

IP Fabrics API  
Sample Applications Documentation  
\$Id: README,v 1.4 2005/01/31 21:36:02 mwhite Exp \$

#### COPYRIGHT NOTICE

Information in this document is furnished in connection with IP Fabrics products. No license, express or implied, to any intellectual property rights is granted by this document. The document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license.

Copyright (c) 2005, IP Fabrics, Inc. All rights reserved.

#### BEFORE COMPILING

Make sure that `../lib/libipfsystem.so` is a symlink to the shared library you wish to use. Generally, this can be done by issuing the following commands:

```
% cd ../lib  
% ln -s libipfsystem.so.1.0.0 libipfsystem.so
```

#### COMPILING

XScale binaries are provided for all sample programs. Should recompilation be necessary, a make file has been provided. The make file gives the following meta-targets, in addition to a target for each individual binary:

```
all          - Build all binaries applicable to the selected platform.  
  
install      - Install compiled binaries to the directory specified by TARGET.  
  
clean        - Remove .o and ~ files, but leave compiled binaries.  
  
distclean   - Do a clean and then remove the compiled binaries as well.
```

The make file also specifies a number of variables that control how the binaries are built and where they are installed. For a full list, see the make file, but a partial list includes:

```
ARCH         - The architecture to build for. By default, this is 'xscale',  
              but it can also be set to 'ia32' to compile for Linux on  
              Intel platforms. Note, that PPL specific programs will only  
              be built for xscale.  
  
TARGET       - The location to install the programs when doing a  
              'make install'.  
  
TOOLDIR      - The location of the PPL compiler and friends.
```

To compile and install all applications for the xscale/ppl system, use the following command line:

```
% make distclean all install
```

To compile and install all applications for the intel/linux system, use this command line:

```
% ARCH=ia32 make distclean all install
```

## PPL\_ARRAYIO

Read/write a PPL array. When reading, the array values are printed to standard output. When writing an array, the values to write are read from standard input. All values are hexadecimal digits with whitespace ignored.

### Arguments:

- c The number of bytes to read/write. It is in fact possible to overflow an array, so this argument should almost always be used when writing an array.
- n The name of the array to read/write, as defined in the PPL program.
- o Offset from the beginning of the array to read/write.
- v Verbose output. Includes offsets from the beginning of the array when printing array values.
- w Write to the array instead of reading from it.

### Examples:

```
% ./ppl_arrayio -n packet_counters -c 20
00000001 00000000 00000003 00000001 00000000

% ./ppl_arrayio -n packet_counters -c 20 -w
00000000 00000000 00000000 00000000 00000000^D
% ./ppl_arrayio -n packet_counters -c 20
00000000 00000000 00000000 00000000 00000000
```

## PPL\_LOG

Read either the PPL system or debug log. Log entries are printed to standard output until the maximum number of log entries to read specified by the user is reached or the program is terminated. When the program starts, it attempts to fetch as many log entries as it can from the back buffer and then it sits in a loop polling for new log entries as they arrive. The poll interval can be specified by the user.

### Arguments:

- c Total number of records to read, or -1 for read forever.
- n Number of records to read per polling interval.
- p Polling interval, in milliseconds.
- t Type of log to read. Possible values are: system (default), debug\_header and debug\_packet.

Examples:

```
% ./ppl_log
log: system counter: 0 timestamp: 90070 value: 00000002 lpn: 2
000000: 4501026e 00000000 04069e87 01010101 14000001 00500080 00000000 00000000
000020: 50000000 96cc0000
```

```
log: system counter: 1 timestamp: 455700 value: 00000002 lpn: 2
000000: 4501026e 00000000 04069e87 01010101 14000001 00500080 00000000 00000000
000020: 50000000 96cc0000
```

... edited for brevity ...

```
% ./ppl_log -t debug_header
log: debug_header counter: 0 timestamp: 90070 event data: 80010202
000000: ffffffff ffffdead beefcafe 08004501 026e0000 00000406 9e870101 01011400
000020: 00010050 00800000 00000000 00005000 000096cc 0000ffff ffffffff ffffffff
000040: ffffffff ffffffff ffffffff ffffffff
```

```
log: debug_header counter: 1 timestamp: 90070 event data: 00030202
000000: ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
000020: ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
000040: ffffffff ffffffff ffffffff ffffffff
```

```
log: debug_header counter: 2 timestamp: 455700 event data: 80010302
000000: ffffffff ffffdead beefcafe 08004501 026e0000 00000406 9e870101 01011400
000020: 00010050 00800000 00000000 00005000 000096cc 0000ffff ffffffff ffffffff
000040: ffffffff ffffffff ffffffff ffffffff
```

