4" maxOccurs="unbounded" />
    </xsi:sequence>
    <xsi:attribute name="length" type="xsi:unsignedByte"
        use="required" />
</xsi:complexType>

<xsi:complexType name="as4_set_type">
    <xsi:choice maxOccurs="unbounded">
        <xsi:element name="AS" type="net:asn4" maxOccurs="unbounded" />
    </xsi:choice>
    <xsi:attribute name="length" type="xsi:unsignedByte"
        use="required" />
</xsi:complexType>
```

```
<!-- Describes the AS4 Aggregator path attribute as defined in RFC 4893.
    Unlike the Aggregator attribute above, this attribute MUST contain a 4-byte
    AS number in addition to the BGP Identifier. -->
<xsi:complexType name="as4_aggregator_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="PARTIAL" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="AS4_AGGREGATOR" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="18" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="AS4_AGGREGATOR">
                    <xsi:complexType>
                        <xsi:sequence>
                            <xsi:element name="AS" type="net:asn4" />
                            <xsi:element name="BGP_IDENTIFIER" type="bgp_id_type" />
                        </xsi:sequence>
                    </xsi:complexType>
                </xsi:element>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                fixed="8" use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the Tunnel Encapsulation path attribute defined in RFC 5512.
    It is optional and transitive, and contains one or more TLV elements which
    each correspond to a specific tunneling technology. -->
<xsi:complexType name="tunnel_encap_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="PARTIAL" minOccurs="0" />
                <xsi:element name="EXTENDED" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="TUNNEL_ENCAPSULATION" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="23" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="TUNNEL" type="tunnel_encap_tlv"
                    maxOccurs="unbounded" />
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedShort"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>
```

```
<!-- Describe the IPv6-Specific Extended Community path attribute defined
    in RFC 5701. While the format of this attribute is extremely similar to the
    Extended Communities attribute above, it is a distinct path attribute with
    its own type code and is not interchangeable. -->
<xsi:complexType name="ipv6_ext_community_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="PARTIAL" minOccurs="0" />
                <xsi:element name="EXTENDED" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="IPV6_SPECIFIC_EXTENDED_COMMUNITIES" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="16" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="COMMUNITY" type="ipv6_extended_community"
                    maxOccurs="unbounded" />
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the ATTR_SET path attribute defined in RFC 6368. This optional
    transitive attribute contains a set of any other path attribute, except for
    MP_REACH and MP_UNREACH. -->
<xsi:complexType name="attr_set_attr">
    <xsi:complexContent>
        <xsi:restriction base="path_attribute">
            <xsi:sequence>
                <xsi:element name="OPTIONAL" minOccurs="1" />
                <xsi:element name="TRANSITIVE" minOccurs="1" />
                <xsi:element name="PARTIAL" minOccurs="0" />
                <xsi:element name="EXTENDED" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="ATTR_SET" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="128" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="ATTR_SET">
                    <xsi:complexType>
                        <xsi:sequence>
                            <xsi:element name="ORIGIN_AS" type="net:asn4" />
                            <xsi:element name="PATH_ATTRIBUTE" type="path_attribute"
                                maxOccurs="unbounded" />
                        </xsi:sequence>
                    </xsi:complexType>
                </xsi:element>
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- SECTION 7: EXTENDED COMMUNITY DEFINITIONS -->
```

```
<!-- Describes the 2-byte AS-Specific Extended Community defined in RFC
    4360. It has two fields: A 2-byte AS number and some locally-administered
    value. -->
<xsi:complexType name="asn2_specific_ext_com">
    <xsi:complexContent>
        <xsi:restriction base="extended_community">
            <xsi:sequence>
                <xsi:choice>
                    <xsi:element name="FCFS" />
                    <xsi:element name="STANDARDS" />
                </xsi:choice>
                <xsi:element name="TRANSITIVE" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="2-OCTET_AS-SPECIFIC" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="0" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="VALUE">
                    <xsi:complexType>
                        <xsi:sequence>
                            <xsi:element name="GLOBAL_ADMINISTRATOR" type="net:asn2" />
                            <xsi:any />
                        </xsi:sequence>
                    </xsi:complexType>
                </xsi:element>
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the IPv4-Specific Extended Community defined in RFC 4360.
    It contains an IPv4 address and some locally-administered value. -->
<xsi:complexType name="ipv4_specific_ext_com">
    <xsi:complexContent>
        <xsi:restriction base="extended_community">
            <xsi:sequence>
                <xsi:choice>
                    <xsi:element name="FCFS" />
                    <xsi:element name="STANDARDS" />
                </xsi:choice>
                <xsi:element name="TRANSITIVE" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="IPV4-SPECIFIC" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="1" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="VALUE">
                    <xsi:complexType>
                        <xsi:sequence>
                            <xsi:element name="GLOBAL_ADMINISTRATOR" type="net:ipv4_addr" />
                            <xsi:any />
                        </xsi:sequence>
                    </xsi:complexType>
                </xsi:element>
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- Describes the Opaque Extended Community. This community has no predefined
    fields, so the content of such a community is preserved here in hex. -->
<xsi:complexType name="opaque_ext_com">
    <xsi:complexContent>
        <xsi:restriction base="extended_community">
            <xsi:sequence>
                <xsi:choice>
                    <xsi:element name="FCFS" />
                    <xsi:element name="STANDARDS" />
                </xsi:choice>
                <xsi:element name="TRANSITIVE" minOccurs="0" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="OPAQUE" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="3" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="VALUE">
                    <xsi:simpleType>
                        <xsi:restriction base="xsi:hexBinary">
                            <xsi:length value="6" />
                        </xsi:restriction>
                    </xsi:simpleType>
                </xsi:element>
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- The following two extended communities are defined in RFC 5512, and
    provide additional functionality in the service of the Tunnel Encapsulation
    attribute. -->
<xsi:complexType name="color_ext_com">
    <xsi:complexContent>
        <xsi:restriction base="extended_community">
            <xsi:sequence>
                <xsi:element name="FCFS" />
                <xsi:element name="TRANSITIVE" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="COLOR" />
                                <xsi:attribute name="value" type="xsi:hexBinary"
                                    fixed="030B" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="COLOR" type="xsi:unsignedInt" />
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<xsi:complexType name="encapsulation_ext_com">
    <xsi:complexContent>
        <xsi:restriction base="extended_community">
            <xsi:sequence>
                <xsi:element name="FCFS" />
                <xsi:element name="TRANSITIVE" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="ENCAPSULATION" />
                                <xsi:attribute name="value" type="xsi:hexBinary"
                                    fixed="030C" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="TUNNEL_TYPE" type="xsi:unsignedShort" />
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- These definitions represent the defined IPv6-Specific Extended Communities.
    These are defined in RFC 5701, and are not interchangeable with the above
    Extended Communities. -->
<xsi:complexType name="ipv6_route_target_ext_com">
    <xsi:complexContent>
        <xsi:restriction base="ipv6_extended_community">
            <xsi:sequence>
                <xsi:element name="TRANSITIVE" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="IPV6-SPECIFIC_ROUTE_TARGET" />
                                <xsi:attribute name="value" type="xsi:hexBinary"
                                    fixed="0002" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="GLOBAL_ADMINISTRATOR" type="net:ipv6_addr" />
                <xsi:element name="LOCAL_ADMINISTRATOR">
                    <xsi:simpleType>
                        <xsi:restriction base="xsi:hexBinary">
                            <xsi:length value="2" />
                        </xsi:restriction>
                    </xsi:simpleType>
                </xsi:element>
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<xsi:complexType name="ipv6_route_origin_ext_com">
    <xsi:complexContent>
        <xsi:restriction base="ipv6_extended_community">
            <xsi:sequence>
                <xsi:element name="TRANSITIVE" />
                <xsi:element name="TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="IPV6-SPECIFIC_ROUTE_ORIGIN" />
                                <xsi:attribute name="value" type="xsi:hexBinary"
                                    fixed="0003" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:element name="GLOBAL_ADMINISTRATOR" type="net:ipv6_addr" />
                <xsi:element name="LOCAL_ADMINISTRATOR">
```

