

XML schema for router configuration data: An annotated DTD

Ladislav Lhotka

April 8, 2003

1 Introduction

Netopeer is a software system for configuring IPv6/IPv4 routers in a device and vendor independent way. Its internal data format is based on the Extensible Markup Language (XML) [xml].

This technical report describes a preliminary version of the XML schema, i.e., the syntax of the internal XML configuration file. It is preliminary in the sense that it does not cover the full scope of parameters and protocols necessary for configuring a typical Internet router. Our goal in this phase is to use the schema (and XML files based on it) and develop all the important parts of the *Netopeer* system – the core repository, command-line and web front-ends and back-ends for PC-based routers, Cisco IOS and JUNOS. We expect that while working on these components we will be able to evaluate our initial assumptions and decisions concerning the structure of the schema and the processing tools. For example, it is not clear yet whether the XSL Transformation language [xslt] is powerful enough for translating the internal XML to the individual target router configuration languages.

Specifically, the schema covers the configuration of basic system parameters like hostname and DNS servers, network interfaces (physical devices, virtual LANs and tunnels), static routing, RIP and RIPng routing protocols and packet/route filtering. We believe this selection of features is sufficiently non-trivial and will thus be a real challenge for the developers of the *Netopeer* software.

In order to describe the XML schema, we will use the Document Type Declaration (DTD) language that is part of the basic XML specification [xml]. It can be argued that other schema languages – for example, the W3C XML Schema [xs-s], [xs-d] or RELAX NG [relax] would better serve our purposes. In fact, it is likely that for a “production” version of the schema we will use one of the more sophisticated schema languages. For the time being, though, DTD seems to be sufficient. It is simple enough so that the developers will not spend much time reading the specification and another advantage is the broad support

for DTD in XML processing tools (including Emacs!), which is not true yet for the other schema languages.

We will use the notation of XPath [xpath] for identifying XML elements and attributes in the text. For example, a subelement of an element will be written as `element/subelement` and its attribute as `element/subelement/@attribute`. In most cases we will be discussing contents of a particular element and use only relative paths (without the leading slash) in the context of that element.

2 Design principles

The design of our XML schema tries to follow general guidelines for a good XML style. In some cases, we reflect specific needs of our application. The design also had to take into account the limitations of the DTD language – these may be removed later with the use of other schema languages.

We assume XML data will in most cases be generated and processed by a program rather than a human. That's why the data structure defined by the DTD is quite rigid in terms of relative ordering of elements – sibling elements must mostly appear in a prescribed order.

2.1 Elements versus attributes

In many cases, the same information can be represented either as the contents of an element or as the value of an attribute. There are two notable cases where elements are the only option:

- if the information is structured, i.e., where subelements are necessary
- if the contents are general (multiline) text, since the values of attributes are subject to the space normalization process [xml].

In all other cases we generally prefer attributes, because they are more concise and also their namespace is restricted to a single element so that name clashes are less likely.

With many attributes we also declare their default values. This information can be utilized, for example, by *Netopeer* front-ends.

2.2 Element names

In an XML schema defined by a DTD, all the elements share a single flat namespace. In other words, we have to guarantee uniqueness of element names

throughout the DTD. On the contrary, the W3C XML Schema allows for a structured namespace where elements defined inside other elements are local and their definition overrides the non-local one, if there is any. While this may be seen as a serious disadvantage of the DTD language, we think that in most cases the overloading of names leads to confusion and should be avoided anyway. Consequently, we used same element names in different contents only if the semantic interpretation of the element is identical, as is the case, for example, of the `description` element.

2.3 Links

The DTD uses in many places simple links as defined by `[xml]`. So far we don't see any need for using the more sophisticated link types defined by `[xlink]`, even though in some cases their use could be justified. This is especially because we don't expect XML files to be ever created by hand and in a program we can easily emulate most of the required functionality just by simple links. For example, the two-way links can be represented as two simple links connecting two resources in both directions. It is then the program's responsibility to guarantee the mutual validity of the links and update both links in sync.

3 Common attributes and elements

Some elements are designed to serve as target for links from other parts of the document. Such elements have two attributes, `@name` and `@id`, declared via the following entity:

```
<!ENTITY % tgt.att
    'name CDATA #REQUIRED
    id ID #REQUIRED'>
```

The value of the `id` attribute has to be unique throughout the configuration and should in most cases be generated by the system and invisible to the user. On the other hand, the `name` attribute serves as a visible reference to the target element and may be used in the user interface, for example as the contents of HTML (clickable) links. The value of the `name` attribute will typically be assigned by the user, although in some cases it may also be provided automatically by the system, if the corresponding object already has a natural name. As opposed to the `id` attribute, the `name` value need not be unique throughout the entire configuration but only within a reasonable scope. For example, the names of network interfaces should be unique, but an interface name could possibly be reused in other contexts, say as a name for a packet filter chain.

The remaining entities are essentially abbreviations for attributes that are common to some elements. Their meaning will be explained later.

```
<!ENTITY % af.att
  'af (ipv4|ipv6) #IMPLIED'>

<!ENTITY % match.att
  'negate (yes|no) "no"'>

<!ENTITY % action.att
  'count (yes|no) "no"'>

<!ENTITY % prefix.att
  'address CDATA #REQUIRED
  length CDATA #IMPLIED'>

<!ENTITY % address.att
  'device IDREF #REQUIRED
  role (primary|secondary) "primary"'>
```

Unlike the attributes, only one element is really common, being reused in many different places of the DTD:

```
<!ELEMENT description (#PCDATA)>
```

This element contains free text describing the parent element in the particular context.

4 The root element

The root element of a complete configuration is `configuration`, defined in the DTD as follows:

```
<!ELEMENT configuration (system, interfaces, prefix-lists?,
                        packet-filters?, routing?)>
<!ATTLIST configuration
  version (0.1) #REQUIRED>
```

The `@version` attribute specifies the version of the DTD. Every complete configuration must carry this information so that a validating parser can immediately recognize a mismatch in version numbers.

In some cases it might be useful to work with a subset of the configuration tree. XML data would then have a root element other than `configuration`. However, such a partial representation must be considered transient and should be handled with care and under well-defined conditions. In particular, if partial data are exchanged between software components, the schema version of the data must either be known to all components or distributed to them by other means.

5 System configuration

In this version the configuration schema covers only a limited subset of global system parameters:

```
<!ELEMENT system (description?, banner?, dns?)>
<!ATTLIST system
    hostname CDATA #REQUIRED
    multicast-routing (on|off) "off">

<!ELEMENT banner (#PCDATA)>

<!ELEMENT dns (dns-server*, search?)>

<!ELEMENT dns-server EMPTY>
<!ATTLIST dns-server
    %af.att;
    address CDATA #REQUIRED>

<!ELEMENT search (domain-suffix+)>

<!ELEMENT domain-suffix EMPTY>
<!ATTLIST domain-suffix
    suffix CDATA #REQUIRED>
```

The `system/@hostname` attribute defines the host name of the router, which usually coincides with one of the domain names that the router has in the DNS.

The `system/@multicast-routing` attribute specifies whether multicast routing is to be activated.

The banner element contains (multiline) text that is displayed on the terminal before the login prompt.

The DNS resolver is configured inside the `dns` element using the following subelements:

- `dns-server` may occur an arbitrary number of times. Beside the IP address of a DNS server (`@address` attribute), an address family ("`ipv4`" or "`ipv6`") can be specified in the `@af` attribute. By default the address family is deduced from the address itself. If several DNS servers are given, they are queried in the order as they appear in the configuration.
- `search` – this element contains an ordered list of domain name suffixes that will be appended in turn to relative domain names (as opposed to absolute or fully qualified domain names that end with a dot) until the name is resolved. Each of the suffixes is represented as the value of a `domain-suffix/@suffix` attribute.

6 Network interfaces

A typical router has a number of network interfaces, often based on disparate technologies and/or media. A considerable part of a router configuration is related to interfaces, in one way or another. Some information (hardware device configuration etc.) is applied solely to an interface and is thus included in a configuration block describing that interface. In other cases like routing protocols and packet filters, specific parameters can be included either with the interface configuration or in other functional subsystems, perhaps with a reference to the interface in question. Router configuration interfaces differ widely in the distribution of information between interfaces and the other parts. *Netopeer* tries to bundle as few parameters as possible with the interface configuration and uses extensively XML links pointing to the interfaces from other subsystems.

6.1 Device setup

It is tempting to structure the configuration of network interfaces after the ISO OSI model into different layers (physical, link, network etc.). However, many real-life situations do not fit into the OSI hierarchy. New headers are often inserted at different places, thus creating virtual layers and interfaces. This is the case of IEEE 802.1q VLANs, MPLS and various tunneling techniques. Some approaches even turn the OSI hierarchy upside down and transport L2 frames in L3 packets.

