Network Working Group
Request for Comments: 2723
Category: Informational

N. Brownlee
The University of Auckland

October 1999

SRL: A Language for Describing Traffic Flows and Specifying Actions for Flow Groups

Status of this Memo

Copyright Notice

Abstract

This document describes a language for specifying rulesets, i.e. configuration files which may be loaded into a traffic flow meter so as to specify which traffic flows are measured by the meter, and the information it will store for each flow.

Table of Contents

1 Purpose and Scope

A ruleset for an RTFM Meter is a sequence of instructions to be executed by the meter's Pattern Matching Engine (PME). The form of these instructions is described in detail in the 'RTFM Architecture' and 'RTFM Meter MIB' documents [RTFM-ARC, RTFM-MIB], but most users - at least initially - find them confusing and difficult to write, mainly because the effect of each instruction is strongly dependent on the state of the meter's Packet Matching Engine at the moment of its execution.

SRL (the Simple Ruleset Language) is a procedural language for creating RTFM rulesets. It has been designed to be simple for people to understand, using statements which help to clarify the execution context in which they operate. SRL programs will be compiled into rulesets which can then be downloaded to RTFM meters.

An SRL compiler is available as part of NeTraMet (a free-software implementation of the RTFM meter and manager), version 4.2 [NETRAMET].

1.1 RTFM Meters and Traffic Flows

The RTFM Architecture [RTFM-ARC] defines a set of 'attributes' which apply to network traffic. Among the attributes are 'address attributes,' such as PeerType, PeerAddress, TransType and TransAddress, which have meaning for many protocols, e.g. for IPv4

traffic (PeerType == 1) PeerAddress is an IP address, TransType is TCP(6), UDP(17), ICMP(1), etc., and TransAddress is usually an IP port number.

An 'RTFM Traffic Flow' is simply a stream of packets observed by a meter as they pass across a network between two end points (or to/from a single end point). Each 'end point' of a flow is specified by the set of values of its address attributes.

An 'RTFM Meter' is a measuring device - e.g. a program running on a Unix or PC host - which observes passing packets and builds 'Flow Data Records' for the flows of interest.

RTFM traffic flows have another important property - they are bi-directional. This means that each flow data record in the meter has two sets of counters, one for packets travelling from source to destination, the other for returning packets. Within the RTFM architecture such counters appear as further attributes of the flow.

An RTFM meter must be configured by the user, which means creating a 'Ruleset' so as to specify which flows are to be measured, and how much information (i.e. which attributes) should be stored for each of them. A ruleset is effectively a program for a minimal virtual machine, the 'Packet Matching Engine (PME),' which is described in detail in [RTFM-ARC]. An RTFM meter may run multiple rule sets, with every passing packet being processed by each of the rulesets. The rule 'actions' in this document are described as though only a single ruleset were running.

In the past creating a ruleset has meant writing machine code for the PME, which has proved rather difficult to do. SRL provides a high-level language which should enable users to create effective rulesets without having to understand the details of the PME.

The language may be useful in other applications, being suitable for any application area which involves selecting traffic flows from a stream of packets.

1.2 SRL Overview

An SRL program is executed from the beginning for each new packet arriving at the meter. It has two essential goals.

(a) Decide whether the current packet is part of a flow which is of interest and, if necessary, determine its direction (i.e. decide which of its end-points is considered to be its source). Other packets will be ignored.

(b) SAVE whatever information is required to identify the flow and accumulate (COUNT) quantitative information for that flow.

At execution, the meter's Packet Matching Engine (PME) begins by using source and destination attributes as they appear 'on the wire.' If the attributes do not match those of a flow to be recorded, the PME will normally execute the program again, this time with the source and destination addresses interchanged. Because of this bi-directional matching, an RTFM meter is able to build up tables of flows with two sets of counters - one for forward packets, the other for backward packets. The programmer can, if required, suppress the reverse-direction matching and assign 'forward' and 'backward' directions which conform to the conventions of the external context.

Goal (a) is achieved using IF statements which perform comparisons on information from the packet or from SRL variables. Goal (b) is achieved using one or more SAVE statements to store the flow's identification attributes; a COUNT statement then increments the statistical data accumulating for it.

2 SRL Language Description

The SRL language is explained below using 'railway diagrams' to describe the syntax. Flow through a diagram is from left to right. The only exception to this is that lines carrying a left arrow may only be traversed right to left. In the diagrams, keywords are written in capital letters; in practice an SRL compiler must be insensitive to case. Lower-case identifiers are explained in the text, or they refer to another diagram.