```
                    <xsi:simpleType>
                        <xsi:restriction base="xsi:hexBinary">
                            <xsi:length value="2" />
                        </xsi:restriction>
                    </xsi:simpleType>
                </xsi:element>
            </xsi:sequence>
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- SECTION 8: TUNNEL ENCAPSULATION TYPES -->

<!-- L2TPv3-IP, GRE, and IP-over-IP are defined in RFC 5512, and the LOAD_BALANCING
    sub-TLV is defined in RFC 5640. RFC 5566 adds the IPsec Tunnel Authenticator
    sub-TLV, in addition to several other tunnel encapsulation types which are
    described below. -->
<xsi:complexType name="l2tpv3_ip_tlv">
    <xsi:complexContent>
        <xsi:restriction base="tunnel_encap_tlv">
            <xsi:sequence>
                <xsi:element name="TUNNEL_TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="L2TPV3_OVER_IP" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="1" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:choice minOccurs="0" maxOccurs="unbounded">
                    <xsi:sequence>
                        <xsi:element name="ENCAPSULATION">
                            <xsi:complexType>
                                <xsi:sequence>
                                    <xsi:element name="SESSION_ID" type="xsi:nonNegativeInteger" />
                                    <xsi:element name="COOKIE" minOccurs="0">
                                        <xsi:simpleType>
                                            <xsi:restriction base="xsi:hexBinary">
                                                <xsi:length value="8" />
                                            </xsi:restriction>
                                        </xsi:simpleType>
                                    </xsi:element>
                                </xsi:sequence>
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="1" use="required" />
                            </xsi:complexType>
                        </xsi:element>
                        <xsi:element name="LOAD_BALANCING" minOccurs="0">
                            <xsi:simpleType>
                                <xsi:restriction base="xsi:hexBinary">
                                    <xsi:length value="2" />
                                </xsi:restriction>
                            </xsi:simpleType>
                        </xsi:element>
                    </xsi:sequence>
                    <xsi:element name="PROTOCOL">
                        <xsi:simpleType>
                            <xsi:restriction base="xsi:hexBinary">
                                <xsi:length value="2" />
                            </xsi:restriction>
                        </xsi:simpleType>
                    </xsi:element>
                    <xsi:element name="COLOR" type="color_ext_com" />
                    <xsi:element name="IPSEC_TUNNEL_AUTH">
                        <xsi:complexType>
```

```
                            <xsi:sequence>
                                <xsi:element name="TYPE" type="xsi:unsignedShort" />
                                <xsi:element name="VALUE" type="xsi:anySimpleType" />
                            </xsi:sequence>
                            <xsi:attribute name="length" type="xsi:unsignedByte" />
                        </xsi:complexType>
                    </xsi:element>
                </xsi:choice>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedShort"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<xsi:complexType name="gre_tlv">
    <xsi:complexContent>
        <xsi:restriction base="tunnel_encap_tlv">
            <xsi:sequence>
                <xsi:element name="TUNNEL_TYPE">
                    <xsi:complexType>
                        <xsi:simpleContent>
                            <xsi:restriction base="net:dual_repr">
                                <xsi:enumeration value="GRE" />
                                <xsi:attribute name="value" type="xsi:unsignedByte"
                                    fixed="2" use="required" />
                            </xsi:restriction>
                        </xsi:simpleContent>
                    </xsi:complexType>
                </xsi:element>
                <xsi:choice minOccurs="0" maxOccurs="unbounded">
                    <xsi:sequence>
                        <xsi:element name="GRE_KEY" type="xsi:unsignedInt" />
                        <xsi:element name="LOAD_BALANCING" minOccurs="0">
                            <xsi:simpleType>
                                <xsi:restriction base="xsi:hexBinary">
                                    <xsi:length value="2" />
                                </xsi:restriction>
                            </xsi:simpleType>
                        </xsi:element>
                    </xsi:sequence>
                    <xsi:element name="PROTOCOL">
                        <xsi:simpleType>
                            <xsi:restriction base="xsi:hexBinary">
                                <xsi:length value="2" />
                            </xsi:restriction>
                        </xsi:simpleType>
                    </xsi:element>
                    <xsi:element name="COLOR" type="color_ext_com" />
                    <xsi:element name="IPSEC_TUNNEL_AUTH">
                        <xsi:complexType>
                            <xsi:sequence>
                                <xsi:element name="TYPE" type="xsi:unsignedShort" />
                                <xsi:element name="VALUE" type="xsi:anySimpleType" />
                            </xsi:sequence>
                            <xsi:attribute name="length" type="xsi:unsignedByte" />
                        </xsi:complexType>
                    </xsi:element>
                </xsi:choice>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<xsi:complexType name="ip_over_ip_tlv">
    <xsi:complexContent>
```

```
            <xsi:restriction base="tunnel_encap_tlv">
                <xsi:sequence>
                    <xsi:element name="TUNNEL_TYPE">
                        <xsi:complexType>
                            <xsi:simpleContent>
                                <xsi:restriction base="net:dual_repr">
                                    <xsi:enumeration value="IP_OVER_IP" />
                                    <xsi:attribute name="value" type="xsi:unsignedByte"
                                        fixed="7" use="required" />
                                </xsi:restriction>
                            </xsi:simpleContent>
                        </xsi:complexType>
                    </xsi:element>
                    <xsi:any minOccurs="0" maxOccurs="unbounded" />
                </xsi:sequence>
                <xsi:attribute name="length" type="xsi:unsignedShort"
                    use="required" />
            </xsi:restriction>
        </xsi:complexContent>
    </xsi:complexType>

    <!-- RFC 5566 adds the following four tunnel types and the IPsec Tunnel
        Authenticator sub-TLV that is used within them. -->
    <xsi:complexType name="transmit_tunnel_end_tlv">
        <xsi:complexContent>
            <xsi:restriction base="tunnel_encap_tlv">
                <xsi:sequence>
                    <xsi:element name="TUNNEL_TYPE">
                        <xsi:complexType>
                            <xsi:simpleContent>
                                <xsi:restriction base="net:dual_repr">
                                    <xsi:enumeration
value="TRANSMIT_TUNNEL_ENDPOINT" />
                                    <xsi:attribute name="value" type="xsi:unsignedByte"
                                        fixed="3" use="required" />
                                </xsi:restriction>
                            </xsi:simpleContent>
                        </xsi:complexType>
                    </xsi:element>
                    <xsi:element name="IPSEC_TUNNEL_AUTH" minOccurs="0">
                        <xsi:complexType>
                            <xsi:sequence>
                                <xsi:element name="TYPE" type="xsi:unsignedShort" />
                                <xsi:element name="VALUE" type="xsi:anySimpleType" />
                            </xsi:sequence>
                            <xsi:attribute name="length" type="xsi:unsignedByte" />
                        </xsi:complexType>
                    </xsi:element>
                </xsi:sequence>
                <xsi:attribute name="length" type="xsi:unsignedShort"
                    use="required" />
            </xsi:restriction>
        </xsi:complexContent>
    </xsi:complexType>

