Linux Implementation Study of Stream Control Transmission Protocol

Karthik Budigere
Department of Commnunication and Networking
Aalto University, Espoo, Finland
karthik.budigere@aalto.fi

ABSTRACT

The Stream Control Transmission Protocol (SCTP) is a one of the new general-purpose transport layer protocol for IP networks. SCTP was first standardized in the year 2000 as RFC 2960. SCTP is developed in complement with the TCP and UDP transport protocols. SCTP improves upon TCP and UDP and it also introduces new features such as multi-homing and multi-streaming, multi-homing feature provides the fault tolerance and the multi-streaming feature addresses the head-of-line blocking problem. This paper describes the high level details about the SCTP implementation in Linux kernel. It mainly focus on SCTP module initialization and registrations for socket layer abstraction, important data structures, SCTP sate machine and packet flow through SCTP stack.

1. INTRODUCTION

Stream control transmission protocol was designed to overcome the limitations of other transport layer protocols on IP such as TCP and UDP. The first SCTP specification was published in October 2000 by the Internet Engineering Task Force (IETF) Signaling Transport (SIGTRAN) working group in the now obsolete RFC 2960 [5]. Since then, the original protocol specification has been slightly modified (checksum change, RFC 3309 [6]) and updated with suggested implementer's fixes (RFC 4460 [4]). Both updates are included in the current protocol specification, RFC 4960 [3] that was released in September 2007. SCTP is rich in new features and capabilities. The capabilities of SCTP would make it suitable as a general transport protocol.

The following are some of the important new features of the SCTP; SCTP inherits some of the features from TCP along with some new exclusive new features.

- Multihoming: SCTP sends packets to one destination IP address and also has capabilities to reroute the messages using alternate route if the current IP address becomes unreachable. Hence SCTP offers resilience to the failed interfaces and faster recovery during network failures.
- Multi-streaming: SCTP supports multiple simultaneous streams of data with in the same connection. When sending the message systems cannot send parts of the same message through different streams; one message must go through one stream. In an ordered delivery option the stream is blocked when the packets are out of order or missing. Only one stream that is affected will be blocked where as the other streams will continue uninterrupted.

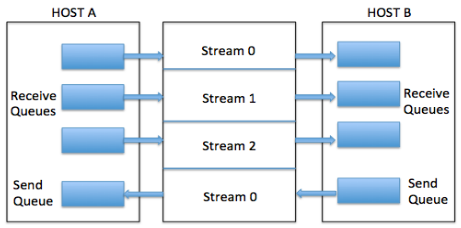

Figure 1: SCTP MultStreaming Association

- Multiple Delivery Modes: SCTP supports delivery modes such as strict order-of transmission (like TCP), partially ordered (per stream), and unordered delivery (like UDP). Message boundary preservation: SCTP preserves applications message boundaries by placing messages inside one or more SCTP data structures, called chunks. Multiple messages can be bundled into a single chunk, or a large message can be spread across multiple chunks.
- Selective acknowledgments. SCTP uses selective acknowledgment scheme, similar to TCP, for packet loss recovery. The SCTP receiver provides SACK to the sender with information regarding the messages to retransmit.
- Heartbeat keep-alive mechanism: SCTP sends heartbeat control packets to idle destination addresses that are part of the association. The protocol declares the IP address to be down once it reaches the threshold of unreturned heartbeat acknowledgments.
- DOS protection. To avoid the impact of TCP SYN flooding like attacks on a target host, SCTP employs a security cookie mechanism during association initialization. User data fragmentation: SCTP will fragment messages to conform to the maximum transmit

unit (MTU) size along a particular routed path between communicating hosts.

The SCTP protocol can be divided into number of functions; these functions are Association startup and shutdown, Sequenced delivery of data within the streams, Data fragmentation, SACK generation and congestion control, chunk bundling, packet validation and path management. All these functions are discussed in detail in the following sections of the paper.

