srl Compiler and Language Users' Guide

Version 4.2

Nevil Brownlee

Information Technology Systems & Services
The University of Auckland
Auckland, New Zealand

August 1998

1. Introduction

1.1. SRL the Simple Ruleset Language

An RTFM meter such as NeTraMet or NetFlowMet must be told which flows are of interest, and what information shoud be saved for each flow. Configuring the meter requires the user to create a 'ruleset,' to be downloaded to the meter by a manager such as *NeMaC* or *nifty*. A ruleset is really a pattern-matching program for the meter's Pattern Matching Engine (PME), described in RFC 2064. A lengthy introduction to rulesets and how to write them appears in RFC 2123. Unfortunately, many users have been discouraged by the apparent difficulties in doing this.

SRL, a language for specifying traffic flows and performing actions upon them, was originally produced so as to provide an easier way to create rulesets. The syntax is documented in an Internet Draft available from the NeTraMet and RTFM home pages.

The following section, documenting an SRL compiler, is copied from the NeTraMet 4.2 Users' Guide.

1.2. *srl* the Ruleset Language Compiler

The program srl is an optimising compiler for SRL, the Simple Ruleset Language. SRL is documented in an Internet Draft, available from the NeTraMet or RTFM home pages. It is used in the same way as other Unix compilers - when the compiler is executed it reads in a source 'srl' file, parses it, and (if it is free from syntax errors) produces an object 'rules' file.

```
    srl [options] source
```

compiles the file 'source', producing a rules file ready to be used by NeMaC. Source files will normally end with .srl and rules files with .rules. For example

```
    srl test-prog.srl
```

produces test-prog.rules.

1.3. Compiler Options

-l	List source program (default only lists statements with errors)	
-s	Syntax check only (don't produce an object file)	
-**ann**	set 'Assembler output' level for object file	
	nn=0	rules in numeric form only. Requires NeMaC v4.2.
	nn=1	attributes and actions given as words. This is the default.
	nn=2	as for nn=1, but don't delete intermediate files.
-Onn	set Optimisation level.	
	nn=0	no optimisation at all.
	nn=1	peephole optimising to delete redundant rules from intermediate files. This is the default.
	nn=2	optimise tests by mask length within expressions (shortest masks first, after allowing for overlapping addresses/masks).
	nn=3	as for nn=2, but optimise expression between if clauses and between statements.
-t	}	Switches for
-v	}	compiler testing

srl extends the language as described in the Internet Draft by adding a number of extra statements:

```
   include fffff ;
```

Will read all the text from source file fffff. Includes may be nested (i.e. an include file may include other files). srl looks for file fffff in the same directory as the source file specified on its command line.

```
   optimise nn ;
   optimise * ;
   optimise ;
```

Allow you to change the optimisation level as required for different parts of your program.

`optimise nn ;`	sets optimising to level nn (described above for the **-O** option).
`optimise ;`	resets the level to the value specified on the command line.
`optimise * ;`	indicates a break between optimised expression groups .

```
   set nn ;
   format aaa .. aaa ;
   statistics ;
```

These three statements are passed on (via the object rules file) to NeMaC. String constants in a format (specifying separators in flow data files) may include C-style constants (including non-ASCII characters introduced with a \).

2. Sample SRL Programs

A collection of SRL programs is provided in the examples/srl directory. Those programs are listed here, with description and commentary. They should provide a good introduction to the language and its capabilities.

2.1. Programs for 'special' packets

2.1.1, broadcast.srl *Watch for broadcast packets*

This program watches for packets with MAC address FF-FF-FF-FF-FF-FF, and saves all their address attributes. It can be used with nm_rc to provide regular lists of broadcast activity on a network.

```
#  broadcast.srl:  Look for Broadcast packets
#

   if SourcePeerType == dummy  # PC meter time-filler packets
      ignore;

   if DestAdjacentAddress == FF-FF-FF-FF-FF-FF {
      save SourcePeerType;
      save SourcePeerAddress/32;
      save DestPeerAddress/32;
      save SourceTransType;
      save SourceTransAddress/16;
      save DestTransAddress/16;
      count;
      }

set 9;
```

The first test is for 'dummy' packets. These are generated (and reported) by the PC versions of the NeTraMet meter as a background activity. The ratio of 'dummy' vs 'normal' packets gives a reasonable estimate of the processor idle time percentage.

The second test looks for a destination MAC address (DestAdjacentAddress) of FF-FF-FF-FF-FF-FF. Packets passing this test cause the compound statement (enclosed in braces) to be executing, saving a list of address attribute values and counting the packet.