    <xsi:complexType name="ipsec_tunnel_tlv">
        <xsi:complexContent>
            <xsi:restriction base="tunnel_encap_tlv">
                <xsi:sequence>
                    <xsi:element name="TUNNEL_TYPE">
                        <xsi:complexType>
                            <xsi:simpleContent>
                                <xsi:restriction base="net:dual_repr">
                                    <xsi:enumeration
value="IPSEC_IN_TUNNEL_MODE" />
                                    <xsi:attribute name="value" type="xsi:unsignedByte"
                                        fixed="4" use="required" />
                                </xsi:restriction>
                            </xsi:simpleContent>
                        </xsi:complexType>
                    </xsi:element>
                    <xsi:element name="IPSEC_TUNNEL_AUTH" minOccurs="0">
                        <xsi:complexType>
                            <xsi:sequence>
                                <xsi:element name="TYPE" type="xsi:unsignedShort" />
                                <xsi:element name="VALUE" type="xsi:anySimpleType" />
                            </xsi:sequence>
                            <xsi:attribute name="length" type="xsi:unsignedByte" />
                        </xsi:complexType>
                    </xsi:element>
                </xsi:sequence>
                <xsi:attribute name="length" type="xsi:unsignedShort"
                    use="required" />
            </xsi:restriction>
        </xsi:complexContent>
    </xsi:complexType>

    <xsi:complexType name="ip_in_ip_with_ipsec_transport_tlv">
        <xsi:complexContent>
            <xsi:restriction base="tunnel_encap_tlv">
                <xsi:sequence>
                    <xsi:element name="TUNNEL_TYPE">
                        <xsi:complexType>
                            <xsi:simpleContent>
                                <xsi:restriction base="net:dual_repr">
                                    <xsi:enumeration
value="IP_IN_IP_WITH_IPSEC_TRANSPORT" />
                                    <xsi:attribute name="value" type="xsi:unsignedByte"
                                        fixed="5" use="required" />
                                </xsi:restriction>
                            </xsi:simpleContent>
                        </xsi:complexType>
                    </xsi:element>
                    <xsi:element name="IPSEC_TUNNEL_AUTH" minOccurs="0">
                        <xsi:complexType>
                            <xsi:sequence>
                                <xsi:element name="TYPE" type="xsi:unsignedShort" />
                                <xsi:element name="VALUE" type="xsi:anySimpleType" />
                            </xsi:sequence>
                            <xsi:attribute name="length" type="xsi:unsignedByte" />
                        </xsi:complexType>
                    </xsi:element>
                </xsi:sequence>
                <xsi:attribute name="length" type="xsi:unsignedShort"
                    use="required" />
            </xsi:restriction>
        </xsi:complexContent>
    </xsi:complexType>

    <xsi:complexType name="mpls_in_ip_with_ipsec_tlv">
        <xsi:complexContent>
            <xsi:restriction base="tunnel_encap_tlv">
                <xsi:sequence>
                    <xsi:element name="TUNNEL_TYPE">
                        <xsi:complexType>
                            <xsi:simpleContent>
                                <xsi:restriction base="net:dual_repr">
                                    <xsi:enumeration
value="MPLS_IN_IP_WITH_IPSEC_TRANSPORT" />
                                    <xsi:attribute name="value" type="xsi:unsignedByte"
                                        fixed="6" use="required" />
                                </xsi:restriction>
                            </xsi:simpleContent>
                        </xsi:complexType>
                    </xsi:element>
                    <xsi:element name="IPSEC_TUNNEL_AUTH" minOccurs="0">
```

```
                    <xsi:complexType>
                        <xsi:sequence>
                            <xsi:element name="TYPE" type="xsi:unsignedShort" />
                            <xsi:element name="VALUE" type="xsi:anySimpleType" />
                        </xsi:sequence>
                        <xsi:attribute name="length" type="xsi:unsignedByte" />
                    </xsi:complexType>
                </xsi:element>
            </xsi:sequence>
            <xsi:attribute name="length" type="xsi:unsignedShort"
                use="required" />
        </xsi:restriction>
    </xsi:complexContent>
</xsi:complexType>

<!-- SECTION 9: NOTIFICATION ERROR TYPES -->

<xsi:complexType name="msg_hdr_error">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error">
            <xsi:enumeration value="MESSAGE_HEADER_ERROR" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="1" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="open_msg_error">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error">
            <xsi:enumeration value="OPEN_MESSAGE_ERROR" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="2" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="update_msg_error">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error">
            <xsi:enumeration value="UPDATE_MESSAGE_ERROR" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="3" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="hold_timer_exp">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error">
            <xsi:enumeration value="HOLD_TIMER_EXPIRED" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="4" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="fsm_error">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error">
            <xsi:enumeration value="FINITE_STATE_MACHINE_ERROR" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="5" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="cease">
```

```
    <xsi:simpleContent>
        <xsi:restriction base="notification_error">
            <xsi:enumeration value="CEASE" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="6" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="unknown_error">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error">
            <xsi:enumeration value="UNKNOWN_ERROR" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<!-- SECTION 10: NOTIFICATION ERROR SUBTYPES -->
<!-- BEGIN OPEN MESSAGE ERROR SUBCODES -->
<xsi:complexType name="connect_not_sync">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="CONNECTION_NOT_SYNCHRONIZED" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="1" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bad_msg_len">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="BAD_MESSAGE_LENGTH" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="2" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bad_msg_type">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="MESSAGE_HEADER_ERROR" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="3" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<!-- BEGIN OPEN MESSAGE ERROR SUBCODES -->

<xsi:complexType name="unsupported_version">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="UNSUPPORTED_VERSION" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="1" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bad_peer_as">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="BAD_PEER_AS" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
```

```
                fixed="2" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bad_bgp_id">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="BAD_BGP_IDENTIFIER" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="3" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="unsupported_parameter">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="UNSUPPORTED_OPTIONAL_PARAMETER" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="4" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="auth_fail">
    <xsi:annotation>
        <xsi:documentation>
            The Authentication Failure error subcode is deprecated.
        </xsi:documentation>
    </xsi:annotation>
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="AUTHENTICATION_FAILURE" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="5" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bad_hold_time">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="UNACCEPTABLE_HOLD_TIME" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="6" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="unsupported_cap">
    <xsi:annotation>
        <xsi:documentation>
            The Unsupported Capability OPEN message error subcode was added
            in RFC 5492.
        </xsi:documentation>
    </xsi:annotation>
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="UNSUPPORTED_CAPABILITY" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="7" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<!-- BEGIN UPDATE MESSAGE ERROR SUBCODES -->
```

