Passive Network Analysis Using Libtrace

Shane Alcock

Outline

- Introduction and Basics
- The Libtrace Tools
- Simple Libtrace Programming
- Advanced Topics

Part Four

- Advanced Topics
  - Advanced protocol analysis
  - A practical example
  - Overview of projects using libtrace
    - Network visualisation with BSOD
  - The future of libtrace
  - Question time

Protocol Analysis

- Libtrace provides functions to jump directly to the header at a particular layer
  - Metadata layer, e.g. RadioTap, Prism, Linux SLL
  - Layer 2 (aka link layer), e.g. Ethernet, 802.11
  - Layer 3 (aka IP layer), e.g. IP, IPv6
  - Transport layer, e.g. TCP, UDP, ICMP

Protocol Analysis

```
void *trace_get_packet_meta(libtrace_packet_t *packet,
 libtrace_linktype_t *linktype, uint32_t *remaining);


void *trace_get_layer2(libtrace_packet_t *packet,
 libtrace_linktype_t *linktype, uint32_t *remaining);


void *trace_get_layer3(libtrace_packet_t *packet, uint16_t
 *ethertype, uint32_t *remaining);


void *trace_get_transport(libtrace_packet_t *packet, uint8_t
 *proto, uint32_t *remaining);
```

Protocol Analysis

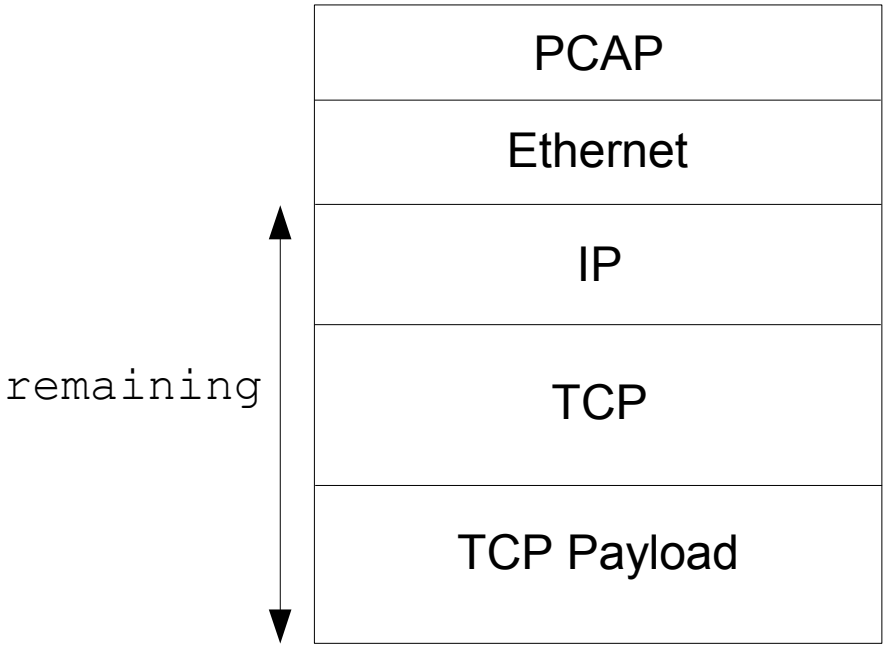
- Each function returns a void pointer to the header
  - If no header is present at that layer, NULL is returned
- Cast the pointer to the appropriate header based on type
  - The second parameter is set to signify the protocol type

Protocol Analysis

- Libtrace defines structures for most common headers
  - IP – libtrace_ip_t
  - IPv6 – libtrace_ip6_t
  - TCP – libtrace_tcp_t
  - UDP – libtrace_udp_t
  - ICMP – libtrace_icmp_t
  - Ethernet – libtrace_ether_t
  - VLAN – libtrace_8021q_t
  - 802.11 – libtrace_80211_t
- Many others as well – check out libtrace.h for a full list