The tokens of an SRL program obey the following rules:

- Comments may appear on any line of an SRL program, following a #
- White space is used to separate tokens
- Semicolon is used as the terminator for most statements
- Identifiers (e.g. for defines and labels) must start with a letter
- Identifiers may contain letters, digits and underscores
- The case of letters is not significant
- Reserved words (shown in upper case in this document) may not be used as identifiers

2.1 Define Directive

```
--- DEFINE -- defname ---- = ---- defined_text ----------------- ;
```

Simple parameterless defines are supported via the syntax above. The define name, defname, is an identifier. The defined text starts after the equal sign, and continues up to (but not including) the

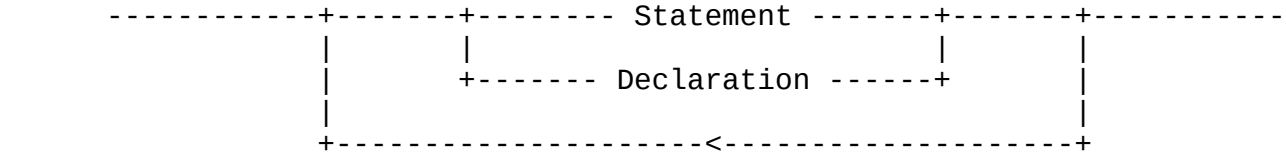
closing semicolon. If a semicolon is required within the defined text it must be preceded by a backslash, i.e. \; in an SRL define produces ; in the text.

Wherever defname appears elsewhere in the program, it will be replaced by the defined text.

For example,

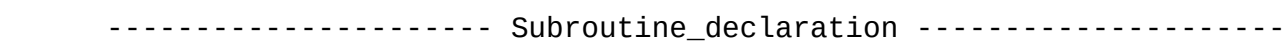
```
DEFINE ftp = (20, 21);  # Well-known Port numbers from [ASG-NBR]
DEFINE telnet = 23;
DEFINE www = 80;
```

2.2 Program



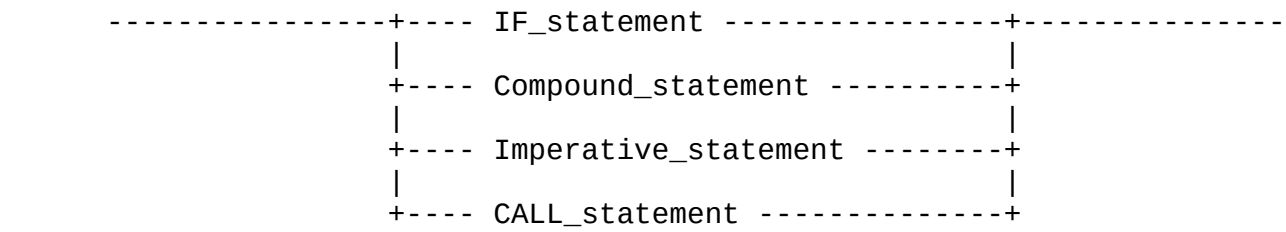
An SRL program is a sequence of statements or declarations. It does not have any special enclosing symbols. Statements and declarations terminate with a semicolon, except for compound statements, which terminate with a right brace.

2.3 Declaration



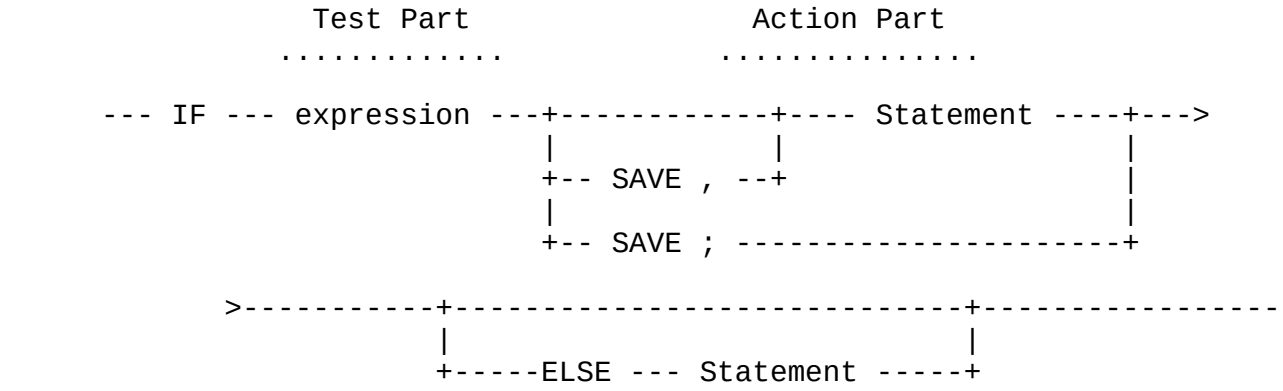
SRL's only explicit declaration is the subroutine declaration. Other implicit declarations are labels (declared where they appear in front of a statement) and subroutine parameters (declared in the subroutine header).

3 Statement

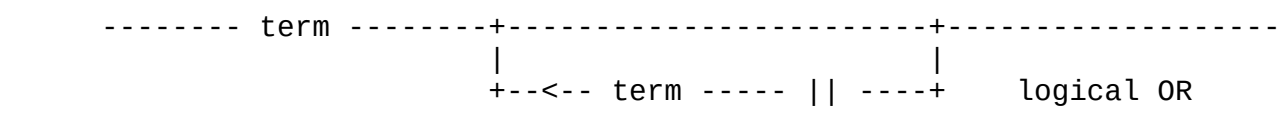


An SRL program is a sequence of SRL statements. There are four kinds of statements, as follows.

