Linux SCTP is catching up and going above

Red Hat, Inc.

Marcelo Ricardo Leitner,  Xin Long

Linux Plumber Conference in Vancouver, 2018
Outline

1. What and Why is SCTP
   - Architecture
   - SCTP vs TCP

2. What We’ve Done on Linux
   - Projects
   - Improvements Made Recently
   - Features Implemented Lately
   - LINUX vs BSD

3. What’s the Next
   - Features Development
   - Code Refactor
   - Hardware Support
What and Why is SCTP

Architecture

Structures

1. Endpoint
2. Association
3. Transport
4. Stream
5. Msg
6. Packet
7. Chunk
SCTP Structures in Linux

- (sctp_)sock
  - + ep

- sctp_endpoint
  - + base.sk
  - + asocs

- sctp_association
  - + ep
  - + asocs
  - + streams
  - + transports

- sctp_stream
  - + out[n] (flexarray)

- sctp_stream_out
  - + ext
  - + outq

- sctp_stream_out_ext
  - + transmit_list
  - + list

- sctp_stream_out
  - + transmit_list
  - + list

- sctp_transport
  - + list
  - + packet

- sctp_transport
  - + list
  - + packet

- sctp_msg
  - + chunks

- sctp_chunk
  - + transmit_list
  - + list
SCTP Procedures in Linux
## SCTP vs TCP/UDP on Feature

<table>
<thead>
<tr>
<th>Feature</th>
<th>SCTP</th>
<th>TCP</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-duplex</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Connection oriented</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Reliable data transmission</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Unreliable data transmission</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Partially reliable data transmission</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>In order delivery</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Out of order delivery</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Flow- and Congestion Control</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>ECN support</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Selective ACKs</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Protection of message boundaries</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Fragmentations</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Multistreaming</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Multihoming</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Protection against SYN flooding</td>
<td>yes</td>
<td>no</td>
<td>n/a</td>
</tr>
<tr>
<td>Half-closed connection</td>
<td>no</td>
<td>yes</td>
<td>n/a</td>
</tr>
</tbody>
</table>
SCTP vs TCP on Performance
Outline

1. What and Why is SCTP
   - Architecture
   - SCTP vs TCP

2. What We’ve Done on Linux
   - Projects
   - Improvements Made Recently
   - Features Implemented Lately
   - LINUX vs BSD

3. What’s the Next
   - Features Development
   - Code Refactor
   - Hardware Support
What We’ve Done on Linux

Projects

lksctp-tools (lib and unit test)

MANIFEST ———
.
|-- bin
|-- doc
|-- man
|-- src
... |-- apps
... |-- func_tests
... |-- include
... |... `– netinet
... |-- lib
... |-- testlib
... `– withsctp

- sctp_darn, sctp_test
- sctp_status, sctp_xconnect
- peel_client, peel_server
- bindx_test, myftp, nagle_rcv, nagle_snd

Unit Test: Look in src/func_tests and in lksctp-tests package for examples of tests. Please do not submit code that fails its own tests or any of the unit tests. If it fails a functional test, please document that with the submission.

- sctp_send, sctp_sendmsg, sctp_recvmsg
- sctp_connectx_orig, sctp_connectx2, sctp_connectx3
- sctp_bindx, sctp_opt_info
- sctp_peeloff, sctp_peeloff_flags
sctp-tests (regression test): 27 test cases so far
What We’ve Done on Linux

Projects

tahi-sctp (conformance test)

RFC4960: Association Initialization
RFC4960: Association Termination
RFC4960: Fault Management
RFC4960: Error Cause
RFC4960: Chunk Bundling
RFC4960: User Data Transfer
RFC4960: Retransmission Timer
RFC4960: Congestion Control
RFC4960: Path MTU Discovery
RFC4960: Multi-Homed Endpoints
RFC4960: Explicit Congestion Notification
RFC4960: Packet Format
RFC4960: Miscellaneous Test
RFC4895: Authentication Chunks
RFC5061: Dynamic Address Reconfiguration
RFC3758: Partial Reliability Extension
RFC3554: Internet Protocol Security
Others

- Syzkaller (fuzz test)
- Codenomicon (fuzz test)
- Packetdrill (conformance test)
- Scapy (packet generating)
- More ?
Transport Rhashtable 1

- ASOCs hash table
  - key: lport + dport
  - key: lport + dport

- TRANSPORT rhash table
  - key: lport + dport + dip
  - key: lport + dport + dip
  - key: lport + dport + dip
  - key: lport + dport + dip
  - key: lport + dport + dip

- ASOC1
  - key: (1.1 + 5000 + 1.2(1.3) + 5001)

- ASOCa
  - key: (1.1 + 5000 + 2(2.3) + 5001)