Protocol Analysis

- Third parameter: `remaining`
  - Set by the protocol analysis function to contain the number of bytes between the start of the header and the end of the packet
  - Other analysis functions require a correct `remaining` value
  - Example: `remaining` after calling trace_get_layer3

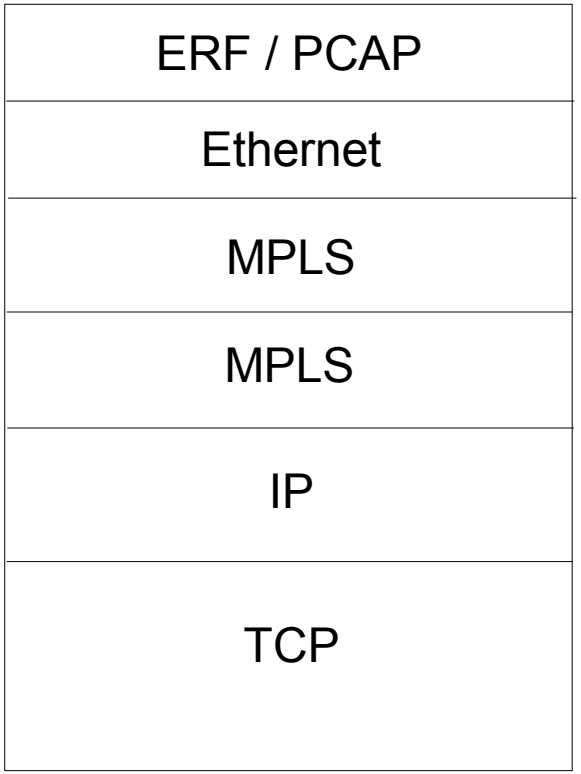


Protocol Analysis

- Example - httpcount.c
  - Rewriting our HTTP counter using the protocol analysis API

Advanced Protocol Analysis

- Remember our MPLS packet from earlier?



Advanced Protocol Analysis

- We can't jump directly to the MPLS headers
  - `trace_get_layer2()` will give us the Ethernet header
  - `trace_get_layer3()` will give us the IP header

Advanced Protocol Analysis

```
void *trace_get_payload_from_layer2(void *l2,
 libtrace_linktype_t linktype, uint16_t *ethertype,
 uint32_t *remaining);
```

- Returns a pointer to the first header after the given layer 2 header
- Returns NULL if the layer 2 header was incomplete
- `linktype` must be set to the type of the layer 2 header
- `ethertype` will be set to indicate the type of the returned header
- `remaining` will be decremented by the size of the skipped header
- Check the value of `remaining` upon return!

Advanced Protocol Analysis

- Example - mplscount.c
  - Counting MPLS packets

Advanced Protocol Analysis

```
void *trace_get_payload_from_mpls(void *mpls, uint16_t
  *type, uint32_t *remaining);
```

- Returns a pointer to the first header after the given MPLS header
- Returns NULL if an MPLS header is not passed in
- Returns NULL if the MPLS header is incomplete
- `type` must be set to the type of the header passed in
- `type` will be updated to indicate the type of the returned header
- `remaining` will be decremented by the size of the skipped header
- Check the value of `remaining` upon return!

Advanced Protocol Analysis

- Example - mplstag.c
  - Printing all the MPLS tags in a packet

Advanced Protocol Analysis

```
void *trace_get_payload_from_vlan(void *vlan, uint16_t
  *type, uint32_t *remaining);
```

- Returns a pointer to the first header after the given VLAN header
- Returns NULL if the header passed in is not a VLAN header
- Returns NULL if the VLAN header is incomplete
- `type` must be set to the type of the header passed in
- `type` will be updated to indicate the type of the returned header
- `remaining` will be decremented by the size of the skipped header
- Check the value of `remaining` upon return!

Advanced Protocol Analysis

```
void *trace_get_payload_from_pppoe(void *pppoe, uint16_t
  *type, uint32_t *remaining);
```

- Returns a pointer to the first header after the given PPPoE header
  - Also skips the subsequent PPP header
- Returns NULL if the PPPoE or PPP header is incomplete
- `type` will be updated to indicate the type of the returned header
- `remaining` will be decremented by the size of the skipped header
- Check the value of `remaining` upon return!

