Application-layer traffic classification

Fulvio Risso
fulvio.risso@polito.it
The problem of traffic classification

- **Simple at L2-L4**
  - “if ethertype == 0x0800, then there is an IP packet”
  - Usually done with a set of *if-then-else* or even *switch-case*

- **Ambiguous at L7**
  - TCP port 80 does not mean automatically “protocol HTTP”
Some notes...

- "Verification” vs “classification”
  - Classification was in the old days, when application were detected by “classifying” the packet based on a field
  - Now, we need to *verify* that the packet keeps to some well defined rules

\[\text{Classification} = \text{Verification}\]

- Protocol verification, or application verification?
  - Skype can use the standard HTTP protocol to get data
  - Is that traffic “Skype” or “HTTP”?
Several approaches to traffic classification

- **Content-based**
  - Port-based (stateless)
  - Payload-based (stateful)
- **Message-based**
- **Packet-based** (e.g., Spatscheck)
- **Payload-based** (stateful)
- **Statistical methods**
  - Host social behaviour (e.g., Faloutsos)
  - Traffic statistics (e.g., Salgarelli, Baiocchi, Moore)
- **Auto-learning methods (e.g., Bayes)**
- **Preclassified bins**

Pre-computed or auto-learning signatures
Protocol verification

- **Different degrees of verification**
  - Protocol is X and not Y
  - Protocol is X, but not well formed

- **Signature-based verification**
  - “Does the message contain some well-known pattern?”
  - Signature + other “simple” checks (e.g., packet length, ...)
  - Can be seen as a *lightweight* syntactical verification, particularly if coupled with TCP and IP normalization

- **Syntactical verification**
  - “Is the message well-formed?”

- **Protocol Conformance verification**
  - “Is the protocol state machine coherent?”

- **Semantic verification**
  - “Is the HTTP packet transporting an image, as it was declared in the header?”
Some notes about protocol verification

- Protocol verification, or application detection?
  - Skype can use the standard HTTP protocol to get data
  - Is that traffic “Skype” or “HTTP”?

> Even the Protocol Conformance Verification cannot get an answer to this problem

- Let’s call this problem “application detection”
  - One step further (is it orthogonal to protocol verification?)
Protocol verification and pre-classification

- **Verification**
  - “Is the protocol what was it supposed to be?”

- **Pre-classification**
  - A smart method to pick the most likely protocol first
    - Usually reduces the number of protocols you have to verify
  - Most-common approaches
    - Port-based
    - **Signature-based**
  - Used by most technologies (also not signature-based)
### Comparing different verification methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
<th>TCP/IP normalization (i.e., per-packet processing)</th>
<th>Cross-packet analysis</th>
<th>State</th>
<th>Session-based correlation (^1)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature-based verification</td>
<td>Low</td>
<td>Not required (cannot detect signatures split over two packets)</td>
<td>Possible</td>
<td>Limited (session table)</td>
<td>Very limited</td>
<td>Low (?)</td>
</tr>
<tr>
<td>Signature-based + TCP/IP normalization</td>
<td>Medium-Low</td>
<td>Required (?)</td>
<td>Possible</td>
<td>Medium (virtual packet + session table)</td>
<td>Very limited</td>
<td>Low (?)</td>
</tr>
<tr>
<td>Syntactical verification</td>
<td>Medium</td>
<td>Required</td>
<td>Yes</td>
<td>Medium (virtual packet + session table)</td>
<td>Possible ((\rightarrow) behavioural)</td>
<td>Medium (?)</td>
</tr>
<tr>
<td>Behavioural verification</td>
<td>Medium-High</td>
<td>Required</td>
<td>Yes</td>
<td>High (virtual packet + session table + per-session state)</td>
<td>Good</td>
<td>Good (?)</td>
</tr>
<tr>
<td>Semantic verification</td>
<td>High</td>
<td>Required</td>
<td>Yes</td>
<td>High (virtual packet + session table + per-session state + ????)</td>
<td>Good</td>
<td>Best (?)</td>
</tr>
</tbody>
</table>