- ASOC3
  - key: (0.1 + 5000 + 0.2(0.3) + 5001)

- TRANSPORT1
  - key: (1.1 + 5000 + 1.2 + 5001)

- TRANSPORT
  - key: (1.1 + 5000 + 1.2 + 5002)

- TRANSPORT
  - key: (1.1 + 5000 + 1.3 + 5001)

- TRANSPORT
  - key: (0.1 + 5000 + 0.2 + 5001)

- TRANSPORT
  - key: (0.1 + 5000 + 0.3 + 5001)

- TRANSPORT
  - key: (1.1 + 5000 + 1.3 + 5002)
Transport Rhashtable 2

1. 1-to-many (with "the same dport and different dip" lookup fast
2. 1-step to find both transport and asoc
3. Rhashtable (rhlist) features: rcu_lock and resize memory
4. Why not use the key with hash(dport, lport, dip, lip) ?
5. Why not make rhashtable per endpoint/socket ?
SCTP Offload 1

Userspace

User Data
(len = 5800)

SCTP

sctp hdr
Frag (1450)  sctp hdr
Frag (1450)  sctp hdr
Frag (1450)  sctp hdr
Frag (1450)

IP

iphdr
sctp hdr
Frag (1450)  iphdr
sctp hdr
Frag (1450)  iphdr
sctp hdr
Frag (1450)  iphdr
sctp hdr
Frag (1450)

LINK

( mtu = 1500)

iphdr
sctp hdr
Frag (1450)  iphdr
sctp hdr
Frag (1450)  iphdr
sctp hdr
Frag (1450)  iphdr
sctp hdr
Frag (1450)
SCTP Offload 2
### SCTP Diag 1

```
[iproute2]# ss --sctp -n -l
State  Recv-Q Send-Q  Local Address:Port      Peer Address:Port
LISTEN 0 128 172.16.254.254:8888    *:**
LISTEN 0 5 127.0.0.1:1234      *:**
LISTEN 0 5 127.0.0.1:1234      *:**
   ESTAB 0 0 127.0.0.1%lo:1234 127.0.0.1:4321
LISTEN 0 128 172.16.254.254:8888 *:**
   ESTAB 0 0 172.16.254.254%eth1:8888 172.16.253.253:8888
   ESTAB 0 0 172.16.254.254%eth1:8888 172.16.1.1:8888
   ESTAB 0 0 172.16.254.254%eth1:8888 172.16.1.2:8888
   ESTAB 0 0 172.16.254.254%eth1:8888 172.16.2.1:8888
   ESTAB 0 0 172.16.254.254%eth1:8888 172.16.2.2:8888
   ESTAB 0 0 172.16.254.254%eth1:8888 172.16.3.1:8888
   ESTAB 0 0 172.16.254.254%eth1:8888 172.16.3.2:8888
LISTEN 0 0 127.0.0.1%lo:4321 127.0.0.1:1234
   ESTAB 0 0 127.0.0.1%lo:4321 127.0.0.1:1234
```

```
[iproute2]# ss -Snai
State  Recv-Q Send-Q  Local Address:Port      Peer Address:Port
LISTEN 0 1 127.0.0.1:27375 127.0.0.1:27375
locals:127.0.0.1,192.168.42.2, v4mapped:1
ESTAB 0 0 127.0.0.1:37636 127.0.0.1:27375
locals:0.0.0.0, v4mapped:1
```

Others

- Dst source addr selection
- Rwnd improvements
- Partial reliability fixes
- MTU handling refactor
- PMTU discovery (critical) fixes
- CRC32c offloading on virtual interfaces
- Some codes cleaning up
- More ...
Overview