2. IMPLEMENTATION OVERVIEW

2.1 SCTP Initialization

SCTP is implemented as an experimental module in the Linux kernel. All the kernel modules have an initialization and exit functions. The SCTP kernel module initialization function is done in sctp_init(). This function is responsible for the initialization of the memory for several data structures and initialization various parameters used by the SCTP module. It also it performs the registration with socket and IP layer.

The important data structures that are initialized in this function are sctp_bucket_cachep of type struct sctp_bind_bucket, this data structure is used for managing the bind/connect, sctp_chunk_cahep that is of type struct sctp_chunk, this data structure is used to store the SCTP chunks. SCTP chunks are the unit of information within an SCTP packet, a chunk consist of a chunk header and chunk specific content, initializes SCTP mib and proc fs directory, initialize the stream counts and association ids handle, memory allocation and initialization of association and endpoint hash tables that are used in connection management, and initialization of SCTP port hash table.

2.1.1 *Socket Layer Registeration*

The socket layer registration of SCTP is done by the functions sctp_v4_protosw_init() and sctp_v6_protosw_init(). The Linux kernel network subsystem data structures, struct proto which is defined in the file /include/net/sock.h and the struct net_proto_family that is defined in the /include/linux/net.h encapsulates the protocol family implementation. In order to register SCTP to TCP/IP stack (using IP as the network layer) we should Initialize an instance of struct proto and register to Linux network sub-system with call proto_register(). The protocol addition and registration with socket layer inetsw protocol switch will done by the function inet_register_protosw(), this function is defined in the net/ipv4/af_inet.c and takes argument of type proto structure defines the transport-specific methods (Upper layer) such as for connect (sctp_connect()), disconnect (sctp_disconnect()), sending (sctp_sndmsg()) and recieving (sctp_rcvmsg()) message etc, while the proto_ops structure defines the general socket methods.

2.1.2 *IP Layer Registeration*

The IP layer registration or adding new transport protocol is performed in the function sctp_v4_add_protocol(). In this function it calls the function register_inetaddr_notifier in order to register to get notifications that notifies whenever there is addition or deletion of inet address and inet_add_protocol() function to add new protocol SCTP with inet layer. The function takes two arguments, first one is of type struct net_protocol in that we can specify the packet handler routine, for SCTP the packet handler routine is sctp_rcv() function and the second argument is protocol identifier that is IPPROTO_SCTP (132). This way the SCTP packets when received in the IP layer are sent to the sctp_rcv() function.

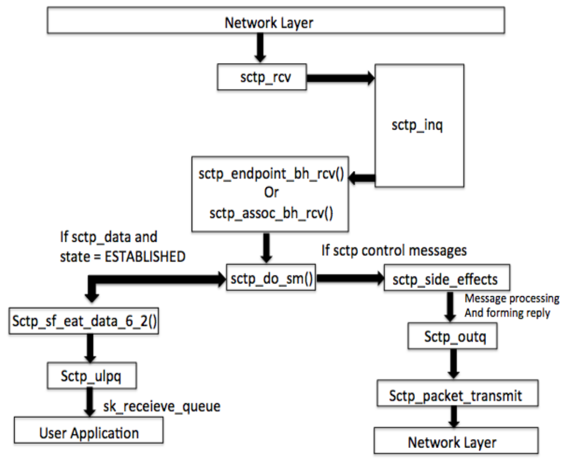

Figure 2: SCTP packet flow from network to userspace

2.2 Packet Flow in SCTP

In this section we present the packet flow discussion of SCTP in the Linux kernel.

2.2.1 *Packet flow from network*

The packet flow from network is as shown in the Fighure 2. The entry point for all the packets from the network layer to the SCTP module is the function sctp_rcv(). This is the function specified as the handler routine during the registeration with Network layer as discussed in the section 2.1.2.