\(^1\) Session-based correlation: capability to correlate different flows (e.g., both directions of a TCP flow; SIP signalling and RTP media, etc.)
Payload-based traffic classification

- **Tools for better Packet Classification**
  - Session Tracking
  - IP and TCP reassembler
    - Requires lot of resources, but there is nothing more to say
  - Protocol state machine
    - Better if we have a language for describing the protocol state machine

- **Other issues that need to be addressed**
  - Application-negotiated sessions
Application-negotiated sessions

- Need to parse application-layer protocol for "child" session ID
  - FTP, SIP

- Incomplete entries
  - E.g. FTP passive

- Needs some "protocol parsing" capabilities
  - Often done also through regex
Session tracking

- “TCP session table”
  - Keeps the correlation among packets belonging to the same session

- Packet-based technologies
  - Use ST to correlate packets (e.g. Sequence Numbers in RTP)
  - Use ST to keep the state of the verification
    - The signature may be present only on the first packet
    - Next packets must not be verified

- Stream-based technologies
  - Use ST to point to the proper instance of the TCP reassembler

- Needs to address the “incomplete entry” problem
  - “masked” entries
And what about application detection?

- This is something orthogonal compared to protocol verification
- Usually done through some signature-based analysis
  - E.g., if the "user-agent" is "emule", probably this is an edonkey download through HTTP
  - At this time, no differences between different protocol verification technologies
Some existing technologies

- **SML, BinPac**
  - Stream-based
  - Cannot be used packet-by-packet

- **NetPDL**
  - Currently packet-based, but it strongly depends on the implementation
  - It has also primitives for packet-based analysis (e.g. `<missing-data>` element)

<table>
<thead>
<tr>
<th></th>
<th>Signature-based preclass.</th>
<th>Signature-based</th>
<th>Signature-based + TCP/IP normalization</th>
<th>Syntactical verification</th>
<th>Behavioural verification</th>
<th>Application verification</th>
<th>Semantic verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinPac</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SML</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>NetPDL</td>
<td>Yes</td>
<td>Yes</td>
<td>Planned</td>
<td>Possible (with extensions)</td>
<td>Possible</td>
<td>Partially implemented</td>
<td>Yes</td>
</tr>
</tbody>
</table>
NetPDL: simple example

<protocol name="Ethernet" longname="Ethernet 802.3">
  <format>
    <fields>
      <field type="fixed" name="dst" longname="MAC Dest." size="6"/>
      <field type="fixed" name="src" longname="MAC Source" size="6"/>
      <field type="fixed" name="type" longname="Ethertype" size="2"/>
    </fields>
  </format>
  <encapsulation>
    <switch expr="buf2int(type)">
      <case value="0x0800"> <nextproto name="#IP"/> </case>
      <case value="0x0806"> <nextproto name="#ARP"/> </case>
    </switch>
  </encapsulation>
</protocol>
Binpac: example (1)

type DNS_message = record {
  header: DNS_header;
  question: DNS_question(this)[header.qdcount];
  answer: DNS_rr(this)[header.ancount];
  authority: DNS_rr(this)[header.nscount];
  additional: DNS_rr(this)[header.arcount];
} &byteorder = bigendian, &exportsourcedata;

type DNS_header = record { ... };

type DNS_question(msg: DNS_message) = record {
  qname: DNS_name(msg);
  qtype: uint16;
  qclass: uint16;
} &let {
  # Generate Bro event dns_request if a query
  bro_gen_request: bool = bro_event_dns_request($context.connection.bro_conn, msg.header, qname, qtype, qclass)
  &if (msg.header.qr == 0); # if a request
};

