IMPLEMENTATION OF SCTP IN AN OPEN SOURCE REAL-TIME OPERATING SYSTEM

Md. Sazzadur Rahman,
Mohammed Atiquzzaman
School of Computer Science,
University of Oklahoma,
Norman, OK 73019-6151
Email: {sazzad,
atiq}@ou.edu

William Ivancic,
Wesley Eddy,
Dave Stewart
NASA Glenn Research Center,
21000 Brookpark Rd. MS 54-8,
Cleveland, OH 44135
Email:
{wivancic,weddy,dstewart}@grc.nasa.gov

***Abstract*— The Stream Control Transmission Protocol (SCTP) is an IETF-standard transport layer protocol in the IP protocol suite. SCTP is very attractive for data communication over satellite networks due to its unique features such as multihoming and multistreaming that are not present in TCP. SIGMA, an SCTP-based mobility management scheme, has been developed for space networks. RTEMS, an open source Real Time Operating System (RTOS), is used by the computers onboard several spacecraft. The network stack of RTEMS does not include SCTP. The objective of this paper is to describe the issues and challenges in implementing SCTP on the RTEMS operating system. The lessons learned will be helpful to implement SCTP in other operating systems.**

***Index Terms*—SCTP, RTEMS, Porting, Networking Stack**

I. INTRODUCTION

The Internet Engineering Task Force (IETF) has developed transport layer protocol, called Stream Control Transmission Protocol (SCTP) [1]. The design of SCTP absorbed many strengths and features (window-based congestion control, error detection and retransmission, etc.) of TCP and its enhancements [2] for satellite networks. The implementation of some of the enhancements in SCTP are, however, different from their corresponding implementations in TCP. SCTP also incorporated several unique features, such as multistreaming and multihoming, that are not available in TCP. Multihoming in SCTP can help endpoints to detect link failures faster than traditional approaches and is transparent to upper-layer applications. Research on SCTP multistreaming in reducing the latency of streaming multimedia in high-loss environments shows that multistreaming results in slower degradation in network throughput as the loss rate increases. These unique features may also help SCTP to achieve better performance than TCP in satellite networks [3], [4]. Moreover, the next generation IPv6 based military and commercial mobile terminals (i.e., Joint Tactical Radio System (JTRS), Future Combat Systems (FCS), and 4G Systems) are likely to be equipped with multiple network interfaces and therefore, will be able to utilize multihoming and multistreaming features of SCTP for better performance.

SCTP-based mobility management schemes such as SIGMA (Seamless IP diversity based generalized Mobility Architecture) [5] and SINEMO (Seamless IP diversity based NEtwork MObility) [6] have been proposed to support inter-satellite handovers in LEO (Low Earth Orbit) constellations.

These attractive features of SCTP have lead to a strong desire to deploy SCTP in satellite networks and gain operational experiences. To be more specific, *our motivation* is to evaluate the performance of SCTP-based mobility management protocol, SIGMA, in an environment realistic to an operational satellite. Surrey Satellite Technology Ltd (SSTL) is a satellite company who provides access to space-based imaging services and runs IP-based satellites [7]. For capturing images, SSTL operates a constellation of satellites including the "United Kingdom - Disaster Monitoring Constellation (UK-DMC)" satellite. UK-DMC has three Solid State Data Recoder (SSDR) devices, two of which are MPC8260 PowerPC based and run the RTEMS operating system [8]. RTEMS is an open source real time operating system and provides a port of the BSD TCP/IP networking stack. Therefore, to deploy SCTP in space via the UK-DMC satellite or test it in a ground engineering model of the satellite, we need to implement SCTP in the networking stack of the RTEMS kernel. The *objective* of this paper is to describe the issues, challenges and steps to implement SCTP in RTEMS. Issues include porting of SCTP from FreeBSD to the RTEMS networking stack, establishment of intercommunication between SCTP and both the network and application layers.

In this paper, we define the pathway SCTP has been implemented in RTEMS for an SSDR. *Our contribution* in this paper is to develop the system model, port SCTP in RTEMS kernel, integrate ported SCTP with application and

The research reported in this paper was funded by NASA Grant NNX06AE44G.
978-1-4244-2677-5/08/$25.00 2008 IEEE

IP layers and test different SCTP features in RTEMS.

The rest of the paper is organized as follows. Sec. II gives an overview of different elements in the system architecture. Sec. III describes the system model. Implementation of SCTP in RTEMS is described in Sec. IV. Sec. V describes testing of SCTP in RTEMS. Sec. VI describes the problems experienced, issues and challenges in implementing SCTP in RTEMS.Finally, concluding remarks are included in Sec. VII.