... edited for brevity ...

```
% ./ppl_log -t debug_packet
log: debug_packet counter: 0 timestamp: 4348656 event data: 80010503
000000: 5b202d6e 203c706f 72743e20 5d207d0a 00000000 00000000 57686572 653a0a00
000020: 20202d68 20202d20 74686973 206d6573 73616765 0a000000 20202d61 20202d20
000040: 616c6c20 74657374 73202873 6b697020 65787465 726e616c 20737973 74656d20
000060: 6c6f6f70 6261636b 20746573 74290a00 20202d66 20202d20 70726573 656e6365
000080: 20746573 740a0000 20202d72 20202d20 72656769 73746572 20616363 65737320
0000a0: 74657374 0a000000 20202d69 20202d20 696e7465 72727570 74207465 73740a00
```

... edited for brevity ...

## PPL\_SENDEVENT

Signal a packetless event on the PPL VM. This will cause the code associated with that event to be executed, though of course doing so on an event that is expecting a packet will generally result in an exception being raised.

### Arguments:

- e The event number to signal. This should be an event defined in the PPL program running on the VM.
- v Verbose output. Causes an acknowledgement of sending to be printed to standard output.

Example:

```
% ./ppl_sendevent -e 4 -v
Event 4 signalled.
% ./ppl_arrayio -n packet_counters -c 20
00000000 00000000 00000000 00000000 00000001
```

## PPL\_SENPKT

Send a packet to a PPL LPN. This will cause the code associated with that LPN to be executed with that packet being used as the packet being operated upon. The packet(s) are read from standard input, separated by double newline characters. Each packet is composed of hexadecimal digits and white space, with the white space being ignored.

Arguments:

- l LPN number to which to send the packet.
- t Outermost header type. Can be: ether or ipv4 (default).
- v Verbose output. Prints an acknowledgement for each packet sent.

Example:

```
% cat test.pkt | ./ppl_sendpkt -l 1 -t ether -v
Packet sent (LPN=1 type=ether len=640)
% ./ppl_arrayio -n packet_counters -c 20
00000000 00000001 00000000 00000000 00000000
```

## PPL\_STAT

Print a set of statistics gathered by each microengine. By default these are interpreted as if the VM were a 'normal' build, but the raw statistics are also available.

Arguments:

- n Number of microengines. Defaults to 16.
- r Display raw output instead of interpreting.
- t Microengine type definition string. Declares what type of code is running on each microengine. This is a comma separated list of at most the value of '-n' elements. Repeating microengine types can be defined by prepending a "nn\*" in front of the microengine type string. Valid types are: ae, be, ce, fwd, null, rx, tx. The default config string is: "ce,be,10\*ae,null,fwd,tx,rx".
- v Verbose output. When raw output is specified, print offset numbers to the left of each row printed.

Examples:

```
% ./ppl_stat
me00 [ce]:
  Pkt Recv: 7  Pkt Sent: 7  Pkt Drop: 0  Pkt Malformed: 0
me01 [be]:
  Pkt Recv: 7  Pkt Sent: 7  Ring Full: 0
```

```

me02 [ae]:
  Early Recv: 1 Late Recv: 1 Late Forward: 0 Late Drop: 0 Exception: 0
me03 [ae]:
  Early Recv: 1 Late Recv: 1 Late Forward: 0 Late Drop: 0 Exception: 0
me04 [ae]:
  Early Recv: 1 Late Recv: 1 Late Forward: 0 Late Drop: 0 Exception: 0
me05 [ae]:
  Early Recv: 1 Late Recv: 1 Late Forward: 0 Late Drop: 0 Exception: 0
me06 [ae]:
  Early Recv: 1 Late Recv: 1 Late Forward: 0 Late Drop: 0 Exception: 0
me07 [ae]:
  Early Recv: 1 Late Recv: 1 Late Forward: 0 Late Drop: 0 Exception: 0
me08 [ae]:
  Early Recv: 1 Late Recv: 1 Late Forward: 1 Late Drop: 0 Exception: 0
me09 [ae]:
  Early Recv: 0 Late Recv: 0 Late Forward: 0 Late Drop: 0 Exception: 0
me10 [ae]:
  Early Recv: 0 Late Recv: 0 Late Forward: 0 Late Drop: 0 Exception: 0
me11 [ae]:
  Early Recv: 0 Late Recv: 0 Late Forward: 0 Late Drop: 0 Exception: 0
me12 [null]:

```

```

me13 [fwd]:
  Pkt Recv: 1 Pkt to TX: 0 Pkt to XScale: 1 Pkt Dropped: 0
me14 [tx]:
  Pkt Recv: 0 Pkt Sent: 0
me15 [rx]:
  Pkt Valid: 0 Pkt Invalid: 0 Pkt Sent: 0 Pkt Dropped: 0