Packets received from the network layer first undergoes the basic checks like checking for the minimum length of the packet received, checking for the out of the blue packets, etc. If the checks fails then the packet is discarded. All the proper packets received are pushed on to the sctp_inq for further processing. The packets are processed by the sctp_endpoint_bh_rcv() or sctp_assoc_bh_rcv() functiosn which schedules packets from the sctp_inq. The processing of the packets are done by the state machine routine sctp_do_sm(). The state machine checks the type of chunks received and process it accordingly. From the network layer we can either receive data or we can receive the SCTP control messages. If we receive data in the ESTABLISHED state then the data is passed on to the user applciation using the sctp_ulpq. If the packet recieved is one of the SCTP control

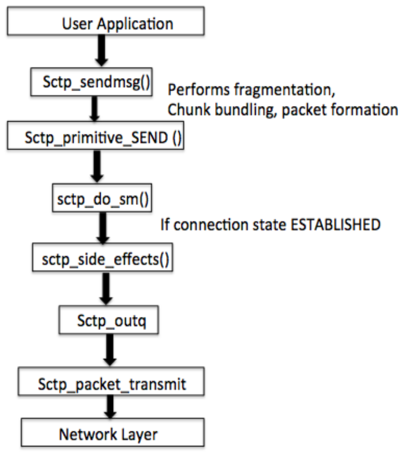

Figure 3: SCTP packet flow from user space

message, then its processed according to the state and suitable action will be taken. If the state machine wants to reply the incoming message, then it forms the reply message and inserts the packet into sctp_outq for the transmission of the reply message to the network layer.

2.2.2 *Packet flow from user space*

The packets sent by the user space is sent to SCTP module using the function sctp_sendmsg(). The packets are checked for the errors, if the packets are free from errors then we call sctp_primitive_SEND() for sendint he packet. The SEND primitive calls the SCTP state machine for performing the suitable actions. The state machine function determines the suitable side effect action to perform and calls the sctp_side_effects() function for the command execution. If the packet is in order and error free then its inserted to the sctp_outq which will be scheduled by the function sctp_packet_transmit() function for transmitting the packet to the network layer.

2.3 Data Structures

The following are some of the important data structures used in the SCTP Linux implementation,

- struct sctp_globals: The entire SCTP module universe is grounded in an instance of struct sctp_globals. This structure holds system wide defaults for things like the maximum number of permitted retransmissions, valid cookie life time, SACK timeout, send and receive buffer policy, several flag variables etc. It contains list of all endpoints on the system, associations on the system and also the port hashtable.

- struct sctp_endpoint: Each UDP-style SCTP socket has an endpoint, represented as a struct SCTP endpoint. The endpoint structure contains a local SCTP socket number and a list of local IP addresses. These two items defines the endpoint uniquely. In addition to endpoint wide default values and statistics, the endpoint maintains a list of associations. This logical sender/receiver of SCTP packets. On a multi-homed host, an SCTP endpoint is represented to its peers as a combination of a set of eligible destination transport addresses to which SCTP packets can be sent and a set of eligible source transport addresses from which SCTP packets can be received.

- struct sctp_association: Each association structure is defined by a local endpoint (a pointer to a struct sctp_endpoint), and a remote endpoint (an SCTP port number and a list of transport addresses). This is one of the most complicated structures in the implementation as it includes a great deal of information mandated by the RFC such as sctp_cookie, counts of various messages, current generated association share key etc. Among many other things, this structure holds the state of the state machine. The list of transport addresses for the remote endpoint is more elaborate than the simple list of IP addresses in the local endpoint data structure since SCTP needs to maintain congestion information about each of the remote transport addresses.

- struct sctp_transport: The struct sctp_transport defined by a remote SCTP port number and an IP address. The structure holds congestion and reachability information for the given address. This is also where we get the list of functions to call to manipulate the specific address family.

- struct sctp_packet: This is the structure which holds the information about the list of chunks along with the SCTP header information. These are getting assembled for the transmission. It has the destination information in the struct sctp_transport in it.