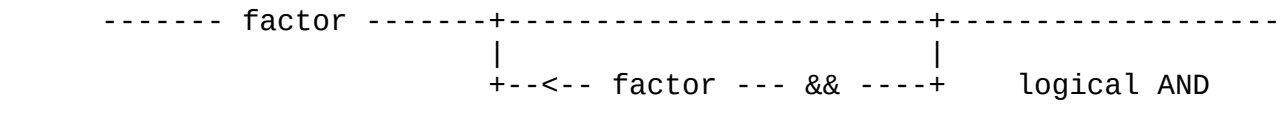
3.1 IF_statement



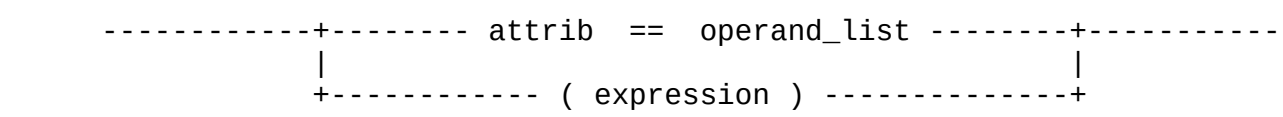
3.1.1 expression



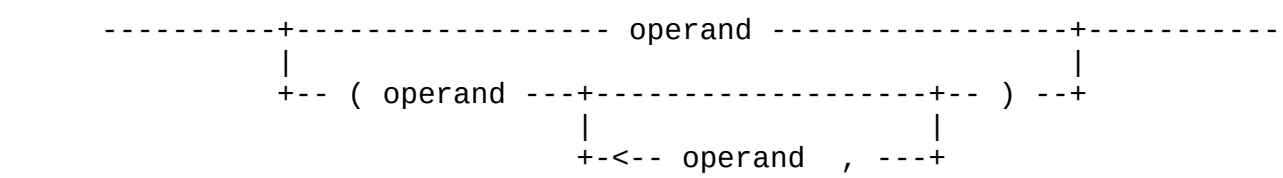
3.1.2 term



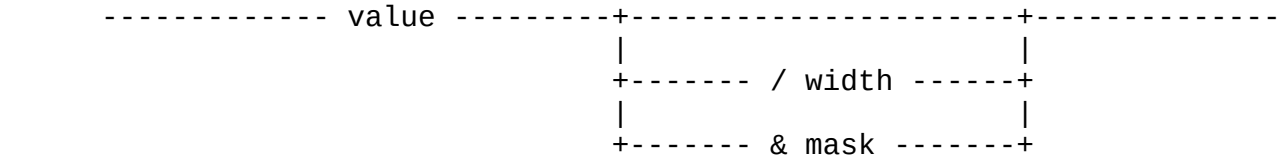
3.1.3 factor



3.1.4 operand_list



3.1.5 operand



3.1.6 Test Part

The IF statement evaluates a logical expression. If the expression value is TRUE, the action indicated in the 'Action Part' of the diagram is executed. If the value is FALSE and the IF has an ELSE clause, that ELSE clause is executed (see below).

The simplest form of expression is a test for equality (== operator); in this an RTFM attribute value (from the packet or from an SRL variable) is ANDed with a mask and compared with a value. A list of RTFM attributes is given in Appendix C. More complicated expressions may be built up using parentheses and the && (logical AND) and || (logical OR) operators.

Operand values may be specified as dotted decimal, hexadecimal or as a character constant (enclosed in apostrophes). The syntax for operand values is given in Appendix B.

Masks may be specified as numbers,

```
        dotted decimal  e.g. &255.255
     or hexadecimal     e.g. &FF-FF
or as a width in bits   e.g. /16
```

If a mask is not specified, an all-ones mask is used.

In SRL a value is always combined with a mask; this combination is referred to as an operand. For example, if we were interested in flows originating from IP network 130.216, we might write:

```
IF SourcePeerAddress == 130.216.0.0 & 255.255.0.0  SAVE;
```

or equivalently

```
IF SourcePeerAddress == 130.216/16  SAVE;
```

A list of values enclosed in parentheses may also be specified; the test succeeds if the masked attribute equals any of the values in the list. For example:

```
IF SourcePeerAddress == ( 130.216.7/24, 130.216.34/24 ) SAVE;
```

As this last example indicates, values are right-padded with zeroes, i.e. the given numbers specify the leading bytes of masks and values.

The operand values and masks used in an IF statement must be consistent with the attribute being tested. For example, a four-byte value is acceptable as a peer address, but would not be accepted as a transport address (which may not be longer than two bytes).