```
<xsi:complexType name="malformed_attributes">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="MALFORMED_ATTRIBUTE_LIST" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="1" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bad_well_known_attr">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="UNRECOGNIZED_WELL_KNOWN_ATTRIBUTE" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="2" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="missing_well_known_attr">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="MISSING_WELL_KNOWN_ATTRIBUTE" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="3" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="attr_flag_error">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="ATTRIBUTE_FLAGS_ERROR" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="4" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="attr_len_error">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="ATTRIBUTE_LENGTH_ERROR" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="5" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="invalid_origin">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="INVALID_ORIGIN_ATTRIBUTE" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="6" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="asn_loop">
    <xsi:annotation>
        <xsi:documentation>
            The AS Routing Loop error subcode is deprecated.
        </xsi:documentation>
    </xsi:annotation>
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
```

```
            <xsi:enumeration value="AS_ROUTING_LOOP" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="7" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bad_next_hop">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="INVALID_NEXT_HOP_ATTRIBUTE" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="8" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="opt_attr_error">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="OPTIONAL_ATTRIBUTE_ERROR" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="9" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="invalid_network_field">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="INVALID_NETWORK_FIELD" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="10" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="bad_as_path">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="MALFORMED_AS_PATH" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="11" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<!-- BEGIN FSM ERROR SUBCODES -->
<xsi:annotation>
    <xsi:documentation>
        Finite State Machine Error Subcodes are defined in RFC 6608.
    </xsi:documentation>
</xsi:annotation>

<xsi:complexType name="unspec_error">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration value="UNSPECIFIED_ERROR" />
            <xsi:attribute name="value" type="xsi:unsignedByte"
                fixed="0" use="required" />
        </xsi:restriction>
    </xsi:simpleContent>
</xsi:complexType>

<xsi:complexType name="opensent_error">
    <xsi:simpleContent>
        <xsi:restriction base="notification_error_sub">
            <xsi:enumeration
```

```
value="RECEIVE_UNEXPECTED_MESSAGE_IN_OPENSENT_STATE" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    fixed="1" use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <xsi:complexType name="openconfirm_error">
        <xsi:simpleContent>
            <xsi:restriction base="notification_error_sub">
                <xsi:enumeration
value="RECEIVE_UNEXPECTED_MESSAGE_IN_OPENCONFIRM_STATE" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    fixed="2" use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <xsi:complexType name="openestablished_error">
        <xsi:simpleContent>
            <xsi:restriction base="notification_error_sub">
                <xsi:enumeration
value="RECEIVE_UNEXPECTED_MESSAGE_IN_OPENESTABLISHED_STATE" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    fixed="3" use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <!-- BEGIN CEASE MESSAGE SUBCODES -->
    <xsi:annotation>
        <xsi:documentation>
            CEASE NOTIFICATION message subcodes were added by RFC 4486.
        </xsi:documentation>
    </xsi:annotation>

    <xsi:complexType name="max_prefixes">
        <xsi:simpleContent>
            <xsi:restriction base="notification_error_sub">
                <xsi:enumeration value="MAXIMUM_NUMBER_OF_PREFIXES_REACHED" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    fixed="1" use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <xsi:complexType name="admin_shutdown">
        <xsi:simpleContent>
            <xsi:restriction base="notification_error_sub">
                <xsi:enumeration value="ADMINISTRATIVE_SHUTDOWN" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    fixed="2" use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <xsi:complexType name="peer_deconfigured">
        <xsi:simpleContent>
            <xsi:restriction base="notification_error_sub">
                <xsi:enumeration value="PEER_DECONFIGURED" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    fixed="3" use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <xsi:complexType name="admin_reset">
        <xsi:simpleContent>
```

```
            <xsi:restriction base="notification_error_sub">
                <xsi:enumeration value="ADMINISTRATIVE_RESET" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    fixed="4" use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <xsi:complexType name="connect_reject">
        <xsi:simpleContent>
            <xsi:restriction base="notification_error_sub">
                <xsi:enumeration value="CONNECTION_REJECTED" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    fixed="5" use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <xsi:complexType name="other_config_change">
        <xsi:simpleContent>
            <xsi:restriction base="notification_error_sub">
                <xsi:enumeration value="OTHER_CONFIGURATION_CHANGE" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    fixed="6" use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <xsi:complexType name="connect_collision_res">
        <xsi:simpleContent>
            <xsi:restriction base="notification_error_sub">
                <xsi:enumeration value="CONNECTION_COLLISION_RESOLUTION" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    fixed="7" use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <xsi:complexType name="out_of_resources">
        <xsi:simpleContent>
            <xsi:restriction base="notification_error_sub">
                <xsi:enumeration value="OUT_OF_RESOURCES" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    fixed="8" use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

    <!-- TYPE TO HANDLE OTHER ERRORS -->

    <xsi:complexType name="unknown_suberror">
        <xsi:simpleContent>
            <xsi:restriction base="notification_error_sub">
                <xsi:enumeration value="UNKNOWN_ERROR" />
                <xsi:attribute name="value" type="xsi:unsignedByte"
                    use="required" />
            </xsi:restriction>
        </xsi:simpleContent>
    </xsi:complexType>

</xsi:schema>
```

C Routing Flow XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xsi:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:ietf:params:xml:ns:routing_flow"
    xmlns="urn:ietf:params:xml:ns:routing_flow"
    xmlns:net="urn:ietf:params:xml:ns:network_base"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xsi:import id="net" namespace="urn:ietf:params:xml:ns:network_base"
        schemaLocation="network_base.xsd" />

    <xsi:annotation>
        <xsi:documentation>
            This XSD provides the general description of a "routing flow"
            between two peer routers. Each peer router is defined by its
            network address/AFI, routing protocol port number, and ASN.
            The flow itself is distinguished by routing protocol, version
            number (if applicable), and flow capabilities.
            Finally, the flow definition includes the messages sent
            during the flow.
        </xsi:documentation>
    </xsi:annotation>

    <!-- The root element that contains a representation of a "routing flow".
        It uses an XSD abstract type to allow any derivation of the basic abstract
        type below to fill in the content of this element. -->
    <xsi:element name="FLOW" type="routing_flow" />

    <!-- The abstract type that provides a basic definition of a routing flow.
        It describes a source and destination peer, the protocol being used, and
        includes a wildcard that SHOULD be replaced with a sequence of protocol-specific
        messages. The definitions of such messages MUST be in separate XSD files. -->
    <xsi:complexType name="routing_flow" abstract="true">
        <xsi:sequence>
            <xsi:element name="SOURCE_PEER" type="peer_type" />
            <xsi:element name="DEST_PEER" type="peer_type" />
            <xsi:element name="PROTOCOL" type="protocol_type" />
            <xsi:any minOccurs="0" maxOccurs="unbounded" />
        </xsi:sequence>
    </xsi:complexType>

    <!-- This type describes a peer router in a routing flow. Any peer must
        have an address with an identifiable AFI, an open port to communicate routing
        messages over, and an Autonomous System that it belongs to. Additionally,
        an implementation MAY include a canonical name for the peer, for example
        to identify a single router with multiple addresses/interfaces. -->
    <xsi:complexType name="peer_type">
        <xsi:sequence>
            <xsi:element name="ADDRESS" type="net:network_addr" />
            <xsi:element name="PORT" type="net:port" />
            <xsi:element name="AS" type="net:asn" />
        </xsi:sequence>
        <xsi:attribute name="name" type="xsi:normalizedString" />
    </xsi:complexType>

    <!-- This type provides the basic description of a routing protocol that
        is in use over a routing flow. It includes the name of the protocol and
        allows inclusion of a version number. Additionally, it describes specific
        capabilities in use for the flow as both a human-readable string and its
        specification-defined machine-readable value. -->
    <xsi:complexType name="protocol_type">
        <xsi:sequence>
            <xsi:element name="NAME" type="xsi:normalizedString" />
            <xsi:element name="CAPABILITY" type="net:dual_repr"
                minOccurs="0" maxOccurs="unbounded" />
        </xsi:sequence>
        <xsi:attribute name="version" type="xsi:unsignedByte" />
    </xsi:complexType>

</xsi:schema>
```