II. System Architecture

We provide a brief overview of different elements of the system in this section.

A. SCTP

The IETF Signaling Transport (SIGTRAN) working group defined the Stream Control Transmission Protocol (SCTP) as a transport layer protocol in 2000. Like TCP, SCTP provides reliable, in-sequence transport of messages with congestion control. Benefits of SCTP include:

1) Multihoming support: One (or both) endpoints of a connection can consist of more than one IP address, enabling transparent fail-over between redundant network paths.
2) Delivery of data in chunks within independent streams: Eliminates unnecessary head-of-line blocking, as opposed to TCP byte-stream delivery.
3) Path selection and monitoring: Selects a "primary" data transmission path and tests the connectivity of the transmission path.
4) Validation and acknowledgment mechanisms: Protects against flooding attacks and provides notification of duplicated or missing data chunks.
5) Improved error detection: Suitable for jumbo ethernet frames.

SCTP has been implemented in many operating systems including BSD and Linux [9]. But it has not been implemented in RTEMS. Since our objective is to test SCTP-based SIGMA in a model of the UK-DMC satellite running on RTEMS, SCTP is required in RTEMS networking stack. Therefore, we need to implement SCTP in RTEMS.

B. RTEMS

RTEMS (Real-Time Executive for Multiprocessor Systems where the original meaning of the letter M was "Missile" and later "Military") was developed by OAR Corporation on behalf of US DoD [10]. It is a free open source real-time operating system and used by military, industrial and scientific projects in both terrestrial and space environments. The key features of RTEMS are:

1) Designed for real-time, embedded systems and has been ported to various target processor architectures like i386, Pentium, PowerPC etc.
2) Designed to support various open API standards including POSIX and uITRON.
3) Includes a port of the FreeBSD TCP/IP stack as well as support for various filesystems including NFS and the FAT filesystem.
4) Does not provide any form of memory management or processes. In POSIX terminology, it implements a single process in a multithreaded environment.

RTEMS also provides several BSP (Board Support Package) definitions for different target CPUs. Each BSP is an implementation of specific support code for a given board and conforms to a given processor. For instance, RTEMS provides pc386 BSP for i386 CPU families and mpc8260 BSP for PowerPC.

In the RTEMS TCP/IP stack, there is no support for SCTP or IPv6 to date.

C. Solid State Data Recorder - SSDR

The Solid State Data Recorder (SSDR) [11] is a general purpose data recording device for space applications. It supports multiple data inputs and can store 1 GBytes of payload data. SSDRs are used on the Disaster Monitoring Constellation satellites. The operating features of a typical SSDR are:

1) Motorola MPC8260 PowerPC or ARM processors
2) RTEMS operating system (POSIX API, BSD sockets) and SSDR BSP (mpc8260 based for PowerPC).
3) Capable of supporting an 8 Mbit/s HDLC or IP-based protocol stream across any one of the four HDLC links at a time.
4) Designed for missions in the LEO environment.

The UK-DMC satellite has three SSDRs and two of them are PowerPC-based, and the other is ARM-based. All three use the RTEMS Operating system [12]. Fig. 1 shows top view of a testbed SSDR located at the NASA Glenn Research Center [13]. As shown in Sec. II-B, RTEMS does not have the SCTP protocol in its TCP/IP stack. Therefore, we intend to implement SCTP in RTEMS with a view to testing SCTP-based SIGMA in the testbed SSDR and possibly the UK-DMC satellite.

D. SIGMA

Our objective is to test the performance of SIGMA in space. SIGMA is an application layer protocol to support low latency and low packet loss in mobile settings. The basic idea of SIGMA is to exploit multihoming to keep the old path alive during the process of setting up the new path to achieve a seamless handover. For multihoming, it uses SCTP and takes the following steps when handoffs occur:

1) Data link layer handover and obtain new IP address.
2) Add IP addresses into the association.
3) Redirect data packets to new IP address.

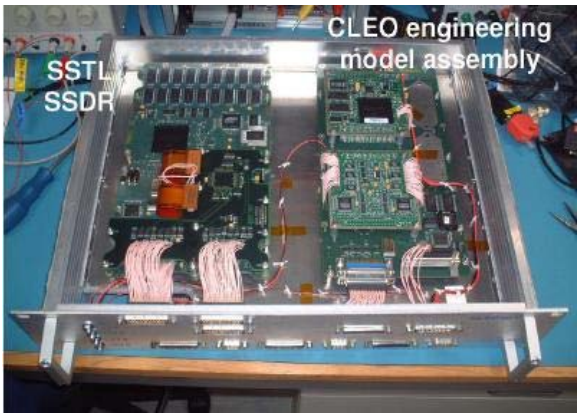

Fig. 1. Top view of SSDR [13].