3.1.7 Action Part

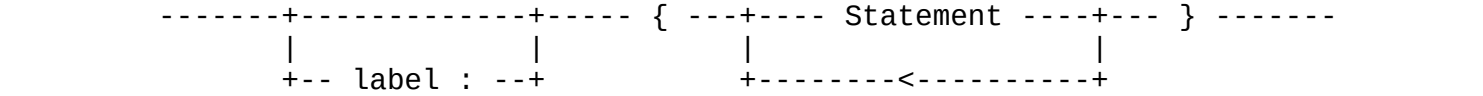
A SAVE action (i.e. SAVE , or SAVE ;) saves attribute(s), mask(s) and value(s) as given in the statement. If the IF expression tests more than one attribute, the masks and values are saved for all the matched attributes. For each value_list in the statement the value saved is the one which the packet actually matched. See below for further description of SAVE statements.

Other actions are described in detail under "Imperative statements" below. Note that the RETURN action is valid only within subroutines.

3.1.8 ELSE Clause

An ELSE Clause provides a statement which will be executed if the IF's test fails. The statement following ELSE will often be another IF statement, providing SRL's version of a 'select' statement. Note that an ELSE clause always matches the immediately preceding IF.

3.2 Compound_statement

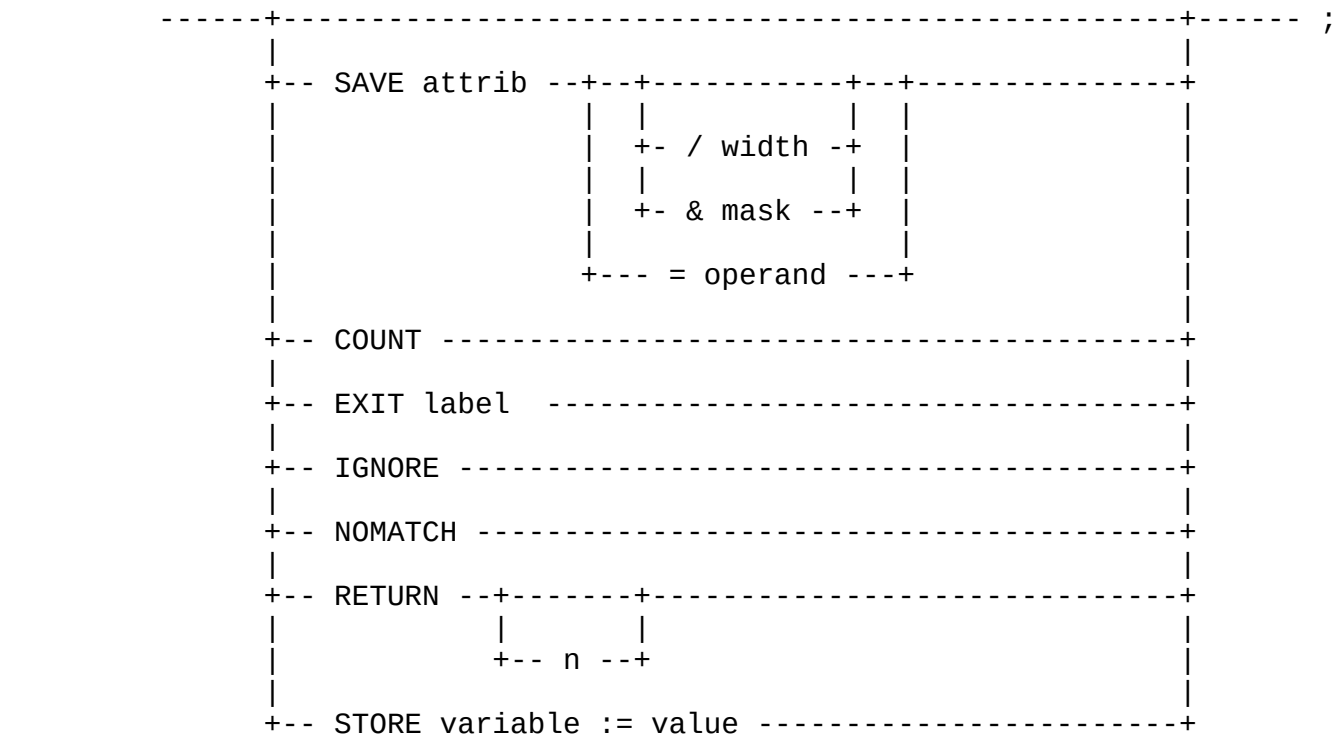


A compound statement is a sequence of statements enclosed in braces. Each statement will terminate with a semicolon, unless it is another compound statement (which terminates with a right brace).

A compound statement may be labelled, i.e. preceded by an identifier followed by a semi-colon. Each statement inside the braces is executed in sequence unless an EXIT statement is performed, as explained below.

Labels have a well-defined scope, within which they must be unique. Labels within a subroutine (i.e. between a SUBROUTINE and its matching ENDSUB) are local to that subroutine and are not visible outside it. Labels outside subroutines are part of a program's outer block.

3.3 Imperative_statement



3.3.1 SAVE Statement

The SAVE statement saves information which will (later) identify the flow in the meter's flow table. It does not actually record anything in the table; this is done when a subsequent COUNT statement executes.

SAVE has two possible forms:

SAVE attrib = operand ; saves the attribute, mask and value as given in the statement. This form of the SAVE statement is similar to that allowed in an IF statement, except that - since imperative statements do not perform a test - you may save an arbitrary value.

SAVE attrib ;
SAVE attrib / width ;
SAVE attrib & mask ; saves the attribute and mask from the statement, and the value resulting from their application to the current packet. This is most useful when used to save a value with a wider mask than than was used to select the packet. For example:

```
IF DestPeerAddress == 130.216/16
        NOMATCH;
ELSE IF SourcePeerAddress == 130.216/16 {
        SAVE SourcePeerAddress /24;
        COUNT;
        }
ELSE IGNORE;
```

3.3.2 COUNT Statement

The COUNT statement appears after all testing and saving is complete; it instructs the PME to build the flow identifier from the attributes which have been SAVEd, find it in the meter's flow table (creating a new entry if this is the first packet observed for the flow), and increment its counters. The meter then moves on to examine the next incoming packet.

3.3.3 EXIT Statement

The EXIT statement exits a labelled compound statement. The next statement to be executed will be the one following that compound statement. This provides a well-defined way to jump to a clearly identified point in a program. For example:

```
outer: {
   ...
   if SourcePeerAddress == 192.168/16
      exit outer;  # exits the statement labelled 'outer'
   ...
   }
# execution resumes here
```

In practice the language provides sufficient logical structure that one seldom - if ever - needs to use the EXIT statement.

3.3.4 IGNORE Statement

The IGNORE statement terminates examination of the current packet without saving any information from it. The meter then moves on to examine the next incoming packet, beginning again at the first statement of its program.

3.3.5 NOMATCH Statement

The NOMATCH statement indicates that matching has failed for this execution of the program. If it is executed when a packet is being processed with its addresses in 'on the wire' order, the PME will

perform the program again from the beginning with source and destination addresses interchanged. If it is executed following such an interchange, the packet will be IGNOREd.

NOMATCH is illustrated in the SAVE example (section 3.3.1), where it is used to ensure that flows having 130.216/16 as an end-point are counted as though 130.216 had been those flows' source peer (IP) address.

3.3.6 STORE Statement

The STORE statement assigns a value to an SRL variable and SAVEs it. There are six SRL variables:

SourceClass	SourceKind
DestClass	DestKind
FlowClass	FlowKind

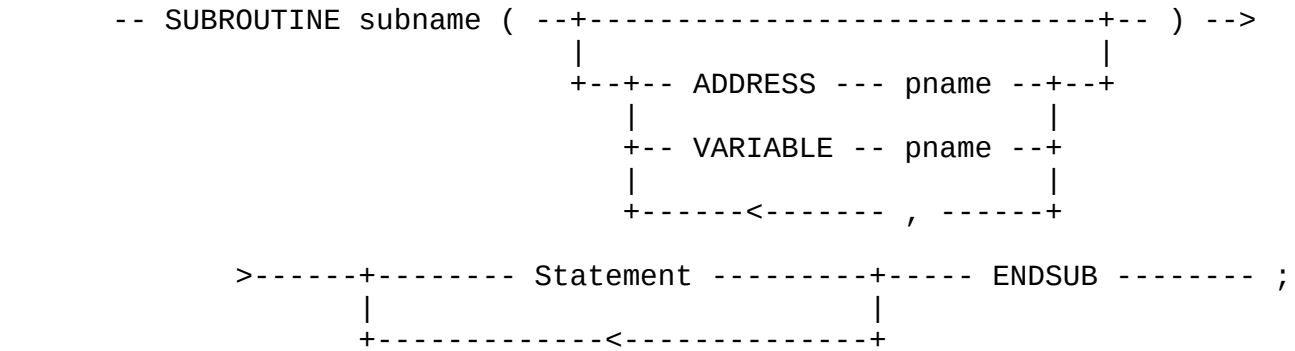
Their names have no particular significance; they were arbitrarily chosen as likely RTFM attributes but can be used to store any single-byte integer values. Their values are set to zero each time examination of a new packet begins. For example:

```
STORE SourceClass := 3;
STORE FlowKind := 'W'
```

3.3.7 RETURN Statement

The RETURN statement is used to return from subroutines and can be used only within the context of a subroutine. It is described in detail below (CALL statement).