D Example XML Instances

The following XML instances are included to illustrate valid instances of the schema in this work. They are not meant to be a comprehensive set, merely examples for reference.

D.1 BGP Network Flow

```
<?xml version="1.0" encoding="UTF-8"?>
<FLOW xmlns="urn:ietf:params:xml:ns:routing_flow"
    xmlns:net="urn:ietf:params:xml:ns:network_base"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xfb="urn:ietf:params:xml:ns:xfb"
    xsi:schemaLocation="urn:ietf:params:xml:ns:routing_flow routing_flow.xsd
                        urn:ietf:params:xml:ns:network_base network_base.xsd
                                urn:ietf:params:xml:ns:xfb xfb_v6.xsd">
    <SOURCE_PEER name="route-views4">
        <ADDRESS afi="1" xsi:type="net:ipv4_addr">128.223.51.15</ADDRESS>
        <PORT>179</PORT>
        <AS byte_len="2" xsi:type="net:asn2">6447</AS>
    </SOURCE_PEER>
    <DEST_PEER name="colostate-bgpmon">
        <ADDRESS afi="1" xsi:type="net:ipv4_addr">129.82.138.6</ADDRESS>
        <PORT>7777</PORT>
        <AS byte_len="2" xsi:type="net:asn2">12145</AS>
    </DEST_PEER>
    <PROTOCOL version="4">
        <NAME>BGP</NAME>
        <CAPABILITY value="65">ASN4</CAPABILITY>
        <CAPABILITY value="2">ROUTE_REFRESH</CAPABILITY>
        <CAPABILITY value="1">MP_IPV6</CAPABILITY>
    </PROTOCOL>
    <BGP_MESSAGE xmlns="urn:ietf:params:xml:ns:xfb" xsi:type="octet_bgp"
length="68">
<OCTETS>FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF004402000000254001010
040021002076E81CF8B415F04D70513150F8AEF400304BB79C121C008046E
81FFDC1857FA69185BBF2F</OCTETS>
    </BGP_MESSAGE>
    <BGP_MESSAGE xsi:type="ascii_update_msg"
xmlns="urn:ietf:params:xml:ns:xfb" length="68" withdrawn_len="0"
path_attr_len="37">
        <MARKER>FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF</MARKER>
        <TYPE value="2">UPDATE</TYPE>
        <PATH_ATTRIBUTE length="1" xsi:type="origin_path_attr">
            <TRANSITIVE />
            <TYPE value="1">ORIGIN</TYPE>
            <ORIGIN value="0" xsi:type="bgp_igp_origin">IGP</ORIGIN>
        </PATH_ATTRIBUTE>
        <PATH_ATTRIBUTE length="16" xsi:type="as_path_attr">
            <TRANSITIVE />
            <TYPE value="2">AS_PATH</TYPE>
            <AS_SEQUENCE length="7">
                <AS byte_len="2" xsi:type="net:asn2">28289</AS>
                <AS byte_len="2" xsi:type="net:asn2">53131</AS>
                <AS byte_len="2" xsi:type="net:asn2">16735</AS>
                <AS byte_len="2" xsi:type="net:asn2">1239</AS>
                <AS byte_len="2" xsi:type="net:asn2">1299</AS>
                <AS byte_len="2" xsi:type="net:asn2">5391</AS>
                <AS byte_len="2" xsi:type="net:asn2">35567</AS>
            </AS_SEQUENCE>
        </PATH_ATTRIBUTE>
        <PATH_ATTRIBUTE length="4" xsi:type="next_hop_attr">
            <TRANSITIVE />
            <TYPE value="3">NEXT_HOP</TYPE>
            <NEXT_HOP afi="1">187.121.193.33</NEXT_HOP>
        </PATH_ATTRIBUTE>
        <PATH_ATTRIBUTE length="4" xsi:type="communities_attr">
            <OPTIONAL />
            <TRANSITIVE />
            <TYPE value="8">COMMUNITIES</TYPE>
            <COMMUNITY>
                <AS byte_len="2">28289</AS>
                <VALUE>65500</VALUE>
            </COMMUNITY>
        </PATH_ATTRIBUTE>
        <NLRI afi="1" safi="1">87.250.105.0/24</NLRI>
        <NLRI afi="1" safi="1">91.191.47.0/24</NLRI>
    </BGP_MESSAGE>
</FLOW>
```

D.2 BGP OPEN Message

```
<?xml version="1.0" encoding="UTF-8"?>

<BGP_MESSAGE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:ietf:params:xml:ns:xfb"
    xmlns:net="urn:ietf:params:xml:ns:network_base"
xsi:schemaLocation="urn:ietf:params:xml:ns:xfb ../xfb_v6.xsd"
    xsi:type="ascii_open_msg" length="77" opt_parm_len="25">
    <MARKER>FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF</MARKER>
    <TYPE value="1">OPEN</TYPE>
    <VERSION>4</VERSION>
    <SRC_ASN byte_len="2">12145</SRC_ASN>
    <HOLD_TIME>30</HOLD_TIME>
    <BGP_IDENTIFIER>0.0.0.0</BGP_IDENTIFIER>
    <PARAMETER xsi:type="authentication_parameter" length="7">
        <TYPE>1</TYPE>
        <AUTHENTICATION>
            <CODE>F1</CODE>
            <DATA>FFFFFFFFFF</DATA>
        </AUTHENTICATION>
    </PARAMETER>
    <PARAMETER xsi:type="capabilities_parameter" length="16">
        <TYPE>2</TYPE>
        <CAPABILITY xsi:type="asn4_capability" length="4">
            <CODE value="65">ASN4</CODE>
            <VALUE xsi:type="net:asn4" byte_len="4">12145</VALUE>
        </CAPABILITY>
        <CAPABILITY xsi:type="mp_capability" length="2">
            <CODE value="1">MULTIPROTOCOL</CODE>
            <VALUE>
                <AFI>2</AFI>
                <SAFI>1</SAFI>
            </VALUE>
        </CAPABILITY>
        <CAPABILITY xsi:type="mp_capability" length="2">
            <CODE value="1">MULTIPROTOCOL</CODE>
            <VALUE>
                <AFI>2</AFI>
                <SAFI>2</SAFI>
            </VALUE>
        </CAPABILITY>
        <CAPABILITY xsi:type="route_refresh_capability" length="0">
            <CODE value="2">ROUTE_REFRESH</CODE>
        </CAPABILITY>
        <CAPABILITY xsi:type="unknown_capability" length="4">
            <CODE value="17">UNKNOWN</CODE>
            <VALUE>FFFFFFFF</VALUE>
        </CAPABILITY>
    </PARAMETER>
    <PARAMETER xsi:type="unknown_parameter" length="6">
        <TYPE>4</TYPE>
        <UNKNOWN>FF09A1</UNKNOWN>
    </PARAMETER>
</BGP_MESSAGE>
```