- struct sctp_chunk This is the most fundamental data structure in SCTP implementation. This holds SCTP chunks both inbound and outbound. It is essentially an extension to struct sk_buff structure. It adds pointers to the various possible SCTP subheaders and a few flags needed specifically for SCTP. One strict convention is that chunk->skb->data is the demarcation line between headers in network byte order and headers in host byte order. All outbound chunks are ALWAYS in network byte order. The first function which needs a field from an inbound chunk converts that full header to host byte order. The stucture also holds information about the subheaders present in the chunk and the sctp_transport information which tells source for an inbound chunk and destination for the outbound chunk.

2.4 Queues

There are four different queues in SCTP Linux implementation. They are sctp_inq, sctp_ulpq, sctp_outq, and sctp_packet. The first two carry information up the stack from the wire to the use and the second two carry information back down the stack. Each queue has one or more structures which define its internal data, and a set of functions which define its external interactions. All the queues have push inputs and external objects explictly put things in by calling methods directly. A pull input is there for a queue and it would need to have a callback function so that it can fetch input in response to some other stimulus. These queue definitions are found in net/sctp/structs.h and net/sctp/ulpqueue.h.

2.4.1 sctp_inq

SCTP inqueue accepts packets and provides chunks. It is responsible for reassembling fragments, unbundling, tracking received TSN's (Transport Sequence Numbers) for acknowledgement, and managing rwnd for congestion control. There is an SCTP inqueue for each endpoint (to handle chunks not related to a specific association) and one for each association. The function sctp_rcv() (which is the receiving function for SCTP registered with IPv4) calls sctp_inq_push() to push packets into the input queue for the appropriate association or endpoint. The function sctp_inq_push() schedules either sctp_endpoint_bh_ rcv() or sctp_assoc_bh_rcv() on the immediate queue to complete delivery. These functions call sctp_inq_pop() to pull data out of the SCTP inqueue. This function does most of the work for this queue. The functions sctp_endpoint_bh_ rcv() and sctp_assoc_bh_rcv() run the state machine on incoming chunks. Among many other side efiects, the state machine can generate events for an upper-layer-protocol (ULP), and/or chunks to go back out on the wire.

2.4.2 sctp_ulpq

sctp_ulpq is the queue which accepts events (either user data messages or notifications) from the state machine and delivers them to the upper layer protocol through the sockets layer. It is responsible for delivering streams of messages in order. There is one sctp_ulpq for every association. The state machine, sctp_do_sm(), pushes data into an sctp_ulpq by calling sctp_ulpq_tail_data(). It pushes notifications with sctp_ulpq_tail_event(). The sockets layer extracts events from an sctp_ulpq with message written sk_data_ready() function sk_buff.

2.4.3 sctp_outq

sctp_outqueue is responsible for bundling logic, transport selection, outbound congestion control, fragmentation, and any necessary data queueing. It knows whether or not data can go out onto the wire yet. With one exception noted below, every outbound chunk goes through an sctp_outq attached to an association. The state machine injects chunks into an sctp_outqueue with sctp_outq_tail(). They automatically push out the other end through a small set of callbacks which are normally attached to an sctp_packet. The state machine is capable of putting a fully formed packet directly on the wire.

2.4.4 sctp_packet

An SCTP packet is a lazy packet transmitter associated with a specific transport. The upper layer pushes data into the packet, usually with sctp_packet_transmit(). The packet blindly bundles the chunks. It transmits the packet to make room for the new chunk. SCTP packet rejects packets which need fragmenting. It is possible to force a packet to transmit immediately with sctp_packet_transmit(). sctp_packet tracks the congestion counters, but handles none of the congestion logic.

2.5 Multihoming in SCTP

There are tow ways to work with multihoming with SCTP. One way is to bind all your addresses through the use of INADDR ANY or IN6ADDR_ANY. This will associate the endpoint with the optimal subset of available local interfaces. The second way is through the use of sctp_bindx(), which allows additional addresses to be added to a socket after the first one is bound with bind(), but before the socket is used to transfer or receive data. The multihoming is implemented in the function sctp_setsockopt_bindx() function. This function takes a argument op which specifies whether to add or remove the address from association. The binding completes by Sending an ASCONF (Address Configuration Change Chunk) chunk with Add IP address parameters to all the peers of the associations that are part of the endpoint indicating that a list of local addresses are added to the endpoint. If any of the addresses is already in the bind address list of the association,than we do not send the chunk for that association. But it will not affect other associations. The associtions are created on the successful reception of the ASCONF_ACK chunk.