3.4 Subroutine_declaration



A Subroutine declaration has three parts:

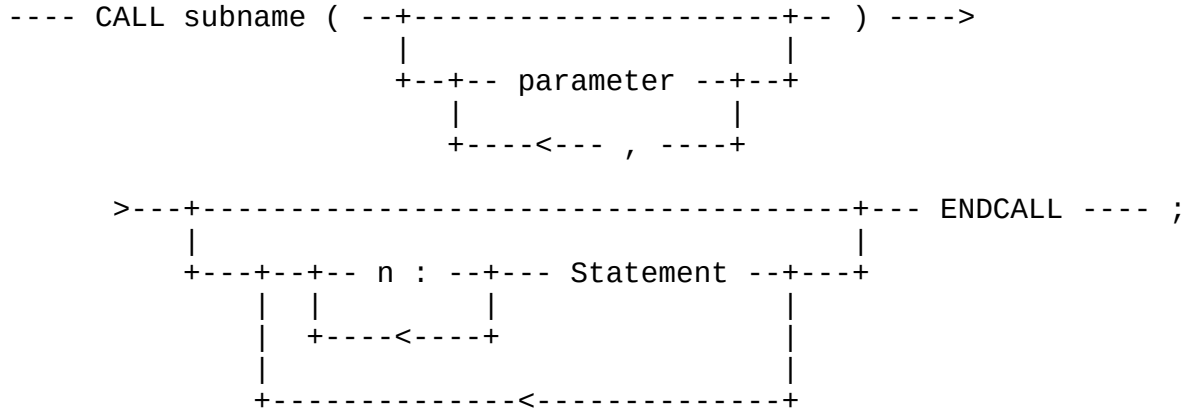
the subname is an identifier, used to name the subroutine.

the parameter list specifies the subroutine's parameters. Each parameter is preceded with a keyword indicating its type - VARIABLE indicates an SRL variable (see the STORE statement above), ADDRESS indicates any other RTFM attribute. A parameter name may be any identifier, and its scope is limited to the subroutine's body.

the body specifies what processing the subroutine will perform. This is simply a sequence of Statements, terminated by the ENDSUB keyword.

Note that EXITs in a subroutine may not refer to labels outside it. The only way to leave a subroutine is via a RETURN statement.

3.5 CALL_statement



The CALL statement invokes an SRL subroutine. The parameters are SRL variables or other RTFM attributes, and their types must match those in the subroutine declaration. Following the parameters is a sequence of statements, each preceded by an integer label. These labels will normally be 1:, 2:, 3:, etc, but they do not have to be contiguous, nor in any particular order. They are referred to in RETURN statements within the subroutine body.

e.g. RETURN 2; would return to the statement labelled 2: within in the CALL statement.

Execution of the labelled statement completes the CALL.

If the return statement does not specify a return label, the first statement executed after RETURN will be the statement immediately following ENDCALL.

4 Example Programs

4.1 Classify IP Port Numbers

```
#
#  Classify IP port numbers
#
   define IPv4 = 1;  # Address Family number from [ASG-NBR]
#
   define ftp = (20, 21);  # Well-Known Port numbers from [ASG-NBR]
   define telnet = 23;
   define www = 80;
#
   define tcp = 6;  # Protocol numbers from [ASG-NBR]
   define udp = 17;
#
   if SourcePeerType == IPv4 save;
   else ignore;  # Not an IPv4 packet
#
   if (SourceTransType == tcp || SourceTransType == udp) save, {
      if SourceTransAddress == (www, ftp, telnet)  nomatch;
         # We want the well-known port as Dest
#
      if DestTransAddress == telnet
         save, store FlowKind := 'T';
      else if DestTransAddress == www
         save, store FlowKind := 'W';
      else if DestTransAddress == ftp
         save, store FlowKind := 'F';
      else {
         save DestTransAddress;
         store FlowKind := '?';
         }
      }
   else save SourceTransType = 0;
#
   save SourcePeerAddress /32;
   save DestPeerAddress   /32;
   count;
#
```

This program counts only IP packets, saving SourceTransType (tcp, udp or 0), Source- and DestPeerAddress (32-bit IP addresses) and FlowKind ('W' for www, 'F' for ftp, 'T' for telnet, '?' for unclassified). The program uses a NOMATCH action to specify the packet direction - its resulting flows will have the well-known ports as their destination.

4.2 Classify Traffic into Groups of Networks

```
#
# SRL program to classify traffic into network groups
#
define my_net = 130.216/16;
define k_nets = ( 130.217/16, 130.123/16, 130.195/16,
                  132.181/16, 138.75/16, 139.80/16 );
#
   call net_kind (SourcePeerAddress, SourceKind)
      endcall;
   call net_kind (DestPeerAddress,   DestKind)
      endcall;
   count;
#
   subroutine net_kind (address addr, variable net)
      if addr == my_net save, {
         store net := 10;  return 1;
         }
      else if addr == k_nets save, {
         store net := 20;  return 2;
         }
      save addr/24;  # Not my_net or in k_nets
      store net := 30;  return 3;
      endsub;
#
```

The net_kind subroutine determines whether addr is my network (130.216), one of the Kawaihiko networks (in the k_nets list), or some other network. It saves the network address from addr (16 bits for my_net and the k_net networks, 24 bits for others), stores a value of 10, 20 or 30 in net, and returns to 1:, 2: or 3:. Note that the network numbers used are contained within the two DEFINEs, making them easy to change.

net_kind is called twice, saving Source- and DestPeerAddress and Source- and DestKind; the COUNT statement produces flows identified by these four RTFM attributes, with no particular source-dest ordering.