D.3 BGP UPDATE Message

```
<?xml version="1.0" encoding="UTF-8"?>

<BGP_MESSAGE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:ietf:params:xml:ns:xfb"
    xmlns:net="urn:ietf:params:xml:ns:network_base"
xsi:schemaLocation="urn:ietf:params:xml:ns:xfb ../xfb_v6.xsd"
    xsi:type="ascii_update_msg" length="9999" withdrawn_len="4" path_attr_len="32">
    <MARKER>FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF</MARKER>
    <TYPE value="2">UPDATE</TYPE>
    <WITHDRAWN afi="1">1.2.3/24</WITHDRAWN>
    <WITHDRAWN afi="1">10/8</WITHDRAWN>
    <PATH_ATTRIBUTE length="1" xsi:type="origin_path_attr">
        <TRANSITIVE />
        <TYPE value="1">ORIGIN</TYPE>
        <ORIGIN value="0" xsi:type="bgp_igp_origin">IGP</ORIGIN>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE length="16" xsi:type="as_path_attr">
        <TRANSITIVE />
        <TYPE value="2">AS_PATH</TYPE>
        <AS_SEQUENCE length="7">
            <AS byte_len="2" xsi:type="net:asn2">28289</AS>
            <AS byte_len="2" xsi:type="net:asn2">53131</AS>
            <AS byte_len="2" xsi:type="net:asn2">16735</AS>
            <AS byte_len="2" xsi:type="net:asn2">1239</AS>
            <AS byte_len="2" xsi:type="net:asn2">1299</AS>
            <AS byte_len="2" xsi:type="net:asn2">5391</AS>
            <AS byte_len="2" xsi:type="net:asn2">35567</AS>
        </AS_SEQUENCE>
        <AS_SET length="3">
            <AS byte_len="2" xsi:type="net:asn2">123</AS>
            <AS byte_len="2" xsi:type="net:asn2">456</AS>
            <AS byte_len="2" xsi:type="net:asn2">789</AS>
        </AS_SET>
        <AS_CONFED_SEQUENCE length="3">
            <AS byte_len="2" xsi:type="net:asn2">34</AS>
            <AS byte_len="2" xsi:type="net:asn2">56</AS>
            <AS byte_len="2" xsi:type="net:asn2">78</AS>
        </AS_CONFED_SEQUENCE>
        <AS_CONFED_SET length="1">
            <AS byte_len="2" xsi:type="net:asn2">90</AS>
        </AS_CONFED_SET>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE length="4" xsi:type="next_hop_attr">
        <TRANSITIVE />
        <TYPE value="3">NEXT_HOP</TYPE>
        <NEXT_HOP afi="1">187.121.193.33</NEXT_HOP>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="med_attr" length="4">
        <OPTIONAL/>
        <TYPE value="4">MULTI_EXIT_DISC</TYPE>
        <MED>7</MED>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="local_pref_attr" length="4">
        <TRANSITIVE/>
        <TYPE value="5">LOCAL_PREF</TYPE>
        <LOCAL_PREF>1</LOCAL_PREF>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="atomic_aggr_attr" length="0">
        <TRANSITIVE/>
        <TYPE value="6">ATOMIC_AGGREGATE</TYPE>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="aggregator_attr" length="6">
        <OPTIONAL/>
        <TRANSITIVE/>
        <TYPE value="7">AGGREGATOR</TYPE>
        <AGGREGATOR>
            <AS byte_len="2">12145</AS>
            <BGP_IDENTIFIER>72639100</BGP_IDENTIFIER>
        </AGGREGATOR>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE length="9999" xsi:type="communities_attr">
        <OPTIONAL />
        <TRANSITIVE />
        <TYPE value="8">COMMUNITIES</TYPE>
        <COMMUNITY>
            <AS byte_len="2">28289</AS>
            <VALUE>65500</VALUE>
        </COMMUNITY>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="originator_id_attr" length="4">
        <OPTIONAL/>
        <TYPE value="9">ORIGINATOR_ID</TYPE>
        <ORIGINATOR_ID>129.82.138.24</ORIGINATOR_ID>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="cluster_list_attr" length="99">
        <OPTIONAL/>
        <TYPE value="10">CLUSTER_LIST</TYPE>
        <CLUSTER_ID>7</CLUSTER_ID>
        <CLUSTER_ID>16</CLUSTER_ID>
        <CLUSTER_ID>24</CLUSTER_ID>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="advertiser_attr" length="4">
        <OPTIONAL/>
        <TYPE value="12">ADVERTISER</TYPE>
        <ADVERTISER xsi:type="net:ipv4_addr" afi="1">123.213.13.32</ADVERTISER>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="mp_reach_attr" length="9999">
        <OPTIONAL/>
        <TYPE value="14">MP_REACH_NLRI</TYPE>
        <MP_REACH_NLRI>
            <AFI>2</AFI>
            <SAFI>1</SAFI>
            <NEXT_HOP_LEN>16</NEXT_HOP_LEN>
            <NEXT_HOP xsi:type="net:ipv6_addr" afi="2">fe80::1</NEXT_HOP>
            <MP_NLRI xsi:type="net:ipv6_prefix" afi="2">
fd68:e916:5287:9f27::/64</MP_NLRI>
        </MP_REACH_NLRI>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="mp_unreach_attr" length="9999">
        <OPTIONAL/>
        <TYPE value="15">MP_UNREACH_NLRI</TYPE>
        <MP_UNREACH_NLRI>
            <AFI>2</AFI>
            <SAFI>1</SAFI>
            <WITHDRAWN xsi:type="net:ipv6_prefix" afi="2">ff08::/16</WITHDRAWN>
            <WITHDRAWN xsi:type="net:ipv6_prefix" afi="2">fd68:e916::/32</WITHDRAWN>
        </MP_UNREACH_NLRI>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="ext_community_attr" length="9999">
        <OPTIONAL/>
        <TRANSITIVE/>
        <EXTENDED/>
        <TYPE value="16">EXTENDED_COMMUNITIES</TYPE>
        <COMMUNITY xsi:type="asn2_specific_ext_com">
            <FCFS/>
            <TRANSITIVE/>
            <TYPE value="0">2-OCTET_AS-SPECIFIC</TYPE>
            <VALUE>
                <GLOBAL_ADMINISTRATOR byte_len="2">12145</GLOBAL_ADMINISTRATOR>
                <LOCAL_VAL>145</LOCAL_VAL>
            </VALUE>
        </COMMUNITY>
        <COMMUNITY xsi:type="color_ext_com">
            <FCFS/>
            <TRANSITIVE/>
            <TYPE value="030B">COLOR</TYPE>
            <COLOR>1776</COLOR>
        </COMMUNITY>
        <COMMUNITY xsi:type="encapsulation_ext_com">
            <FCFS/>
            <TRANSITIVE/>
            <TYPE value="030C">ENCAPSULATION</TYPE>
            <TUNNEL_TYPE>2</TUNNEL_TYPE>
        </COMMUNITY>
        <COMMUNITY xsi:type="ipv4_specific_ext_com">
            <FCFS/>
            <TRANSITIVE/>
            <TYPE value="1">IPV4-SPECIFIC</TYPE>
            <VALUE>
                <GLOBAL_ADMINISTRATOR afi="1">1.2.3.4</GLOBAL_ADMINISTRATOR>
                <LOCAL_VALUE>blah</LOCAL_VALUE>
            </VALUE>
        </COMMUNITY>
        <COMMUNITY xsi:type="opaque_ext_com">
            <STANDARDS/>
            <TRANSITIVE/>
            <TYPE value="3">OPAQUE</TYPE>
            <VALUE>AA00BBCCDDFF</VALUE>
        </COMMUNITY>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="as4_path_attr" length="9999">
        <OPTIONAL/>
        <TRANSITIVE/>
        <TYPE value="17">AS4_PATH</TYPE>
        <AS_SET length="6">
            <AS byte_len="4">12145</AS>
            <AS byte_len="4">75621926</AS>
            <AS byte_len="4">1455562</AS>
            <AS byte_len="4">1284</AS>