Faced with the given variety of header combinations we don't think it is useful to force any OSI-based hierarchy on interface configuration data. Consequently,

in our XML schema we use several independent blocks handling different types of network interfaces and use XML links for defining the actual chains of packet headers:

1. *network-devices* contains physical interfaces present in the router. Some parameters of their configuration may not be applicable to virtual devices like VLAN interfaces, e.g., transmission speed or duplex mode.
2. *virtual-lans* are essentially virtual L2 interfaces created on top of a physical network interface. They are identified by numeric tags carried in VLAN headers that are inserted between L2 (Ethernet) and L3 headers. Apart from the IEEE standard 802.1q, Cisco routers and switches still support their proprietary VLAN encapsulation named ISL. Each VLAN interface element must refer, through a link, to the base network device.
3. *tunnels* are virtual L3 interfaces created on top of another L3 interface, the most common species being IPv6 in IPv4 [RFC2893] or GRE (*Generic Route Encapsulation*) [RFC2784]. As opposed to VLANs, tunnels are often not bound to a single network device – the local end can be identified by any IP address configured on the local host, e.g., an address of a loopback interface.

The corresponding part of the DTD looks as follows:

```
<!ELEMENT interfaces (network-devices, virtual-lans?,
                      tunnels?, l3-addresses)>

<!ELEMENT network-devices (device+)>

<!ELEMENT device (description?)>
<!ATTLIST device
  disable (yes|no) "no"
  arp (on|off) "on"
  duplex (full|half|auto) "auto"
  speed CDATA #IMPLIED
  multicast (on|off) "off"
  txqlen CDATA #IMPLIED
  mtu CDATA #IMPLIED
  macaddr CDATA #IMPLIED
  encapsulation (ppp|hdlc|none) "none"
  filter-in IDREF #IMPLIED
  filter-out IDREF #IMPLIED
  %tgt.att;>
```

```

<!ELEMENT virtual-lans (vlan-interface+)>

<!ELEMENT vlan-interface (description?)>
<!ATTLIST vlan-interface
    device IDREF #REQUIRED
    encapsulation (802.1q|isl) "802.1q"
    tag CDATA #REQUIRED
    multicast (on|off) "off"
    mtu CDATA #IMPLIED
    filter-in IDREF #IMPLIED
    filter-out IDREF #IMPLIED
    %tgt.att;>

<!ELEMENT tunnels (tunnel+)>

<!ELEMENT tunnel (description?,
    (tunnel-source-address|tunnel-source-interface),
    tunnel-destination-address)>
<!ATTLIST tunnel
    mode (ipip|v6inv4|gre) "v6inv4"
    ttl CDATA #IMPLIED
    device IDREF #IMPLIED
    multicast (on|off) "off"
    mtu CDATA #IMPLIED
    macaddr CDATA #IMPLIED
    pmtudisc (on|off) "on"
    key CDATA #IMPLIED
    checksum (on|off) "off"
    sequence (on|off) "off"
    filter-in IDREF #IMPLIED
    filter-out IDREF #IMPLIED
    %tgt.att;>

<!ELEMENT tunnel-source-address EMPTY>
<!ATTLIST tunnel-source-address
    address CDATA #REQUIRED>

<!ELEMENT tunnel-source-interface EMPTY>
<!ATTLIST tunnel-source-interface
    device IDREF #REQUIRED>

<!ELEMENT tunnel-destination-address EMPTY>

```

```
<!ATTLIST tunnel-destination-address
    address CDATA #REQUIRED>
```

Interface elements of all three types (`device`, `vlan-interface` and `tunnel`) often serve as link targets and are thus required to have the `@name` and `@id` attributes (defined through the `tgt.att` entity, see section 3).

In the case of network interfaces, the `@name` attribute deserves a careful consideration in order to keep the *Netopeer* configurations reasonably platform independent. Designers of operating system demonstrated immense creativity in naming network devices, so for example a Gigabit Ethernet interface can be identified by the following names, depending on the architecture:

- `GigabitEthernet0/1` in Cisco IOS – the numbers represent the slot and port in the router chassis.
- `ge-0/1/2` in JUNOS – the numbers again represent the slot, physical interface card (PIC) and port.
- `wm0` in Unix systems based on BSD – the first part (`wm`) identifies the device driver and is thus hardware specific; if there are more cards with the same driver, they are distinguished by the number (0, 1, etc.) in the order as they are detected by the system.
- `eth1` in Linux – this is similar to the previous case, only the first part (`eth`) is common to all Ethernet drivers.

Faced with this variety, we decided to handle the issue of network device names in the following way:

1. In a front-end, an interface can be assigned a name by the user or the *Netopeer* system, which may or may not correspond to the actual name of the interface in the target router (if there is any at all). This is necessary for identifying the interface in front-ends.
2. The back-end, which translates the configuration for a particular router, may use the XML configuration data along with another table that binds correct interface names to the `@id` attributes of XML elements representing interfaces

In other words, the invariant “handle” to each interface in the XML data is the `@id` attribute, while the `@name` attribute is essentially arbitrary until the final instantiation of the configuration for the target router.

VLAN interfaces are special in that their names are on all router platforms composed of the name of the parent physical interface and a numeric suffix, usually separated from the name by a dot, for example `GigabitEthernet0/1.1` in Cisco IOS. Therefore, the `vlan-interface/@name` attributes are required to contain just the numeric suffix – with them back-ends will be able to compose proper names of VLAN interfaces.

Apart from the `@id` and `@name` attributes, the interface elements share few others, which are all optional:

- `@mtu` – Maximum Transfer Unit, default is device dependent.
- `@filter-in`, `@filter-out` – refer to packet filter chains that are activated on the interface. The former applies to incoming and the latter to outgoing packets. By default no chains are attached.

The `@device` element has the following specific attributes, all of them optional:

- `@disable` – if set to yes, the network device will be disabled. Default is no.
- `@arp` – enable/disable the ARP protocol. Default is on.
- `@duplex` – set full or half duplex mode on certain interface types (e.g., Ethernet). The default is auto meaning that the duplex mode is autonegotiated.
- `@speed` – set the interface transmission rate in bits per second, optionally the symbolic factors "K" (kilo), "M" (mega) and "G" (giga) may be used, for example 9600, 128K, 100M or 2.5G. Default is device dependent and typically autonegotiated.
- `@multicast` – enable/disable the IGMP functions on the interface. Default is off.
- `@txqlen` – for devices that allow it, set the length of the transmit queue as the number of packets the queue can hold. The default is device dependent.
- `@macaddr` – set the MAC address of the interface. For most physical network devices this value is hardwired and should not be changed.
- `@encapsulation` – for serial interfaces, define the encapsulation (PPP or HDLC) to be used on the interface. Default is no encapsulation.

The `vlan-interface` element has two required attributes:

- @device – link to the underlying network device.
- @tag – VLAN tag.

The `tunnel` element contains, beside the optional `description` subelements that define the local and remote ends of the tunnel:

- `tunnel-destination-address` contains in its @address attribute IP address of the remote end.
- IP address of the local end can be specified either explicitly using the `tunnel-source-address` element, or by referring to an *active* network interface using the `tunnel-source-interface` – its @device attribute should contain the ID of that interface.

The required attribute `tunnel/@mode` defines the mode (type) of the tunnel. Three modes are supported: IPv6 in IPV4 (default), IPv4 in IPv4 and GRE (*Generic Route Encapsulation*).

The following attributes of `tunnel` are optional:

- @ttl – time to live (or hop count in IPv6 terminology) that is set in the IP headers of the (outer) tunnel packets. By default, the tunnel packets inherit the value from the inner (tunneled) packets.
- @device – bind the tunnel to a specific network interface. In this case the tunnel packets can be sent only out of this interface.
- @pmtudisc – enable/disable path MTU discovery on this tunnel. Default is on.
- @key (GRE tunnels only) – 32-bit value, which can be used as a (rather weak) security measure. By default no key is used.
- @checksum (GRE tunnels only) – enable/disable end-to-end checksumming of tunnel packets. Default is "off".
- @sequence (GRE tunnels only) – if set to "on", the tunnel interface will drop packets that arrive out of order.

6.2 Addressing

Any interface that is configured or defined in the above three classes may be assigned one or more network-layer addresses (IPv4, IPv6, or potentially others). This is done separately in yet another block, `interfaces/13-addresses`.

Each address defined within this block is attached to an interface by means of an XML link.

In formal terms:

```
<!ELEMENT 13-addresses ((ipv4-address|ipv6-address)*)>
```

Note that the addresses may appear in an arbitrary order.

```
<!ELEMENT ipv4-address EMPTY>
<!ATTLIST ipv4-address
    address CDATA #REQUIRED
    masklen CDATA #REQUIRED
    peer-address CDATA #IMPLIED
    broadcast CDATA #IMPLIED
    %address.att;>
```

```
<!ELEMENT ipv6-address EMPTY>
<!ATTLIST ipv6-address
    address CDATA #REQUIRED
    masklen CDATA #REQUIRED
    peer-address CDATA #IMPLIED
    scope (global|site|link|local) #IMPLIED
    %address.att;>
```

The `@address` and `@masklen` attributes specify the IP address and length of the subnet mask in bits, respectively. The `peer-address` can only be included for point-to-point links and contains the address of the opposite end of the link. The `ipv4-address/@broadcast` attribute gives the broadcast IP address. The default should be fine in most cases – the host part of the address is filled with ones (in a binary representation). Finally, the `ipv6-address/@scope` attribute defines the *scope* of the address – it should rarely be set explicitly, normally it is determined from the address itself, see [RFC2373].