This program has only one 'NeMaC' statement, indicating that it is to be run (on a meter) using '9' as its set name. It doesn't have a format statement, since nm_rc will use its built-in default format.

Here is some sample output obtained from nm_rc running this program’s output rules file ..

```
/examples/srl> ../../nm_rc -c30 -r broadcast.rules bluebottle.itss Nevil-32

nm_rc: Remote Console for NeTraMet: V4.2
Using MIB file: /dept/itc/nevil/au-snmp/mib/mib.txt
>> manager 1:  Status=1, Owner=NeTraMet
>> ruleset 1:  Status=1, Owner=NeTraMet, Name=1

#Format: flowruleset flowindex firsttime  sourcepeertype sourcepeeraddress destpeeraddress
 sourcetranstype sourcetransaddress desttransaddress  topdus frompdus  tooctets fromoctets

#---  bluebottle.itss le0  2 flows     0pps   16Bps   17:21:19 Mon 10 Aug 1998  ---
50%  5   6   0s  ip4 130.216.169.115 255.255.255.255  udp  nbio nbio    1    0   243B    0B
50%  5   7   0s  ip4 130.216.96.251  255.255.255.255  udp  nbio nbio    1    0   243B    0B
 0%  bytes in 0 other flows

#---  bluebottle.itss le0  143 flows     7pps   2kBps   17:21:30 Mon 10 Aug 1998  ---
 7%  5  44   8s  ip4 130.216.4.253   255.255.255.255  udp  520 520      7    0    4kB    0B
 6%  5  29  10s  ip4 130.216.97.74   255.255.255.255  udp  nbio nbio   13    0    3kB    0B
 5%  5  26  11s  ip4 130.216.97.201  255.255.255.255  udp  nbio nbio   11    0    3kB    0B
 4%  5  40   9s  ip4 130.216.4.254   130.216.4.255    udp  520 520      4    0    2kB    0B
 3%  5  34  10s  ip4 130.216.61.42   255.255.255.255  udp  nbio nbio    6    0    2kB    0B
 3%  5 103   5s  ip4 130.216.7.14    255.255.255.255  udp  nbio nbio    6    0    1kB    0B
 3%  5  42   8s  ip4 130.216.7.137   255.255.255.255  udp  nbio nbio    6    0    1kB    0B
 2%  5  43   8s  ip4 130.216.97.72   255.255.255.255  udp  nbio nbio    5    0    1kB    0B
 2%  5  27  11s  ip4 130.216.4.10    255.255.255.255  udp  snmp 162     8    0   896B    0B
 1%  5  36   9s  ip4 130.216.97.85   255.255.255.255  udp  nbio nbio    3    0   771B    0B
65%  bytes in 133 other flows
```

2.1.2. pr+bc.srl *Count broadcasts, summarise protocols*

This program watches for broadcast packets too. Non-broadcast packets are also summarised, creating one flow for each PeerType (network protocol) observed on the network.

```
#  pr+bc.srl:  broadcast ruleset, in SRL
#

   if SourcePeerType == dummy ignore;  # PC meter time-filler packets

   if DestAdjacentAddress == 01-00 & 01-00 {
      store FlowClass := 1;
      if SourcePeerType == IP
         save SourcePeerAddress/24;
      else if SourcePeerType == ethertalk
         save SourcePeerAddress/16;
      else save SourcePeerAddress/32;
      save SourcePeerType;
      count;
      }
   else if SourceAdjacentAddress == 01-00 & 01-00
      nomatch;  # Broadcast packet with direction reversed ???
   else {  # Not a broadcast packet
      save SourcePeerType;
      count;
      }

SET 4;  # 'NeMaC' commands
FORMAT
   SourcePeerType "  "  ToPDUs "  " ToOctets "  #  "
   FlowClass "  " SourcePeerAddress;
```

The test used for 'broadcast' packets only looks at the bottom bit of the first AdjacentAddress byte. Note the trailing zeroes in the address and mask for this test - without it the compiler would assume that the value was to be right-justified in the six-bit attribute!

The FlowClass variable is used to indicate whether the flow is a 'broadcast' one. It is set to one by the Store statement for broadcast packets. The SourcePeerAddress os saved for broadcasts, bu the SourcePeerType is used to determine how many bits of the address should be saved; 24 for IP addresses, 16 for Ethertalk and 32 for any other protocols.

The ruleset will be run using set name '4,' and a format statement is used to tell NeMaC which attribute values should be read and saved in flow data files. String constants (delimited by quote marks) are used to separate groups of attribute values.

