FFPF: Fairly Fast Packet Filters

Herbert Bos
Vrije Universiteit Amsterdam
herbertb @ cs.vu.nl
http://ffpf.sourceforge.net
Outline

1. What is FFPF?
2. How does it work?
3. How can I use it?
   • hardware support
   • applications
   • examples

Herbert Bos, VU, http://www.cs.vu.nl/~herbertb
Network Monitoring

- Increasingly important
  - traffic characterisation, security
  - traffic engineering, SLAs, billing, etc.

spread of SAPPHIRE in 30 minutes
What is it?

- **single framework for monitoring that**
  - uses all levels in the processing hierarchy
  - is language neutral
  - offers advanced processing in NIC
  - supports stateless and stateful filters
  - is backward compatible with pcap (while supporting a more powerful language)
  - helps users to build complex monitoring applications by ‘clicking components together’

---

Herbert Bos, VU, [http://www.cs.vu.nl/~herbertb](http://www.cs.vu.nl/~herbertb)
reduce copying

- FFPF avoids both ‘horizontal’ and ‘vertical’ copies

- no ‘vertical’ copies
- no ‘horizontal’ copies within flow group
- more than ‘just filtering’ in kernel (e.g., statistics)
• PacketBuf
  – circular buffer with N slots
  – large enough to hold packet

• IndexBuf
  – circular buffer with N slots
  – pointers to packets in PacketBuf
• PacketBuf
  – circular buffer with N slots
  – large enough to hold packet

• IndexBuf
  – circular buffer with N slots
  – pointers to packets in PacketBuf
Buffers

- PacketBuf
  - circular buffer with N slots
  - large enough to hold packet
- IndexBuf
  - circular buffer with N slots
  - pointers to packets in PacketBuf

Herbert Bos, VU, http://www.cs.vu.nl/~herbertb
what to do if writer catches up with slowest reader?

• slow reader preference
  – drop new packets
    (traditional way of dealing with this)
  – overall speed set by slowest reader

• fast reader preference
  – overwrite existing packets
  – application responsible for keeping up
    • check whether packets have been overwritten
    • different drop rates for different apps
How to use it?

we build complex applications out of simple components
Software structure

- **userspace**: FFPF toolkit, MAPI
- **userspace-kernel boundary**: libpcap, FFPF
- **kernelspace**: FFPF
- **host-NIC boundary**: FFPF
- **ixp2xxx**: FFPF

Herbert Bos, VU, [http://www.cs.vu.nl/~herbertb](http://www.cs.vu.nl/~herbertb)
Extensible

modular framework
language agnostic
plug-in filters
Example: GUI for creating flowgraphs

Hovering over nodes will display configuration (more keybindings: see help menu)
simple GUI support
simple GUI support
simple GUI support
FFPF is responsible for mapping the flowgraph on the underlying hardware.
Instantiating flow graphs

**USERSPACE**

- `sampler`
- `fpl1`
- `accept`
- `bytecount`

**KERNEL**

- `localspace`
- `device`
- `buf_dee77000`
- `groupbuf_1`
- `lo`
- `any`
- `eth0`
- `device`
- `pid`
Instantiating flow graphs

KERNEL

USERSPACE

functions available in kernel
Three flavours of packet processing on IXP & host

- **Regular**
  - copy only when needed
  - may be slow depending on access pattern

- **Zero copy**
  - *never* copy
  - may be slow depending on access pattern

- **Copy once**
  - copy always
Example 1: writing applications

```c
1. int main(int argc, char** argv) {
2.   void *opaque_handle;
3.   int count, returnval;
4.   /** pass the string to the subsystem. It initializes and returns an opaque pointer */
5.   opaque_handle = ffpf_open(argv[1], strlen(argv[1]));
6.   if (start_listener(asynchronous_listener, NULL))
7.   {
8.     printf (WARNING,"spawn failed\n");
9.     ffpf_close (opaque_handle); return -1;
10. }
11.   count = ffpf_process(FFPF_PROCESS_FOREVER);
12.   stop_listener(1);
13.   returnval = ffpf_close(opaque_handle);
14.   printf(,"processed %d packets\n",count);
15.   return returnval;
16. }
```

Herbert Bos, VU, http://www.cs.vu.nl/~herbertb
1. `void* asynchronous_listener(void* arg) {`
2.     `int i;`
3.     `while (!should_stop()) {`
4.         `sleep(5);`
5.     `for (i=0; i<export_len; i++) {`
6.         `printf("flowgrabber %d: read %d, written%d\n", i,`
               `get_stats_read(exported[i]),`
               `get_stats_processed(exported[i]));`
7.         `while ((pkt = get_packet_next(exported[i])) {`
8.             `dump_pkt (pkt);`
9.         `}`
10.     `}`
11. `}
12.     `return NULL;`
13. `}`
Example 2: FPL-2