7 Packet filtering

In the harsh environment of today's Internet, routers represent an important line of defense against various threats. This is especially true for edge routers at a boundary of different administrative domains. Basic requirements each router must satisfy are specified in [RFC2827]. For example, routers must block ingress traffic with "spoofed" addresses. In addition, most peripheral (institutional) networks allow connections from the global Internet only to a small subset of

services (SSH, SMTP, etc.) and/or define specific hosts that can receive such external connections.

This functionality of routers is often denoted as *packet filtering* or *stateless firewalling*. Stateless means that the decision about the fate of every received packet is based only on the information contained in the headers of that packet and does not keep any state information about established connections.

Packet filters are mostly configured as a sequence of rules specifying

1. Properties of the packet, like incoming/outgoing interface, MAC and IP addresses and values of specific fields and flags in the L2, L3 and L4 headers.
2. Action that is to be taken if the packet *matches* the rule, i.e., satisfies all the specified properties. We also say that the rule *fires* in this case.

Existing packet filter configuration languages differ rather widely in the ways how this principle is implemented a interpreted and it is often not easy to translate the rules from one language to another.

One of the important differences stems from the fact that some router architectures filter packets in hardware, mostly in network interface cards, while others do it in software. In the first case, each sequence of filtering rules must be bound to an interface where they are applied. Software filtering, as it is done for example in many Unix-based operating systems, allows a centralized approach where rules apply regardless of the incoming or outgoing interface (of course, an interface can nonetheless be specified as a required property of the packet).

The logic of the packet filter configuration in the present XML schema is modeled after the *Netfilter* subsystem of the Linux kernel, version 2.4 [Rus2002].

7.1 Packet filter chains

An ordered sequence of filtering rules is also known as *packet filter chain*. All chains are collected under an element `/configuration/packet-filters` and defined in the DTD as follows:

```
<!ELEMENT packet-filters (packet-filter-chain*)>
<!ATTLIST packet-filters
    global-in IDREF #IMPLIED
    global-out IDREF #IMPLIED>

<!ELEMENT packet-filter-chain (description?, packet-filter-rule+)>
<!ATTLIST packet-filter-chain
```

```
policy (accept|drop|reject|log) "drop"  
%tgt.att;>
```

A packet filter chain thus contains an optional description and one or more rules. For each packet subject to the chain, the rules are applied one by one in the specified order until the first rule that the packet matches. This matching rule then defines the action that will be performed with the packet (see below).

If no matching rule is found, the packet “falls through” the chain. In this case the attribute `packet-filter-chain/@policy` determines the default action. For example, selecting the value of “accept” means essentially “what is not explicitly forbidden is allowed” while the value of “drop” means the opposite: “what is not allowed is forbidden”. The values of the `@policy` attribute are related to packet filter actions that will be described in section 7.5.

The `packet-filter-chain` elements are expected to become a target of an XML link and so they possess `@id` and `@name` attributes. A filter chain is activated by being linked from an appropriate place. Our XML schema allows packet filter chains to be linked either from a specific network interface (from `@filter-in` and `@filter-out` of `device`, `vlan-interface` and `tunnel`) or from `packet-filters/@global-in` and `packet-filters/@global-out`. In the latter case the chains act as global filters that apply to all packets regardless of the interface. In both positions, though, the direction of data flow is significant – a chain is applied to either incoming or outgoing packets.

An interesting situation would arise if both global and interface-bound chains were configured for the same data flow direction. This semantic problem is resolved, by definition, as follows:

- For the *incoming* direction: First applied is the chain defined in the `@filter-in` attribute of the interface on which the packet arrives. If the packet matches no rule in this chain *and* its policy is “accept”, the packet is further submitted to the global chain that is linked from the attribute `packet-filters/@global-in`.
- For the *outgoing* direction, the interface-bound and global chains exchange their roles with respect to the previous case: First the global chain defined in `packet-filters/@global-out` is applied and if no rule fires and its policy is “accept”, the `@filter-out` chain bound to the outgoing interfaces is applied.

In practical terms, the implementors of *Netopeer* back-ends will have to cope with the following situations (for incoming packets, the solution for outgoing packets is analogical):

1. If the target router platform supports only interface-bound packet filters, the back-end should compose the packet filter for each interface *I* as follows: first all the rules of the chain attached to *I* in the XML configuration are used and then all the rules in the global filter (linked from `packet-filters/@global-in`), which either specify interface *I* among its criteria *or* do not specify any interface at all – in the latter case the same rule should be used on all interfaces.
2. If the target platform supports only a global packet filter, then it should include first all the interface-bound chains from the XML configuration and add the criterion specifying the particular interface to each rule. At the end, the global chain should be appended, if there is any. The order of inclusion of interface-bound chains is not important since they are mutually exclusive due to the interface criterion.

7.2 Packet filter rules

The contents of a packet filter rule are defined by our DTD as follows:

```
<!ELEMENT packet-filter-rule (description?, packet-match-list,
                             packet-action-list)>
```

An optional `description` is followed by two required elements defining the two parts of the rule, namely `packet-match-list` and `packet-action-list` with the following contents:

```
<!ELEMENT packet-match-list (match-interface?, match-mac?,
                             match-ipv4?, match-ipv6?,
                             match-tcp?, match-udp?,
                             match-icmp?)>
```

```
<!ELEMENT packet-action-list (log-action?,noop-action?,
                              (accept-action|drop-action|
                               reject-action|gosub-action)?)>
```

The `packet-match-list` element contains the conditions that have to be satisfied *simultaneously* for the rule to match. If this happens, all the actions specified by the `packet-action-list` are performed.

7.3 Prefix lists

In packet filter rules, as well as route filter rules that will be described later, one often needs to specify a certain subset of IPv4 or IPv6 address space. It can be done using the `/configuration/prefix-lists/prefix-list` element:

```

<!ELEMENT prefix-list (description?, match-prefix+)>
<!ATTLIST prefix-list
    %tgt.att;>

<!ELEMENT match-prefix EMPTY>
<!ATTLIST match-prefix
    %af.att;
    ge CDATA #IMPLIED
    le CDATA #IMPLIED
%prefix.att;>

```

The `prefix-list` element contains, apart from an optional `description`, one or more `match-prefix` elements representing a block of IP addresses as follows (see also the definitions of the entities `af.att` and `prefix.att`):

- The `@af` attribute defines the address family of the prefix (“`ipv4`” or “`ipv6`”). The default is implied from the form of the prefix.
- The `@address` attribute gives the IPv4/IPv6 address (address prefix).
- `@length` defines the prefix length, i.e., the number of leftmost bits of the address that will be used for matching.
- The `@ge` and `@le` attributes are only useful for matching route prefixes and will be explained in section 8.1.

An IP address (or route prefix) matches the prefix list if it matches at least one of the prefixes.

Any number of prefix lists can be defined using the `prefix-list` elements. They are all collected under the `/configuration/prefix-lists` element:

```

<!ELEMENT prefix-lists (prefix-list*)>

```

In order to apply a prefix list, one has to link it from a packet (or route) filter rule.

7.4 Match conditions

The DTD specifies the following condition against which a packet can be matched. Please note that all leaf elements include (through the `match.att` entity) the `@negate` attribute, which allows specifying the logically opposite condition.

```
<!ELEMENT match-interface EMPTY>
<!ATTLIST match-interface
    if IDREF #REQUIRED
    %match.att;>
```

This condition specifies the incoming or outgoing interface for the packet by using the link in the @if attribute. The direction is determined implicitly from the enclosing chain (-in or -out).

```
<!ELEMENT match-mac EMPTY>
<!ATTLIST match-mac
    src CDATA #REQUIRED
    %match.att;>
```

Using this condition the *source* MAC address can be checked. The @src attribute should contain the MAC address in the usual notation, i.e., six bytes expressed in hexadecimal and separated by colons. Example: *00:30:4f:07:a9:a2*.

```
<!ELEMENT match-ipv4 (match-source?, match-destination?,
    match-dsfield?, match-ecnfield?)>
<!ATTLIST match-ipv4
    fragment (all|first|subseq) "all">

<!ELEMENT match-ipv6 (match-source?, match-destination?,
    match-dsfield?, match-ecnfield?,
    match-flowlabel?)>
```

The elements match-ipv4 and match-ipv6 are non-leaf and contain several elements related to IPv4/IPv6 headers. These will be described in more detail below. For IPv4, the attribute match-ipv4/@fragment can also be used to specify whether the condition is to be applied to just the first IP fragment or all fragments except the first one. By default all fragments are checked.

```
<!ELEMENT match-source EMPTY>
<!ATTLIST match-source
    list IDREF #REQUIRED
    %match.att;>
```

```
<!ELEMENT match-destination EMPTY>
<!ATTLIST match-destination
    list IDREF #REQUIRED
    %match.att;>
```

These two elements represent conditions on source or destination IP addresses. The `list` attribute is a link to a prefix list (see section 7.3) that defines the IP prefixes to match the address against.

```
<!ELEMENT match-dsfield EMPTY>
<!ATTLIST match-dsfield
    dscp CDATA #REQUIRED
    %match.att;>
```

```
<!ELEMENT match-ecnfield EMPTY>
<!ATTLIST match-ecnfield
    ect (on|off|dontcare) "dontcare"
    ce (on|off|dontcare) "dontcare"
    %match.att;>
```

These two elements serve for matching the contents of the former TOS field in the IPv4 header or traffic class field in the IPv6 header. According to [RFC3260], the original TOS/traffic class octet is subdivided into two portions:

- The six most significant bits are used for DSCP (*Differentiated Services Code Point*).
- The two least significant bits are used for Explicit Congestion Notification (ECN), see [RFC3168].