2.1.3. icmp.srl *Watch ICMP packets*

This program watches for ICMP packets, and sets the FlowKind variable to a letter indicating the ICMP packet type. FlowKind values are used by *nifty* as plotting symbols for the flows with the most traffic.

```
#  icmp.srl:  Look at ICMP packets
#

   if SourcePeerType == IP && SourceTransType == ICMP save, {

      if SourceTransAddress == 0        # Echo reply
         store FlowKind := 'e';
      else if SourceTransAddress ==  3  # Destination unreachable
         store FlowKind := 'U';
      else if SourceTransAddress ==  8  # Echo request
         store FlowKind := 'E';
      else if SourceTransAddress == 11  # Time exceeded
         store FlowKind := 'X';
      else if SourceTransAddress == 13  # Timestamp request
         store FlowKind := 'T';
      else if SourceTransAddress == 14  # Timestamp reply
         store FlowKind := 't';
      else if SourceTransAddress == 17  # Address Mask request
         store FlowKind := 'M';
      else if SourceTransAddress == 18  # Address Mask reply
         store FlowKind := 'm';
      else store FlowKind := '?';

      save SourceTransAddress;  # ICMP type
      save DestTransAddress;    # ICMP dest
      save SourcePeerAddress;
      save DestPeerAddress;
      count;
      }

   else ignore;  # Not an IP packet

set 5;
```

This program uses an expression containing two comparisons ANDed together, allowing it to find IP/ICMP packets with a single IF statement. The compiler groups the code together for the tests in the sequence of else- clauses – this allows the NeTraMet meter to execute the else- tests as a single hashed comparison.

2.1.4. other.srl *Summarise ‘other’ packets*

The NeTraMet meter looks at the Ethernet type and LSAP fields of every packet it sees, so as to determine the packet’s PeerType. If it’s not a PeerType which RTFM knows about, NeTraMet gives it PeerType ‘other,’ and puts its Ethertype and LSAP values into the Source- and DestPeerAddress attribute. This program looks for ‘other’ packets and saves their Ethertypes and LSAPs.

```
#  other.srl:  Look at 'other' packets (this is a NeTraMet
#     implementation feature, not part of the RTFM Architecture)
#

   if SourcePeerType == other  save, {

      # Since IP wasn't tested for, NeTraMet counts
      #    all packets as 'other'

      save SourcePeerAddress;  # Ethertype
      save DestPeerAddress;    # LSAP
      count;
      }

   else ignore;  # Not an IP packet


set 5;
#format
#   SourceKind DestKind FlowKind "  "
#   SourcePeerType SourcePeerAddress DestPeerAddress "  "
#   SourceTransType SourceTransAddress DestTransAddress;
```

(The format statement was commented out because I normally use this program with nm_rc, which provides its own default format).

2.2. Programs for classifying IP ports

2.2.1. cs2.srl *Simple example classifier*

This program is from the SRL Internet Draft, and is a very much simplified version of nifty.srl. It finds IP packets with port numbers we are interested in (www, ftp and telnet), sets FlowKind to a plotting symbol suitable for *nifty,* and saves the Source- and DestPeerAddresses.

```
#  cs2.srl:  Classify IP port numbers  - with compound statements
#

   IF SourcePeerType == IP 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 := '?';
         }
      SAVE SourcePeerAddress /32;
      SAVE DestPeerAddress   /32;
      COUNT;
      }

   IGNORE;  # Not an IP packet

SET  7;  # NeMaC commands
FORMAT
 firsttime "  " topdus frompdus " # " tooctets fromoctets
 sourcepeertype " \" "
 sourcepeeraddress destpeeraddress desttransaddress " \x3F "
 flowkind;  #  Must have the semicolon !
STATISTICS;
```