            <AS byte_len="4">14528</AS>
            <AS byte_len="4">481287</AS>
        </AS_SET>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="as4_aggregator_attr" length="8">
        <OPTIONAL/>
        <TRANSITIVE/>
        <TYPE value="18">AS4_AGGREGATOR</TYPE>
        <AS4_AGGREGATOR>
            <AS byte_len="4">13258720</AS>
            <BGP_IDENTIFIER>571039246</BGP_IDENTIFIER>
        </AS4_AGGREGATOR>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="tunnel_encap_attr" length="9999">
        <OPTIONAL/>
        <TRANSITIVE/>
        <EXTENDED/>
        <TYPE value="23">TUNNEL_ENCAPSULATION</TYPE>
        <TUNNEL xsi:type="gre_tlv" length="99">
            <TUNNEL_TYPE value="2">GRE</TUNNEL_TYPE>
            <GRE_KEY>7</GRE_KEY>
            <LOAD_BALANCING>FFFF</LOAD_BALANCING>
        </TUNNEL>
        <TUNNEL xsi:type="ip_in_ip_with_ipsec_transport_tlv" length="9999">
            <TUNNEL_TYPE value="5">IP_IN_IP_WITH_IPSEC_TRANSPORT</TUNNEL_TYPE>
            <IPSEC_TUNNEL_AUTH>
                <TYPE>13</TYPE>
                <VALUE>17</VALUE>
            </IPSEC_TUNNEL_AUTH>
        </TUNNEL>
        <TUNNEL xsi:type="ip_over_ip_tlv" length="9999">
            <TUNNEL_TYPE value="7">IP_OVER_IP</TUNNEL_TYPE>
        </TUNNEL>
        <TUNNEL xsi:type="ipsec_tunnel_tlv" length="9999">
            <TUNNEL_TYPE value="4">IPSEC_IN_TUNNEL_MODE</TUNNEL_TYPE>
            <IPSEC_TUNNEL_AUTH>
                <TYPE>13</TYPE>
                <VALUE>17</VALUE>
            </IPSEC_TUNNEL_AUTH>
        </TUNNEL>
        <TUNNEL xsi:type="l2tpv3_ip_tlv" length="9999">
            <TUNNEL_TYPE value="1">L2TPV3_OVER_IP</TUNNEL_TYPE>
            <COLOR>
                <FCFS/>
                <TRANSITIVE/>
                <TYPE value="030B">COLOR</TYPE>
                <COLOR>17</COLOR>
            </COLOR>
            <ENCAPSULATION value="1">
                <SESSION_ID>7</SESSION_ID>
                <COOKIE>FFFFFFFFFFFFFFFF</COOKIE>
            </ENCAPSULATION>
            <LOAD_BALANCING>FFFF</LOAD_BALANCING>
            <PROTOCOL>AABB</PROTOCOL>
            <IPSEC_TUNNEL_AUTH>
                <TYPE>13</TYPE>
                <VALUE>17</VALUE>
            </IPSEC_TUNNEL_AUTH>
        </TUNNEL>
        <TUNNEL xsi:type="mpls_in_ip_with_ipsec_tlv" length="9999">
            <TUNNEL_TYPE value="6">MPLS_IN_IP_WITH_IPSEC_TRANSPORT</TUNNEL_TYPE>
            <IPSEC_TUNNEL_AUTH>
                <TYPE>13</TYPE>
                <VALUE>17</VALUE>
            </IPSEC_TUNNEL_AUTH>
        </TUNNEL>
        <TUNNEL xsi:type="transmit_tunnel_end_tlv" length="9999">
            <TUNNEL_TYPE value="3">TRANSMIT_TUNNEL_ENDPOINT</TUNNEL_TYPE>
            <IPSEC_TUNNEL_AUTH length="99">
                <TYPE>14</TYPE>
                <VALUE>198</VALUE>
            </IPSEC_TUNNEL_AUTH>
        </TUNNEL>
    </PATH_ATTRIBUTE>
    <PATH_ATTRIBUTE xsi:type="ipv6_ext_community_attr" length="9999">
        <OPTIONAL/>
        <TRANSITIVE/>
        <EXTENDED/>
        <TYPE value="16">IPV6_SPECIFIC_EXTENDED_COMMUNITIES</TYPE>
        <COMMUNITY xsi:type="ipv6_route_origin_ext_com">
            <TRANSITIVE/>
            <TYPE value="0003">IPV6-SPECIFIC_ROUTE_ORIGIN</TYPE>
            <GLOBAL_ADMINISTRATOR afi="2">fe80::1</GLOBAL_ADMINISTRATOR>
            <LOCAL_ADMINISTRATOR>FFFF</LOCAL_ADMINISTRATOR>
        </COMMUNITY>
        <COMMUNITY xsi:type="ipv6_route_target_ext_com">
            <TRANSITIVE/>
            <TYPE value="0002">IPV6-SPECIFIC_ROUTE_TARGET</TYPE>
            <GLOBAL_ADMINISTRATOR afi="2">fd68:e916::c0a8:100</GLOBAL_ADMINISTRATOR>
            <LOCAL_ADMINISTRATOR>AA00</LOCAL_ADMINISTRATOR>
        </COMMUNITY>
    </PATH_ATTRIBUTE>
```

```
    <PATH_ATTRIBUTE length="9999" xsi:type="attr_set_attr">
        <OPTIONAL/>
        <TRANSITIVE/>
        <EXTENDED/>
        <TYPE value="128">ATTR_SET</TYPE>
        <ATTR_SET>
            <ORIGIN_AS byte_len="4">12145</ORIGIN_AS>
            <PATH_ATTRIBUTE length="1" xsi:type="origin_path_attr">
                <TRANSITIVE />
                <TYPE value="1">ORIGIN</TYPE>
                <ORIGIN value="0" xsi:type="bgp_igp_origin">IGP</ORIGIN>
            </PATH_ATTRIBUTE>
            <PATH_ATTRIBUTE length="16" xsi:type="as_path_attr">
                <TRANSITIVE />
                <TYPE value="2">AS_PATH</TYPE>
                <AS_SEQUENCE length="7">
                    <AS byte_len="2" xsi:type="net:asn2">28289</AS>
                    <AS byte_len="2" xsi:type="net:asn2">53131</AS>
                    <AS byte_len="2" xsi:type="net:asn2">16735</AS>
                    <AS byte_len="2" xsi:type="net:asn2">1239</AS>
                    <AS byte_len="2" xsi:type="net:asn2">1299</AS>
                    <AS byte_len="2" xsi:type="net:asn2">5391</AS>
                    <AS byte_len="2" xsi:type="net:asn2">35567</AS>
                </AS_SEQUENCE>
            </PATH_ATTRIBUTE>
            <PATH_ATTRIBUTE length="4" xsi:type="next_hop_attr">
                <TRANSITIVE />
                <TYPE value="3">NEXT_HOP</TYPE>
                <NEXT_HOP afi="1">187.121.193.33</NEXT_HOP>
            </PATH_ATTRIBUTE>
            <PATH_ATTRIBUTE xsi:type="med_attr" length="4">
                <OPTIONAL/>
                <TYPE value="4">MULTI_EXIT_DISC</TYPE>
                <MED>7</MED>
            </PATH_ATTRIBUTE>
            <PATH_ATTRIBUTE xsi:type="local_pref_attr" length="4">
                <TRANSITIVE/>
                <TYPE value="5">LOCAL_PREF</TYPE>
                <LOCAL_PREF>1</LOCAL_PREF>
            </PATH_ATTRIBUTE>
            <PATH_ATTRIBUTE xsi:type="atomic_aggr_attr" length="0">
                <TRANSITIVE/>
                <TYPE value="6">ATOMIC_AGGREGATE</TYPE>
            </PATH_ATTRIBUTE>
            <PATH_ATTRIBUTE xsi:type="aggregator_attr" length="6">
                <OPTIONAL/>
                <TRANSITIVE/>
                <TYPE value="7">AGGREGATOR</TYPE>
                <AGGREGATOR>
                    <AS byte_len="2">12145</AS>
                    <BGP_IDENTIFIER>72639100</BGP_IDENTIFIER>
                </AGGREGATOR>
            </PATH_ATTRIBUTE>
            <PATH_ATTRIBUTE length="4" xsi:type="communities_attr">
                <OPTIONAL />
                <TRANSITIVE />
                <TYPE value="8">COMMUNITIES</TYPE>
                <COMMUNITY>
                    <AS byte_len="2">28289</AS>
                    <VALUE>65500</VALUE>
                </COMMUNITY>
            </PATH_ATTRIBUTE>
        </ATTR_SET>
    </PATH_ATTRIBUTE>
    <NLRI afi="1" safi="1">87.250.105.0/24</NLRI>
    <NLRI afi="1" safi="1">91.191.47.0/24</NLRI>
</BGP_MESSAGE>
```