3. STATE MACHINE, ALGORITHMS AND OPTIONS

3.1 State machine

The state machine in SCTP implementation is quite literal. SCTP implementation has an explicit state table which keys to specific state functions which are tied directly back to parts of the RFC. The core of the state machine is implementaed in the function sctp_do_sm().
Each state function produces a description of the side effects (in the form of a struct sctp sm_retval) needed to handle the particular event. A separate side effect processor, sctp_side_effects(), converts this structure into actions.
Events fall into four categories. The first category is about the state transitions associated with arriving chunks. The second category is the transitions due to primitive requests from upper layers, Not defined completely in the standards so its implementation specific. The third category of events is

Table 1: Partial SCTP state machine table

Message	CLOSED	COOKIE_WAIT	COOKIE_ECHOED	ESTABLISHED
SCTP_DATA	sctp_sf_ootb	Discard	Discard	sctp_sf_eat_data_
INIT	sctp_sf_do_5_1B_init	sctp_sf_do_5_2_1_siminit	sctp_sf_do_5_2_1_siminit	sctp_sf_do_5_2_2
INIT_ACK	sctp_sf_do_5_2_3_initack	sctp_sf_do_5_1C_ack	Discard	Discard
COOKIE_ECHO	sctp_sf_do_5_1D_ce	sctp_sf_do_5_2_4_dupcook	sctp_sf_do_5_2_4_dupcook	sctp_sf_do_5_2_4
COOKIE_ECHO_ACK	Discard	Discard	sctp_sf_do_5_1E_ca	Discard
SCTP_HEARTBEAT	sctp_sf_ootb	Discard	sctp_sf_beat_8_3	sctp_sf_beat_8_3
SCTP_HEARTBEAT_ACK	sctp_sf_ootb	sctp_sf_violation	Discard	sctp_sf_backbeat_

timeouts. The final category is a catch all for odd events like queues emptying. In order to create an explicit state machine, it was necessary to first create an explicit state table. Table 1 shows the partial state machine table with functions for different kind of chunks during different states.

3.1.1 *SCTP connection Initiation*

As SCTP and TCP are both connection oriented,they require communications state on each host. A TCP connection is defined by two IP addresses and two port numbers. Given two hosts, A and Z, a TCP connection is defined by [IP-A]+[Port-A]+[IPZ]+[Port-Z] where IP-A and Port-A are one end of the connection and IP-Z and Port-Z are the other. An SCTP association is defined as [a set of IP addresses at A]+[Port-A]+[a set of IP addresses at Z]+[Port-Z]. Any of the IP addresses on either host can be used as a source or destination in the IP packet and still properly identify the association. Before data can be exchanged, the two SCTP hosts must exchange the communications state (including the IP addresses involved) using a fourway handshake, TCPâĂŹs three-way handshake, a four-way handshake eliminates exposure to the aforementioned TCP SYN flooding attacks. The receiver of the initial (INIT) contact message in a four-way handshake does not need to save any state information or allocate any resources. Instead, it responds with an INIT-ACK message, which includes a state cookie that holds all the information needed by the sender of the INIT-ACK to construct its state. The state cookie is digitally signed via a mechanism. Both the INIT and INIT-ACK messages include several parameters used in setting up the initial state:

- A list of all IP addresses that will be a part of the association.
- An initial transport sequence number that will be used to reliably transfer data.
- An initiation tag that must be included on every inbound SCTP packet.
- The number of outbound streams that each side is requesting.
- The number of inbound streams that each side is capable of supporting.