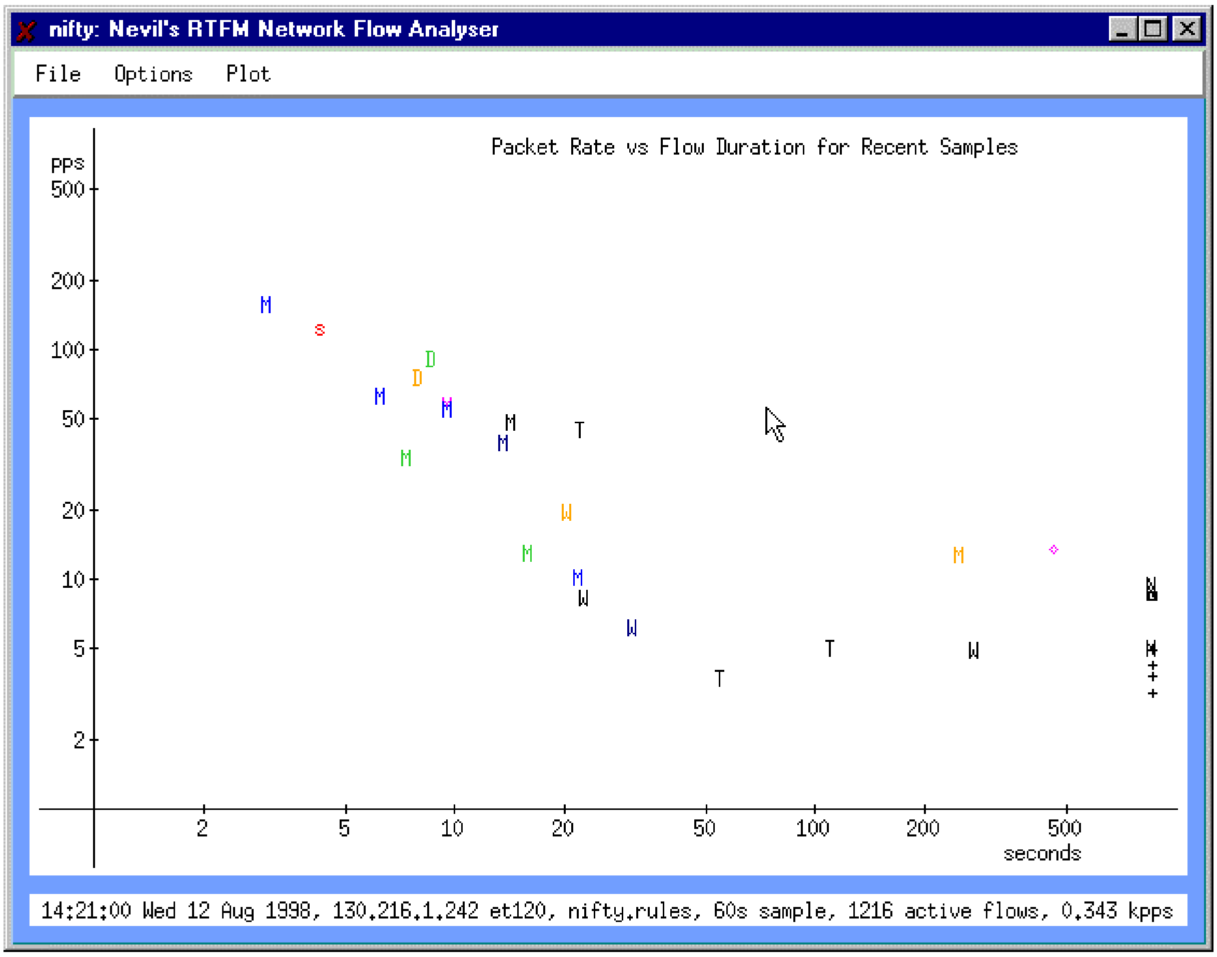
The second if- statement is interesting for two reasons. One, it tests a list of operands (values and masks) in a single comparison. Since no explicit mask is specified, the whole value field is used in the test. Two, it executes a 'nomatch' statement if the comparison succeeds. The effect of this is to ensure that the ports we are interested in always appear as the DestTransAddress of the reported flows, which makes processing the collected flow data a little bit easier.