type DNS_rr(msg: DNS_message) = record {
  rr_name: DNS_name(msg);
  rr_type: uint16;
  rr_class: uint16;
  rr_ttl: uint32;
  rr_rdlen: uint16;
  rr_rdata: DNS_rdata(msg, rr_type, rr_class)
  &length = rr_rdlen;
} &let {
  bro_gen_A_reply: bool = bro_event_dns_A_reply($context.connection.bro_conn, msg.header, this, rr_rdata.type_a)
  &if (rr_type == 1);
  bro_gen_NS_reply: bool = bro_event_dns_NS_reply(...);
  &if (rr_type == 2);
};

type DNS_rdata(msg: DNS_message, rr_type: uint16, 33 rr_class: uint16) = case rr_type of {
  1 -> type_a: uint32 &check(rr_class == CLASS_IN);
  2 -> type_ns: DNS_name(msg);
  # Omitted: TYPE_PTR, TYPE_MX, ...
  default -> unknown: bytestring &restofdata;
};
Binpac: example (2)

# A DNS name is a sequence of DNS labels

```pascal
type DNS_name(msg: DNS_message) = record
  labels: DNS_label(msg)[] &until($element.last);
end;
```

# A label contains a byte string or a name pointer

```pascal
type DNS_label(msg: DNS_message) = record
  length: uint8;
  data: case label_type of
    0 -> label: bytestring &length = length;
    3 -> ptr_lo: uint8; # the lower 8-bit of offset
  end;
end;

&let {
  label_type: uint8 = length >> 6;
  last: bool = (length == 0) || (label_type == 3);
}

# If the label is a pointer ...

```pascal
ptr_offset: uint16 = (length & 0x3f) << 8 + ptr_lo
  \[\text{if} \ (label_type == 3);\]
ptr: DNS_name(msg)
  \[\text{withinput} \ {\text{msgdata}}(msg.sourcedata, ptr_offset)\]
  \[\text{if} \ (label_type == 3);\]
```
NetPDL and TCP normalization
NetPDL and TCP encapsulation

Dynamic entries

```
<if expr="checklookuptable($tcpsessiontable, ...) >
    <if-true>
        <nextproto proto="$tcpsession.nextproto"/>
    </if-true>
</if>
```

"Well-known" entries

```
<switch expr="sourceport">
    <case value="21"><nextproto-candidate proto="#ftp"/> </case>
    <case value="80"><nextproto-candidate proto="#http"/></case>
</switch>
```

"Try and see" entries