- **Stream Schedulers and User Message Interleaving** for the Stream Control Transmission Protocol [RFC8260]
- Additional Policies for the Partially Reliable Stream Control Transmission Protocol Extension [RFC7496]
- Stream Control Transmission Protocol (SCTP) Stream Reconfiguration [RFC6525]
- Sockets API Extensions for the Stream Control Transmission Protocol (SCTP) [RFC6458]
- Full SELinux support
- More...
Stream Schedulers

```c
struct sctp_sched_ops {
    /**< Property handling for a given stream */
    int (*set)(struct sctp_stream *stream, __u16 sid, __u16 value,
               gfp_t gfp);
    int (*get)(struct sctp_stream *stream, __u16 sid, __u16 *value);

    /**< Init the specific scheduler */
    int (*init)(struct sctp_stream *stream);
    /**< Init a stream */
    int (*init_sid)(struct sctp_stream *stream, __u16 sid, gfp_t gfp);
    /**< Free the entire thing */
    void (*free)(struct sctp_stream *stream);

    /**< Enqueue a chunk */
    void (*enqueue)(struct sctp_outq *q, struct sctp_dataseg *msg);
    /**< Dequeue a chunk */
    struct sctp_chunk *(*dequeue)(struct sctp_outq *q);
    /**< Called only if the chunk fit the packet */
    void (*dequeue_done)(struct sctp_outq *q, struct sctp_chunk *chunk);
    /**< Sched all chunks already enqueued */
    void (*sched_all)(struct sctp_stream *stream);
    /**< Unschedule all chunks already enqueued */
    void (*unsched_all)(struct sctp_stream *stream);
};
```
**Message Interleaving**

```
struct sctp_stream_interleave {
    __u16   data_chunk_len;
    __u16   ftsn_chunk_len;

    /* (I-)DATA process */
    struct sctp_chunk *(*make_datafrag)(const struct sctp_association *asoc,
                                            const struct sctp_sndrcvinfo *sinfo,
                                            int len, __u8 flags, gfp_t gfp);

    void (*assign_number)(struct sctp_chunk *chunk);
    bool (*validate_data)(struct sctp_chunk *chunk);
    int (*ulpgevent)(struct ulpq *ulpq,
                        struct sctp_chunk *chunk, gfp_t gfp);
    int (*enqueue_event)(struct ulpq *ulpq,
                          struct sctp_ulpevent *event);
    void (*renage_events)(struct ulpq *ulpq,
                           struct sctp_chunk *chunk, gfp_t gfp);
    void (*start_pd)(struct ulpq *ulpq, gfp_t gfp);
    void (*abort_pd)(struct ulpq *ulpq, gfp_t gfp);

    /* (I-)FORWARD-TSN process */
    void (*generate_ftsn)(struct sctp_outq *q, __u32 otsn);
    bool (*validate_ftsn)(struct sctp_chunk *chunk);
    void (*report_ftsn)(struct ulpq *ulpq, __u32 ftsn);
    void (*handle_ftsn)(struct ulpq *ulpq,
                        struct sctp_chunk *chunk);
};
```
PR_SCTP policies

1. Timed Reliability
   SCTP_PR_SCTP_TTL
   - When dequeuing chunks from A
   - When dequeuing chunks from C
   - When moving chunks from B to C
   - After receiving a SACK, check B and C

2. Limited Retransmissions Policy
   SCTP_PR_SCTP_RTX
   - When dequeuing chunks from A
   - When dequeuing chunks from C
   - When moving chunks from B to C
   - After receiving a SACK, check B and C

3. Priority Policy
   SCTP_PR_SCTP_PRIO
   - Before enqueuing chunk into A
   - And No Enough TX Buffer
   - Then try to drop C -> B -> A.
Stream Reconfig

1 Add Outgoing Streams: No restrictions
2 Add Incoming Streams: No restrictions
3 Reset Outgoing Streams: Reset stream 1, b have to be empty
4 Reset Incoming Streams: Peer will send Outgoing Stream request for which it has to follow the above rule
5 Reset SSN/TSN: All queues have to be empty: A, B, C, a, b, c
Socket APIs

1. User APIs
   - sctp_sendv
   - sctp_recvv

2. Snd Info Flags
   - SENDALL
   - MSG_MORE

3. Cmsgs
   - PR_INFO
   - AUTH_INFO
   - DSTv4
   - DSTv6
# Linux vs BSD on Features

<table>
<thead>
<tr>
<th>Chunks</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINUX:</td>
<td>LINUX:</td>
</tr>
<tr>
<td>ongoing</td>
<td>sctp_do_sm()</td>
</tr>
<tr>
<td></td>
<td>transport rhashtable</td>
</tr>
<tr>
<td>BSD:</td>
<td>offload</td>
</tr>
<tr>
<td>SCTP_NR_SELECTIVE_ACK (draft)</td>
<td>diag</td>
</tr>
<tr>
<td>SCTP_PACKET_DROPPED (draft)</td>
<td>BSD:</td>
</tr>
<tr>
<td>SCTP_PAD_CHUNK</td>
<td>sctp_cc_functions</td>
</tr>
</tbody>
</table>
Outline

1. What and Why is SCTP
   - Architecture
   - SCTP vs TCP

2. What We’ve Done on Linux
   - Projects
   - Improvements Made Recently
   - Features Implemented Lately
   - LINUX vs BSD

3. What’s the Next
   - Features Development
   - Code Refactor
   - Hardware Support
Features Development

- Support more Chunks, Apis, Sockopts, Notifications.
- Other features from Draft RFC, like SCTP NAT and CMT.
- SCTP Performance Improvement (including sndbuf auto-tuning)
- Add more test cases in sctp-tests.
Code Refactor

- Some huge and messy functions.
- Congestion framework.
- Refactor lksctp-tools.
Hardware Support

- GSO x frag_list x frags.
- Checksum.
- Offload.
The end.

Thanks for listening.