2.2.2. nifty.srl *Generalised classifier (nifty default)*

This program is an srl version of the original *nifty* ruleset. It is a generalisation of cs2.srl (above).

```
SET  4;  # NeMaC commands
FORMAT FlowRuleSet FlowIndex FirstTime "  "
   SourcePeerType SourcePeerAddress DestPeerAddress "  "
   SourceTransType SourceTransAddress DestTransAddress "  "
   ToPDUs FromPDUs "  " ToOctets FromOctets;

#  nifty ruleset, in SRL
#
#  Nevil Brownlee, ITSS Technology Development, The University of Auckland
#

   if SourcePeerType == IP
      save;  # Fall through to IP handling below
   else if SourcePeerType == other {  # ethertype/LSAP in source/dest peer
      store FlowKind := 3;  # Plot as SQUARE
      save SourcePeerAddress/16;
      save DestPeerAddress/16;
      count;
      }
   else if SourcePeerType == dummy
      ignore;
   else {
      store FlowKind := 3;  # Plot as SQUARE
      count;
      }

   if SourceTransType == (tcp, udp) save, {  # Look at well-known ports

      if SourceTransAddress == (
         domain, ftp, ftpdata, gopher, nntp, ntp, smtp, snmp, telnet, www,
         79, 110, 143, 513, 515,  # finger, pop, imap, login, printer
         137, 138, 139,  # NETBIOS name service, datagram, session
         2049,           # NFS
         1080, 8080,     # UA socks gateway, www proxy
         3128, 3130,     # Squid cache, cache control
         6000            # X-Windows
         ) nomatch;  # We want the well-known port as Dest
      if DestTransAddress == (137, 138, 139)  # NETBIOS
         save, store FlowKind := 'B';
      else if DestTransAddress == 3128  # Squid data
         save, store FlowKind := 'C';
      else if DestTransAddress == 3130  # Squid control
         save, store FlowKind := 'c';
      else if DestTransAddress == domain
         save, store FlowKind := 'D';
      else if DestTransAddress == (ftp, ftpdata)
         save, store FlowKind := 'F';
      else if DestTransAddress == 143  # imap
         save, store FlowKind := 'I';
      else if DestTransAddress == nntp
         save, store FlowKind := 'N';
      else if DestTransAddress == 110  # pop
         save, store FlowKind := 'P';
      else if DestTransAddress == smtp
         save, store FlowKind := 'M';
      else if DestTransAddress == 1080  # UA socks gateway
         save, store FlowKind := 'S';
      else if DestTransAddress == snmp
         save, store FlowKind := 's';
      else if DestTransAddress == telnet
         save, store FlowKind := 'T';
      else if DestTransAddress == (www, 8080)  # UA WWW proxy
         save, store FlowKind := 'W';
      else if DestTransAddress == 6000  # xwin
         save, store FlowKind := 'X';

      else if SourceTransType == udp
         store FlowKind := 2;  # Plot as PLUS
      else if SourceTransType == tcp
         store FlowKind := 1;  # Plot as DIAMOND
      else
         store FlowKind := 3;  # Plot as SQUARE
      }

   else {  # Not tcp or udp
      store FlowKind := 3;  # Plot as SQUARE
      save SourceTransType;
      }

   save SourcePeerAddress/32;
   save DestPeerAddress/32;
   save SourceTransAddress/16;
   save DestTransAddress/16;
   count;
```

The sample plot which follows was produced by *nifty* while running this program on a PC NeTraMet meter. The plot colour indicates how many samples have been collected since there was any activity for this flow (black = currently active, red = idle for 8 samples).


nifty: Nevil's RTFM Network Flow Analyser
File
Options
Plot
Packet Rate vs Flow Duration for Recent Samples
pps
500
200
100
50
20
10
5
2
2
5
10
20
50
100
200
500
seconds

14:21:00 Wed 12 Aug 1998, 130.216.1.242 et120, nifty.rules, 60s sample, 1216 active flows, 0.343 kpps

2.3. Programs which work with lists of IP networks

2.3.1. k1.srl *Simple example, Kawaihiko network*

This program is one of a series which displays flows from the Kawaihiko network.

It uses define statements to specify lists of network addresses (each with a specified address width, e.g. /16 indicates a mask of 255.255.0.0), and these defines are used in if-statements to form expressions testing long lists of networks.

```
#  k1.srl:  Display flows from the Kawaihiko network
#

define Auckland_B =
   130.216/16;
define AIT_B =
   156.62/16;
define Waikato_B =
   130.217/16,
   163.7/16,
   166.83/16;
define Canterbury_B =
   132.181/16,
   153.111/16,
   155.32/16;
 CCNET3define Lincoln_B =
   138.75/16;
define Otago_B =
   139.80/16;
```

```
define Kawaihiko_B =
   Auckland_B, AIT_B, Waikato_B, Canterbury_B, Lincoln_B, Otago_B;


   if SourcePeerType == IP save;
   else ignore;

   if DestPeerAddress == (Kawaihiko_B) nomatch;
         # We want the Kawaihiko site as source

   if SourcePeerAddress == (Auckland_B)
      save, store FlowKind := 'A';
   else if SourcePeerAddress == (AIT_B)
      save, store FlowKind := 'T';
   else if SourcePeerAddress == (Waikato_B)
      save, store FlowKind := 'W';
   else if SourcePeerAddress == (Canterbury_B)
      save, store FlowKind := 'C';
   else if SourcePeerAddress == (Lincoln_B)
      save, store FlowKind := 'L';
   else if SourcePeerAddress == (Otago_B)
      save, store FlowKind := 'O';
   else {
      save SourcePeerAddress;
      store FlowKind := '?';
      }

   save DestPeerAddress;
   count;


set 3;
format
   FlowKind "  "
   SourcePeerType SourcePeerAddress DestPeerAddress "  "
   SourceTransType SourceTransAddress DestTransAddress;
```

2.3.2 k8.srl *Display Kawaihiko network*

This program is a more elaborate one in the series of programs which display flows from the Kawaihiko network.