- new pkt processing language: FPL-2
  - for IXP, kernel and userspace
  - simple, efficient and flexible
  - simple example: filter all webtraffic
    \[
    \text{IF} \quad (\text{PKT.IP_PROTO} == \text{PROTO_TCP}) \quad \&\& \quad (\text{PKT.TCP_PORT} == 80) \quad \text{THEN RETURN 1;}
    \]
  
  - more complex example: count pkts in all TCP flows
    \[
    \text{IF} \quad (\text{PKT.IP_PROTO} == \text{PROTO_TCP}) \quad \text{THEN}
    \]
    \[
    \text{R}[0] = \text{Hash}[14, 12, 1024];
    \text{M}[\text{R}[0]]++;
    \]
    \[
    \text{FI}
    \]
FPL-2

- all common arithmetic and bitwise operations
- all common logical ops
- all common integer types
  - for packet
  - for buffer (useful for keeping state!)

- statements
  - Hash
  - External functions
    - to call hardware implementations
    - to call fast C implementations
  - If ... then ... else
  - For ... break; ... ROF
  - Return

Herbert Bos, VU, http://www.cs.vu.nl/~herbertb
Example application: dynamic ports

1. // R[0] stores no. of dynports found (initially 0)
2. IF (PKT.IP_PROTO==PROTO_TCP) THEN
3.   IF (PKT.TCP_DPORT==554) THEN
4.     M[R[0]]=EXTERN("GetDynTCPDPortFromRTSP",0,0);
5.     R[0]++;
6. ELSE
7.   FOR (R[1]=0; R[1] < R[0]; R[1]++)
8.     IF (PKT.TCP_DPORT == M[R[1]]) THEN
9.       RETURN TRUE;
10.  FI
11.  ROF
11.  FI
12.  FI
12. RETURN FALSE;

Herbert Bos, VU, http://www.cs.vu.nl/~herbertb
Summary

- concept of ‘flow’ ➔ generalised
- copying and context switching ➔ minimised
- processing in kernel/NIC ➔ complex apps + ‘pipes’
- FPL: FFPF Packet Languages ➔ fast + flexible
- persistent storage ➔ flow-specific state
- flow groups ➔ applications sharing packet buffers
- commandline/GUIs ➔ easy to use

Herbert Bos, VU, http://www.cs.vu.nl/~herbertb
http://ffpf.sourceforge.net/

**Introduction**

The fairly fast packet filter (FFPF) is an approach to network packet processing that adds many new features to existing filtering solutions like BPF. FFPF is designed for high speed by pushing computations by...
### Pkt processing on IXP

<table>
<thead>
<tr>
<th>operator-type</th>
<th>operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>+, -, /, *, %, --, ++</td>
</tr>
<tr>
<td>Assignment</td>
<td>=, *=, /=, %=, +=, -=</td>
</tr>
<tr>
<td>Logical/Relational</td>
<td>==, !=, &gt;, &lt;, &gt;=, &lt;=,</td>
</tr>
<tr>
<td></td>
<td>&amp; &amp; ,</td>
</tr>
<tr>
<td>Bitwise</td>
<td>&amp; ,</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>statement-type</th>
<th>operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>if/then/else</td>
<td>IF (expr) THEN stmt1 ELSE stmt2 FI</td>
</tr>
<tr>
<td>for()</td>
<td>FOR (initialize; test; update) stmts; BREAK; stmts; ROF</td>
</tr>
<tr>
<td>external function</td>
<td>EXTERN(filter, input, output)</td>
</tr>
<tr>
<td>hash()</td>
<td>HASH(start_byte, size)</td>
</tr>
<tr>
<td>return value</td>
<td>RETURN (val)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data type</th>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register ( n )</td>
<td>R[( n )]</td>
</tr>
<tr>
<td>Memory location ( n )</td>
<td>M[( n )]</td>
</tr>
<tr>
<td>Packets access:</td>
<td></td>
</tr>
<tr>
<td>- byte ( f(n) )</td>
<td>PKT.B[( f(n) )]</td>
</tr>
<tr>
<td>- word ( f(n) )</td>
<td>PKT.W[( f(n) )]</td>
</tr>
<tr>
<td>- bit ( m ) in byte ( n )</td>
<td>PKT.B[( n )].U1[( m )]</td>
</tr>
<tr>
<td>- byte ( m ) in word ( n )</td>
<td>PKT.W[( n )].U8[( m )]</td>
</tr>
<tr>
<td>etc.</td>
<td>(many options, including macros)</td>
</tr>
</tbody>
</table>

Fig. 8. FPL-2 language constructs