In the program no use is made of return numbers and they could have been omitted. However, we might wish to re-use the subroutine in another program doing different things for different return numbers, as in the version below.

```
call net_kind (DestPeerAddress, DestKind)
   1: nomatch;  # We want my_net as source
      endcall;
call net_kind (SourcePeerAddress, SourceKind)
   1: count;    # my_net -> other networks
      endcall;
save SourcePeerAddress /24;
save DestPeerAddress /24;
count;
```

This version uses a NOMATCH statement to ensure that its resulting flows have my_net as their source. The NOMATCH also rejects my_net -> my_net traffic. Traffic which doesn't have my_net as source or destination saves 24 bits of its peer addresses (the subroutine might only have saved 16) before counting such an unusual flow.

5 Security Considerations

SRL is a language for creating rulesets (i.e. configuration files) for RTFM Traffic Meters - it does not present any security issues in itself.

On the other hand, flow data gathered using such rulesets may well be valuable. It is therefore important to take proper precautions to ensure that access to the meter and its data is secure. Ways to achieve this are discussed in detail in the Architecture and Meter MIB documents [RTFM-ARC, RTFM-MIB].

6 IANA Considerations

Appendix C below lists the RTFM attributes by name. Since SRL only refers to attributes by name, SRL users do not have to know the attribute numbers.

The size (in bytes) of the various attribute values is also listed in Appendix C. These sizes reflect the object sizes for the attribute values as they are stored in the RTFM Meter MIB [RTFM-MIB].

IANA considerations for allocating new attributes are discussed in detail in the RTFM Architecture document [RTFM-ARC].

7 APPENDICES

7.1 Appendix A: SRL Syntax in BNF

```
      <SRL program>    ::=  <S or D> | <SRL program> <S or D>

      <S or D>         ::=  <statement> | <declaration>

      <declaration>    ::=  <Subroutine declaration>

      <statement>      ::=  <IF statement> |
                            <Compound statement> |
                            <Imperative statement> |
                            <CALL statement>

      <IF statement>   ::=  IF <expression> <if action> <opt else>

      <if action>      ::=  SAVE ; |
                            SAVE , <statement> |
                            <statement>

      <opt else>       ::=  <null> |
                            ELSE <statement>

      <expression>     ::=  <term> | <term> || <term>

      <term>           ::=  <factor> | <factor> && <factor>

      <factor>         ::=  <attribute> == <operand list> |
                            ( <expression> )

      <operand list>   ::=  <operand> | ( <actual operand list> )

      <actual operand list> ::= <operand> |
                            <actual operand list> , <operand>

      <operand>        ::=  <value> |
                            <value> / <width> |
                            <value> & <mask>

      <Compound statement> ::= <opt label> { <statement seq> }

      <opt label>      ::=  <null> |
                            <identifier> :

      <statement seq>  ::=  <statement> | <statement seq> <statement>

      <Imperative statement> ::=  ; |
```

```
                        SAVE <attribute> <opt operand> ; |
                        COUNT ; |
                        EXIT <label> ; |
                        IGNORE ; |
                        NOMATCH ; |
                        RETURN <integer> ; |
                        RETURN ; |
                        STORE <variable> := <value> ;

    <opt operand>    ::=  <null> |
                        <width or mask> |
                        = <operand>

    <width or mask>   ::= / <width> | & <mask>

    <Subroutine declaration> ::=
                        SUBROUTINE <sub header> <sub body> ENDSUB ;

    <sub header>     ::=  <subname> ( ) |
                        <subname> ( <sub param list> )

    <sub param list> ::= <sub param> | <sub param list> , <sub param>

    <sub param>      ::=  ADDRESS <pname> | VARIABLE <pname>

    <pname>          ::=  <identifier>

    <sub body>       ::=  <statement sequence>

    <CALL statement> ::=  CALL <call header> <opt call body> ENDCALL ;

    <call header>    ::=  <subname> ( ) |
                        <subname> ( <call param list> )

    <call param list> ::= <call param> |
                        <call param list> , <call param>

    <call param>     ::=  <attribute> | <variable>

    <opt call body>  ::=  <null> |
                        <actual call body>

    <actual call body> ::=  <numbered statement> |
                        <actual call body> <numbered statement>

    <numbered statement> ::= <int label seq> <statement>

    <int label seq>  ::=  <integer> : | <int label seq> <integer> :
```

The following are terminals, recognised by the scanner:

```
<identifier>     Described in section 2
<integer>        A decimal integer

<attribute>      Attribute name, as listed in Appendix C

<value>, <mask>  Described in section 5.2

<width>     ::= <integer>
<label>     ::= <identifier>

<variable>  ::=  SourceClass | DestClass | FlowClass |
                  SourceKind | DestKind | FlowKind
```

7.2 Appendix B: Syntax for Values and Masks

Values and masks consist of sequences of numeric fields, each of one or more bytes. The non-blank character following a field indicates the field width, and whether the number is decimal or hexadecimal. These 'field type' characters may be:

```
.  period      decimal, single byte
-  minus       hex,     single byte
!  exclaim     decimal, two bytes
```

For example, 130.216.0.0 is an IP address (in dotted decimal), and FF-FF-00-00 is an IP address in hexadecimal.