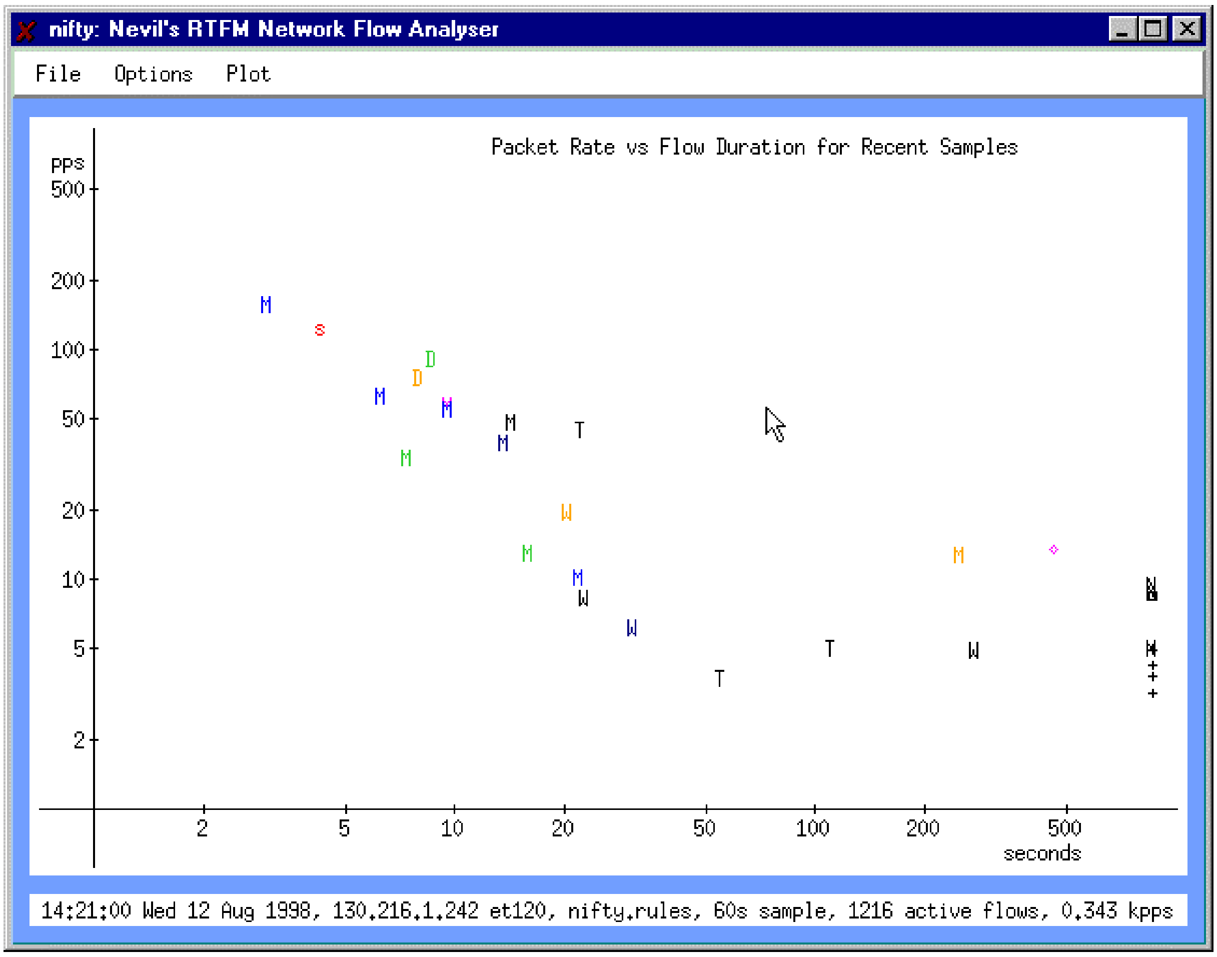
Include statements are used to read in files of define statements. These are similar to the defines in k1.srl (above), except that they contain defines for all the Kawaihiko networks (k-nets.srl) and all the New Zealand networks (nz-nets.srl).