Advanced Protocol Analysis

```
void *trace_get_payload_from_ip(libtrace_ip_t *ip, uint8_t
  *proto, uint32_t *remaining);
```

```
void *trace_get_payload_from_ip6(libtrace_ip6_t *ip, uint8_t
  *proto, uint32_t *remaining);
```

- Returns a pointer to the first header after the given IP header
- Returns NULL if the IP header is incomplete
- `proto` is set to indicate the protocol of the returned header
- `remaining` operates just as with the previous functions

Advanced Protocol Analysis

```
void *trace_get_payload_from_tcp(libtrace_tcp_t *tcp,
 uint32_t *remaining);
```

```
void *trace_get_payload_from_udp(libtrace_udp_t *udp,
 uint32_t *remaining);
```

```
void *trace_get_payload_from_icmp(libtrace_icmp_t *icmp,
 uint32_t *remaining);
```

- Returns a pointer to the data after the given transport header
- Returns NULL if the header is incomplete
- `remaining` operates just as with the previous functions
- No indication is given as to the protocol of the returned data

A Practical Example

- Determining the amount of header overhead
  - Step through each of the headers to calculate total for a packet
    - Include link and meta-data layers
      - Therefore, I can't jump straight to the IP header
  - Also calculate post-transport payload size to compare against
  - Produce statistics for both TCP and UDP traffic
  - Periodically output stats so we can create a pretty graph

- End result: headerdemo.c

Ruby Libtrace

- Written by Nevil Brownlee (University of Auckland)

- Combine the features of ruby with libtrace
  - Exception handling
  - Iterators
  - Garbage collection

- Other languages
  - Python bindings suffered from poor performance
  - Implementations from libtrace users are most welcome

Libtrace Projects

- WDCap
  - http://research.wand.net.nz/software/wdcap.php
  - Tool for capturing and writing traces
  - Driving force behind much of the early libtrace development
  - Modular components allow capture to be customised
    - Packet snapping and anonymisation
    - Output trace file rotation
    - Direction tagging
    - Exporting of captured packets over a network

Libtrace Projects

- Maji – an implementation of an IPFIX meter
  - http://research.wand.net.nz/software/maji.php
  - Packets are read using libtrace
  - Many information elements are extracted using libtrace functions
    - i.e. any elements that can be found in a protocol header
  - Hoping for a release before the end of 2008
- More information about IPFIX (including links to RFCs)
  - http://www.ietf.org/html.charters/ipfix-charter.html

Libtrace Projects

- Nettest
  - http://nettest.wand.net.nz/
  - Passive network performance measurement applet
  - Collects performance statistics from NZ broadband users
  - Measurements are all done using libtrace
  - Compare results
    - ISPs
    - Service plans
    - Cities

Libtrace Projects

- TCP object extraction
  - Determine application-level objects using only packet headers
  - Search for non-MSS sized packets to find object boundaries
  - Paper published at ATNAC 2007
    - http://www.wand.net.nz/pubDetail.php?id=224

Libtrace Projects

- BSOD
  - http://research.wand.net.nz/software/visualisation.php
  - Real-time 3D graphical view of network traffic
  - Input can be any libtrace-supported format
    - Especially live capture formats!
  - BSOD server processes the trace or live capture
  - BSOD client displays the 3D visualisation

Libtrace Projects

- BSOD demonstration

The Future

- Enhancements planned for upcoming libtrace releases
  - New IO system enabling (de)compression in a separate thread
  - Revamp of the tool UI to fix inconsistencies
  - Support for new protocols and trace formats
  - General performance enhancements

- Further suggestions are also welcome!

The End

- Any final questions?

WAND Network Research Group
Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

www.crc.net.nz
www.wand.net.nz
www.waikato.ac.nz

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato