<u>RivuS</u>

Stream Control Transmission Protocol (SCTP) on FreeBSD

<u>Project By</u>:

Jayesh V. Rane

Nitin N. Kumbhar

Kedar S. Sovani

PICT.

<u>Guidance</u>:

Prof. Rajesh B. Ingle, PICT.

Mr. Aditya Kini, Calsoft Pvt. Ltd.

Pune Institute of Computer Technology

Affiliated to University of Pune

CERTIFICATE

This is to certify that the project

RivuS

Stream Control Transmission Protocol (SCTP) on FreeBSD

Has been successfully completed by:

Mr. Jayesh V. Rane (B2054310)
Mr. Nitin N. Kumbhar (B2054270)
Mr. Kedar S. Sovani (B2054321)

of BE Computers towards the partial fulfillment of the Bachelors Degree in Computer Engineering from University of Pune for the Academic Year 2001-2002

Prof. Rajesh Ingle	Prof. Dr. CVK Rao
Internal Project Guide,	Head of the Department,
Dept. of Comp. Engg.	Dept. of Comp. Engg.
PICT, Pune.	PICT, Pune.

ACKNOWLEDGEMENT

Executing a research project involves quite a lot of effort. We take this opportunity to thank all those people who have been instrumental in making our project successful.

We would like to thank our Head of Department, Dr. CVK Rao for providing us with necessary computing resources and facilities. We also extend our sincere thanks to Mr. Adityashankar Kini, Mr. Anshuman Kanetkar and Dr. Anupam Bhide, CEO, Calsoft Pvt. Ltd. for their perseverance, eagerness to help us in all possible ways.

We are grateful to Prof. Rajesh Ingle, our guide from Pune Institute of Computer Technology for his undivided attention and support without which this project could not have been completed.

We are thankful to Mr. Randall Stewart, Cisco Inc. and Mr. Louis Gerberge, Apple Computer Inc. for providing us help during progress of RivuS. Also we are grateful to Mr. M. McKusick, University of Berkeley, California from providing us help on FreeBSD Network stack.

Also we would like to thank all our colleagues for all the support and encouragement that they have extended to us.

The RivuS Team.

ABSTRACT

Our project, RivuS, involves the entire implementation of SCTP protocol in the 4.3 FreeBSD Network Stack [10][11], along with the implementation of our idea of Load Sharing, proposed as an extension to SCTP.

SCTP is Stream Control Transmission Protocol, proposed in RFC 2960. It has remarkable advantages over TCP. The main features of SCTP are MultiStreaming, Multihoming, Resistance to DOS attacks, and Partial ordered delivery among others. Our implementation is in the form of a kernel module and provides TCP-style socket APIs.

We are exploiting the Multihoming feature of SCTP to get high performance gains. In the Path Aggregation concept, we multiplex the data over multiple paths, which exist between the two endpoints. The multiplexing of data is done after considering the static and dynamic characteristics of each path. We claim that Load Sharing gives much better Performance than using normal TCP.

During this research work on Multihoming feature of SCTP, we had submitted a Research Paper 'Exploiting Multihoming Feature of SCTP for High Performance' at International USENIX Conference 2002, California, USA. FreeNIX Conference for Poster Session at California had invited us.

INDEX

1. Introduction:

The Internet has grown manifold in the last few years. There are millions and millions of clients and servers using this infrastructure for a variety of applications. The impact of this phenomenal growth has had some unexpected and worrisome developments. The first important development of this growth is that it has led to many new and complex network services in addition to the traditional network services. These services and applications have more advanced and demanding requirements compared to the traditional services.

1.1 Our Approach and Contributions

1.1.1 Approach

As more and more data transfer intensive applications find their way to the Internet, the demand for high data transfer rates is constantly rising. Thus, performance has become a critical issue.

For this reason, the transport layer protocols should be capable enough to give enhanced performance. Stream Control Transmission Protocol (SCTP) [1] is a reliable protocol operating on the top of a potentially unreliable connectionless packet service such as IP. SCTP is more advantageous than existing Transport Layer protocol i.e. Transmission Control Protocol (TCP)[2] because of its acknowledged error-free non-duplicated transfer of data.

1.1.2 Contribution

The main contributions of our research work are

- We have proposed new extension to SCTP as ‘Path Aggregation’
- We have implemented SCTP as the kernel module for 4.3 FreeBSD Operating System.
- We have also implemented SCTP as a user space (and partially kernel space) library in the 4.3 FreeBSD Operating System [5].

Using our extension, ‘Path Aggregation’, we are multiplexing data over the different paths between two ends and gaining better performance. Path Aggregation concept is explained in detail in section 9. The performance figures using RivuS Path

Aggregation implementation shows that we are gaining tremendously. And this implementation is far better than other current solution available, like Channel Bonding [3], IP Multipathing [4].

1. What is RivuS? :

RivuS is a Latin word for “Stream”. SCTP has two core features as, MultiStreaming and MultiHoming.

MultiStreaming means that within the connection we have multiple streams of packets that can be sent or received on either end. In TCP, in-order sequence data transfer takes place through only one stream, so there is possibility of Head of Line Blocking, which degrades performance of protocol. Using the SCTP we are avoiding this drawback. As the protocol has message-oriented nature for data transfer, there will be always restriction on the message boundaries i.e. every time the message boundaries will be preserved. So you can bundle multiple streams within the same connection.

Second feature of SCTP is MultiHoming, the node having multiple Network Interface cards on it, is called as MultiHomed node. During Network failures, we can actually transfer the load from one interface card to other interface card, if the former fails to provide service. Using this feature of SCTP, we are providing an extension to SCTP as Path Aggregation.

Another feature of SCTP is Resistance to DOS attack. In TCP, we have Three-way handshaking for the association purpose but here in the SCTP association, we have Four-way handshaking between two ends. The mechanism uses encryption algorithms such as MD5 for the purpose.

3. Purpose:

TCP has performed immense service as the primary means of reliable data transfer in IP networks. However, an increasing number of recent applications have found TCP too limiting, and have incorporated their own reliable data transfer protocol on top of UDP.

SCTP is a reliable transport protocol operating on top of a connectionless packet network such as IP. It offers the following services to its users:

a) Acknowledged error-free non-duplicated transfer of user data

b) Data fragmentation to conform to discovered path MTU size

c) Sequenced delivery of user messages within multiple streams, with an option for order-of-arrival delivery of individual user messages

d) Optional bundling of multiple user messages into a single SCTP packet

e) Network-level fault tolerance through supporting of multi- homing at either or both ends of an association.

4. Scope:

The RivuS has to offer two functionalities, explained in following sections.

4.1 SCTP implementation:

The protocol has been defined in the RFC 2960. The implementation [7] strictly follows the RFC, as far as the features enlisted below are considered.

The following features of the protocol are implemented:

Association establishment

- Data transfer
- Association shutdown
- Multi-streaming
- Multi-homing
- Retransmissions and time-outs
- Packet bundling
- Congestion control
- Fragmentation

4.2 Path Aggregation extension:

The Path Aggregation extension multiplexes data over multiple paths that exist between two multi-homed hosts. MultiHomed host means the host having multiple network interface cards (NICs). Thus all these paths will be used in the data transfer, and multiplexing of data will be done depending on the static and dynamic characteristics of these paths i.e. RTT, cwnd, PMTU [10].

Hence the Path Aggregation extension is capable of providing high data transfer rates to user applications. Furthermore, the user can have an option regarding whether to use this feature or not.

5. Specific Requirements

5.1 Protocol requirements:

TCP has two major drawbacks that the new protocol should avoid. The new protocol should meet the following requirements,

- There should be support for path fail over,
- The unnecessary head of line blocking should be avoided.

Using a protocol that supports multi-homing and multi-streaming respectively can fulfill these two requirements. This can be achieved by the use of SCTP.

5.2 User Interface:

This being an implementation of a protocol, it should provide user-protocol interface analogous to the socket layer. This will facilitate in ease of use for the user. The primitives provided to the user should access all the features of the protocol, explained in section 8.4.

5.3 System Interface:

The implementation should use standard protocol-protocol interfaces provided by the 4.3FreeBSD operating system to communicate with the IP layer, explained in section 8.5.

5.4 Communications Interfaces:

The SCTP implementation installed on two endpoints will be using the SCTP protocol for the communication purposes. The protocol used should be conformant to RFC2960.

6. Test plan

Implementation of the protocol is as kernel module and user space library for 4.3 FreeBSD. Performance testing done according to this plan for library implementation will give less effective figures.

Testing goal for this implementation is to make this transport layer protocol suitable to be used in the Internet world.

6.1 Features to test

The parts of the SCTP protocol, which we have tested, are divided into following parts

1. Primitives provided

The implementation provides primitives [6] analogous to socket layer primitives. All the functions will serve, as primitives will be tested. Functions sctpSocket, sctpBind, sctpConnect, sctpListen, sctpAccept are used in association creation. They are tested for handling of data structures required in SCTP association during association creation.

sctpSend & sctpReceive are the functions used in data transfer. They are tested for data transfer on different streams within the association.

sctpShutdown, sctpClose are used in shutdown process. SCTP protocol provides two ways (graceful and ungraceful) of shutting down the association. The processes of closing the association are tested for both the cases with different test cases.

2. Association (Connection) establishment

In this the four way handshake used for association creation are tested. Cookie mechanism used for security purpose during association creation is the main point.

3. Data transfer

Data transfer between to hosts is tested here. Testing for delivery of messages in their entirety on proper stream within the association is done.

4. Timer handling

    Here test is performed for retransmissions of packets containing data or control information. Special software is used to emulate the WAN during testing.

5. Shutting down association

    The three way handshake used in graceful shutdown is tested. In this testing the delivery of remaining data in transmission queue is checked.

6. Path fail over

    The Multihoming feature of SCTP protocol is tested here. Properties of path like packet loss, delay, bandwidth, noise decide failure of the path. So to emulate all these properties of path special software like dummynet in FreeBSD is used.

7. Path Aggregation extension

    Testing for this proposed extension to protocol would include testing of algorithm to give priority to paths, amount of data transferred on each path.

6.2 Types of Testing

Following types of testing is used in testing of project.

6.2.1 Unit testing

This type of testing is used to test the modules that are made up of components like functions or some part of functions. Modules can be categorized as below.

a. Kernel component

    This is the only part of implementation that is in kernel. For testing of this module driver programs are written that test data structures used in kernel, code flow, and other functions used.

b. Sctp input processing

    This part of the protocol does all input processing. It is tested for handling of different chunks in which it will take appropriate action. Reception of data according to streams is tested.

c. Sctp output processing

Sctp output is the only outlet from the protocol to others. Testing for this will include sending appropriate chunks according to state of association, handling of transmission queues, and other data structures.

d. Sctp primitives

All the primitives are tested for maintaining proper state of associate on, data handling in case of send and receive primitives.

6.2.2 Bottom-up integration testing

Integration testing is done from bottom to up. Kernel modules form a cluster. Driver programs use this cluster. These driver programs are replaced by cluster of user space sctp input and output processing module and other driver programs that simulate primitives provided.

Finally all the tested clusters are combined together for testing of whole software.

6.2.3 System testing

Recovery testing

Path fail over mechanism is tested under this recovery testing. One of the many paths used in association is subjected to fail. For this special software like dummynet is used. Path fail over mechanism automatically detects the path failure and switch all the traffic on that failed path to one of the active path in association.

6.2.4 Stress testing

In this type of testing working of protocol is tested for large amount of data transfer. Test cases check stress on memory in case of data sender and receiver. Reception of data packets at high rates at receiver, gives processing ability of input component.

6.3 Performance testing

Performance of the protocol is tested for amount of time taken for data transfer. Tests are conducted with different network conditions like low bandwidth, high delay, and packet loss.

6.4 Test Case Specification (TCS)

Since this is an implementation of a protocol, directly programs, which are using socket layer primitives, are used as test cases to check working of protocol. These test cases test all the type of communication between hosts. These include test cases for data transfer on different streams, associations with many addresses involved in it; simultaneous data transfer between two hosts, among others.

All test cases must be executed in WAN environment. For this purpose software called dummynet is used. It is special software to handle the properties of the network paths are created.

Five different scenarios are used for testing. These Five scenarios will simulate the WAN environment. These are as follows.

6.4.1 No transmission errors and no delay

Test case 1:

Goal: Association setup with one and two addresses in the INIT and INIT-ACK chunks.

Procedure: Run test programs with one or two addresses used by bind system call.

Expected result: A connection gets successfully established.

Test result: Same as expected.

Test case 2:

Goal: Association setup with a wide range of the number of streams to test negotiation

Procedure: Run test programs with a wide range of inbound (IS) and Outbound (OS) streams

Expected result: A connection gets established if the no of inbound streams of one host is equal to the no of outbound steams of other host and no of outbound streams of former is equal to the no of inbound streams of later. Otherwise connection is not established.

Test Result: Same as expected.

Test case 3:

Goal: Data transfer on (at least) two streams with ordered and unordered delivery.

Procedure: Run test program which emulates the streaming applications.

Expected result: Within each stream data is correctly delivered.

Test Result: Same as expected.

Test case 4:

Goal: Association shutdown.

Procedure: Run test program which shuts down an established connection.

Expected result: The connection is closed with proper handshake. All the memory allocated for TCB, TABs, among others is freed.

Test Result: Same as expected.

Test case 5:

Goal: Path usage: Observe the DATA flow in a single and multi-homed environment. specially the primary path should be changed if one PATH goes down.

Procedure: Run test program which uses at least two addresses for DATA transfer

Expected result: DATA transfer happens on the primary path that is the best path among the others. If the primary path fails the DATA transfer is shifted to other available beat path transparent to application.

Test Result: Same as expected.

Test case 6:

Goal: Association handling in case of no available paths.

Procedure: Run test program. Subject all the paths available between the two hosts to fail using the dummynet.

Expected result: Protocol tries to retransmit the data for maximum times as configured. If it is not able to send the data to other host it closes the association giving an error to application.

Test Result: Same as expected.

6.4.2 No transmission errors and delay

Test case 7:

Goal: Association setup with large delays (cookie lifetime).

Procedure: Run test program. Set some delay for paths between the to hosts using dummynet.

Expected result: Protocol checks the time taken by cookie to return. If the cookie arrives Late, in time more then the cookie lifetime, then stale cookie error is sent to other end to reject the connection.

Test Result: Same as expected.

Test case 8:

Goal: handle INIT collision correctly.

Procedure: Run test program. Set some delay for paths between the to hosts using dummynet.

Expected result: Protocol checks state of the association and takes appropriate action.

Test Result: Same as expected.

Test case 9:

Goal: Reordering of out of sequence arrived data chunks.

Procedure: Run test program. Set some delay for paths between the to hosts using dummynet.

Expected result: Protocol checks whether DATA is in sequence or not. If DATA is in sequence then it is directly delivered to application. If DATA is not in sequence then it temporarily stores DATA in buffers and makes an entry into a map used to manage unordered DATA. When the map is full it directly drops the DATA.

Test Result: Same as expected.

Test case 10:

Goal: Immediate delivery of datagrams marked for out of sequence delivery.

Procedure: Run test program.

Expected result: Protocol delivers the DATA that is out of sequence directly to the application.

Test Result: Same as expected.

Test case 11:

Goal: Association shutdown (SHUTDOWN collision).

Procedure: Run test program.

Expected result: Whenever application calls SHUTDOWN system call to close established connection, SHUTDOWN chunk is sent to peer end only if some host haven’t yet received SHUTDWN from peer end.

Test Result: Same as expected.

6.4.3 Transmission errors and no delay

Test case 12:

Goal: Association setup (loss of chunks used in association setup handshake).

Procedure: Run test program.

Expected result: The ICS timer is set. If the timer expires then considering the state of association, retransmit the corresponding chunk. If retransmissions happen maximum times as defined by protocol then give an error to upper layer that it can not connect to the peer host.

Test Result: Same as expected.

Test case 13:

Goal: Retransmission mechanism based on Retransmission timer (exponential back-off).

Procedure: Run test program.

Expected result: When DATA is transmitted to peer end the retransmission timer is set. If the timer expires DATA is retransmitted and again timer is set with exponential backed off value. If the retransmissions happen maximum times as defined by protocol the DATA transfer is shifted to other available path.

Test Result: Same as expected.

Test case 14:

Goal: Shutdown of association (loss of chunks used in association shutdown handshake).

Procedure: Run test program.

Expected result: The ICS timer is set. If the timer expires then considering the state of association, retransmit the corresponding chunk. If retransmissions happen maximum times as defined by protocol then close connection without handshake.

Test Result: Same as expected.

6.4.4 Transmission errors and delay

Test case 15:

Goal: Association setup (loss and collisions of chunks used in association setup handshake).

Procedure: Run test program.

Expected result: The ICS timer is set. If the timer expires then considering the state of association, retransmit the corresponding chunk. If retransmissions happen maximum times as defined by protocol then give an error to upper layer that it can not connect to the peer host. If collisions happen then Protocol checks state of the association and takes appropriate action.

Test Result: Same as expected.

Test case 16:

Goal: Retransmissions based on timer rules.

Procedure: Run test program.

Expected result: Retransmission timer is started, stopped or restarted according to received SACK.

Test Result: Same as expected.

Test case 17:

Goal: Shutdown of association (loss and collisions of chunks used in association shutdown handshake).

Procedure: Run test program.

Expected result: The ICS timer is set. If the timer expires then considering the state of association, retransmit the corresponding chunk. If retransmissions happen maximum times as defined by protocol then close connection without handshake.

Test Result: Same as expected.

6.4.5 Security

Test case 18:

Goal: Cookie mechanism

Procedure: Run test program. Trap a SCTP packet containing INIT ACK or COOKIE ECHO chunk. Modify some parameter like IP address in COOKIE as if some intruder is modifying it.

Expected result: Connection is rejected.

Test Result: Same as expected.

Test case 19:

Goal: CRC32 checksum

Procedure: Run test program. Trap a SCTP packet. Modify some bits from the packet so that this will give different checksum than previous one.

Expected result: packet is dropped

Test Result: Same as expected.

6.4.6 Other Tests

Test case 20:

Goal: Two or more simultaneous connections between two hosts

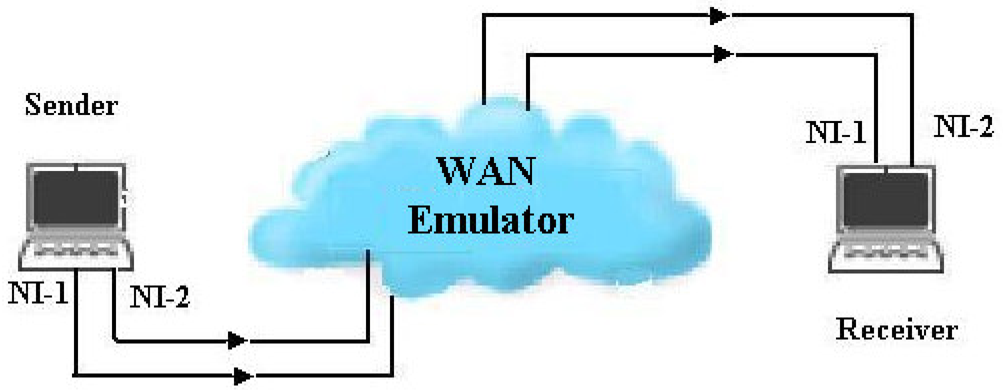
Procedure: Run test program.

Expected result: Demultiplexing of data is done properly according to 5-tuple used to Identify a connection.

Test Result: Same as expected.

6.5 Resources

Following setup is used for the testing purpose.



The setup contains three machines. One act as WAN emulator on which DUMMYNET [9] is running to emulate WAN. It will control the bandwidth, delay, and packet loss on a path.

Sender and receiver are communicating with each other using transport layer SCTP protocol. Test cases are used to check the data transfer between endpoints.

7. Software Quality plan

7.1 Reviews

Reviews and tests are used to ensure the software quality. Reviews are conducted after each milestone. Tests are conducted according test plan to ensure the required quality of the protocol implementation.

Software requirements review:

Software requirements for this implementation of protocol are specified in the requirement specification document. The reviews of these requirements are done to meet the software quality according to standard.

Preliminary design review:

The high level design of the software is reviewed. At any state of the software development the review of the high level design removes some errors in it, deviations in previous from design.

Critical design review

The low level design of the software gives details in depth. The review of this ensures that there is no deviation from previous plan. It may collect some statistics that gives measure of the software quality.

7.2 Tests

Testing of the implementation is done in such a way that it will meet all the requirements of protocol. Also performing tests of many types ensures software quality.

Following types of testing are conducted to ensure the quality of the software.

1. Unit testing

This ensures that algorithms used in software are working properly. Quality degradation due to some logical errors, improper working of software at boundary conditions is reduced.

2. Integration testing

This type of testing ensures that the software will work properly after integration of many components like functions into one component like a major function.

3. System testing

System testing ensures the quality of the software in many ways. It checks working of the software during recovery of it from a failure like path failure. Software must pass the test cases made for security testing in cookie mechanism. Performance testing ensures the quality of the protocol so that it will be used as a transport layer.

8. Estimation & schedule

Date	Phase	Description	Date Completed
16/08/2001	Start	Beginning of the Project Work	-
25/08/2001	Requirement Analysis	Literature Survey, Feasibility Reports.	27/8/2001
01/11/2001	Project Study	Study of the protocols SCTP (RFC 2960), TCP (RFC 0796), FreeBSD kernel internals, and FreeBSD kernel TCP/IP implementation, Interaction with the socket and IP layer.	01/11/2001
10/11/2001	Paper Work	Drafting of the paper to be submitted to USENIX.	12/11/2001
20/12/2001	Design	Design of the entire SCTP implementation.	23/12/2001
05/01/2002	i) Coding ii) Design	i) Completion of the coding for the implementation ii) Design of the Path Aggregation extension	10/01/2002 08/01/2002
30/01/2002	i) Testing ii) Coding & Testing	i) Testing of the SCTP implementation.(Multi-homing not included). ii) Coding and testing of the Path Aggregation extension	31/01/2002 31/01/2002
07/02/2002	Testing	Testing of the multi-homing feature.	07/02/2002
08/02/2002	Coding	Beginning of coding in the	

		kernel space. Application.	09/02/2002
20/02/2002	i) Coding ii) Application	i) Conversion of the user space implementation into a kernel module. ii) searching, designing and coding of the application. (design and coding applicable only if writing application from scratch)	20/02/2002
05/03/2002	Testing	i) Testing of the kernel module. ii) Testing of the application	05/03/2002
12/03/2002	Completion	End of project Work	12/03/2002

9. Literature Survey:

9.1 SCTP Implementation:

This implementation of the SCTP protocol is for the 4.3FreeBSD operating system. There is no full-fledged implementation of SCTP in BSD as of date. There is only one implementation available in the FreeBSD. This implementation supports UDP style APIs and is not totally conformant to the RFC 2960.

9.2 Path Aggregation extension:

A concept similar to the Path Aggregation extension is used by Channel Bonding and Sun Microsystems' IP Multipathing.

Channel bonding requires hardware changes for its functioning across multiple networks. Channel Bonding works at the data link layer. Thus when it is being used to span networks it needs the switches and other hardware to be aware of Channel Bonding. Also Channel Bonding does not have any statistics as Path Aggregation has since it is at the Transport Layer.

IP Multipathing exists on Solaris 8.0 © Sun Microsystems. IP Multipathing is a proprietary product of the Sun Microsystems. No openly available documents for design or implementation of the same are available. On the other hand, Path Aggregation is available as design and implementation under Open Source.

10. High Level Design:

10.1 Library Design:

10.1.1 Design Architecture:

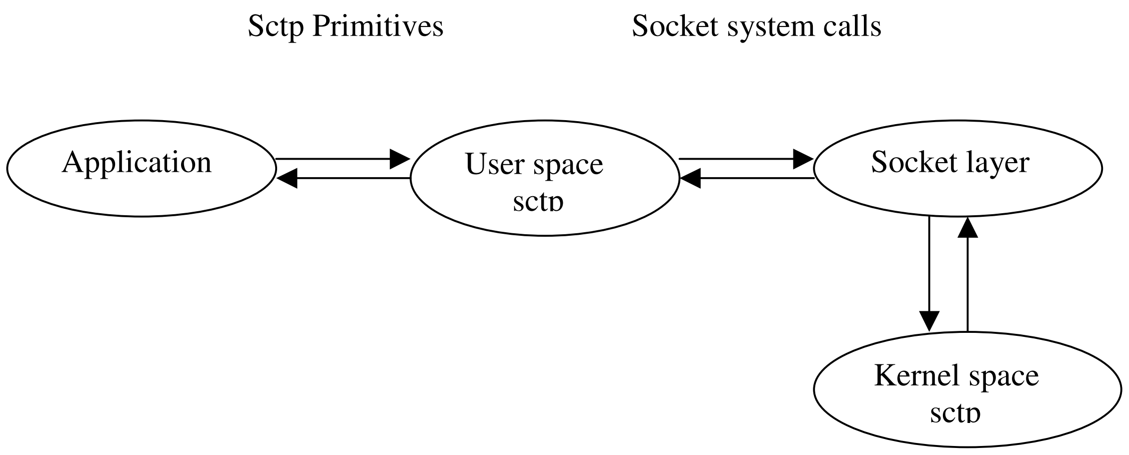
Our implementation is kernel space as well as user space implementation. The part in the kernel will be supporting the user space part to implement the protocol.

In our design we keep the kernel part as a trivial part, which in most cases will not be manipulating data, but will act as placeholder. The part in the kernel is defined as connection-oriented.

We provide the user with a set of primitives, as we will see in this section soon. Our user space part does most of the data manipulation defined by the protocol, and finally maps these primitives to existing socket system calls, which in turn calls the functions that we have provided in the kernel.

The Protocol Control Block for SCTP is kept in the user space, whereas the one for Internet PCB will reside in the kernel itself. All the buffers (retransmission queues, receive buffers) are also be a part of the user space.

The design is analogous to the design of the TCP used in the kernel.



10.1.2 Modification to kernel data structure:

We use the existing inpcb for the protocol. The fields local address and foreign address, in our case are not needed instead these should be replaced by lists for the corresponding, so, we have modified a lot of functions in the kernel like in_pcblookup among others. We majorly use the inpcb for demultiplexing of data that has arrived at each socket. To enable us use the existing demultiplexing function, we use the field to store the 'Initiate Tag', which is unique per association. Thus, this also participates in the demultiplexing, but in a manner transparent to the demultiplexing function.

We modify the current sockaddr_in structure to facilitate the storage of lists of addresses. This we do by the use of the extra 8 bytes that are available in the socket address structure. We use 4 bytes of these for the pointer to next address structure

10.1.3 Kernel Functions:

Following is a brief description of the sequence in which the different actions take place.

sctp_usrreq:

The SCTP user request function has the following commands implemented.

1. PRU_ATTACH: -

    soreserve (allocate socket)

    in_pcballoc (allocate internet PCB)

2. PRU_BIND: -

    call in_pcbbind (to bind the local addresses to the inpcb, if no port is specified an ephemeral port will be allocated)

3. PRU_LISTEN: -

    if port is not assigned one is assigned by call to in_pcbbind (NULL argument for ephemeral port)

4. PRU_CONNECT: -

get ephemeral port if not one.

in_pcbconnect (Here we pass the Initiate tag to the in_pcbconnect function as the foreign address.

soisconnected.

5. PRU_SEND: -

call sctp_output with the mbuf received.

m_freem to free the mbuf (check this ***)

6.PRU_DISCONNECT: -
7.PRU_DETACH: -
8. PRU_ABORT: -

in_pcbdetach

soisdisconnected

9. PRU_ACCEPT: -
10. PRU_RCVD: -

Nil

sctp_input:

a) in_pcblookup to get the PCB to which data belongs. There can be a problem in terms of the sockets wishing to 'accept'. (If there is no INADDR_ANY there while the socket is accepting we can do so in the call to PRU_ACCEPT, or PRU_LISTEN.
b) Traverse thru each chunk in the packet and see if there is INIT or COOKIE_ECHO, if not append the packet to the data buffer and goto(d) , or else goto (c)
c) If INIT or COOKIE_ECHO then check if the state of the socket is SO_ACCEPTCONN, and only if so, then call sonewconn, then soisconnected and

append data to the new sockets buffer. If the socket is not in SO_ACCEPTCONN then drop the packet.

d) Write the new Verification Tag and the fport to the inpcb.

e) return

sctp_output:

call ip_output with the packet and with flag IP_RAWOUTPUT.

10.1.4 Primitives for User-Protocol Interface :

We provide the following primitives along with the corresponding responsibilities

<u>Underlined parameters are the return values.</u>

1. sctpSocket (domain, type, protocol)
    a) call "socket" (internally calls PRU_ATTACH),get sockfd
    b) create sctpcb and return associd id mapping it to the sockfd in the sctpcb (call sctpcballoc for information see file subroutines.doc)
    c) link it to per process list of PCBs.

2. sctpBindx(associd, local address list, port, address count)
    a) find the sctpcb with the given associd
    b) check if the list is valid (we'll have to find a way out for this) and if so, put the given values in the sctpcb.
    c) call "bind" with address INADDR_ANY and port =port (internally calls PRU_BIND)
    d) if any ephemeral port is assigned get the port using 'getsockname').
    e) return appropriate return values (1, -1)

3. sctpListen(associd, backlog)
    a) find the sctpcb from associd
    b) state = LISTEN

c) call "listen" (internally causes the socket option for this socket to be set to SO_ACCEPTCONN)

4. sctpAccept(associd, <u>no. of outbound & inbound streams, dest. Address, port, address count</u>)

a) Find sctpcb from associd.
b) If state = LISTEN continue.
c) call "accept", (sleeps until scpt_input wakes it up on the occurrence of INIT or COOKIE
ECHO. After waking up calls soaccept -> PRU_ACCEPT.
d) "recv" on the new socket (sctp_input has stored the packet that has arrived on the new socket)
e) if packet = INIT, continue, else , goto (i)
f) state = INIT_RCVD create new sctpcb and update values in the sctpcb from the INIT arrived (negotiate)
g) call usr_sctp_output
h) close the new socket, destroy the new sctpcb and goto (a) above. (while closing PRU_CLOSE or PRU_DISCONNECT will be called, MIND THAT DEPENDENCY)
i) if packet = COOKIE_ECHO continue, else drop the packet "close" the socket
j) validate cookie, create new sctpcb, update values from the cookie into sctpcb, if not validated drop the packet and 'close' the packet.
k) state = COOKIE_RCVD.
l) call usr_sctp_output.
m) call setitimer & handler for SIGALRM
n) go for asynchronous i/o for socket & handler for SIGIO
o) Return the new associd and other values

5. sctpConnect(associd, , no. of outbound & inbound streams, dest. Address, port, address count)

a) find sctpcb from associd, update values from the parameters
b) if local address list is empty, initialize it to INADDR_ANY.
c) call “connect” system call with parameters (Initiate Tag and fport) (this will internally put the socket in soisconnected state and hence can be used for data transfer). If lport is 0, then after the call to connect, we’ll have an ephemeral port assigned. Get it using ‘getsockname’ and update the sctpcb value.
d) call setitimer & handler for SIGALRM.
e) state = COOKIE_WAIT. Start T1-init timer
f) call usr_sctp_output
g) “recv” from socket, if packet = INIT ACK, stop T1-init timer, update parameters in the sctpcb
h) state = COOKIE_ECHOED, write the cookie to the send (retransmission Q). start T1-cookie timer
i) call usr_sctp_output
j) “recv”, if packet = COOKIE_ACK, stop T1-cookie timer, state = ESTABLISHED
k) delete COOKIE from the send Q
p) go for asynchronous i/o for socket & handler for SIGIO
l) return appropriate values

6. sctpSend(associd,buffer address, byte count, stream id, flags(unordered, unbundled) payload protocol id)

a) find sctpcb from associd
b) create data chunks and append to send (retransmission) Q. assign SSNs and TSNs and update the values in the sctpcb.
c) update ‘send sequence Nos.’
d) if T3-RTX timer not running start it with RTO value
e) call usr_sctp_output

7. sctpReceive(associd, buffer address, byte count, stream id, partial flag, payload protocol id)

(NON-BLOCKING)

stream id is accepted as a parameter

a) read the particular streams recv buffer and if data present then return the data for the next expected data chunk (in-sequence in the stream). Else return EWOULDBLOCK.
b) remove the data that has been read from the receive buffers. Update a_rwnd & set ACK flag
c) Call usr_sctp_output to send window update

Notes: 1. Here the receive buffers will directly contain data chunks, the processing of IP packets is done in the usr_sctp_input function.

2. For usr_sctp_input function, see if we get data packet-by-packet, due to setting of PR_ATOMIC, or do we have to read the first 20 bytes (IP Header), and from the length field then determine how many bytes to be read.

8. sctpShutdown(associd,)

a) state = SHUTDOWN_PENDING
b) call usr_sctp_output

10.1.5 Primitives for Protocol-Protocol Interface:

1. usr_sctp_output (pointer to PCB, pointer to Transport Address Block (TAB))

Local variable - IP packet structure with maximum PMTU.

1) If state = ESTABLISHED/SHUTDOWN_PENDING/SHUTDOWN_RCVD, else goto 10.
2) Call ‘fill_headers’ to fill the IP & SCTP header.

3) If the flags field in the PCB has ACK flag set, generate a SACK chunk store in the IP packet, reset ACK flag in the PCB. (SACK can be generated from the info. in PCB)
4) Check the idle counter for the destination, and calculate the cwnd accordingly, reset the idle counter. (This is to be done only if data transfer is scheduled).
5) Read the Transmission Queue for the given address and copy data chunks from snd_nxt to the end of transmission Queue (snd_max) into the IP packet. While doing so, see that the size is allowable by cwnd (the one after that in pt. 4 above), rwnd and PMTU, Calculate the length for this in bytes from the data chunk headers).
6) If some more data chunks remain in the transmit Queue, due to smaller PMTU, then set a flag to indicate, "come back here to send another packet".
7) Update the Peer's rwnd, the no. of outstanding bytes.
8) Move snd_nxt to the proper position.
9) If not timing any segment, time a segment. (set rtt to 1), store the first TSN in this packet in the field provided in PCB.
10) Update the values of length and checksum in the IP packet. As 'fill_headers; would not do it for you.
11) Send the IP packet so formed.
12) Do processing for other states,
    a) if state = INIT_RCVD, create INIT_ACK, cookie, send it,
    b) if state = COOKIE_RCVD, then generate COOKIE_ACK, send it
    state = ESTABLISHED.
    c) if state = COOKIE_WAIT, generate INIT and send it
    d) if state = COOKIE_ECHOED, read cookie form the Q and send it
    e) if state = SHUTDOWN_SENT, send SHUTDOWN chunk.
    f) if state = SHUTDOWN_ACK_RCVD, send SHUTDOWN_COMPLETE, do "close" and delete PCB.

2. usr_sctp_input (SIGIO Handler)

1) "recv" data from the socket. Use select if need be.

2) lookup the PCB.

3) Arrived SACK Processing:

   I) for each TAB (Transport Address Block) do the following

   a) remove data chunks from transmission queues that have been acknowledged by TSN ACK in the SACK chunk

   b) manage cwnd, no. of outstanding data bytes

   c) if data still present in transmission queues restart the T3-rtx timer, else stop the timer

   d) If the Cum. TSN Acked by the SACK moves ahead of the TSN being timed, reset rtt counter, and make RTO measurements

   e) Call usr_sctp_output. (Do this only if snd_nxt != snd_max, because as SACKs are processed more data may be pumped out that had remained due to cwnd restrictions).

   II) Note the Peer's new rwnd.

4) If state = SHUTDOWN_PENDING/SHUTDOWN_RCVD then check whether for all TABs

   snd_una == snd_max.

   a) if state = SHUTDOWN_PENDING and snd_una == snd_max
   state = SHUTDOWN_SENT, call usr_sctp_output.

   b) if state = SHUTDOWN_SENT and receive = SHUTDOWN_ACK
   state = SHUTDOWN_ACK_RCVD, call usr_sctp_output.

   c) if state = SHUTDOWN_SENT and receive = SHUTDOWN
   state = SHUTDOWN_ACK_SENT , call usr_sctp_output.

   d) if state = SHUTDOWN_ACK_SENT and receive = SHUTDOWN_ACK
   state = SHUTDOWN_ACK_RCVD , call usr_sctp_output.

   a. if state = SHUTDOWN_ACK_SENT and receive= SHUTDOWN_COMPLETE, stop timers, "close" , and delete PCB.

   e) if state = ESTABLISHED and receive = SHUTDOWN
      state = SHUTDOWN_RCVD , call usr_sctp_output.
   f) if state = SHUTDOWN_RCVD and snd_una == snd_max
      state = SHUTDOWN_ACK_SENT , call usr_sctp_output.

5) Handle abortive conditions create ABORT chunks and call snd_abort.
6) Arrived DATA Processing:
   a) Check the destination addresses for the arrived packet and if these do not match with any address in the local address list, then, then drop the packet. If the address list has INADDR_ANY instead then neglect this test and accept the packet in any case.
   b) Set ACK flag
   c) Decrement a_rwnd
   d) Update cwnd depending on the idle counter
   e) Reset idle counter
   f) Increment ACK state
   g) If ACK state ==2, call usr_sctp_output.
   h) Advance last rcvd TSN (from the arrived data chunks)
   i) Update mapping array relative to the last rcvd TSN(Cum. TSN ACK pt.).
   j) Demultiplex data and reassemble it according to SSNs into different streams' buffers.
7) Loopback to "recv" until u get EWOULDBLOCK.

10.1.6 SCTP Timers:

We use the following 6 counters/timers for the protocol. All are decremented unless specifically stated.

a) init timer
b) cookie timer
c) retransmission timer (per TAB)
d) shutdown timer
e) idle counter (per TAB, should be incremented)
f) rtt counter (per TAB, should be incremented)

init, cookie and shutdown timers are mutually exclusive and hence only a single counter called I-C-S counter will be used. A ‘0’ indicates that the timer/counter is disabled, whereas, a ‘0’ after decrementing indicates a time-out.

10.1.6.1 I-C-S timer :

Decrement if non-zero, if zero after decrementing then do the following,

Case INIT:

a) Reset the value to exponential Backoff
b) Increment no. of retrans (pt. 31) in sctpcb
c) If exceeds Max Init Retransmissions, then give user notification
d) Else call usr_sctp_output

Case COOKIE:

Same as INIT only compare with Max Cookie Retrans

Case SHUTDOWN:

Same as INIT only compare with Max Shutdown Retrans

The remaining timer/counters are done on a per TAB basis,

10.1.6.2 Retransmission timer:

Decrement if non-zero, if becomes ‘0’ on decrementing, do the following

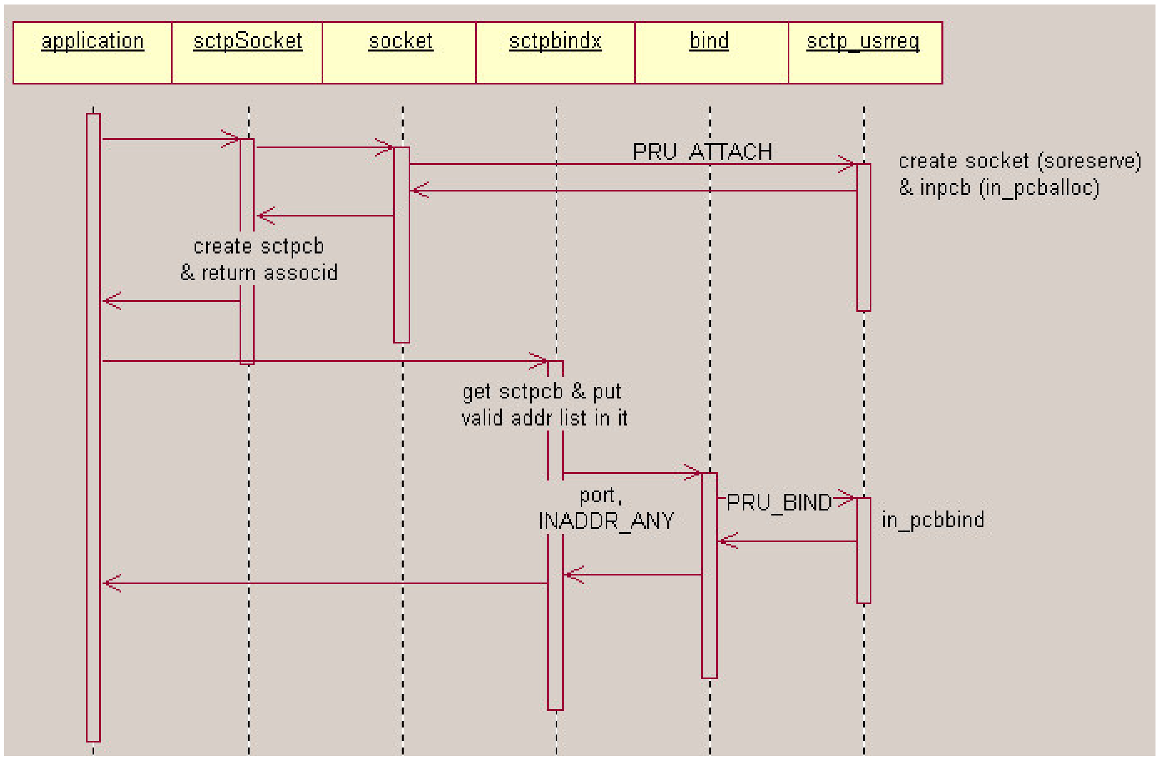
a) if segment is being timed stop it.
b) Increment no. of times retransmission has occurred on the address
c) If the no. of retransmissions exceeds Path Max Retrans then move the transmission Queue to some proper address in the TAB list.
d) Update peer’s rwnd, (snd_nxt – snd_una)
e) Update cwnd, ssthresh, no. of outstanding bytes.
f) Reset counter value to exponential back-off.
g) Call usr_sctp_output with the current TAB pointer, as the second parameter.

10.1.6.3 Idle counters: only increment.

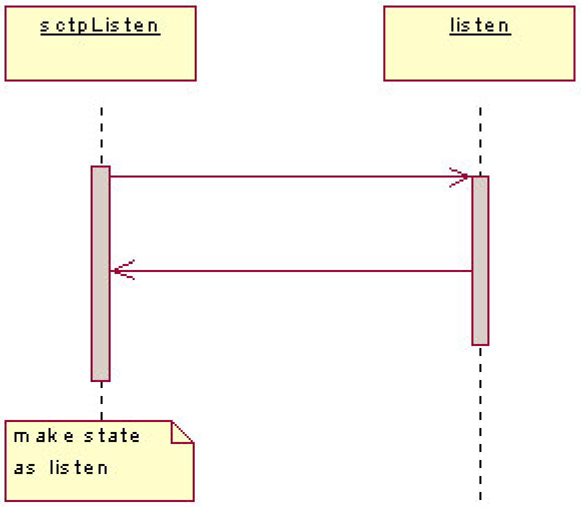
10.1.6.4 RTT counter: if non-zero, then increment.

10.1.7 Sequence diagrams:

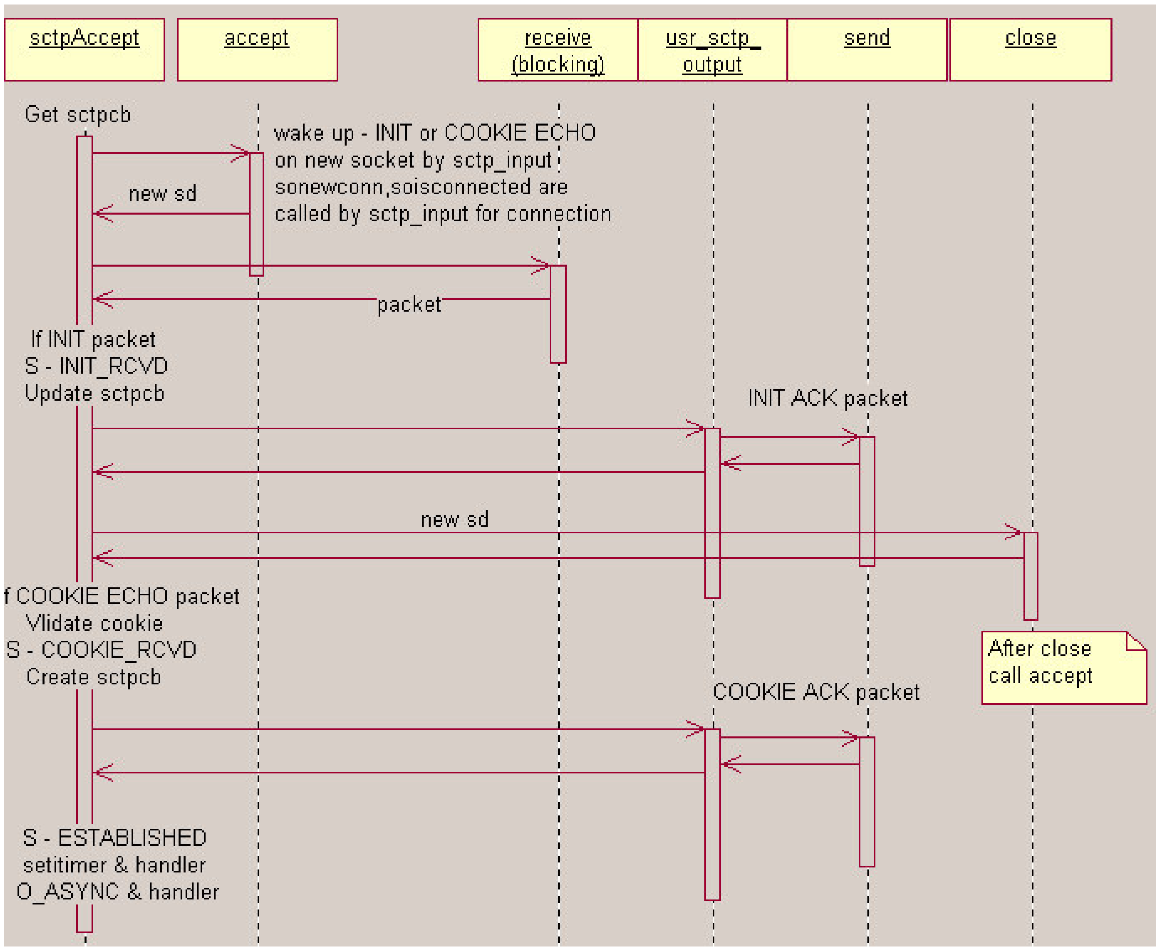
- Sequence diagram for sctpSocket and sctpBindx



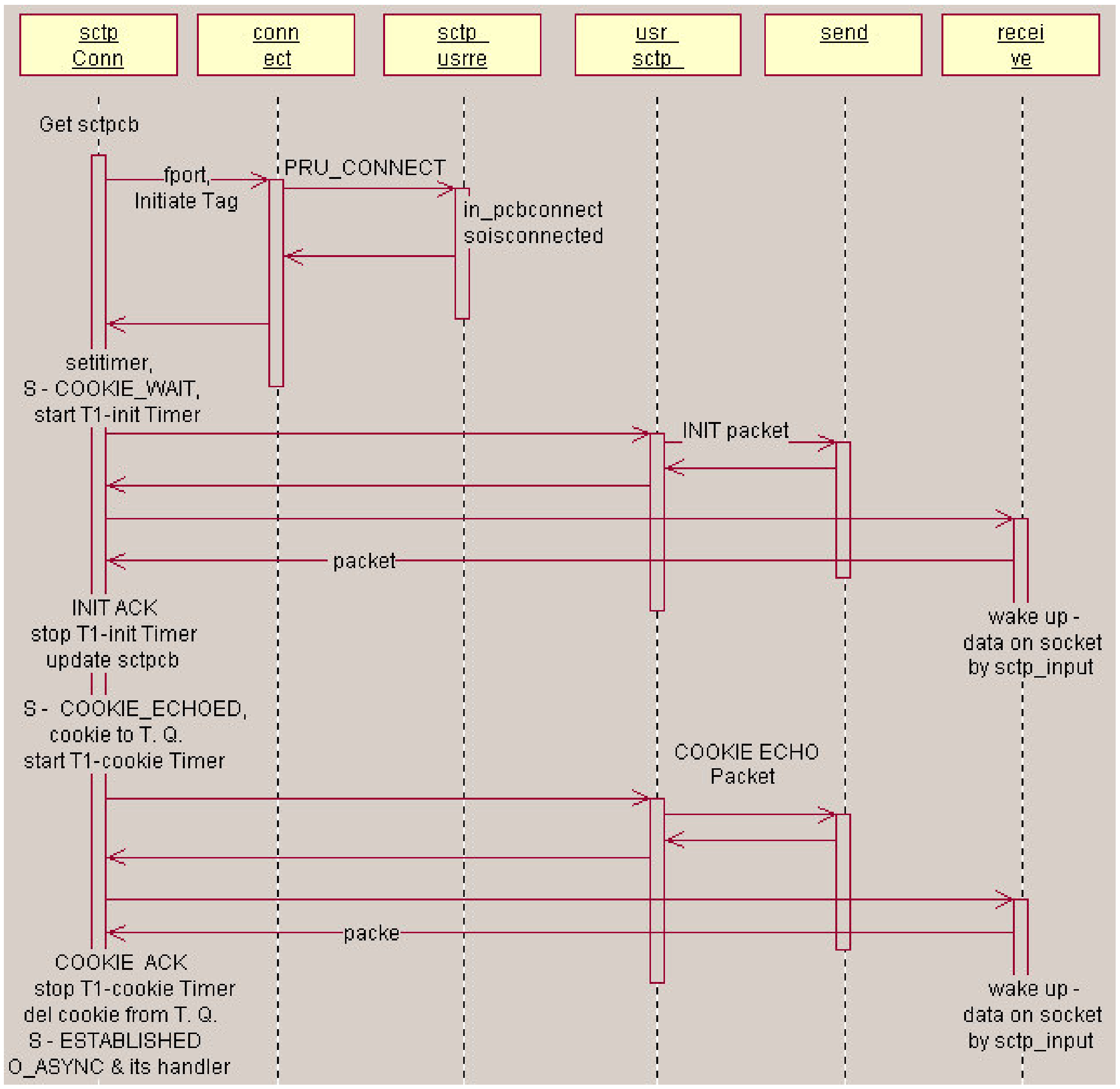
- Sequence diagram for sctpListen



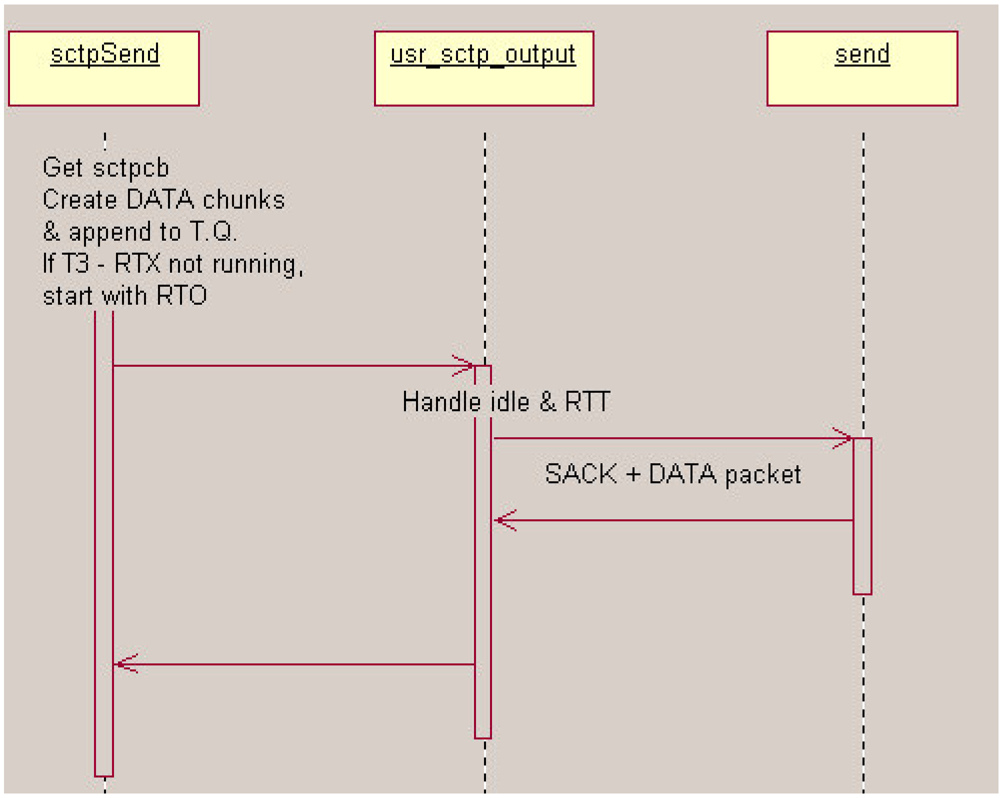
- Sequence diagram for sctpAccept



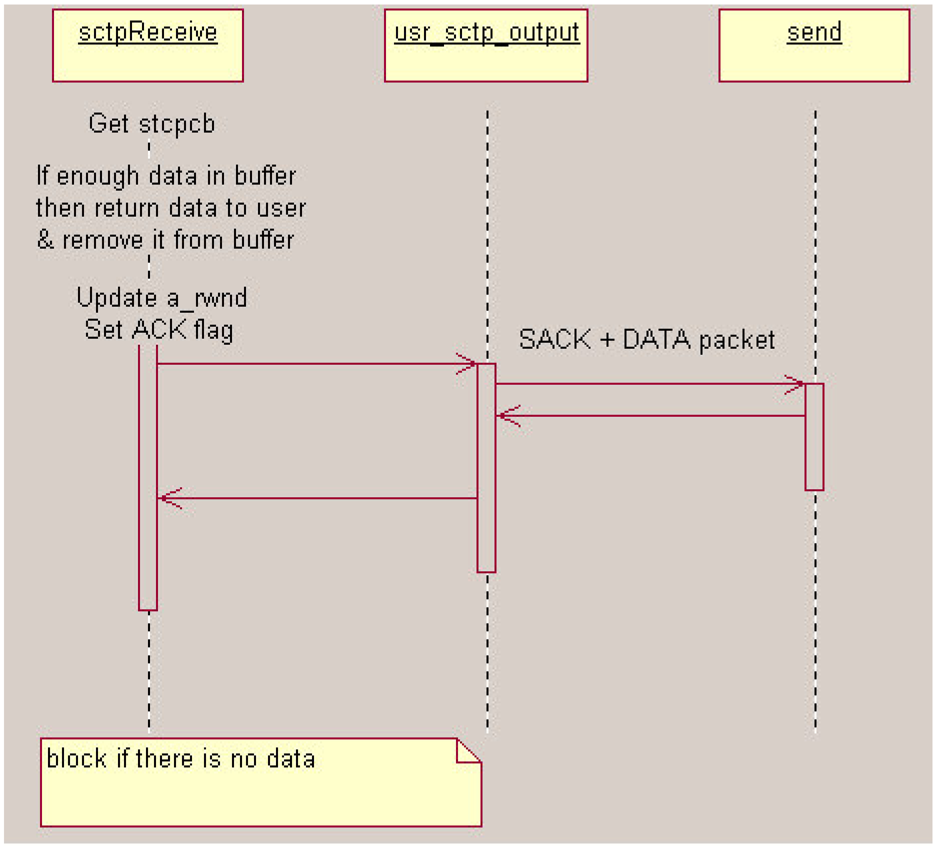
- Sequence diagram for sctpConnect


sctp Conn
conn ect
sctp_ usrre
usr_ sctp_
send
recei ve
Get sctpcb
fport, Initiate Tag
PRU_CONNECT
in_pcbconnect soisconnected
setitimer, S - COOKIE_WAIT, start T1-init Timer
INIT packet
packet
INIT ACK stop T1-init Timer update sctpcb
wake up - data on socket by sctp_input
S - COOKIE_ECHOED, cookie to T. Q. start T1-cookie Timer
COOKIE ECHO Packet
packe
COOKIE ACK stop T1-cookie Timer del cookie from T. Q. S - ESTABLISHED O_ASYNC & its handler
wake up - data on socket by sctp_input

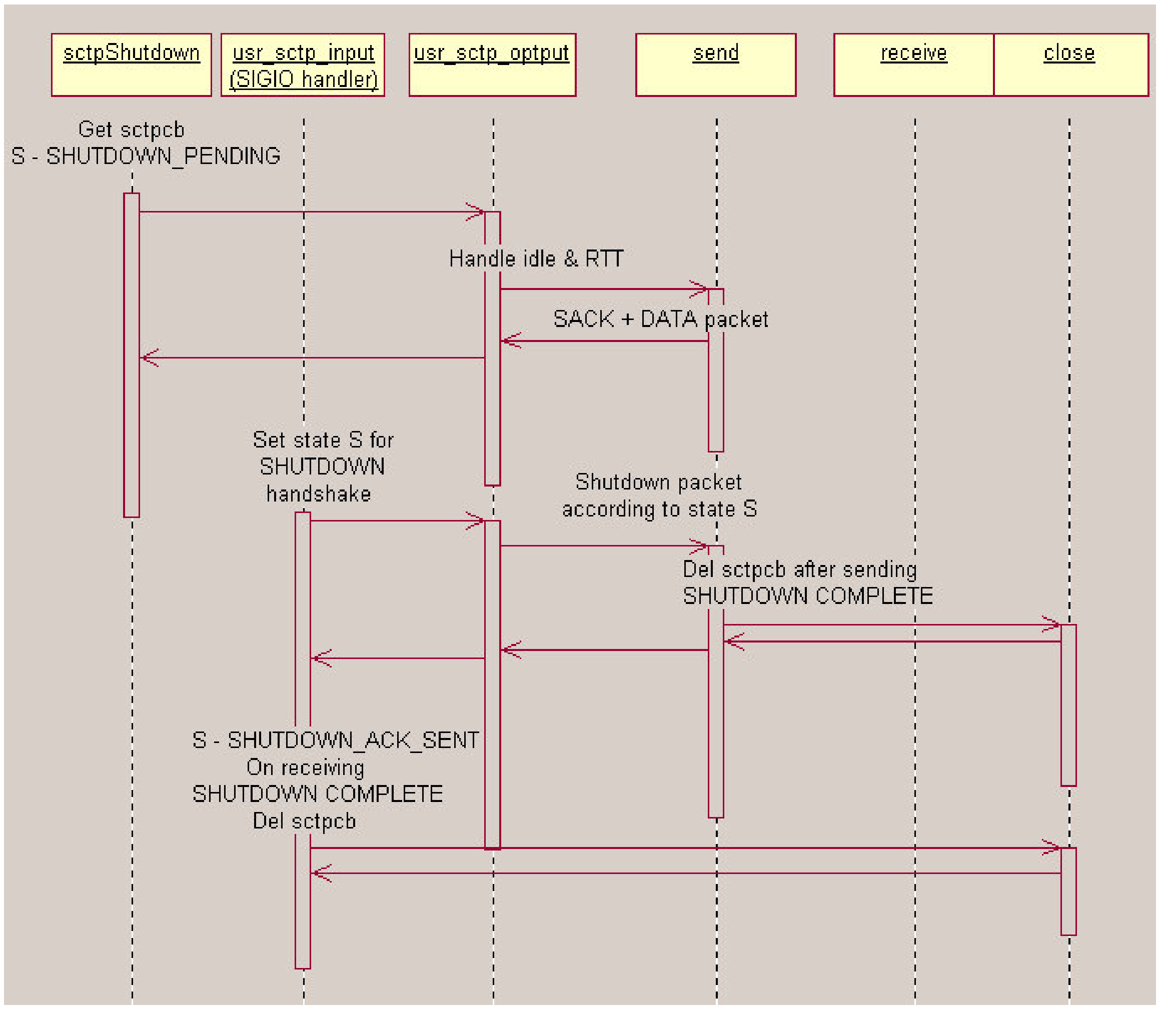
- Sequence diagram for sctpSend



- Sequence diagram for sctpReceive

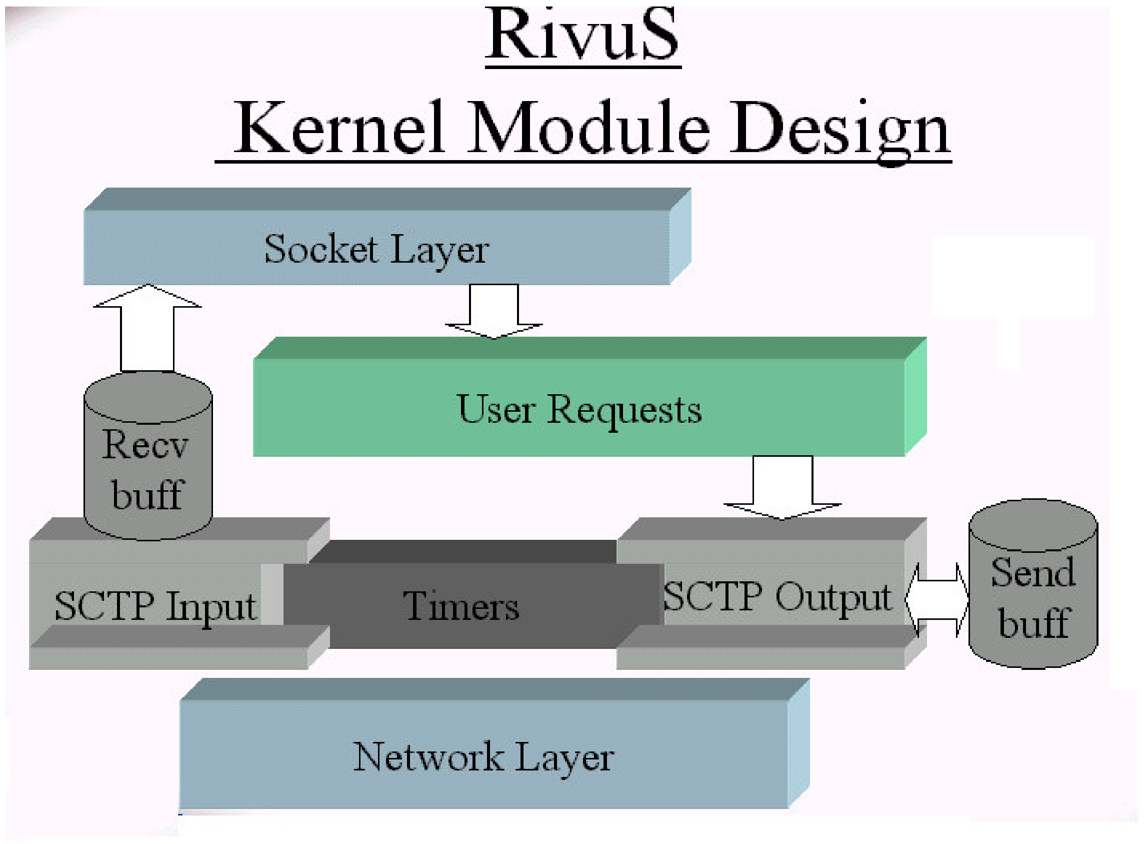


- Sequence diagram for sctpShutdown


sctpShutdown
usr_sctp_input (SIGIO handler)
usr_sctp_optput
send
receive
close
Get sctpcb
S - SHUTDOWN_PENDING
Handle idle & RTT
SACK + DATA packet
Set state S for SHUTDOWN handshake
Shutdown packet according to state S
Del sctpcb after sending SHUTDOWN COMPLETE
S - SHUTDOWN_ACK_SENT
On receiving SHUTDOWN COMPLETE
Del sctpcb

10.2 Kernel Space:

The SCTP protocol is implemented as a Kernel module for 4.3 FreeBSD. The Socket layer communicates with Network layer through our SCTP Kernel module. As can be seen from the diagram below, our layer is between the socket layer above and the network layer beneath.



All the data structures of User space library are shifted here in the Kernel space, to get the Kernel Module for RivuS.

The main data structure – SCTP Protocol Control Block i.e. sctpcb is pointed by the Internet Protocol Control Block i.e. inpcb which is in the kernel space. In this sctpcb, we have all information regarding Local Address List, Foreign Address List and List for Receive buffers for each path.

We provide same Socket Layer APIs as TCP provides. So our design is such that the use of protocol is totally transparent to user. We have done the kernel space design such that the normal user has to just change the third argument to the socket system call, rather than creating a socket of type SOCK_STREAM with TCP, user has to just change it to SCTP. That gives the application transparency.

For the kernel space design, we have to take care of the replacement all the memory buffers into kernel space i.e. m_bufs. All the socket system calls point to the SCTP user requests functions like socket( ) system call internally calls sctp_usr_attach routine from the User Request function.. Likewise this is kernel module design is kept similar to TCP.

11. Implementation of SCTP

This section elaborates on the implementation details of SCTP. The semantics of TCP have largely been preserved intact.

11.1 Overview of the Kernel Data Structures:

The network protocol stack used for the implementation of SCTP is the common reference implementation of TCP/IP from the Computer Systems Research Group at the University of California at Berkeley. This has been distributed with the 4.x BSD operating system. The official name of the software is the 4.4 BSD-Lite distribution, its also referred to as Net/3. The open source operating system (that distributes the 4.4 BSD Lite source code) that was used in the project is FreeBSD (http://www.freebsd.org).

11.1.1 Primary Data Structures and their relationships:

The socket and the PCBs (Protocol Control Blocks) are the primary data structures relevant in our discussion and are modified to a large extent in the implementation of SCTP.

11.1.2 The socket Structure

A socket structure represents one end of a communication link and holds or points to all the information associated with the link. This includes the protocol to use, state information for the protocol (which includes source and destination addresses), queues of arriving connections, data buffers and option flags. The following are the important and relevant fields of the socket structure:

1. so type: This field identifies the communication semantics to be supported by the socket and the associated protocol. For SCTP, it is SOCK STREAM.
2. so options: It's a collection of flags that modify the behavior of a socket. Some additional flags were introduced in addition to the default ones in the implementation of SCTP.
3. so state: Represents the internal state and additional characteristics of the socket. The implementation of SCTP introduced some additional socket states.

4. so pcb : This field points to a protocol control block that contains protocol specific state information and parameters for the socket. Each protocol defines its own control block structure, so so pcb is defined to be a generic parameter. In the case of TCP, the control block structures are struct inpcb (for IP specific information) and struct tcpcb (for TCP specific information).

5. so proto: This field points to the protosw structure of the protocol selected by the process during the socket system call.

6. so error holds an error code until it can be reported to a process during the next sys-tem call that references the socket.

7. so upcall and so upcallarg: so upcall is a function pointer, and so upcallarg is designed to be a parameter to that function. Whenever important events happen, such as a segment arrival, the so upcall function is invoked with the so upcallarg as parameter.

11.1.3 Protocol Control Blocks

As mentioned earlier, each protocol maintains state and information in its protocol control block. Some protocols like UDP, have very little state that they do not have the need to keep a separate control block and use the same control block as IP (Internet Protocol Control Block). The protocol control blocks of relevance to our implementation are the Internet Protocol Control Block (struct inpcb) and the SCTP Control Block (struct sctpcb).

Internet Protocol Control Block

The Internet PCB contains the information common to all UDP and TCP endpoints: foreign and local IP addresses, foreign and local port numbers, IP header prototype, IP options to use for this endpoint, and a pointer to the routing table entry for the destination of this endpoint. The SCTP control block contains all the state information that SCTP maintains for each connection:
sequence numbers in both directions, window sizes, retransmission timers and the like.

The socket and the inpcb structure point to each other by means of the so pcb and the inp socket members respectively. The inpcb structure also contains the (local address, local port, foreign address, foreign port) tuple for this endpoint.
The SCTP control block struct sctpcb stores SCTP specific information for this connection. The inpcb and the sctpcb structures also point to each other.

SCTP Control Block

The SCTP Protocol Control Block is a very large and complex structure. This sctpcb control block contains all the necessary parameters per association.

1. Peer Verification Tag : Tag value to be sent in every packet and is received in the INIT or INIT ACK chunk.
2. My Verification Tag : Tag expected in every inbound packet and sent in the INIT or INIT ACK chunk.
3. State : A state variable indicating what state the association is in, i.e. COOKIE-WAIT, COOKIE-ECHOED, ESTABLISHED, SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED, SHUTDOWN-ACK-SENT.
4. Peer Transport Address List : A list of SCTP transport addresses that the peer is bound to. This information is derived from the INIT or INIT ACK and is used to associate an inbound packet with a given association. Normally this information is hashed or keyed for quick lookup and access of the TCB.
5. Primary Path : This is the current primary destination transport address of the peer endpoint. It may also specify a source transport address on this endpoint.
6. Peer Rwnd : Current calculated value of the peer’s rwnd.
7. Next TSN : The next TSN number to be assigned to a new DATA chunk. This is sent in the INIT or INIT ACK chunk to the peer and incremented each time a DATA chunk is assigned a TSN (normally just prior to transmit or during fragmentation).
8. Last Rcvd TSN : This is the last TSN received in sequence. This value is set initially by taking the peer’s Initial TSN, received in the INIT or INIT ACK chunk, and subtracting one from it.

9. Mapping Array : An array of bits or bytes indicating which out of order TSN’s have been received (relative to the Last Rcvd TSN). If no gaps exist, i.e. no out of order packets have been received, this array will be set to all zero. This structure may be in the form of a circular buffer or bit array.
10. Ack State : This flag indicates if the next received packet is to be responded to with a SACK. This is initialized to 0. When a packet is received it is incremented. If this value reaches 2 or more, a SACK is sent and the value is reset to 0. Note: This is used only when no DATA chunks are received out of order. When DATA chunks are out of order, SACK’s are not delayed.
11. Inbound Streams : An array of structures to track the inbound streams. Normally including the next sequence number expected and possibly the stream number.
12. Outbound Streams : An array of structures to track the outbound streams. Normally including the next sequence number to be sent on the stream.
13. Reasm Queue : A re-assembly queue.
14. Local Transport Address List : The list of local IP addresses bound in to this association.
15. Association PMTU : The smallest PMTU discovered for all of the peer’s transport addresses.

Per Transport Address Data

For each destination transport address in the peer’s address list derived from the INIT or INIT ACK chunk, a number of data elements needs to be maintained including:
1.cwnd : The current congestion window.
2.ssthresh : The current ssthresh value.
3. RTO : The current retransmission timeout value.
4. SRTT : The current smoothed round trip time.
5. RTTVAR : The current RTT variation.
6. partial bytes acked: The tracking method for increase of cwnd when in congestion avoidance mode.

7. state : The current state of this destination, i.e. DOWN, UP, ALLOW-HB, NO-HEARTBEAT, etc.

8. PMTU : The current known path MTU.

9. Per Destination Timer : A timer used by each destination.1

10. RTO-Pending : A flag used to track if one of the DATA chunks sent to this address is currently being used to compute a RTT. If this flag is 0, the next DATA chunk sent to this destination should be used to compute a RTT and this flag should be set. Every time the RTT calculation completes (i.e. the DATA chunk is SACK'd) clear this flag.

11.2 File Organization:

11.2.1 User Library:

The files structure for the user space implementation is divided into 2 parts, files in the kernel, and those in the user space.

Kernel Space Files:

- in.h – This file contains the Protocol ID of SCTP (which is 132). This file is already present, only the particular ID is added to it.
- in_proto.c – This file contains the protocol switch structure. This is also originally contained in the FreeBSD network stack code. We add our own protocol control block to it.
- sctp.h – This file contains the definitions of all the structures that are needed in the
- sctp_var.h – This file contains extern definitions of all the functions and variables in the kernel.
- sctp_usrreq.c – This file contains the functions that interface between the socket layer and the transport layer (i.e. SCTP).

User space files:

- sctp.h – This file contains the definitions of the structures which are all the different headers that are used within different packets and chunks, by the protocol.
- sctp_input.c – This file contains functions for handling of the data that is received by the host. The major function being sctp_input which runs at network interrupt priority level.
- sctp_ls.c – This file contains function for configuration of the Load Sharing (i.e. Path Aggregation) feature.
- sctp_mac.c – This file contains the code for the handling of the MD5 hash that needs to be calculated during the initialization of the association.
- sctp_mac.h – This is the header file that is needed for the sctp_mac.c file.

- sctp_output.c – This file handles all the processing of the data that will be sent down the network stack by a given host.
- sctp_prims.c – This file has the APIs that are exported to the user by *The RivuS*.
- sctp_prims.h – This is the header file needed by the corresponding *.c file.
- sctp_subr.c – This file contains the sub – routines that are used through out the Transport layer code. These routines are all combined together in this file.
- sctp_timer.c – This file contains the handling of the different timers that the *The RivuS* needs for its functioning.
- sctp_timer.h – This is the header file needed by the corresponding *.c file.
- sctp_var.h – This is the header file which has all the extern definitions of all the functions used through the entire *RivuS* SCTP.
- sctpcb.h – This file has the structures for the Protocol Control Block for SCTP and also TAB (Transport Address Block).

11.2.2 Kernel Module:

The following files are a patch to the FreeBSD kernel.

- in_proto.c – This file contains protocol switch structures for every protocol..
- sctp_error.c – This file has error routines which give error messages if no module is loaded.
- sctp_var.h – This file has the extern definitions for the functions used by sctp_error.c

The following files form the 'module' part of *The RivuS*.

- Makefile – This is the makefile used with the 'make' utility for proper compilation and loading of the module.
- sctp.h – This file contains the structures used for creating headers for different chunks and the SCTP packet.
- sctp_bindx.c – This file generates a System Call module which defines a new system call 'sctp_bindx' used for the binding of multiple addresses.

- sctp_csum.c – This file creates the checksum [8] of the SCTP packet by the crc32c checksum mechanism.
- sctp_input.c – This file contains functions for handling of the data that is received by the host. The major function being sctp_input which runs at network interrupt priority level.
- sctp_ls.c – This file contains function for configuration of the Load Sharing (i.e. Path Aggregation) feature.
- sctp_mac.c – This file contains the code for the handling of the MD5 hash that needs to be calculated during the initialization of the association.
- sctp_mac.h – This is the header file needed by the corresponding *.c file.
- sctp_module.c – This file is responsible for the appropriate creation and the loading of the Kernel module.
- sctp_output.c – This file handles all the processing of the data that will be sent down the network stack by a given host.
- sctp_subr.c – This file contains the sub – routines that are used through out the Transport layer code. These routines are all combined together in this file.
- sctp_timer.c – This file contains the handling of the different timers that the *The RivuS* needs for its functioning.
- sctp_timer.h – This is the header file needed by the corresponding *.c file.
- sctp_usrreq.c – This file handles all the user requests generated by the user and processed via the socket layer system calls.
- sctp_var.h – This is the header file which has all the extern definitions of all the functions used through the entire *RivuS* SCTP.
- sctpcb.h –This file has the extern definitions for the functions used by sctp_error.c

11.3 Code Sample:

As a sample of our code we present the sctp_usr_connect function from the file sctp_usrreq.c. This function is called by the socket layer when the user issues the ‘connect’ system call. The code does the initialization necessary for starting up the handshake with the server, and also conditions to sleep till the connection is established. Further the function hands over control to the sctp_output function.

```
/*
 * This is called when a user wishes to connect to some foreing address
 * The function should set appropriate conditions and call the main output
 * function for the same.
 * so - pointer to socket structure
 * nam - pointer to socket address structure to connect  to
 * p - pointer to proc structure for the process
 */
int
sctp_usr_connect(struct socket *so, struct sockaddr *nam, struct proc *p)
{
        int s = splnet();
        int error = 0;
        struct inpcb *inp = sotoinpcb(so);
        struct sctpcb *sp;

        DTPRINTF("* usrreq : CONNECT\n");
        if(inp == 0) {
                error = EINVAL;
                goto out;
        }

        sp = intosctpcb(inp);

        /* if no port bound, get one */
        if(inp->inp_lport == 0) {

                struct sockaddr_in sin;

                sin.sin_len = 0;
                sin.sin_family = AF_INET;
                sin.sin_port = sin.sin_addr.s_addr = 0;

                error = sctp_usr_bind(so, (struct sockaddr *)&sin, p);
                if(error)
                        goto out;
        }
        /*
         * We do not support REUSEADDR and REUSEPORT hence no
         * question of multiple connections for the same ports on both side.
         */

        /* The mis and os are set in the usr_attach itself */
        /* We build the TAB for the destination address specified */

        sp->primary_path = tab_alloc(sp, NULL,
                                ((struct sockaddr_in *)nam)->sin_addr);

        if(sp->primary_path == 0) {
                error = ENOBUFS;
                goto out;
        }

        sp->faddr_count = 1;

        inp->inp_faddr.s_addr = sp->initiate_tag;
        inp->inp_fport = ((struct sockaddr_in *)nam)->sin_port;

        in_pcbrehash(inp);
        sp->s_template = get_sctp_template(sp);
        /*
         * This causes the connect to sleep in the
         * code for 'connect' system call
         */
        soisconnecting(so);
        sp->state = SCTPS_COOKIE_WAIT;

        /* start the INIT T1-timer */
        sp->timers[SCTPT_ICS] = init_backoff[0];
        callout_reset(&sp->t_ics, sp->timers[SCTPT_ICS], sctp_timer_ics, sp);

        DTPRINTF("! usrreq CONNECT : state is\n");
        sctp_inpcb_state(inp);

        sctp_output(sp, &sp->primary_path->tab);

 out:
        splx(s);
        DTPRINTF("* usrreq : CONNECT returning\n");
        return error;
}
```

The code shows that the *BSD kernel Programming style(9)* has been followed for

- Naming of variables.
- Naming of MACROs.
- Indentation
- Function definition.
- Commenting

11.4 Concurrency and Versioning:

Maintaining concurrency in the files available with different developers is an important issue in a large project, especially like RivuS. Also, maintaining consistency between different versions that are created by different developers is critical. Negligence towards these issues can cause severe problems and incompatibilities as the project grows.

Concurrent Version System (CVS) is a tool used to maintain the consistency and versioning amongst developers and versions. Logs of all the changes made as well as all the versions are maintained by the CVS server.

Documents being simultaneously edited by different developers are merged together into the CVS repository and conflicts, if any, are pointed out immediately to take appropriate actions.

12. Path Aggregation using multi-homing

Our main goal is to achieve high rates of data transfer between endpoints connected across the network and provide extreme reliability & scalability.

Our idea is to gain high data transfer rates between the two ends by using the multiple paths available between them. Multi-homed hosts (a host with multiple network interfaces) can benefit by the use of the multiple paths to achieve such rates. The data that is sent can be multiplexed over these paths at the data sender, depending upon 4 factors, mentioned in section 4, and demultiplexed back at the data receiver end. We term this as Path Aggregation. We focus on gaining enhanced data transfer rates by the use of Path Aggregation.

Channel Bonding and IP Multipathing are based on a similar concept, but they have their inherent drawbacks. In channel bonding, hardware changes in the system are required and IP Multipathing solution is developed just for Solaris 8©. Our concept is transparent to application and also it is not restricted to only one operating system.

12.1 Path Aggregation Goal:

We propose Path Aggregation as an extension to SCTP protocol, which has inherent support for Multihoming and Multistreaming. We have implemented the entire SCTP transport layer protocol in 4.3 FreeBSD. Following are the major goals of RivuS.

12.1.1. High transfer rates:

The multiplexing of data over multiple paths will lead to high data transfer rates between the data sender and the data receiver. Thus, greater bandwidth can be made available to the data transfer.

12.1.2. Performance:

We provide high data transfer rates combined with dynamic adjustment to network conditions. We improve efficiency by avoiding fragmentation, by intelligently selecting the path for sending data. We are proposing an extension to the protocol to optionally support byte-oriented transfer. Thus we achieve higher performance benefits.

12.1.3. Transparency to the application:

The Path Aggregation will be transparent to the application riding above the protocol. Also, by intelligently performing the Path Aggregation, no modifications need be done at the data receiver. Thus, modifications need to be done only at the data sender, so our solution is non-intrusive and requires minimal changes to be done to the existing SCTP stack.

12.1.4. Minimal changes to SCTP stack:

The Path Aggregation extension can be incorporated in any SCTP protocol stack with minimal changes to the protocol.

12.2 Features of Path Aggregation:

- Use of Multihoming feature in SCTP
- Define an association to be of load-sharing type.
- A load-sharing association makes use of multiple paths available between its endpoints, and multiplexes data between these paths. Thus, it achieves high data transfer rates.
- Depending upon the number of paths available, values of parameters like, the size of retransmission queues, the buffer size and the receiver window are kept proportional to the number of paths available.

  i.e. if there are 3 paths then size of receiver window is thrice that of the normal receiver window.
- The TSNs are shared between these paths as the association is the same.
- As the RFC defines each path will have its own congestion window, PMTU and RTT.

- The loading of individual path needs to be defined upon many factors, essentially, as network parameters change frequently; these loading factors must be configured dynamically.
- Also, congestion avoidance, slow start, RTT calculations need to be performed seperately for each path.

12.3 Path Aggregation Concept:

Any sender of a Path Aggregation association takes a 'multiplexing decision' for each packet that it receives from the upper layer. This 'multiplexing decision' involves choosing the perfect path to the destination. The decision is crucial as far as performance is considered. The factors those affect this decision are path Maximum Transmission Unit (PMTU), congestion window (cwnd), Round Trip Time (RTT), errors [10] among others. These factors are discussed in detail in later parts of this section.

At the data receiver of a Path Aggregation association, the packets received on all the interfaces are re-assembled together into buffers for that particular association and delivered to upper layers in strict sequence. Hence, the range of sequence numbers is shared among the multiple paths that exist between the hosts. Thus, though there exists a single association, the paths that the data packets take are different as shown in figure 1. The figure shows that bandwidth 'x' is available on each of the path, but for the sender and receiver a total of '2x' is available for the data transfer.

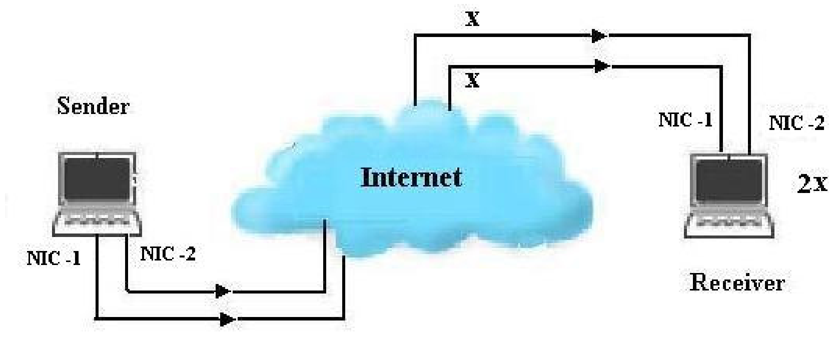


Figure 1: Data Transfer between multi-homed hosts

We now explain the factors that affect the ‘multiplexing decision’.

1. Path Maximum Transmission Unit (PMTU):

The association will discover the PMTU of each path that is involved in the Path Aggregation association, as defined by the protocol, SCTP. The path with a larger PMTU should be preferred for transmission of larger size packets. This would avoid unnecessary fragmentation of packets that would otherwise take place, causing delays and overheads. For packets with smaller size, though there would not be any appreciable difference. Thus in addition to Path Aggregation, we achieve efficiency in terms of fragmentation and reassembly.

2. Round Trip Time (RTT):

The Path Aggregation association determines the RTT of each path that is involved in the association. Clearly, a path with a smaller RTT should be preferred over a path having an RTT of higher value. Thus more data should be prioritized on a path with a smaller RTT.

3. Congestion Window (cwnd):

Congestion window is affected by many other factors like packet loss on the path, congestion at the path, amount of unacknowledged data (i.e. in-flight data) at the sender. Hence a path with a higher value of congestion window would provide higher transfer rates than a path with a lower value, and hence should be preferred for delivering data.

4. Errors:

To minimize the loss of packets and avoid unnecessary retransmissions, an error free path should be preferred. The amount of error for a given path can be found out using the statistics maintained at each end.

12.4 Path Aggregation Design:

We now explain the different components of the system. We have the RivuS SCTP, as shown in figure 2, which implements the core functionality provided by SCTP. These include all the rules defined by the protocol.

1. Data sender:

The RivuS SCTP is notified whether the current association is a Path Aggregation one or not. Depending upon this information the RivuS SCTP initializes parameters of the association. Parameters like size of the retransmission queue and receive buffers are suitably enlarged for a Path Aggregation association as compared to a normal association. The RivuS SCTP checks whether multiple paths are available. This is done by scanning the list of IP addresses the other end offers to the host at association initiation. A host can specify the number of interfaces that it wishes to use in the association at the initialization of the SCTP association. Only if multiple paths exist, then the Path Aggregation mechanism will be effective, otherwise it will be treated as a normal association.

The RivuS SCTP daemon processes the data received during the operation and hands it over to the multiplexer. The multiplexer surveys the factors that affect the multiplexing decision, which we discussed in design section and takes a decision as to which path, is to be chosen for the particular packet. The multiplexer will accordingly choose the destination address of the chosen path for that packet. We utilize the routing infrastructure available to maintain distinctness among the paths. Here I think we should mention that instead of doing this for each packet, we do it after a specific interval of time, and the priorities assigned at that instant are used for taking the multiplexing decision for that particular interval.

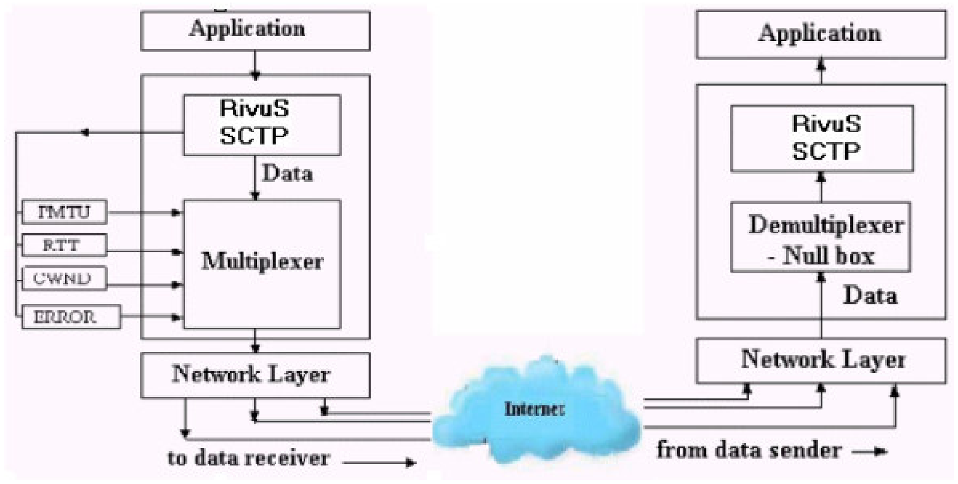


Figure 2: Architecture of RivuS

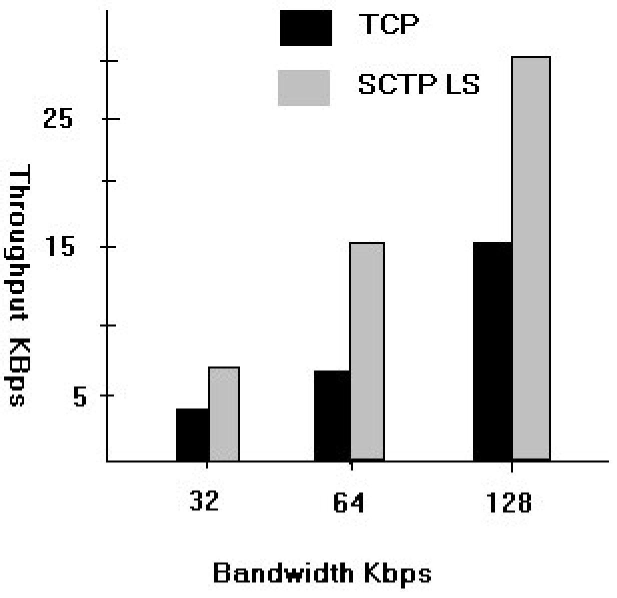
2 .*Data receiver:*

The multiplexing at the data sender avoids any overhead of demultiplexing among the paths at the receiver end. Thus, the data arriving at the receiver will only be given to the RivuS SCTP and the demultiplexer at the other end is a null box. There is no need to make changes to the SCTP stack code at the data receiver, since we share the same range of sequence numbers for all the paths used for the association, we don't need to explicitly have demultiplexer at the receiving end. Also, the cost of demultiplexing and its overheads at the receiver end are eliminated.

13. Performance Analysis :

We have tested our implementation on our Local Area Network. The machines we have used are Pentium 600 MHz, 64 MB RAM, each having two network interface cards (all cards are RealTek 8039 10Mbps).

We have conducted different tests with throttling of the network. And we have got very good gain in performance, while comparing the existing TCP stack with our implemented SCTP stack. The following figure shows that we are getting remarkable gain in performance as compared to normal TCP. So if we compare our implementation with other implementations like, Channel bonding or IP Multipathing, then we surely say that our Kernel space implementation gives more performance than others with minimal changes in the existing installations.



14. Future Work :

We propose to exploit multihoming for gaining extreme data integrity from data redundancy between the paths. This is useful in case of applications that need extreme data integrity at the cost of much more bandwidth being used.

After completing the implementation of SCTP in FreeBSD network stack We will be porting RivuS to Linux to provide contribution in Linux 2.5 summit.

Also we are currently working on design phase for 'Web Server over SCTP'. With the use of MultiStreaming feature of SCTP, a Web Server can gain much better performance.

15. Conclusion:

We thus conclude that, SCTP has been implemented in the 4.3 FreeBSD Operating System as a library as well as a Kernel module. Also Software Engineering principles were followed for analysis, design, coding, testing and documentation. The implementation conforms to the RFC 2960 and also to drafts issued for issues like checksum and APIs.

We also conclude that by exploiting the multihoming feature of SCTP, we gain higher performance in terms of data transfer.. Also the implementation is transparent to the user and minimal changes need to be made in the socket library options. Intelligent use of the routing table structure is utilized for maintaining distinctness among the paths. We also eliminate the cost of demultiplexing at the receiver end.

Though IP Multipathing and Channel bonding are similar in approach, they have there own set of inherent drawbacks. Channel bonding with its requirement for changes in the hardware devices and IP Multipathing in Solaris 8, a non-portable solution, are inadequate for applications. We provide a portable solution with minimal changes to existing installations.

16. References:

[1] RFC 2960: Randall Stewart and Q. Xie, *Stream Control Transmission Protocol (SCTP)*, 2000

[2] RFC 793: J. Postel, *Transmission Control Protocol (TCP)* , 1981.

[3] Cisco Systems Inc., “Channel Bonding”,
Available online: http://www.cisco.com/

[4] Sun Microsystems, “IP Multipathing”,
Available online: http://www.sun.com.co/blueprints/

[5] The FreeBSD Operating System, http://www.freebsd.org/

[6] Sockets API Extensions for SCTP
Available online: www.ietf.org/internet-drafts/draft-ietf-tsvwg-sctpsocket-03.txt

[7] SCTP Implementer’s Guide
Available online: www.ietf.org/internet-drafts/draft-ietf-tsvwg-sctpimpguide-04.txt

[8] Stream Control Transmission Protocol (SCTP) Checksum Change
Available online: www.ietf.org/internet-drafts/draft-ietf-tsvwg-sctpcsum-05.txt

[9] DummyNET Software for 4.3 FreeBSD

[10] *TCP/IP Illustrated Volume I & II* by W. Richard Stevens and Gary Wright 2000 by Pearson Education Asia Pvt. Ltd.

[11] Code walkthrough CDs set of FreeBSD by Mr. M. McKusick