The `match-dsfield/@dscp` attribute should contain the value of DSCP. The two bits of the ECN field are given in the two attributes `match-ecnfield/@ect` (ECN-capable transport) and `match-ecnfield/@ce` (Congestion Experienced). The default value "dontcare" means that the bit can have any value (is not checked).

```
<!ELEMENT match-flowlabel EMPTY>
<!ATTLIST match-flowlabel
    value CDATA #REQUIRED
    %match.att;>
```

Using this element the value of the *flow label* field in the IPv6 header can be matched.

```
<!ELEMENT match-tcp (match-source-port-range?,
    match-destination-port-range?,
    match-tcp-flags?, match-tcp-ecn?,
    match-tcp-option*)>
```

This element is composed of five leaf elements that enable matching different items in the TCP header.

```
<!ELEMENT match-source-port-range EMPTY>
<!ATTLIST match-source-port-range
    lo CDATA #REQUIRED
    hi CDATA #REQUIRED
    %match.att;>

<!ELEMENT match-destination-port-range EMPTY>
<!ATTLIST match-destination-port-range
    lo CDATA #REQUIRED
    hi CDATA #REQUIRED
    %match.att;>
```

These two conditions are satisfied if the source or destination port number is within the limits given by the values of @lo and @hi. This element is used for both TCP and UDP.

```
<!ELEMENT match-tcp-flags EMPTY>
<!ATTLIST match-tcp-flags
    syn (on|off|dontcare) "dontcare"
    ack (on|off|dontcare) "dontcare"
    fin (on|off|dontcare) "dontcare"
    rst (on|off|dontcare) "dontcare"
    urg (on|off|dontcare) "dontcare"
    psh (on|off|dontcare) "dontcare">
```

Using this element the values of all standard TCP flags can be specified. The default value of "dontcare" means that the particular TCP flag value is irrelevant.

```
<!ELEMENT match-tcp-ecn EMPTY>
<!ATTLIST match-tcp-ecn
    cwr (on|off|dontcare) "dontcare"
    ece (on|off|dontcare) "dontcare">
```

This condition allows matching the Explicit Congestion Notification (ECN) flags in the TCP header – Congestion Window Reaction (CWR) and ECN Echo (ECE), see [RFC3168].

```
<!ELEMENT match-tcp-option EMPTY>
<!ATTLIST match-tcp-option
    kind CDATA #REQUIRED
    %match.att;>
```

TCP header can be augmented by various options (currently about 25 of them are defined, only few being really useful though). Specific TCP option can be matched by filling in the @kind option by the appropriate option number.

```
<!ELEMENT match-udp (match-source-port-range?,
    match-destination-port-range?)>
```

This element contains the conditions available for matching UDP headers – source and destination port ranges.

```
<!ELEMENT match-icmp EMPTY>
<!ATTLIST match-icmp
    type CDATA #REQUIRED
    code CDATA #IMPLIED>
```

Using this element ICMP packets can be matched. In order to match ICMP messages of a specific type, the attributes @type and @code can be filled in with appropriate values, see [RFC0792] for IPv4 and [RFC2463] for IPv6.

7.5 Packet filter actions

If a packet matches the conditions specified in the first part of a filter rule, an action is performed according to the second part of the same rule. The following paragraphs describe the elements that represent the available actions.

Each of the action elements has a @count attribute (declared through the action.att entity). If it is set to "yes", the system should increment packet and byte counters for each packet that matches the rule, along with performing the specified action. However, some platforms may not support this feature.

The declaration of the packet-action-list indicates that four of the allowed actions, namely accept-action, drop-action, reject-action and gosub-action, are mutually exclusive. The other two actions, log-action and noop-action are may coexist with other actions (usually with just a single terminal action) in a single packet action list.

The actions accept-action, drop-action and reject-action are *terminal* in the sense that after being executed they always terminate the processing of the enclosing filter chain.

If a rule fires and contains only non-terminal actions, the filtering process then continues with the next rule in the chain.

```
<!ELEMENT accept-action EMPTY>
<!ATTLIST accept-action
    %action.att;>
```

Accept the packet, i.e., allow it to pass through the “checkpoint”.

```
<!ELEMENT drop-action EMPTY>
<!ATTLIST drop-action
    %action.att;>
```

Drop the packet silently.

```
<!ELEMENT reject-action EMPTY>
<!ATTLIST reject-action
    type CDATA #IMPLIED
    code CDATA #IMPLIED
    %action.att;>
```

This action is similar to the previous one, but in addition to dropping the packet an ICMP error message is sent to the packet originator. ICMP message type can be selected using the @type and @code attributes – it should contain a numeric value as defined by [RFC0792] for IPv4 and [RFC2463]. The default is port “unreachable” (type 3 and code 3 for IPv4, or type 1 and code 4 for IPv6).

```
<!ELEMENT gosub-action EMPTY>
<!ATTLIST gosub-action
    goto IDREF #REQUIRED
    %action.att;>
```

This action enables “jumps” or “subroutine calls” in the otherwise linear sequence of filter rules. It works in the following way:

- The @goto attribute contains a link to another chain whose rules will be examined in sequence as if they were part of the original (calling) chain.
- If a packet falls through the “subroutine” chain, i.e., matches either no rules at all or only rules with non-terminal actions, then further processing depends on the value of the @policy attribute in the subroutine

chain. If it corresponds to a terminal action (values "accept", "drop" or "reject"), the packet is handled accordingly and the processing stops here. If, on the other hand, @policy is either not specified for the subroutine chain or its value is "log", the processing continues with the rule in the original (calling) chain immediately after the rule with gosub-action that called the subroutine chain.

```
<!ELEMENT log-action EMPTY>
<!ATTLIST log-action
    level (debug|info|notice|warning|
          err|crit|alert|emerg) "info"
    %action.att;>
```

Log the information about the packet, typically via the *syslog* system. The importance of the message can be modified by setting the @level attribute (the default is "info").

```
<!ELEMENT noop-action EMPTY>
<!ATTLIST noop-action
    %action.att;>
```

This action is a no-operation, i.e., it does nothing. It can be useful though for counting packets that match the rule and their aggregate size.

8 Routing

Every IP router deserving this name must support a number of routing protocols and means for managing and manipulating routing tables. In this version of *Netopeer* XML schema we include only a limited functionality – static routes and RIP/RIPng protocols.

The entire routing subsystem is found under the /configuration/routing element:

```
<!ELEMENT routing (route-filters?, static-routes?,
                  rip?, ripng?)>
```

8.1 Route filters

Routing table is a central data structure used by the router control plane. Routes that are inserted into the routing table may come from different sources (routing

protocols or statically configured information). A proper administration of an IP router requires careful management of the routing table and related components, in particular to be able to:

- control which routes learned by a particular routing protocol are inserted into the routing table,
- control which routes are advertised by individual routing protocols,
- control which routes are transferred to the forwarding table.

Philosophy of our route filtering subsystem is based on the model known from Juniper Networks routers, where all routes must pass through the routing table, which thus serves as a route exchange center. It means that

- routing table accepts routes from active routing protocols (or configured static routes) – this is called *route import*;
- routing table provides routes to routing protocols or the forwarding table – this is called *route export*.

By default, all routes learned by routing protocols are imported to the routing table. In the opposite direction the default strategy depends on the routing protocol. In general, so called redistribution of routes between different protocols is not performed by default and must be explicitly configured as necessary.

It is not uncommon that routers have to work with more than one routing table in order to support multicast routing or routing policies. Such features will be handled in future versions of the *Netopeer* XML schema. In this version we assume a single routing table.

Configuration of route filter rules resembles the packet filter framework described in section 7. The topmost element is `routing/route-filters`:

```
<!ELEMENT route-filters (route-filter-chain*)>
```

Under this element an arbitrary number of route filter chains may be defined:

```
<!ELEMENT route-filter-chain (description?,  
                                route-filter-rule+)>  
<!ATTLIST route-filter-chain  
%tgt.att;>
```

Each `route-filter-chain` element can be a target for an XML link. In order to activate a route filter chain, it must be linked from an appropriate place (e.g., routing protocol).

Route filter chains contain route filter rules that have the same structure as their packet filter counterparts:

```
<!ELEMENT route-filter-rule (description?, route-match-list,
                             route-action-list)>
```

After an optional description, each rule contains `route-match-list` and `route-action-list` elements:

```
<!ELEMENT route-match-list (match-destination?, match-nexthop?,
                             match-metric?, match-interface?,
                             match-route-source?,
                             match-route-type?)>
```

```
<!ELEMENT route-action-list (set-nexthop?, set-metric?,
                              accept-action?, drop-action?,
                              log-action?, gosub-action?,
                              noop-action?)>
```

8.1.1 Matching route properties

Each route carries a number of properties that can be matched in a route filter rule. Two of the elements – `match-destination` and `match-interface` – are the same as in packet filter rules. However, if prefix lists are used for route matching, the elements `prefix-list/match-prefix` may specify the `@ge` and `@le` attributes. The routing prefixes are then matched in the following way:

- If neither `@ge` nor `@le` is used, the routing prefix must match *exactly* the prefix in the prefix list.
- If `@ge` or `@le` (or both) are specified, then the routing prefix can match only if its length is greater or equal than the value of `@ge` and less or equal than the value of `@le`. For each `prefix-list/match-prefix` element, the values of its attributes must satisfy the inequalities

$@length \leq @ge \leq @le \leq adrlen,$

where *adrlen* is 32 for IPv4 and 128 for IPv6.

Other match conditions are specific for route filters:

```
<!ELEMENT match-nexthop EMPTY>
<!ATTLIST match-nexthop
    list IDREF #REQUIRED
    %match.att;>
```

This element is used for matching the next hop address against a prefix list pointed to by the `@list` attribute. In this case the `@ge` and `@le` attributes don't apply.

```
<!ELEMENT match-metric EMPTY>
<!ATTLIST match-metric
    metric CDATA #REQUIRED
    %match.att;>
```

The metric associated with the route can be matched using this element.

```
<!ELEMENT match-route-source EMPTY>
<!ATTLIST match-route-source
    source (static|connected|rip|ripng|
    ospf|ospf3|isis|bgp) #REQUIRED
    %match.att;>
```

Each route carries information about the source it comes from. By setting the value of the `@source` attribute we can select routes coming from a particular routing protocol or static routes.

```
<!ELEMENT match-route-type EMPTY>
<!ATTLIST match-route-type
    type (normal|local|blackhole|
    unreachable|prohibit) "normal"
    %match.att;>
```

Routes are also characterized by their type. Most routes are of the "normal" type. Routes to host-local destinations (loopback interfaces etc.) have the type "local". The other values signal that the destination is unreachable and the routing subsystem should behave as follows: For "blackhole" routes the packet is silently discarded. It is also the case for the other two types but, in addition, an ICMP message is also generated, namely "Host Unreachable" for "unreachable" routes and "Communication Administratively Prohibited" for "prohibit" routes.

8.1.2 Route filter actions

Some of the available route filter actions have the same or similar meaning as in packet filter rules: `accept-action`, `drop-action`, `drop-action`, `log-action`, `gosub-action` and `noop-action`. However, apart from simply accepting or rejecting a route it is often necessary to modify one or more attributes of the route. For these purposes, there is a set of new *non-terminal* actions/elements:

```
<!ELEMENT set-nexthop (nexthop+)>
```

```
<!ELEMENT nexthop (description?)>
```

```
<!ATTLIST nexthop
    via CDATA #IMPLIED
    device IDREF #IMPLIED
    weight CDATA #IMPLIED>
```

Using this action one or more next hop routers can be specified. The `@via` attribute contains the IP address of the next hop router and `@device` optionally a link to the output network device. For multipath routes (those having multiple next hops), the value of the `@weight` attribute determines the relative preference of each path.

```
<!ELEMENT set-metric EMPTY>
```

```
<!ATTLIST set-metric
    metric CDATA #REQUIRED>
```

This way the metric associated with the route can be modified.

8.1.3 Activating route filters

The elements corresponding to various routing protocols contain the following two subelements that activate a route filter chain:

```
<!ELEMENT route-import EMPTY>
<!ATTLIST route-import
    chain IDREF #REQUIRED>
```

```
<!ELEMENT route-export EMPTY>
<!ATTLIST route-export
    chain IDREF #REQUIRED>
```

The @chain attributes refer to the selected chain. Remember that the terms “import” and “export” reflect the viewpoint of the routing table. Therefore, by including the element route-import in the configuration of a routing protocol we affect the routes coming *from* that routing protocol to the routing table.

8.2 Static routes

Static routes are configured inside the routing/static-routes element:

```
<!ELEMENT static-routes (route-import?, route+)>
```

Using the route-import element, we can control which static routes are imported to the routing table. On the other hand, routes from the routing table cannot be exported to static routes since the main virtue of static routes is that they must be manually configured by the router administrator.

The static-routes element then contains one or more elements representing static routes:

```
<!ELEMENT route (description?, destination, nexthop+)>
<!ATTLIST route
    %af.att;
    type (normal|local|reject|prohibit|blackhole) "normal"
    metric CDATA #IMPLIED
    preference CDATA #IMPLIED
    scope (link|site|global) #IMPLIED
    zone-index CDATA #IMPLIED>
```

Beside an optional description, each route must include exactly one destination element and one or more nexthop elements. The latter was defined above and destination is just a wrapper for a destination address prefix:

```
<!ELEMENT destination EMPTY>
<!ATTLIST destination
    %prefix.att;>
```

We can specify a number of other properties through the values of following attributes:

- @af – address family, “ipv4” or “ipv6”. Usually it can be deduced from the form of the destination address.
- @type – type of route as described in subsection 8.1.1.

- @metric – this value affects the routing algorithm: destination prefix length is the primary criterion (routes with longer matching prefixes are preferred) and for the same prefix length the route with minimum metric is preferred.
- @preference – preference of static routes with respect to individual routing protocols. The relative ordering of route sources through preference values is platform dependent.
- @scope – this attribute is mainly useful for IPv6 routes and should be usually determined automatically from the destination address: The link-local scope is associated with IPv6 addresses starting with the prefix *fe80::/10* and site-local with *fec0::/10*, see [RFC2373].
- @zone-index – value of this attribute distinguishes different zones of the same address scope, see [Deer2002].

8.3 RIP

Routing Information Protocol (RIP) is one of the oldest IP interior gateway protocols. The original version 1 was defined in [RFC1058] and the improved version 2 in [RFC2543]. In most backbone networks it has already been replaced by modern IGP protocols like OSPF or IS-IS.

The `routing/rip` element and its subelements are defined as follows:

```
<!ELEMENT rip (route-import?, route-export?,
               rip-interface+, rip-neighbor*)>
<!ATTLIST rip
  disable (yes|no) "no"
  version (1|2|12in1out) "12in1out"
  broadcast (yes|no) #IMPLIED
  check-zero (yes|no) "yes"
  preference CDATA #IMPLIED
  default-metric CDATA #IMPLIED>
```

Unlike static routes, routes can be both imported and exported from RIP to the routing table and vice versa. The attributes have the following meaning:

- @disable – if set to "yes", RIP is disabled. Default is "no".
- @version – specifies the RIP version to be used by default (this can be overridden on each interface). The default value of this attribute is "12in1out", which means that packets of both versions will be received and only version 1 packets sent.

- @broadcast – if set to "yes", RIP will broadcast version 1 packets even if no rip/rip-interface (see below) is defined. The default is "no" and the behavior is then interface specific.
- @check-zero – The specifications require that the reserved fields in the RIP packet be filled with zeros. If this attribute is set to "yes" (default), RIP packets that violate this requirement are discarded. Some RIP implementations use the reserved fields for nonstandard features. If you want to interoperate with them, set this attribute to "no".
- @preference – relative preference of routes learned from RIP in comparison to other sources. This is a platform-dependent value.
- @default-metric – value of the metric that will be used when advertising routes learned from other sources in RIP. The default is 1.

RIP version 1 normally broadcast its packets out of all participating interface. On non-broadcast links this is not possible and so the IP address of the neighboring router must be specified explicitly using the following element:

```
<!ELEMENT rip-neighbor EMPTY>
<!ATTLIST rip-neighbor
    address CDATA #REQUIRED>
```

For each interface that is to participate in the RIP protocol, a rip-interface element must be included, unless we use RIPv1 with the broadcast option.

```
<!ELEMENT rip-interface (rip-authentication?)>
<!ATTLIST rip-interface
    device IDREF #REQUIRED
    receive (yes|no) "yes"
    send (yes|no) "yes"
    in-version (1|2|both) "both"
    out-version (1|2) "2"
    split-horizon (yes|no) "yes"
    metric-in CDATA #IMPLIED
    metric-out CDATA #IMPLIED>
```

The configuration applies to the interface that is referred to by the attribute @device. The other attributes have the following meaning:

- @receive – determines whether RIP packets are received on the interface.
- @send – determines whether RIP packets are sent via the interface.

- `@in-version` – if "1" or "2" is specified, only packets of that version are accepted on the interface. The default is "both", which means that both versions are accepted.
- `@out-version` – this attribute specifies the RIP version that is used for packets sent out of the interface. The default is "2".
- `@split-horizon` – enables or disables the split horizon mode (RIP updates received on the interface are not sent back to the same interface). The default value is "yes" and should be used in most cases.
- `@metric-in` – value to be added to the metric of all routes received on the interface. The default is 1.
- `@metric-out` – value to be added to the metric of all routes send via the interface. The default is 0.

RIP version 2 introduced optional authentication of routing updates.

```
<!ELEMENT rip-authentication EMPTY>
<!ATTLIST rip-authentication
    type (simple|md5) "simple"
    password CDATA #REQUIRED>
```

Two authentication methods are available and can be selected by the value of the attribute `@type`:

- "simple" means that a string specified in the `@password` attribute (up to 16 characters) is included in every update sent on the interface and all RIP packets received on the interface are also checked whether they contain the same password. This method actually provides almost no security since the password travels unencrypted.
- "md5" is a much more robust authentication method, which computes a hash value (message digest) of both the RIP packet *and* a password. The digest is then sent with the RIP packet.

