The UDP-Lite protocol (RFC 3828)

UDP-Lite is a Standards-Track IETF transport protocol whose characteristic is a variable-length checksum. This has advantages for transport of multimedia (video, VoIP) over wireless networks, as partly damaged packets can still be fed into the codec instead of being discarded due to a failed checksum test.

This file briefly describes the existing kernel support and the socket API. For in-depth information, you can consult:

- The UDP-Lite Homepage: http://web.archive.org/web/%2E/http://www.erg.abdn.ac.uk/users/gerrit/udp-lite/

  From here you can also download some example application source code.
- The UDP-Lite HOWTO on http://web.archive.org/web/%2E/http://www.erg.abdn.ac.uk/users/gerrit/udp-lite/files/UDP-Lite-HOWTO.txt
- The Wireshark UDP-Lite WiKi (with capture files): https://wiki.wireshark.org/Lightweight_User_Datagram_Protocol
- The Protocol Spec, RFC 3828, http://www.ietf.org/rfc/rfc3828.txt

1. Applications

Several applications have been ported successfully to UDP-Lite. Ethereal (now called wireshark) has UDP-Litev4/v6 support by default.

Porting applications to UDP-Lite is straightforward: only socket level and IPPROTO need to be changed; senders additionally set the checksum coverage length (default = header length = 8). Details are in the next section.

2. Programming API

UDP-Lite provides a connectionless, unreliable datagram service and hence uses the same socket type as UDP. In fact, porting from UDP to UDP-Lite is very easy: simply add `IPPROTO_UDPLITE` as the last argument of the socket(2) call so that the statement looks like:

```
s = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDPLITE);
```

or, respectively,

```
s = socket(PF_INET6, SOCK_DGRAM, IPPROTO_UDPLITE);
```

With just the above change you are able to run UDP-Lite services or connect to UDP-Lite servers. The kernel will assume that you are not interested in using partial checksum coverage and so emulate UDP mode (full coverage).

To make use of the partial checksum coverage facilities requires setting a single socket option, which takes an integer specifying the coverage length:

- Sender checksum coverage: UDPLITE_SEND_CSCOV

  For example:

  ```
  int val = 20;
  setsockopt(s, SOL_UDPLITE, UDPLITE_SEND_CSCOV, &val, sizeof(int));
  ```

  sets the checksum coverage length to 20 bytes (12b data + 8b header). Of each packet only the first 20 bytes (plus the pseudo-header) will be checksummed. This is useful for RTP applications which have a 12-byte base header.
- Receiver checksum coverage: UDPLITE_RECV_CSCOV

  This option is the receiver-side analogue. It is truly optional, i.e. not required to enable traffic with partial checksum coverage. Its function is that of a traffic filter: when enabled, it instructs the kernel to drop all packets which have a coverage _less_ than this value. For example, if RTP and UDP headers are to be protected, a receiver can enforce that only packets with a minimum coverage of 20 are admitted:

  ```
  int min = 20;
  setsockopt(s, SOL_UDPLITE, UDPLITE_RECV_CSCOV, &min, sizeof(int));
  ```

The calls to getsockopt(2) are analogous. Being an extension and not a stand- alone protocol, all socket options known from UDP can be used in exactly the same manner as before, e.g. UDP_CORK or UDP_ENCAP.

A detailed discussion of UDP-Lite checksum coverage options is in section IV.

3. Header Files

The socket API requires support through header files in /usr/include:

- /usr/include/netinet/in.h to define IPPROTO_UDPLITE
- /usr/include/netinet/udplite.h for UDP-Lite header fields and protocol constants

For testing purposes, the following can serve as a `mini` header file:

```
#define IPPROTO_UDPLITE       136
#define SOL_UDPLITE           136
#define UDPLITE_SEND_CSCOV     10
#define UDPLITE_RECV_CSCOV     11
```

Ready-made header files for various distros are in the UDP-Lite tarball.

4. Kernel Behaviour with Regards to the Various Socket Options

To enable debugging messages, the log level need to be set to 8, as most messages use the KERN_DEBUG level (7).

1. Sender Socket Options

If the sender specifies a value of 0 as coverage length, the module assumes full coverage, transmits a packet with coverage length of 0 and according checksum. If the sender specifies a coverage < 8 and different from 0, the kernel assumes 8 as default value. Finally, if the specified coverage length exceeds the packet length, the packet length is used instead as coverage length.

2. Receiver Socket Options

The receiver specifies the minimum value of the coverage length it is willing to accept. A value of 0 here indicates that the receiver always wants the whole of the packet covered. In this case, all partially covered packets are dropped and an error is logged.

It is not possible to specify illegal values (<0 and <8); in these cases the default of 8 is assumed.

All packets arriving with a coverage value less than the specified threshold are discarded, these events are also logged.

3. Disabling the Checksum Computation

On both sender and receiver, checksumming will always be performed and cannot be disabled using SO_NO_CHECK. Thus:

```
setsockopt(sockfd, SOL_SOCKET, SO_NO_CHECK,  ... );
```

will always will be ignored, while the value of:

```
getsockopt(sockfd, SOL_SOCKET, SO_NO_CHECK, &value, ...);
```

is meaningless (as in TCP). Packets with a zero checksum field are illegal (cf. RFC 3828, sec. 3.1) and will be silently discarded.