```
<nextproto-candidate proto="#http"/>
<nextproto-candidate proto="#skype"/>
```
# NetPDL: indicative processing cost

<table>
<thead>
<tr>
<th>Test mode</th>
<th>Processing cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old NetPDL, no session tracking</td>
<td>265 µs/packet</td>
</tr>
<tr>
<td>Session tracking, no protocol verification</td>
<td>41 µs/packet</td>
</tr>
<tr>
<td>Session tracking with protocol verification</td>
<td>110 µs/packet</td>
</tr>
<tr>
<td>Session tracking with protocol verification and “try and see” entries</td>
<td>243 µs/packet</td>
</tr>
</tbody>
</table>

Pentium 4, 3GHz, 2GB RAM  
800MB data, full payload, 1.5M packets  
PacketDecoder (based on the NetBee library)
Evaluating the results

**Coverage**
# of supported protocol

**Accuracy**
Quality of the classification for protocol X

**False positives**
"this proto is X", while it is not

**False negatives**
"this protocol is not X", while it is

**Unclassified**
"this proto is unknown", while it is X

**Wrong classification**
"this proto is Y", while it is X

- It has to be done "per protocol"
- Only for protocol in our "coverage" domain
Coverage

Link-Layer Protocols
- ethernet
- fddi
- tokenring
- vlan
- llc
- snap
- stp-rstp

BlueTooth Protocols
- hci_acl_data
- hci_command
- hci_error_message
- hci_event
- hci_negotiation
- hci_packet_type
- hci_sco_data
- hci_unknown_type
- l2cap_command

IPX Protocols
- ipx
- ripx
- ipx_sap

Transport Protocols
- tcp
- udp

Network Protocols
- bootp
- dhcp
- icmp
- igmp
- arp
- ip
- ipfrag
- ipv6
- icmp6
- dns
- dns_tcp

Routing Protocols
- rip
- rip6
- igrp
- eigrp
- ospf
- ospf6
- BGP
- dvmrp
- pim
- pim6

Messaging Protocols
- skype
- yahoomsg
- msnmsg
- irc

P2P Protocols
- ares
- winmx
- bittorrent
- edonk
- edonkudp
- fasttrack
- gnutella
- dcpp
- peerenabler
- slsk

Routing Protocols
- rip
- rip6
- igrp
- eigrp
- ospf
- ospf6
- BGP
- dvmrp
- pim
- pim6

Mail Protocols
- imap
- pop3
- simap4
- smtp
- spop3
- ssntp

RPC Protocols
- dce_rpc_tcp
- dce_rpc_udp
- onc_rpc_udp
- mnt
- rpcbind
- nfs

Other Protocols
- btsdp
- cdp
- hsrp
- vrrp
- cpp
- chap
- gre
- icmp
- lcp
- ppp
- ptp
- esp
- kerberos
- netbios
- netbiosdgm
- netbiosssn
- ntp
- pppoe
- pppoed

Application Protocols
- ftp
- ftpdata
- http
- rtsp
- sip
- stun
- snmp
- telnet

Other Protocols
- radius
- rpcap
- rtcp
- rtp
- auth
- cldap
- icp
- ipp
- isakmp
- jrmi
- ldap
- microsoftds
- rdp
- ssdp
- ssh
- ssl
- syslog
- xmpp
- fsecure
- sql
- oracle_sql
- samba
- rfb
- wins
- ms_sql_monitor
- ms_sql_server
- nt_security_log
- cvs
- pcanywhere
Comments about coverage

Current status

- A large number of supported protocols
  - 122 protocols detected
  - Several P2P applications

- The quality of the classification may vary
  - In fact, this is a very huge amount of work

Comments

- We can say that ALL IETF protocols are (or can be) supported
- All the protocol that are “well-behaving” (also P2P with dynamic ports, …) can be supported
- We cannot support encrypted protocols
  - But all the payload-based technologies have this limit
# Accuracy: Edonkey, Kazaa

<table>
<thead>
<tr>
<th>File</th>
<th>Packets</th>
<th>OK</th>
<th>%</th>
<th>FP</th>
<th>FN (unknown)</th>
<th>FN (wrong)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>File1</td>
<td>1077</td>
<td>1077</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File2</td>
<td>48</td>
<td>28</td>
<td>58.33%</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>Missing signature for Manolito</td>
</tr>
<tr>
<td>File3</td>
<td>382</td>
<td>382</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File4</td>
<td>10745</td>
<td>10742</td>
<td>99.97%</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>Packet too short (signature does not match)</td>
</tr>
<tr>
<td>File5</td>
<td>10723</td>
<td>10720</td>
<td>99.97%</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>Packet too short (signature does not match)</td>
</tr>
<tr>
<td>File6</td>
<td>8124</td>
<td>8118</td>
<td>99.93%</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>Session was dropped due to a timeout</td>
</tr>
<tr>
<td>File7</td>
<td>982</td>
<td>982</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File8</td>
<td>10</td>
<td>10</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File</th>
<th>Packets</th>
<th>OK</th>
<th>%</th>
<th>FP</th>
<th>FN (unknown)</th>
<th>FN (wrong)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>File1</td>
<td>1118</td>
<td>1118</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File2</td>
<td>49</td>
<td>49</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File3</td>
<td>49</td>
<td>49</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File4</td>
<td>49</td>
<td>49</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File5</td>
<td>5650</td>
<td>5650</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File6</td>
<td>1118</td>
<td>1118</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File7</td>
<td>59</td>
<td>29</td>
<td>49.15%</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>Missing signature?</td>
</tr>
<tr>
<td>File8</td>
<td>15</td>
<td>15</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
## Accuracy: Gnutella

<table>
<thead>
<tr>
<th>File</th>
<th>Packets</th>
<th>OK</th>
<th>%</th>
<th>Errors</th>
<th>FN (unknown)</th>
<th>FN (wrong)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>File1</td>
<td>13</td>
<td>0</td>
<td>0.00%</td>
<td></td>
<td>0</td>
<td>13</td>
<td>0 Missing signature (UDP traffic)</td>
</tr>
<tr>
<td>File2</td>
<td>1762</td>
<td>1762</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File3</td>
<td>1762</td>
<td>1762</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File4</td>
<td>1762</td>
<td>1762</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File5</td>
<td>1762</td>
<td>608</td>
<td>34.51%</td>
<td>0</td>
<td>0</td>
<td>1154</td>
<td>Detected as HTTP; missing signature?</td>
</tr>
<tr>
<td>File6</td>
<td>1762</td>
<td>1762</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File7</td>
<td>660</td>
<td>660</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File8</td>
<td>425</td>
<td>425</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File9</td>
<td>300</td>
<td>300</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File10</td>
<td>167</td>
<td>167</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File11</td>
<td>192</td>
<td>192</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File12</td>
<td>143</td>
<td>143</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File13</td>
<td>143</td>
<td>143</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File14</td>
<td>1762</td>
<td>1762</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File15</td>
<td>1762</td>
<td>608</td>
<td>34.51%</td>
<td>0</td>
<td>0</td>
<td>1154</td>
<td>Detected as HTTP; missing signature?</td>
</tr>
<tr>
<td>File16</td>
<td>1762</td>
<td>1762</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File17</td>
<td>88</td>
<td>88</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>File18</td>
<td>79</td>
<td>79</td>
<td>100.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Other protocols:**
- FTP: 20 files, 46.2MB, 100%
- RTP: 7 files, 7.1MB, 100%
- HTTP tunnelling: 4 files, 2.1MB, 0%
Comments about accuracy

- The accuracy is very high

  So far, it is influenced only by the quality of the “signature”
  The NetPDL language provides all the mechanisms we need

- In fact, almost all the errors were related to missing signatures
  - It’s a matter of signatures, even if a protocol wants to screw you up

- False Positives depend on the protocol (e.g. much likely for HTTP)

- Limits related to the PacketBased-PerFlowState (PBFS)
  - Signature split across two packets
  - Missing protocol state machine
  - Do not seem to be relevant if no 100% accuracy is required
Classification results: top protos (bytes)

Nov 06, 9.5GB, 9.5 hours
- edonk: 44%
- http: 21%
- ms_sql_server: 13%
- samba: 11%
- other: 6%

Feb 07, 9.3GB, 11 hours
- edonk: 34%
- http: 32%
- edonkudp: 23%
- sip: 2%
- rdp: 2%
- ssl: 1%
- samba: 1%
- other: 6%

May 07, 7.8GB, 22 min
- edonk: 76%
- http: 5%
- smtp: 3%
- imap: 3%
- ftpdata: 2%
- sel: 2%
- rtsp: 2%
- other: 7%

(real traces)
Memory (#of TCP session entries)

- Nov 06, 9.5GB, 9.5 hours
- Feb 07, 9.3GB, 11 hours
Memory (total number of entries)

Nov 06, 9.5GB, 9.5 hours
Feb 07, 9.3GB, 11 hours
May 07, 7.8GB, 22 min
Nov 06, 9.5GB, 9.5 hours
Cleaning up traces: TCP activity timeout

![Graph showing TCP activity timeout and dropped packets percentage over time.](image)
“fully stateful” classification: BRO