8.4 RIPng

RIPng is essentially a modification (and simplification) of RIP version 2 for the IPv6 address family. Therefore, its representation in the DTD is very similar to RIP:

```

<!ELEMENT ripng (route-import?, route-export?,
                 ripng-interface?)>
<!ATTLIST ripng
  disable (yes|no) "no"
  preference CDATA #IMPLIED
  default-metric CDATA #IMPLIED>

<!ELEMENT ripng-interface EMPTY>
<!ATTLIST ripng-interface
  device IDREF #REQUIRED
  receive (yes|no) "yes"
  send (yes|no) "yes">

```

Note that there are no authentication options – RIPng relies on the IPSec framework that is (or should be) part of every IPv6 implementation.

9 An example configuration

Here is an example of a valid XML configuration that covers most of the features supported in the current version of the DTD:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE configuration SYSTEM "netopeer.dtd">

<configuration version="0.1">
  <system hostname="burcak" multicast-routing="on">
    <banner>Welcome to burcak!</banner>
    <dns>
      <dns-server address="2001:718:1:1::2"/>
      <search>
<domain-suffix suffix="cesnet.cz"/>
<domain-suffix suffix="muni.cz"/>
      </search>
    </dns>
  </system>
  <interfaces>
    <network-devices>
      <device duplex="full" name="FastEthernet0/0" id="i00">
<description>VLAN trunk</description>
      </device>
      <device name="FastEthernet0/1" id="i01">

```

```

<description>Library</description>
  </device>
  <device disable="yes" name="FastEthernet0/2" id="i02">
<description>Lab</description>
  </device>
  <device name="Serial1/0" mtu="512"
        encapsulation="ppp" id="i10">
<description>Connection to the Internet</description>
  </device>
</network-devices>
<virtual-lans>
  <vlan-interface name="1" device="i00" tag="101" id="i001">
<description>Management</description>
  </vlan-interface>
  <vlan-interface name="2" device="i00" tag="102" id="i002">
<description>Marketing</description>
  </vlan-interface>
  <vlan-interface device="i00" tag="103" multicast="on"
name="3" id="i003">
<description>Research</description>
  </vlan-interface>
</virtual-lans>
<tunnels>
  <tunnel mode="gre" ttl="64" name="Tunnel0" id="i10t0">
<description>IPv6 tunnel to our branch office</description>
<tunnel-source-interface device="i10"/>
<tunnel-destination-address address="222.2.2.2"/>
  </tunnel>
</tunnels>
<l3-addresses>
  <ipv4-address address="10.1.0.1" masklen="16"
        device="i001"/>
  <ipv6-address address="2001:718:0:1::1" masklen="64"
        device="i001"/>
  <ipv4-address address="10.2.0.1" masklen="16"
        device="i002"/>
  <ipv6-address address="2001:718:0:2::1" masklen="64"
        device="i002"/>
  <ipv4-address address="10.3.0.1" masklen="16"
        device="i003"/>
  <ipv6-address address="2001:718:0:3::1" masklen="64"
        device="i003"/>
  <ipv4-address address="192.168.1.1" masklen="30"

```

```

        device="i01"/>
<ipv6-address address="2001:718:0:0::1" masklen="64"
        device="i01"/>
<ipv4-address address="195.113.1.33" masklen="27"
        device="i02"/>
<ipv4-address address="10.4.0.1" masklen="16"
        device="i02" role="secondary"/>
<ipv6-address address="2001:718:0:5::1" masklen="64"
        device="i02"/>
<ipv4-address address="111.1.1.121" masklen="30"
        peer-address="111.1.1.122" device="i10"/>
<ipv6-address address="2001:718:1:1::1" masklen="64"
        device="i10"/>
<ipv6-address address="2001:718:0:111::1" masklen="64"
        device="i10t0"/>
</13-addresses>
</interfaces>
<prefix-lists>
<prefix-list name="nospoof4" id="lkjoz3r">
    <match-prefix address="192.168.0.0" length="16"/>
    <match-prefix address="10.0.0.0" length="8"/>
    <match-prefix address="195.113.1.0" length="24"/>
</prefix-list>
<prefix-list name="nospoof6" id="gogo42">
    <match-prefix af="ipv6" address="2001:718:0::"
        length="48"/>
</prefix-list>
<prefix-list name="trusted4" id="pokj09">
    <match-prefix address="10.1.0.0" length="16"/>
    <match-prefix address="10.2.0.0" length="15"/>
</prefix-list>
<prefix-list name="trusted6" id="uhvg51">
    <match-prefix address="2001:718:0:1::" length="64"/>
    <match-prefix address="2001:718:0:2::" length="63"/>
</prefix-list>
</prefix-lists>
<packet-filters global-in="bflm5">
    <packet-filter-chain name="main" policy="drop" id="bflm5">
        <packet-filter-rule>
<description>From LAN Management - accept</description>
<packet-match-list>
    <match-interface if="i001"/>
</packet-match-list>

```

```

<packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
<description>From LAN Marketing - accept</description>
<packet-match-list>
  <match-interface if="i002"/>
</packet-match-list>
<packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
<description>From LAN Research - accept</description>
<packet-match-list>
  <match-interface if="i003"/>
</packet-match-list>
<packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
<description>Anti-spoofing v4</description>
<packet-match-list>
  <match-interface if="i10"/>
  <match-ipv4>
    <match-source list="lkjoz3r"/>
  </match-ipv4>
</packet-match-list>
<packet-action-list>
  <drop-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
<description>Anti-spoofing v6</description>
<packet-match-list>
  <match-interface if="i10"/>
  <match-ipv4>
    <match-source list="gogo42"/>
  </match-ipv4>
</packet-match-list>
<packet-action-list>

```

```

    <drop-action/>
</packet-action-list>
    </packet-filter-rule>
    <packet-filter-rule>
<description>Accept v4 destinations outside trusted LANs</description>
<packet-match-list>
    <match-ipv4>
        <match-destination list="pokj09" negate="yes"/>
    </match-ipv4>
</packet-match-list>
<packet-action-list>
    <accept-action/>
</packet-action-list>
    </packet-filter-rule>
    <packet-filter-rule>
<description>Accept v6 destinations outside trusted LANs</description>
<packet-match-list>
    <match-ipv6>
        <match-destination list="uhvg51" negate="yes"/>
    </match-ipv6>
</packet-match-list>
<packet-action-list>
    <accept-action/>
</packet-action-list>
    </packet-filter-rule>
    <packet-filter-rule>
<packet-match-list>
    <match-tcp/>
</packet-match-list>
<packet-action-list>
    <gosub-action goto="ux1b"/>
</packet-action-list>
    </packet-filter-rule>
    <packet-filter-rule>
<packet-match-list>
    <match-udp/>
</packet-match-list>
<packet-action-list>
    <gosub-action goto="zzR6"/>
</packet-action-list>
    </packet-filter-rule>
</packet-filter-chain>
<packet-filter-chain name="tcp" id="ux1b">

```

```

        <packet-filter-rule>
<packet-match-list>
  <match-tcp>
    <match-destination-port-range lo="ssh" hi="ssh"/>
  </match-tcp>
</packet-match-list>
<packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
<packet-match-list>
  <match-tcp>
    <match-destination-port-range lo="smtp" hi="smtp"/>
  </match-tcp>
</packet-match-list>
<packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
<packet-match-list>
  <match-tcp>
    <match-destination-port-range lo="imaps" hi="imaps"/>
  </match-tcp>
</packet-match-list>
<packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
<packet-match-list>
  <match-tcp>
    <match-destination-port-range lo="h323hostcall"
                                  hi="h323hostcall"/>
  </match-tcp>
</packet-match-list>
<packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
<packet-match-list>

```

```

    <match-tcp>
      <match-destination-port-range lo="30000" hi="30010"/>
    </match-tcp>
  </packet-match-list>
</packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
<description>Drop other TCPs initiated from outside</description>
<packet-match-list>
  <match-tcp>
    <match-tcp-flags syn="on" ack="off" rst="off"/>
  </match-tcp>
</packet-match-list>
<packet-action-list>
  <drop-action/>
</packet-action-list>
  </packet-filter-rule>
</packet-filter-chain>
  <packet-filter-chain name="udp" id="zzR6">
    <packet-filter-rule>
<packet-match-list>
  <match-udp>
    <match-destination-port-range lo="domain" hi="domain"/>
  </match-udp>
</packet-match-list>
<packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
<packet-match-list>
  <match-udp>
    <match-source-port-range lo="domain" hi="domain"/>
  </match-udp>
</packet-match-list>
<packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
<packet-match-list>

```

```

    <match-udp>
      <match-destination-port-range lo="ntp" hi="ntp"/>
    </match-udp>
  </packet-match-list>
</packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
</packet-match-list>
  <match-udp>
    <match-source-port-range lo="ntp" hi="ntp"/>
  </match-udp>
</packet-match-list>
</packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
</packet-match-list>
  <match-udp>
    <match-destination-port-range lo="5000" hi="5003"/>
  </match-udp>
</packet-match-list>
</packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
</packet-match-list>
  <match-udp>
    <match-source-port-range lo="5000" hi="5003"/>
  </match-udp>
</packet-match-list>
</packet-action-list>
  <accept-action/>
</packet-action-list>
  </packet-filter-rule>
  <packet-filter-rule>
</description>Drop everything else</description>
</packet-match-list>
</packet-match-list>
</packet-action-list>

```