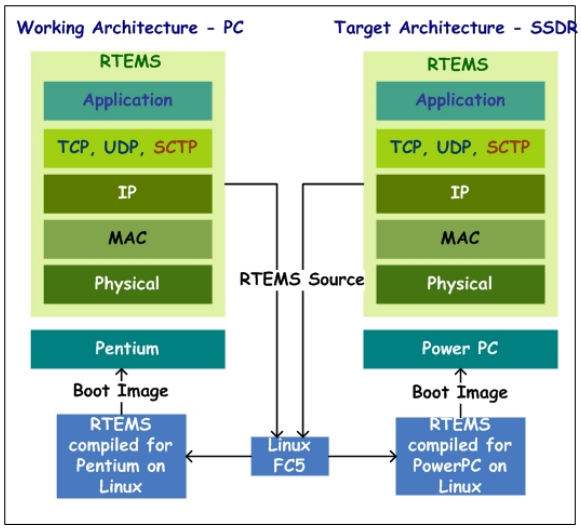

Fig. 2. System model of SCTP in RTEMS.

4) Update location manager (LM).
5) Delete or deactivate obsolete IP address.

SIGMA, as an end-to-end mobility management scheme in satellite environment, increases connectivity of Internet nodes by seamless handover between satellites, exhibits low handover latency and low packet loss rate [14]. The performance of SIGMA leads to the motivation of implementing and testing it in on operational satellites. Since SIGMA relies on SCTP as its underlying protocol and SCTP is not present in RTEMS, our aim is to implement SCTP in RTEMS first.

III. System Models

Fig. 2 shows a system model for implementing SCTP in RTEMS. We developed SCTP for RTEMS in an i386 based machine with Fedora Core in our lab, cross-compiled it for the i386 architecture and pc386 BSP (See Sec. II-B) and ran the executable on an i386 machine. After testing SCTP on i386, we changed the target architecture to PowerPC

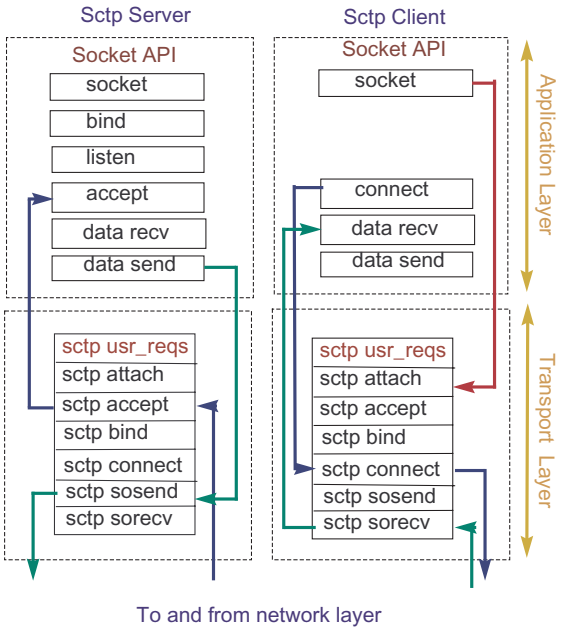

Fig. 3. Intercommunication between application and transport layers.

(instead of i386) and changed the target BSP to the SSDR BSP (See Sec. II-C) (instead of pc386) and tried to run the executable in the SSDR at NASA Glenn Research Center.

In this section, we describe of SCTP integration with different layers in the RTEMS networking stack.

A. Intercommunication between Application and Transport layers

Intercommunication between application layer and transport layer is provided by socket API (Application Programming Interface) functions. SCTP uses a structure for user requests, called sctp_usrreqs, to handle socket API calls from the application layer. For all kernel and user-space SCTP socket operations, this structure has appropriate callback functions as shown in Fig. 3. For instance, application calls to the socket() function with the parameter IPPROTO_SCTP result in calls to the sctp_attach() callback function of sctp_usrreqs structure in transport layer. This works in the same way for other socket API functions as well.