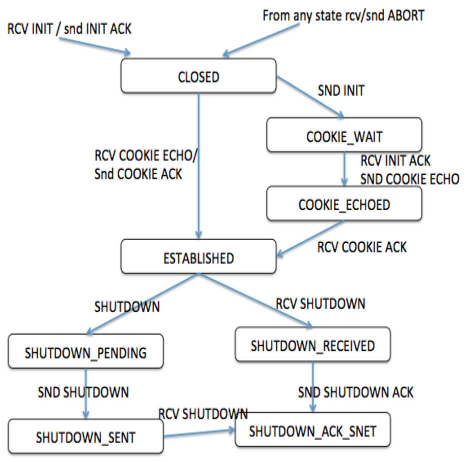

Figure 4: SCTP State Machine

After exchanging these messages, the sender of the INIT echoes back the state cookie in the form of a COOKIE-ECHO message that might have user DATA messages bundled onto it as well (subject to pathMTU constraints). Upon receiving the COOKIEECHO, the receiver fully reconstructs its state and sends back a COOKIE-ACK message to acknowledge that the setup is complete. This COOKIE-ACK can also bundle user DATA messages with it.

3.1.2 *SCTP Shutdown*

A connection-oriented transport protocol needs a graceful method for shutting down an association. SCTP uses a three-way handshake with one difference from the one used in TCP: A TCP end point can engage the shutdown procedure while keeping the connection open and receiving new data from the peer. SCTP does not support this âĂIJhalf closedâĂİ state, which means that both sides are prohibited from sending new data by their upper layer once a graceful shutdown sequence is initiated. In this example, the application in host A wishes to shut down and terminate the association with host Z. SCTP enters the SHUTDOWN_PENDING state in which it will accept no data from the application but will still send new data that is queued for transmission to host Z. After acknowledging all queued data, host A sends a SHUTDOWN chunk and enters the SHUTDOWN_SENT state. Upon receiving the SHUTDOWN chunk, host Z notifies its upper layer, stops accepting new data from it, and enters the SHUTDOWN_RECEIVED state. Z transmits any remaining data to A, which follows with subsequent SHUTDOWN chunks that inform Z of the dataâĂŹs arrival and reaffirm that the association is shutting down. Once it a host A sends a subsequent SHUTDOWN_ACK chunk, followed by a SHUTDOWN_COMPLETE chunk that completes the association shutdown.

4. CONCLUSION

Despite a considerable amount of research, SCTP still lacks a killer application that could motivate its widespread adoption into the well-established IP networks protocol stack. Hence, SCTP is still not part of the vendor-supplied TCP/IP stack for widespread OSes[1]. One of the important milestone towards a broader adoption of SCTP was the decision within the mobile communications industry to select SCTP as a transport protocol for the Long Term Evolution (LTE) networks to support signaling message exchange between network nodes. SCTP is also the key transport component in current SIGTRAN suites used for transporting SS7 signaling information over packet-based networks. Hence, SCTP is used in progressively adopted Voice over IP (VoIP) architectures and thus becomes part of related signaling gateways, media gateway controllers, and IP-based service control points that are used to develop convergent voice and data solutions.[2]

5. REFERENCES

[1] Lukasz Budzisz, Johan Garcia, Anna Brunstrom, and Ferr. A taxonomy and survey of sctp research. *ACM Comput. Surv.*, 44(4):18:1–18:36, September 2012.

[2] Preethi Natarajan, Janardhan R. Iyengar, Paul D. Amer, and Randall Stewart. Sctp: an innovative transport layer protocol for the web. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 615–624, New York, NY, USA, 2006. ACM.

[3] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007.

[4] R. Stewart, I. Arias-Rodriguez, K. Poon, A. Caro, and M. Tuexen. Stream Control Transmission Protocol (SCTP) Specification Errata and Issues. RFC 4460 (Informational), April 2006.

[5] R. Stewart, Q. Xie, K. Morneault, and C. Sharp. RFC 2960, Stream control transmission protocol. http://www.faqs.org/rfcs/rfc2960.html, October 2000.

[6] J. Stone, R. Stewart, and D. Otis. Stream Control Transmission Protocol (SCTP) Checksum Change. RFC 3309 (Proposed Standard), September 2002. Obsoleted by RFC 4960.