```

    <drop-action/>
</packet-action-list>
    </packet-filter-rule>
</packet-filter-chain>
</packet-filters>
<routing>
    <route-filters>
        <route-filter-chain name="static-routes" id="1562">
<route-filter-rule>
    <route-match-list>
        <match-route-source source="static"/>
</route-match-list>
    <route-action-list>
        <accept-action/>
</route-action-list>
</route-filter-rule>
        </route-filter-chain>
    </route-filters>
    <static-routes>
        <route preference="110">
<destination address="192.168.2.0" length="24"/>
<nexthop via="192.168.1.2"/>
        </route>
        <route af="ipv6">
<destination address="2001:718:0:4::" length="64"/>
<nexthop via="2001:718::2"/>
        </route>
    </static-routes>
    <rip preference="120" default-metric="2">
        <route-export chain="1562"/>
        <rip-interface device="i10" in-version="2"
            out-version="2" split-horizon="no">
<rip-authentication type="md5" password="mnam-mnam"/>
        </rip-interface>
    </rip>
    <ripng>
        <route-export chain="1562"/>
        <ripng-interface device="i10t0"/>
    </ripng>
</routing>
</configuration>

```

10 The complete DTD

This is the current version of the DTD in its entirety:

```
<!--
```

```
Program name: DTD for router configuration
Copyright (C) 2003 CESNET
Author(s): Ladislav Lhotka <Lhotka@cesnet.cz>
```

```
$Id: netopeer.dtd,v 1.7 2003/04/09 06:21:40 lhotka Exp $
```

```
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version 2 of the
License, or (at your option) any later version.
```

```
This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
02111-1307, USA.
```

```
-->
```

```
<!-- Common attributes -->
```

```
<!ENTITY % tgt.att
  'name CDATA #REQUIRED
  id ID #REQUIRED'>
```

```
<!ENTITY % af.att
  'af (ipv4|ipv6) #IMPLIED'>
```

```
<!ENTITY % match.att
  'negate (yes|no) "no"'>
```

```
<!ENTITY % action.att
  'count (yes|no) "no"'>
```

```

<!ENTITY % prefix.att
  'address CDATA #REQUIRED
  length CDATA #IMPLIED'>

<!ENTITY % address.att
  'device IDREF #REQUIRED
  role (primary|secondary) "primary"'>

<!-- Common elements -->

<!ELEMENT description (#PCDATA)>

<!-- Root element -->

<!ELEMENT configuration (system, interfaces, prefix-lists?,
                        packet-filters?, routing?)>
<!ATTLIST configuration
  version (0.1) #REQUIRED>

<!-- System -->

<!ELEMENT system (description?, banner?, dns?)>
<!ATTLIST system
  hostname CDATA #REQUIRED
  multicast-routing (on|off) "off">

<!ELEMENT banner (#PCDATA)>

<!ELEMENT dns (dns-server*, search?)>

<!ELEMENT dns-server EMPTY>
<!ATTLIST dns-server
  %af.att;
  address CDATA #REQUIRED>

<!ELEMENT search (domain-suffix+)>

<!ELEMENT domain-suffix EMPTY>
<!ATTLIST domain-suffix
  suffix CDATA #REQUIRED>

<!-- Interfaces -->

```

```

<!ELEMENT interfaces (network-devices, virtual-lans?,
                      tunnels?, 13-addresses)>

<!ELEMENT network-devices (device+)>

<!ELEMENT device (description?)>
<!ATTLIST device
  disable (yes|no) "no"
  arp (on|off) "on"
  duplex (full|half|auto) "auto"
  speed CDATA #IMPLIED
  multicast (on|off) "off"
  txqlen CDATA #IMPLIED
  mtu CDATA #IMPLIED
  macaddr CDATA #IMPLIED
  encapsulation (ppp|hdlc|none) "none"
  filter-in IDREF #IMPLIED
  filter-out IDREF #IMPLIED
  %tgt.att;>

<!ELEMENT virtual-lans (vlan-interface+)>

<!ELEMENT vlan-interface (description?)>
<!ATTLIST vlan-interface
  device IDREF #REQUIRED
  encapsulation (802.1q|isl) "802.1q"
  tag CDATA #REQUIRED
  multicast (on|off) "off"
  mtu CDATA #IMPLIED
  filter-in IDREF #IMPLIED
  filter-out IDREF #IMPLIED
  %tgt.att;>

<!ELEMENT tunnels (tunnel+)>

<!ELEMENT tunnel (description?,
                 (tunnel-source-address|tunnel-source-interface),
                 tunnel-destination-address)>
<!ATTLIST tunnel
  mode (ipip|v6inv4|gre) "v6inv4"
  ttl CDATA #IMPLIED
  device IDREF #IMPLIED

```

```

        multicast (on|off) "off"
        mtu CDATA #IMPLIED
        macaddr CDATA #IMPLIED
        pmtudisc (on|off) "on"
        key CDATA #IMPLIED
        checksum (on|off) "off"
        sequence (on|off) "off"
        filter-in IDREF #IMPLIED
        filter-out IDREF #IMPLIED
        %tgt.att;>

<!ELEMENT tunnel-source-address EMPTY>
<!ATTLIST tunnel-source-address
        address CDATA #REQUIRED>

<!ELEMENT tunnel-source-interface EMPTY>
<!ATTLIST tunnel-source-interface
        device IDREF #REQUIRED>

<!ELEMENT tunnel-destination-address EMPTY>
<!ATTLIST tunnel-destination-address
        address CDATA #REQUIRED>

<!ELEMENT l3-addresses ((ipv4-address|ipv6-address)*)>

<!ELEMENT ipv4-address EMPTY>
<!ATTLIST ipv4-address
        address CDATA #REQUIRED
        masklen CDATA #REQUIRED
        peer-address CDATA #IMPLIED
        broadcast CDATA #IMPLIED
        %address.att;>

<!ELEMENT ipv6-address EMPTY>
<!ATTLIST ipv6-address
        address CDATA #REQUIRED
        masklen CDATA #REQUIRED
        peer-address CDATA #IMPLIED
        scope (global|site|link|local) #IMPLIED
        %address.att;>

<!-- Prefix lists -->

```

```

<!ELEMENT prefix-lists (prefix-list*)>

<!ELEMENT prefix-list (description?, match-prefix+)>
<!ATTLIST prefix-list
      %tgt.att;>

<!ELEMENT match-prefix EMPTY>
<!ATTLIST match-prefix
      %af.att;
      ge CDATA #IMPLIED
      le CDATA #IMPLIED
%prefix.att;>

<!-- Packet filter -->
<!ELEMENT packet-filters (packet-filter-chain*)>
<!ATTLIST packet-filters
      global-in IDREF #IMPLIED
      global-out IDREF #IMPLIED>

<!ELEMENT packet-filter-chain (description?,
                                packet-filter-rule+)>
<!ATTLIST packet-filter-chain
      policy (accept|drop|reject|log) "drop"
      %tgt.att;>

<!ELEMENT packet-filter-rule (description?, packet-match-list,
                                packet-action-list)>

<!ELEMENT packet-match-list (match-interface?, match-mac?,
                                match-ipv4?, match-ipv6?,
                                match-tcp?, match-udp?,
                                match-icmp?)>

<!ELEMENT packet-action-list (log-action?,noop-action?,
                                (accept-action|drop-action|
                                reject-action|gosection?))>

<!ELEMENT match-interface EMPTY>
<!ATTLIST match-interface
      if IDREF #REQUIRED
      %match.att;>

<!ELEMENT match-mac EMPTY>

```

```

<!ATTLIST match-mac
    src CDATA #REQUIRED
    %match.att;>

<!ELEMENT match-ipv4 (match-source?, match-destination?,
    match-dsfield?, match-ecnfield?)>
<!ATTLIST match-ipv4
    fragment (all|first|subseq) "all">

<!ELEMENT match-source EMPTY>
<!ATTLIST match-source
    list IDREF #REQUIRED
    %match.att;>

<!ELEMENT match-destination EMPTY>
<!ATTLIST match-destination
    list IDREF #REQUIRED
    %match.att;>

<!ELEMENT match-dsfield EMPTY>
<!ATTLIST match-dsfield
    dscp CDATA #REQUIRED
    %match.att;>

<!ELEMENT match-ecnfield EMPTY>
<!ATTLIST match-ecnfield
    ect (on|off|dontcare) "dontcare"
    ce (on|off|dontcare) "dontcare"
    %match.att;>

<!ELEMENT match-ipv6 (match-source?, match-destination?,
    match-dsfield?, match-ecnfield?,
    match-flowlabel?)>

<!ELEMENT match-flowlabel EMPTY>
<!ATTLIST match-flowlabel
    value CDATA #REQUIRED
    %match.att;>

<!ELEMENT match-tcp (match-source-port-range?,
    match-destination-port-range?,
    match-tcp-flags?, match-tcp-ecn?,
    match-tcp-option*)>

```

```

<!ELEMENT match-source-port-range EMPTY>
<!ATTLIST match-source-port-range
    lo CDATA #REQUIRED
    hi CDATA #REQUIRED
    %match.att;>

<!ELEMENT match-destination-port-range EMPTY>
<!ATTLIST match-destination-port-range
    lo CDATA #REQUIRED
    hi CDATA #REQUIRED
    %match.att;>

<!ELEMENT match-tcp-flags EMPTY>
<!ATTLIST match-tcp-flags
    syn (on|off|dontcare) "dontcare"
    ack (on|off|dontcare) "dontcare"
    fin (on|off|dontcare) "dontcare"
    rst (on|off|dontcare) "dontcare"
    urg (on|off|dontcare) "dontcare"
    psh (on|off|dontcare) "dontcare">

<!ELEMENT match-tcp-ecn EMPTY>
<!ATTLIST match-tcp-ecn
    cwr (on|off|dontcare) "dontcare"
    ece (on|off|dontcare) "dontcare">

<!ELEMENT match-tcp-option EMPTY>
<!ATTLIST match-tcp-option
    kind CDATA #REQUIRED
    %match.att;>

<!ELEMENT match-udp (match-source-port-range?,
                    match-destination-port-range?)>

<!ELEMENT match-icmp EMPTY>
<!ATTLIST match-icmp
    type CDATA #REQUIRED
    code CDATA #IMPLIED>

<!ELEMENT accept-action EMPTY>
<!ATTLIST accept-action
    %action.att;>

```

```

<!ELEMENT drop-action EMPTY>
<!ATTLIST drop-action
    %action.att;>

<!ELEMENT reject-action EMPTY>
<!ATTLIST reject-action
    type CDATA #IMPLIED
    code CDATA #IMPLIED
    %action.att;>

<!ELEMENT log-action EMPTY>
<!ATTLIST log-action
    level (debug|info|notice|warning|
        err|crit|alert|emerg) "info"
    %action.att;>

<!ELEMENT gosub-action EMPTY>
<!ATTLIST gosub-action
    goto IDREF #REQUIRED
    %action.att;>

<!ELEMENT noop-action EMPTY>
<!ATTLIST noop-action
    %action.att;>

<!-- Routing -->
<!ELEMENT routing (route-filters?, static-routes?, rip?, ripng?)>

<!-- Route filters -->

<!ELEMENT route-filters (route-filter-chain*)>

<!ELEMENT route-filter-chain (description?, route-filter-rule+)>
<!ATTLIST route-filter-chain
    %tgt.att;>

<!ELEMENT route-filter-rule (description?, route-match-list,
    route-action-list)>

<!ELEMENT route-match-list (match-destination?, match-nexthop?,
    match-metric?, match-interface?,
    match-route-source?,

```

```

                                match-route-type?))>

<!ELEMENT route-action-list (set-nexthop?, set-metric?,
                                accept-action?, drop-action?,
                                log-action?, gosub-action?,
                                noop-action?))>

<!ELEMENT match-nexthop EMPTY>
<!ATTLIST match-nexthop
    list IDREF #REQUIRED
    %match.att;>

<!ELEMENT match-metric EMPTY>
<!ATTLIST match-metric
    metric CDATA #REQUIRED
    %match.att;>

<!ELEMENT match-route-source EMPTY>
<!ATTLIST match-route-source
    source (static|connected|rip|ripng|ospf|ospf3|isis|bgp) #REQUIRED
    %match.att;>

<!ELEMENT match-route-type EMPTY>
<!ATTLIST match-route-type
    type (normal|local|blackhole|unreachable|prohibit) #REQUIRED
    %match.att;>

<!ELEMENT set-nexthop (nexthop+)>

<!ELEMENT nexthop (description?)>
<!ATTLIST nexthop
    via CDATA #IMPLIED
    device IDREF #IMPLIED
    weight CDATA #IMPLIED>

<!ELEMENT set-metric EMPTY>
<!ATTLIST set-metric
    metric CDATA #REQUIRED>

<!ELEMENT route-import EMPTY>
<!ATTLIST route-import
    chain IDREF #REQUIRED>

```

```

<!ELEMENT route-export EMPTY>
<!ATTLIST route-export
    chain IDREF #REQUIRED>

<!-- Static routes -->
<!ELEMENT static-routes (route-import?, route+)>

<!ELEMENT route (description?, destination, nexthop+)>
<!ATTLIST route
    %af.att;
    type (normal|local|reject|prohibit|blackhole) "normal"
    metric CDATA #IMPLIED
    preference CDATA #IMPLIED
    scope (link|site|global) #IMPLIED
    zone-index CDATA #IMPLIED>

<!ELEMENT destination EMPTY>
<!ATTLIST destination
    %prefix.att;>

<!ELEMENT rip (route-import?, route-export?,
    rip-interface+, rip-neighbor*)>
<!ATTLIST rip
    disable (yes|no) "no"
    version (1|2|12in1out) "12in1out"
    broadcast (yes|no) #IMPLIED
    check-zero (yes|no) "yes"
    preference CDATA #IMPLIED
    default-metric CDATA #IMPLIED>

<!ELEMENT rip-neighbor EMPTY>
<!ATTLIST rip-neighbor
    address CDATA #REQUIRED>

<!ELEMENT rip-interface (rip-authentication?)>
<!ATTLIST rip-interface
    device IDREF #REQUIRED
    receive (yes|no) "yes"
    send (yes|no) "yes"
    in-version (1|2|both) "both"
    out-version (1|2) "2"
    split-horizon (yes|no) "yes"
    metric-in CDATA #IMPLIED

```

```

metric-out CDATA #IMPLIED>

<!ELEMENT rip-authentication EMPTY>
<!ATTLIST rip-authentication
    type (simple|md5) "simple"
    password CDATA #REQUIRED>

<!ELEMENT ripng (route-import?, route-export?, ripng-interface?)>
<!ATTLIST ripng
    disable (yes|no) "no"
    preference CDATA #IMPLIED
    default-metric CDATA #IMPLIED>

<!ELEMENT ripng-interface EMPTY>
<!ATTLIST ripng-interface
    device IDREF #REQUIRED
    receive (yes|no) "yes"
    send (yes|no) "yes">

```

References

- [xs-d] Biron P.V., Malhotra A. (Ed.): *XML Schema Part 2: Datatypes*. W3C Recommendation 2 May 2001. <http://www.w3.org/TR/xmlschema-2/>
- [xml] Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. (Ed.): *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C Recommendation 6 October 2000. <http://www.w3.org/TR/REC-xml>
- [xslt] Clark, J. (Ed.): *XSL Transformations (XSLT) Version 1.0*. W3C Recommendation 16 November 1999. <http://www.w3.org/TR/xslt>
- [relax] Clark, J. (Ed.): *RELAX NG Specification*. OASIS Committee Specification 3 December 2001. <http://www.oasis-open.org/committees/relaxng/spec-20011203.html>
- [RFC2463] Conta A., Deering S.: *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. RFC 2463, IETF, December 1998.
- [Deer2002] Deering S., Haberman B., Jinmei T., Nordmark E., Onoe A., Zill B.: *IPv6 Scoped Address Architecture*, draft-ietf-ipngwg-scoping-arch-04.txt, IETF, June 2002.

- [xlink] DeRose S., Maler E., Orchard D.: *XML Linking Language (XLink) Version 1.0*. W3C Recommendation 27 June 2001. <http://www.w3.org/TR/2001/REC-xlink-20010627/>
- [RFC2784] Farinacci D., Li T., Hanks S., Meyer D., Traina P. *Generic Routing Encapsulation (GRE)*. RFC 2784, IETF, March 2000.
- [RFC2827] Ferguson P., Senie, D.: *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*. RFC 2827, IETF, January 1998.
- [RFC2893] Gilligan R., Nordmark E.: *Transition Mechanisms for IPv6 Hosts and Routers*. RFC 2893, IETF, August 2000.
- [RFC3260] Grossman D.: *New Terminology and Clarifications for Diffserv*. RFC 3260, IETF, April 2002.
- [RFC1058] Hedrick C.L.: *Routing Information Protocol*. RFC 1058, IETF, June 1988.
- [RFC2373] Hinden R., Deering S.: *IP Version 6 Addressing Architecture*. RFC 2373, IETF, July 1998.
- [RFC2543] Malkin G.: *RIP version 2*. RFC 2453, IETF, November 1998.
- [RFC0792] Postel J.: *Internet Control Message Protocol*. RFC 792, IETF, September 1981.
- [RFC3168] Ramakrishnan K., Floyd S., Black D. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168, IETF, September 2001.
- [Rus2002] Russel R. *Linux 2.4 Packet Filtering HOWTO*, revision 1.26, January 2002. <http://www.netfilter.org/documentation/>
- [xs-s] Thompson H.S., Beech D., Maloney M., Mendelsohn N. (Ed.): *XML Schema Part 1: Structures*. W3C Recommendation 2 May 2001. <http://www.w3.org/TR/xmlschema-1/>
- [xpath] Clark J., DeRose S.: *XML Path Language (XPath) Version 1.0*. W3C Recommendation 16 November 1999. <http://www.w3.org/TR/xpath>