Subroutine net-group() is used to classify a peer address, and to set a variable indicating which of the Kawiahiko networks the peer address belongs to.

The program sets optimise level 3, requesting the compiler to group the comparisons into the largest groups it can manage, and uses an optimise * statement to make sure that network 10.0 is tested on its own, after all the other New Zealand networks.

```
#  k8.srl:  Display flows from the Kawaihiko network
#

include k-nets.srl;   # Kawaihiko nets
include nz-nets.srl;  # New Zealand nets (including Kawaihiko)

optimise 3;  # Over expressions and statements

   if SourcePeerType == IP save;
   else ignore;
   call net_group(SourcePeerAddress, FlowKind)
```

```
      1: {  # Kawaihiko
         call net_group(DestPeerAddress, DestKind)
            1:  # Kawaihiko
               store FlowKind := '*';
            2:  # NZIX
               store FlowKind := 'Z';
            3: {  # Not Kawaihiko
                  if DestPeerAddress == (nz_nets)  # Test the NZ nets
                   store DestKind := '@';
optimise *;  # Force break in cross-statement optimisation
                  else if DestPeerAddress == 10.0/8  # Test 10.0 last
                     store DestKind := '@';
                  else store DestKind := '?';
                  if DestKind == '@' {  # LC letters for Kawaiahiko-NZ
                     if FlowKind == 'A' store FlowKind := 'a';
                     else if FlowKind == 'T' store FlowKind := 't';
                     else if FlowKind == 'W' store FlowKind := 'w';
                     else if FlowKind == 'C' store FlowKind := 'c';
                     else if FlowKind == 'L' store FlowKind := 'l';
                     else if FlowKind == 'O' store FlowKind := 'o';
                     }
                  }
            endcall;
         save SourcePeerAddress;  save DestPeerAddress;
         save SourceTransType;
         save SourceTransAddress;  save DestTransAddress;
         count;
         }
      endcall;

   if MatchingStoD == 1 nomatch;  # We want Kawaihiko as source
   else {
      save SourcePeerAddress;  save DestPeerAddress;
      save SourceTransType;
      save SourceTransAddress;  save DestTransAddress;
      count;
      }


   subroutine net_group(ADDRESS peer, VARIABLE kind)
      if peer == (Auckland_B) save, {  # Test the /16 nets first
         store kind := 'A';  return 1;
         }
      else if peer == (AIT_B) save, {
         store kind := 'T';  return 1;
         }
      else if peer == (Waikato_B) save, {
         store kind := 'W';  return 1;
         }
      else if peer == (Canterbury_B) save, {
         store kind := 'C';  return 1;
         }
      else if peer == (Lincoln_B) save, {
         store kind := 'L';  return 1;
         }
      else if peer == (Otago_B) save, {
         store kind := 'O';  return 1;
         }

      else if peer == (NZIX) save, {  # Test the other Kawaihiko nets
         store kind := 'Z';  return 2;
         }
      else if  # peer == (Auckland) ||
            peer == (Auckland_N) save, {
         store kind := 'A';  return 1;
         }
      else if peer == (Waikato) || peer == (Waikato_N) save, {
         store kind := 'W';  return 1;
         }
      else if peer == (Canterbury) || peer == (Canterbury_N) save, {
         store kind := 'C';  return 1;
         }
      else if peer == (Lincoln) || peer == (Lincoln_N) save, {
         store kind := 'L';  return 1;
         }
      else if peer == (Otago) || peer == (Otago_N) save, {
         store kind := 'O';  return 1;
         }

      else {  # Not a Kawaihiko net
         save peer;
         store kind := '?';  return 3;
         }
      endsub;


set 3;
#format
#   SourceKind DestKind FlowKind "  "
#   SourcePeerType SourcePeerAddress DestPeerAddress "  "
#   SourceTransType SourceTransAddress DestTransAddress;
```


nifty: Nevil's RTFM Network Flow Analyser
File
Options
Plot
Packet Rate vs Flow Duration for Recent Samples
pps
500
200
100
50
20
10
5
2
2
5
10
20
50
100
200
500
seconds

14:21:00 Wed 12 Aug 1998, 130.216.1.242 et120, nifty.rules, 60s sample, 1216 active flows, 0.343 kpps

The above sample plot was produced by *nifty* while running this program on a NetFlowMet meter. Flows from Kawaihiko sites to other New Zealand networks are plotted with lower-case letters (e.g. ‘c’ is Canterbury), from Kawaihiko to networks outside New Zealand in upper-case letters (e.g. ‘W’ is Waikato). The flow plotted as ‘?’ (pointed to by the cursor) is from a non-Kawaihiko site to the Kawaihiko web cache.

