UDP Options
Implementation Experience
draft-ietf-tsvwg-udp-options-02

Tom Jones
tom@erg.abdn.ac.uk

NEAT is funded by the European Union’s Horizon 2020 research and innovation programme under grant agreement no. 644334.
UDP Option Area

IP transport payload

| IP Hdr | UDP Hdr | UDP user data | surplus area |

+--------+---------+----------------+--------------+
UDP Option TLV

+--------+       +--------+       +--------+       +--------+
| Kind=0 |       | Kind=1 |       | Kind=2 | Ones8 |       | Kind=3 | Len=4 | CRC16sum |
| EOL    |       | NOP    |       | OCS    |       | ACS    |

neat
# UDP Options

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td>–</td>
<td>End of Options List (EOL)</td>
</tr>
<tr>
<td>1*</td>
<td>–</td>
<td>No operation (NOP)</td>
</tr>
<tr>
<td>2*</td>
<td>2</td>
<td>Option checksum (OCS)</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>Alternate checksum (ACS)</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Lite (LITE)</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Maximum segment size (MSS)</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>Timestamps (TIME)</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>Fragmentation (FRAG)</td>
</tr>
<tr>
<td>8</td>
<td>(varies)</td>
<td>Authentication and Encryption (AE)</td>
</tr>
<tr>
<td>9–126</td>
<td>(varies)</td>
<td>UNASSIGNED (assignable by IANA)</td>
</tr>
<tr>
<td>127–253</td>
<td></td>
<td>RESERVED</td>
</tr>
<tr>
<td>254</td>
<td>N(&gt;=4)</td>
<td>RFC 3692–style experiments (EXP)</td>
</tr>
<tr>
<td>255</td>
<td></td>
<td>RESERVED</td>
</tr>
</tbody>
</table>
## UDP Options

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td>-</td>
<td>End of Options List (EOL)</td>
</tr>
<tr>
<td>1*</td>
<td>-</td>
<td>No operation (NOP)</td>
</tr>
<tr>
<td>2*</td>
<td>2</td>
<td>Option checksum (OCS)</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>Alternate checksum (ACS)</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Lite (LITE)</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Maximum segment size (MSS)</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>Timestamps (TIME)</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>Fragmentation (FRAG)</td>
</tr>
<tr>
<td>8</td>
<td>(varies)</td>
<td>Authentication and Encryption (AE)</td>
</tr>
<tr>
<td>9-126</td>
<td>(varies)</td>
<td>UNASSIGNED (assignable by IANA)</td>
</tr>
<tr>
<td>127-253</td>
<td></td>
<td>RESERVED</td>
</tr>
<tr>
<td>254</td>
<td>N(&gt;=4)</td>
<td>RFC 3692–style experiments (EXP)</td>
</tr>
<tr>
<td>255</td>
<td></td>
<td>RESERVED</td>
</tr>
</tbody>
</table>
Implementation Status

- Implementation for FreeBSD
  - [https://github.com/uoaerg/freebsd](https://github.com/uoaerg/freebsd)
    - branch udpoptions-ietf101
- Wireshark dissector
- PacketDrill Tests
New Options to support DPLPMTUD

- Add two New Options
  - Echo Request
    ```
    +---------+--------+-----------------+
    | Kind=9  | Len=6  |     Token       |
    +---------+--------+-----------------+
    1 byte   1 byte       4 bytes
    ```
  - Echo Response
    ```
    +---------+--------+-----------------+
    | Kind=10 | Len=6  |     Token       |
    +---------+--------+-----------------+
    1 byte   1 byte       4 bytes
    ```
  - Probe: Echo request with entire padding packet
Why do we need two options

- Each data direction is a separate unidirectional stream
- Use echo request and response in each direction
  - Uses:
    - To verify UDP Options is supported on the path
    - As a connectivity check
    - To verify PMTU Probe is received
    - Could verify remote receives a specific “option”
Why do we need two options

- Each data direction is a separate unidirectional stream
- Use echo request and response in each direction
  - Uses:
    - To verify UDP Options is supported on the path
    - As a connectivity check
    - To verify PMTU Probe is received
    - Could verify remote receives a specific “option”
Why do we need two options

- Each data direction is a separate unidirectional stream
- Use echo request and response in each direction
  - Uses:
    - To verify UDP Options is supported on the path
    - As a connectivity check
    - To verify PMTU Probe is received
    - Could verify remote receives a specific “option”
Why do we need two options

- Each data direction is a separate unidirectional stream
- Use echo request and response in each direction
  - Uses:
    - To verify UDP Options is supported on the path
    - As a connectivity check
    - To verify PMTU Probe is received
    - Could verify remote receives a specific “option”
Why do we need two options

- Each data direction is a separate unidirectional stream
- Use echo request and response in each direction
  - Uses:
    - To verify UDP Options is supported on the path
    - As a connectivity check
    - To verify PMTU Probe is received
    - Could verify remote receives a specific “option”
Problem with OCS

“I was proposing that the current OCS option be discarded, and replaced with a fixed length 16 bit checksum field at the start of the surplus area…

- Derek Fawcus
Problem with OCS

“I was proposing that the current OCS option be discarded, and replaced with a fixed length 16 bit checksum field at the start of the surplus area…

So I am again proposing:

1. Remove the OCS option.
2. Add a fixed length 16 bit checksum field at a known location (The start of the surplus area).
3. Potentially allow one to store 0x0000 in that field to indicate the options themselves are not protected by a checksum.”

- Derek Fawcus
Problem with OCS

• Option space may need to start with padding
• OCS cannot be fixed at start of option space
Suggestion:

- Redefine EOL as (EOL+OCS)
- Always placed at end of packet
- Makes option layout easier
- Reduces minimum option space Len by 1
- Option space becomes:
  - OPT,OPT,\(\text{NOP},\)EOL+OCS
Next Steps

• My implementation will be done in June
• Users?
• Other implementations?
Acknowledgement

NEAT is funded by the European Union’s Horizon 2020 research and innovation programme under grant agreement no. 644334.
void in_delayed_cksum(struct mbuf *m) {
    struct ip *ip;
    uint16_t csum, offset, ip_len;

    ip = mtod(m, struct ip *);
    offset = ip->ip_hl << 2;
    ip_len = ntohs(ip->ip_len);
    csum = in_cksum_skip(m, ip_len, offset);
    if (m->m_pkthdr.csum_flags & CSUM_UDP && csum == 0)
        csum = 0xffff;
    offset += m->m_pkthdr.csum_data; /* checksum offset */
    /* find the mbuf in the chain where the checksum starts*/
    while ((m != NULL) && (offset >= m->m_len)) {
        offset -= m->m_len;
        m = m->m_next;
    }
    *(u_short *)(m->m_data + offset) = csum;
}