The last field of a value or mask has no field width character. Instead it takes the same width as the preceding field. For example, 1.3.10!50 and 1.3.0.10.0.50 are two different ways to specify the same value.

Unspecified fields (at the right-hand side of a value or mask) are set to zero, i.e. 130.216 is the same as 130.216.0.0.

If only a single field is specified (no field width character), the value given fills the whole field. For example, 23 and 0.23 specify the same value for a SourceTransAddress operand. For variables (which have one-byte values) a C-style character constant may also be used.

IPv6 addresses and masks may also be used, following the conventions set out in the IP Version 6 Addressing Architecture RFC [V6-ADR].

7.3 Appendix C: RTFM Attribute Information

The following attributes may be tested in an IF statement, and their values may be SAVEd (except for MatchingStoD). Their maximum size (in bytes) is shown to the left, and a brief description is given for each. The names given here are reserved words in SRL (they are <attribute> terminals in the grammar given in Appendix A).

Note that this table gives only a very brief summary. The Meter MIB [RTFM-MIB] provides the definitive specification of attributes and their allowed values. The MIB variables which represent flow attributes have 'flowData' prepended to their names to indicate that they belong to the MIB's flowData table.

1 SourceInterface, DestInterface
Interface(s) on which the flow was observed

1 SourceAdjacentType, DestAdjacentType
Indicates the interface type(s), i.e. an ifType from [ASG-NBR], or an Address Family Number (if metering within a tunnel)

0 SourceAdjacentAddress, DestAdjacentAddress
For IEEE 802.x interfaces, the MAC addresses for the flow

1 SourcePeerType, DestPeerType
Peer protocol types, i.e. Address Family Number from [ASG-NBR], such as IPv4, Novell, Ethertalk, ..

0 SourcePeerAddress, DestPeerAddress
Peer Addresses (size varies, e.g. 4 for IPv4, 3 for Ethertalk))

1 SourceTransType, DestTransType
Transport layer type, i.e. Protocol Number from [ASG-NBR] such as tcp(6), udp(17), ospf(89), ..

2 SourceTransAddress, DestTransAddress
Transport layer addresses (e.g. port numbers for TCP and UDP)

1 FlowRuleset
Rule set number for the flow

1 MatchingStoD
Indicates whether the packet is being matched with its addresses in 'wire order.' See [RTFM-ARC] for details.

The following variables may be tested in an IF, and their values may be set by a STORE. They all have one-byte values.

SourceClass, DestClass, FlowClass,
SourceKind, DestKind, FlowKind

The following RTFM attributes are not address attributes - they are measured attributes of a flow. Their values may be read from an RTFM meter. (For example, NeTraMet uses a FORMAT statement to specify which attribute values are to be read from the meter.)

8 ToOctets, FromOctets
Total number of octets seen for each direction of the flow

8 ToPDUs, FromPDUs
Total number of PDUs seen for each direction of the flow

4 FirstTime, LastActiveTime
Time (in centiseconds) that first and last PDUs were seen for the flow

Other attributes will be defined by the RTFM working group from time to time.

8 Acknowledgments

The SRL language is part of the RTFM Working Group's efforts to make the RTFM traffic measurement system easier to use. Initial work on the language was done by Cyndi Mills and Brad Frazee in Boston. SRL was developed in Auckland; it was greatly assisted by detailed discussion with John White and Russell Fulton. Discussion has continued on the RTFM and NeTraMet mailing lists.

9 References

[ASG-NBR] Reynolds, J. and J. Postel, "Assigned Numbers", STD 2, RFC 1700, October 1994.

[NETRAMET] Brownlee, N., NeTraMet home page, http://www.auckland.ac.nz/net/NeTraMet

[RTFM-ARC] Brownlee, N., Mills, C. and G. Ruth, "Traffic Flow Measurement: Architecture", RFC 2722, October 1999.

[RTFM-MIB] Brownlee, N., "Traffic Flow Measurement: Meter MIB", RFC 2720, October 1999.

[V6-ADDR] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture," RFC 2373, July 1998.

10 Author's Address

Nevil Brownlee
Information Technology Systems & Services
The University of Auckland
Private Bag 92-019
Auckland, New Zealand

Phone: +64 9 373 7599 x8941
EMail: n.brownlee@auckland.ac.nz

11 Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.