```

```

% ./ppl_stat -r
00000007 00000007 00000000 00000000 00000000 00000000 00000000 013b3ac0
00000007 00000007 00000000 00000000 00000000 00000000 00000000 00000000
00000001 00000000 00000001 00000000 00000000 00000000 00000000 013b3abf
00000001 00000000 00000001 00000000 00000000 00000000 00000000 013b3abe
00000001 00000000 00000001 00000000 00000000 00000000 00000000 013b3abe
00000001 00000000 00000001 00000000 00000000 00000000 00000000 013b3abe
00000001 00000000 00000001 00000000 00000000 00000000 00000000 013b3abd
00000001 00000000 00000001 00000000 00000000 00000000 00000000 013b3abd
00000001 00000000 00000001 00000001 00000000 00000000 00000000 013b3abd
00000000 00000000 00000000 00000000 00000000 00000000 00000000 013b3abd
00000000 00000000 00000000 00000000 00000000 00000000 00000000 013b3abd
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000001 00000000 00000000 00000001 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

```

## PPL\_PKTIO

Cycles a packet that it receives between itself(a XScale user app) and the VM. The number of times the packet is cycled is based on a command line argument. It is recommended that the VM run the pktio\_test.ppl program in order to exercise this sample application.

LIBPCAP library calls are used to access the packets that are sent up to this XScale user application. The packets are sent back to the VM using

the IPF\_SEND\_PACKET API.

Arguments:

- i Interface that the packet arrives on. Defaults to ixdev0.
- n Number of iterations that the packet cycles. Defaults to 32.
- l The lpn value to be used when the packet is sent to the VM
- v Verbose output. Will print the packet in hex.

Examples:

```
% ./ppl_pktio -i ixdev0 -n 10 -l 0 -v  
Warning: arptype 65535 not supported by libpcap - falling back to cooked socket
```

```
Received a packet of length 149  
45100085 69b84000 400626b2 c0a8147d c0a8142b 0017b6ed 6b0888b3 6e0188a0  
801816a0 78030000 0101080a 0001061a 16d46359 5761726e 696e673a 20617270  
74797065 20363535 3335206e 6f742073 7570706f 72746564 20627920 6c696270  
63617020 2d206661 6c6c696e 67206261 636b2074 6f20636f 6f6b6564 20736f63  
6b65740d 0a
```

```
Received a packet of length 112  
ffffffff ffff0000 00000002 08004500 00520000 00000406 e1fd1400 0001c0a8  
00000063 00630000 00000000 00005000 00003491 00000001 02030405 06071122  
33440c0d 0e0f1011 12131415 16171819 1a1b1c1d 1e1f2021 22232425 26272829  
Sent a packet of length 96  
The Return Value = 0 at Iteration = 1
```

```
Received a packet of length 112  
ffffffff ffff0000 00000002 08004500 00520000 00000406 e1fd1400 0001c0a8  
00000063 00630000 00000000 00005000 00003491 00000001 02030405 06071122  
33440c0d 0e0f1011 12131415 16171819 1a1b1c1d 1e1f2021 22232425 26272829  
Sent a packet of length 96  
The Return Value = 0 at Iteration = 2
```

... edited for brevity ...

```
Received a packet of length 112  
ffffffff ffff0000 00000002 08004500 00520000 00000406 e1fd1400 0001c0a8  
00000063 00630000 00000000 00005000 00003491 00000001 02030405 06071122  
33440c0d 0e0f1011 12131415 16171819 1a1b1c1d 1e1f2021 22232425 26272829  
Sent a packet of length 96  
The Return Value = 0 at Iteration = 9
```

```
Received a packet of length 112  
ffffffff ffff0000 00000002 08004500 00520000 00000406 e1fd1400 0001c0a8  
00000063 00630000 00000000 00005000 00003491 00000001 02030405 06071122  
33440c0d 0e0f1011 12131415 16171819 1a1b1c1d 1e1f2021 22232425 26272829  
Sent a packet of length 96  
The Return Value = 0 at Iteration = 10
```

```
Received a packet of length 112  
ffffffff ffff0000 00000002 08004500 00520000 00000406 e1fd1400 0001c0a8  
00000063 00630000 00000000 00005000 00003491 00000001 02030405 06071122  
33440c0d 0e0f1011 12131415 16171819 1a1b1c1d 1e1f2021 22232425 26272829
```

## SENDPKT

Send one or more packets out a Linux Ethernet interface. This program is the Linux equivalent of PPL\_SENDPKT. The packet(s) are read from standard input, separated by double newline characters. Each packet is composed of hexadecimal digits and white space, with the white space being ignored.

### Arguments:

- i The Ethernet interface to transmit packets from, defaults to eth0.
- v Print an acknowledgement for each packet sent.