4. Fragmentation

The checksum computation respects both buffersize and MTU. The size of UDP-Lite packets is determined by the size of the send buffer. The minimum size of the send buffer is 2048 (defined as SOCK_MIN_SNDBUF in include/net/sock.h), the default value is configurable as net.core.wmem_default or via setting the SO_SNDBUF socket(7) option. The maximum upper bound for the send buffer is determined by net.core.wmem_max.

Given a payload size larger than the send buffer size, UDP-Lite will split the payload into several individual packets, filling up the send buffer size in each case.

The precise value also depends on the interface MTU. The interface MTU, in turn, may trigger IP fragmentation. In this case, the generated UDP-Lite packet is split into several IP packets, of which only the first one contains the L4 header.

The send buffer size has implications on the checksum coverage length. Consider the following example:

```
Payload: 1536 bytes          Send Buffer:     1024 bytes
MTU:     1500 bytes          Coverage Length:  856 bytes
```

UDP-Lite will ship the 1536 bytes in two separate packets:

```
Packet 1: 1024 payload + 8 byte header + 20 byte IP header = 1052 bytes
Packet 2:  512 payload + 8 byte header + 20 byte IP header =  540 bytes
```

The coverage packet covers the UDP-Lite header and 848 bytes of the payload in the first packet, the second packet is fully covered. Note that for the second packet, the coverage length exceeds the packet length. The kernel always re-adjusts the coverage length to the packet length in such cases.

As an example of what happens when one UDP-Lite packet is split into several tiny fragments, consider the following

example:

```
Payload: 1024 bytes            Send buffer size: 1024 bytes
MTU:      300 bytes            Coverage length:   575 bytes

+-+-----------+-------------+-------------+-------------+
|8|    272    |      280    |     280     |     280     |
+-+-----------+-------------+-------------+-------------+
            280            560           840           1032
                                        ^
*****checksum coverage*************
```

The UDP-Lite module generates one 1032 byte packet (1024 + 8 byte header). According to the interface MTU, these are split into 4 IP packets (280 byte IP payload + 20 byte IP header). The kernel module sums the contents of the entire first two packets, plus 15 bytes of the last packet before releasing the fragments to the IP module.

To see the analogous case for IPv6 fragmentation, consider a link MTU of 1280 bytes and a write buffer of 3356 bytes. If the checksum coverage is less than 1232 bytes (MTU minus IPv6/fragment header lengths), only the first fragment needs to be considered. When using larger checksum coverage lengths, each eligible fragment needs to be checksummed. Suppose we have a checksum coverage of 3062. The buffer of 3356 bytes will be split into the following fragments:

```
Fragment 1: 1280 bytes carrying  1232 bytes of UDP-Lite data
Fragment 2: 1280 bytes carrying  1232 bytes of UDP-Lite data
Fragment 3:  948 bytes carrying   900 bytes of UDP-Lite data
```

The first two fragments have to be checksummed in full, of the last fragment only 598 (= 3062 - 2*1232) bytes are checksummed.

While it is important that such cases are dealt with correctly, they are (annoyingly) rare: UDP-Lite is designed for optimising multimedia performance over wireless (or generally noisy) links and thus smaller coverage lengths are likely to be expected.

5. UDP-Lite Runtime Statistics and their Meaning

Exceptional and error conditions are logged to syslog at the KERN_DEBUG level. Live statistics about UDP-Lite are available in /proc/net/snmp and can (with newer versions of netstat) be viewed using:

```
netstat -svu
```

This displays UDP-Lite statistics variables, whose meaning is as follows.

InDatagrams	The total number of datagrams delivered to users.
NoPorts	Number of packets received to an unknown port. These cases are counted separately (not as InErrors).
InErrors	Number of erroneous UDP-Lite packets. Errors include: • internal socket queue receive errors • packet too short (less than 8 bytes or stated coverage length exceeds received length) • xfrm4_policy_check() returned with error • application has specified larger min. coverage length than that of incoming packet • checksum coverage violated • bad checksum
OutDatagrams	Total number of sent datagrams.

These statistics derive from the UDP MIB (RFC 2013).

6. IPtables

There is packet match support for UDP-Lite as well as support for the LOG target. If you copy and paste the following line into /etc/protocols:

```
udplite 136     UDP-Lite        # UDP-Lite [RFC 3828]
```

then:

```
iptables -A INPUT -p udplite -j LOG
```

will produce logging output to syslog. Dropping and rejecting packets also works.

7. Maintainer Address

The UDP-Lite patch was developed at

University of Aberdeen Electronics Research Group Department of Engineering Fraser Noble Building Aberdeen AB24 3UE; UK

The current maintainer is Gerrit Renker, <gerrit@erg.abdn.ac.uk>. Initial code was developed by William Stanislaus, <william@erg.abdn.ac.uk>.