D.4 BGP NOTIFICATION Message

```
<?xml version="1.0" encoding="UTF-8"?>

<BGP_MESSAGE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:ietf:params:xml:ns:xfb"
    xmlns:net="urn:ietf:params:xml:ns:network_base"
xsi:schemaLocation="urn:ietf:params:xml:ns:xfb ../xfb_v6.xsd"
    xsi:type="open_msg_error_message" length="21">
    <MARKER>FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF</MARKER>
    <TYPE value="3">NOTIFICATION</TYPE>
    <ERROR_CODE value="2">OPEN_MESSAGE_ERROR</ERROR_CODE>
    <ERROR_SUBCODE xsi:type="bad_peer_as">
        <CODE value="2">BAD_PEER_AS</CODE>
    </ERROR_SUBCODE>
</BGP_MESSAGE>

<?xml version="1.0" encoding="UTF-8"?>

<BGP_MESSAGE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:ietf:params:xml:ns:xfb"
    xmlns:net="urn:ietf:params:xml:ns:network_base"
xsi:schemaLocation="urn:ietf:params:xml:ns:xfb ../xfb_v6.xsd"
    xsi:type="update_msg_error_message" length="21">
    <MARKER>FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF</MARKER>
    <TYPE value="3">NOTIFICATION</TYPE>
    <ERROR_CODE value="3">UPDATE_MESSAGE_ERROR</ERROR_CODE>
    <ERROR_SUBCODE xsi:type="unrecognized_well_known_attr">
        <CODE value="2">UNRECOGNIZED_WELL_KNOWN_ATTRIBUTE</CODE>
        <DATA xsi:type="med_attr" length="4">
            <OPTIONAL/>
            <TYPE value="4">MULTI_EXIT_DISC</TYPE>
            <MED>7</MED>
        </DATA>
    </ERROR_SUBCODE>
</BGP_MESSAGE>

<?xml version="1.0" encoding="UTF-8"?>

<BGP_MESSAGE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:ietf:params:xml:ns:xfb"
    xmlns:net="urn:ietf:params:xml:ns:network_base"
xsi:schemaLocation="urn:ietf:params:xml:ns:xfb ../xfb_v6.xsd"
    xsi:type="fsm_error_message" length="21">
    <MARKER>FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF</MARKER>
    <TYPE value="3">NOTIFICATION</TYPE>
    <ERROR_CODE value="5">FINITE_STATE_MACHINE_ERROR</ERROR_CODE>
    <ERROR_SUBCODE xsi:type="opensent_error">
        <CODE value="1">RECEIVE_UNEXPECTED_MESSAGE_IN_OPENSENT_STATE</CODE>
        <DATA>2</DATA>
    </ERROR_SUBCODE>
</BGP_MESSAGE>
```

D.5 BGP KEEPALIVE Message

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Valid KEEPALIVE message -->
<BGP_MESSAGE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:ietf:params:xml:ns:xfb"
    xmlns:net="urn:ietf:params:xml:ns:network_base"
xsi:schemaLocation="urn:ietf:params:xml:ns:xfb ../xfb_v6.xsd"
    xsi:type="ascii_keepalive_msg" length="19">
   <MARKER>FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF</MARKER>
   <TYPE value="4">KEEPALIVE</TYPE>
</BGP_MESSAGE>
```

D.6 BGP ROUTE-REFRESH Message

```
<?xml version="1.0" encoding="UTF-8"?>

<BGP_MESSAGE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:ietf:params:xml:ns:xfb"
    xmlns:net="urn:ietf:params:xml:ns:network_base"
xsi:schemaLocation="urn:ietf:params:xml:ns:xfb ../xfb_v6.xsd"
    xsi:type="ascii_rr_msg" length="21">
    <MARKER>FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF</MARKER>
    <TYPE value="5">ROUTE_REFRESH</TYPE>
    <AFI>1</AFI>
    <SAFI>1</SAFI>
</BGP_MESSAGE>
```