- Signature used for detecting protocols on unknown ports
  - Unfortunately, only a few protocols have this feature
  - http, smtp, ftp, ssh, irc

- Protocol conformance classification
  - Through BinPac dissectors
  - dns, dce_rpc, http, ncp, rpc, smb

- For our tests, the only suitable protocol is HTTP
  - It is detected also on non-default ports (thanks to the signature)
  - It is parsed through a BinPac dissector (behavioral classification)
“Signature-based” vs. “fully stateful”

<table>
<thead>
<tr>
<th></th>
<th>Test1</th>
<th>Test2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of HTTP sessions</td>
<td>22897</td>
<td>38018</td>
<td></td>
</tr>
<tr>
<td>Classified by both tools</td>
<td>22875</td>
<td>37986</td>
<td></td>
</tr>
<tr>
<td>Classified only by BRO</td>
<td>2</td>
<td>0</td>
<td>Signature split across packets (2)</td>
</tr>
<tr>
<td>Classified only by NetPDL</td>
<td>20</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>False negatives¹</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bro</td>
<td>20</td>
<td>32</td>
<td>Unidirectional signature (17/29), drops (3/3)</td>
</tr>
<tr>
<td>NetPDL</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>False positives¹</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bro</td>
<td>7²</td>
<td>13²</td>
<td>Skype (6/11), Edonkey (1/2)</td>
</tr>
<tr>
<td>NetPDL</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

¹ False negatives and false positives are computed taking into account the set of sessions detected by at least one technology. For instance, a valid HTTP session that was undetected by both Bro and NetPDL, is not included in the “false positives”.

² These false positives are more related to “application detection”; in fact, formally these sessions are valid HTTP sessions.
Some comments (1)

- "fully stateful" classification is
  - More precise
    - ...as expected, if everything is correct
    - Packets with "unidirectional signatures" are, in fact, not valid HTTP packets
  - Less robust, e.g., to packet losses
    - Losses in the network, low CPU (see Telecom Italia)
  - Can be used only on the access slide
    - Asymmetric routing problem

- BRO
  - Does not implement "application detection"
  - So, some sessions (which are valid HTTP sessions indeed) are not recognized as "skype traffic"
Some comments (2)

- **Packet-based vs stream-based classification**
  - According to these tests, it is not an issue
  - Only 4 sessions have the signature split across two packets
    - Two of them (due to HTTP 1.1) have been classified correctly in any case by NetPDL, although at a later time (second HTTP request)
Conclusions

- PacketBased-PerFlowState vs MessageBased-PerProtocolState
  - Aka “NetPDL vs Pcube”, although it is not really correct

- From QoS perspective
  - MBPS has more, but...
    - Syntactical verification: not important for QoS
    - Stream based (splitted signatures): we can do without it (see current accuracy results)
    - Behavioural verification: idem
  - PBPS has everything you need, with enough (very high) accuracy
  - Please note that NetPDL is PBPS, but it can become MBFS
    - Adding Syntactical verification and Stream reassembly

- From the security perspective
  - The story is very different
Future work (1)

- **More on protocol Classification Accuracy**
  - ... in terms of False Positives and False Negatives
  - The technology is pretty similar to the one in NBAR; so, we’re confident that the accuracy will be very high (it is just a matter of tuning signatures and such)
  - Different test conditions
    - What happens when a snapshot is processed?
    - Different capture traces (University, ISP)
    - Memory requirements
    - Heuristics (also to cope with encrypted payloads)

- **Robustness**
  - Packet losses, asymmetric routing

- **Scalability**
  - Processing time, memory requirements
Future work (2)

- **Protocol description (header format)**
  - We are still unable to describe some L7 protocols
    - New types of fields will probably be required
      - XML, ASN.1 BER, TLV
    - New structures (sequence; set, choice?)

- **More dissectors**
  - Currently, 122 protocols are classified, but not fully decoded
Some links

- **NetBee Library**
  - http://www.nbee.org

- **Protocol database and NetPDL latest specs**
  - http://www.nbee.org/netpdl/
Questions?