2.3.3 2ip8.srl *Classify traffic from/between routers*

This program is an SRL version of /examples/rules.two-ip-groups.

```
#  2ip8.srl: rules.two-ip-groups in SRL
#
#  Nevil Brownlee, ITSS Technology Development, University of Auckland
#
#
#  Rule specification file to tally IP packets in three groups:
#     UA to AIT, UA to elsewhere, AIT to elsewhere
#
#     -------+-------------------+----------------+--------
#            |                   |                |
#       +----+-----+        +----+-----+        +---+---+
#       |   UA     |        |   AIT    |        | meter |
#       +-+-+-+-+--+        +-+-+-+-+--+        +-------+
#
#  The rules work as follows:
#     Non-IP packets are ignored
#     The 'classify' subroutine is used to determine which IP addresses
#        are part of 'UA' and which are part of 'AIT.'
#     ua-to-ait is the 'forward' direction for these flows; ait to ua
#        packets are retried
#

DEFINE ua  = (130.216/16, 202.37.88/24);
DEFINE ait = (156.62/16, 192.73.21/24);

   IF SourcePeerType == IP SAVE ;
   ELSE IGNORE;  # Not IP

   CALL classify(SourcePeerAddress)
   1: {  # from ua
         CALL classify(DestPeerAddress)
         1: IGNORE;    # ua -> ua
         2: COUNT;     # ua -> ait
            ENDCALL;
         COUNT;  # Not ua or ait
         }
   2: {  # from ait
         CALL classify(DestPeerAddress)
         1: NOMATCH;   # ait -> ua
         2: IGNORE;    # ait -> ait
            ENDCALL;
         COUNT;        # ait -> other
         }
      ENDCALL;  # End of outer CALL classify()


   SUBROUTINE classify (ADDRESS peer)
      IF peer == ua {
         SAVE peer/24;  RETURN 1;         # ua
         }
      ELSE IF peer == ait {
         SAVE peer/24;  RETURN 2;      # ait
         }
      SAVE peer/32;              # other
      ENDSUB;
```

```
SET 5;
FORMAT FlowRuleSet FlowIndex FirstTime " "
   SourcePeerType " "
   SourcePeerAddress DestPeerAddress "  "
   ToOctets FromOctets;
STATISTICS ;

# end of file
```

Here is some sample output obtained from nm_rc running this program’s output rules file ..

```
/examples/srl > ../../nm_rc -c30 -r 2ip8.rules bluebottle.itss Nevil-32

nm_rc: Remote Console for NeTraMet: V4.2
Using MIB file: /dept/itc/nevil/au-snmp/mib/mib.txt
>> manager 1:  Status=1, Owner=NeTraMet
>> ruleset 1:  Status=1, Owner=NeTraMet, Name=1

#Format: flowruleset flowindex firsttime sourcepeertype sourcepeeraddress
destpeeraddress  tooctets fromoctets

#---  bluebottle.itss udp-9996  0 flows    0pps   0Bps
         17:26:14 Mon 10 Aug 1998  ---
100%  bytes in 0 other flows

#---  bluebottle.itss udp-9996 120 flows  37pps  18kBps
         17:26:30 Mon 10 Aug 1998  ---
16%  17   2  21s  ip4 130.216.1.0   140.200.128.19    0B  82kB
 8%  17  89  28s  ip4 130.216.4.0   195.116.85.71     0B  45kB
 8%  17  52  18s  ip4 130.216.191.0 207.48.52.225     0B  42kB
 8%  17  12  24s  ip4 130.216.96.0  207.82.250.251    0B  40kB
 7%  17  67  19s  ip4 130.216.92.0  209.83.166.88     0B  38kB
 5%  17  13  21s  ip4 156.62.3.0    207.200.75.4      0B  27kB
 4%  17 104  20s  ip4 156.62.3.0    204.162.96.57     0B  21kB
 4%  17  81  16s  ip4 156.62.3.0    208.10.192.196    0B  21kB
 4%  17  40  44s  ip4 130.216.12.0  210.65.0.33       0B  20kB
 4%  17  60  17s  ip4 130.216.191.0 62.144.115.98     0B  19kB
33%  bytes in 110 other flows
```

3. Author's Address

Please send any comments, suggestions, bug reports to me, Nevil Brownlee, i.e.

n.brownlee@auckland.ac.nz