The way data transfer is done in typical SCTP client/server applications is shown in Fig. 3. We assumed SCTP server as data source and SCTP client as data sink. The server application calls the send() function with data payloads to send from application layer and this then uses the sctp_sosend() callback function of the sctp_usrreqs structure in the transport layer. This callback function takes all responsibility to ensure sending data to the client application on an already connected socket. Depending on the status of the operation, the send() function returns a

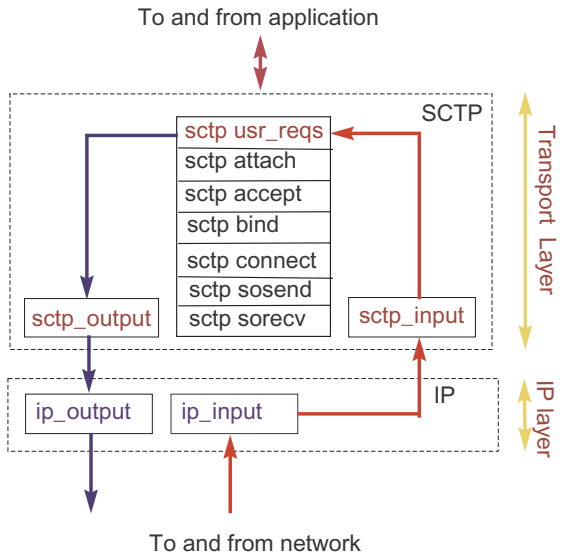

Fig. 4. Intercommunication between transport and IP layers.

code to the application indicating success or error conditions. On the client side, the IP layer gets the incoming packet and uses the sctp_soreceive callback function of the sctp_usrreqs structure in the transport layer. This callback function process the incoming data and delivers it when the application layer calls the recv() function. These phenomena are shown in Fig. 3

B. Intercommunication between Transport and Network Layers

The network layer is just below the transport layer. Outgoing SCTP data or control packets from user space to transport layer eventually go throuth IP in the network layer for further processing. Any incoming data or control packets are also first processed by IP in the network layer before arriving in the transport layer.

Outbound SCTP packets are passed from sctp_output() to the network layer via the ip_output() interface of IP, which adds an IP header to the packet and passes it on to the MAC layer. For any inbound packet, the MAC layer passes the packet to IP, which removes the IP header from the packet and passes it to the transport layer through the sctp_input() callback function of SCTP. This sctp_input() function then can determine sctp_usrreqs function to be invoked by looking at the SCTP headers in the packet or examining at SCTP association status etc. This process is shown in Fig. 4.

IV. Porting SCTP in RTEMS

This section describes the RTEMS development environment, our approach to port SCTP in the RTEMS networking stack and relevant porting issues and aspects.

A. RTEMS development environment

The design goal of RTEMS was to make it a true RTOS targeting embedded systems with small memory space. RTEMS uses the GNU compiler tool chain for development. It also features POSIX, ITRON and classic APIs in both the C and ADA languages. Different RTOS components such as multitasking (thread creation) and synchronization mechanisms (mutex, semaphore, events, message queues etc.), schedulers, clocks etc. are available in RTEMS. Due to its real time characteristics, application tasks can directly use any system libraries and services [10]. To build an application task, it is required to specify all necessary configuration parameters and link the application with the desired RTEMS libraries, BSP etc. The application executable can then be directly run on the target architecture.

B. Choice of BSD over Linux for SCTP porting

There are several open source SCTP implementations available in different operating systems. FreeBSD SCTP and Linux kernel SCTP are among the options for kernel-space SCTP implementations. In RTEMS, the TCP/IP networking stack has been ported from FreeBSD by Eric Norum of the Saskatchewan Accelator Laboratory. Since the TCP/IP stack has been ported from FreeBSD, we thought it was a natural choice to port the SCTP from FreeBSD, rather than Linux or some other implementation. Therefore, we decided to port the SCTP implementation in FreeBSD to RTEMS.

C. Our approach for porting SCTP from FreeBSD to RTEMS

We ported the FreeBSD6.1 SCTP patch source code to RTEMS as follows:

1) Extract the necessary source files from the FreeBSD6.1 SCTP patch.
2) Put the source files in RTEMS source tree where TCP and UDP code currently resides, add them in the libnetworking Makefile.
3) Compile the whole project and resolve any issues (See Sec. IV-D).
4) Establish intercommunication between SCTP and the application layer by adding system calls or API functions that can be called within the application layer.
5) Establish intercommunication between SCTP and IP in the network layer.
6) Run tests with SCTP client/server loopback applications.
7) Run tests with SCTP client/server applications between two different RTEMS machines over the Internet.

D. Porting issues

As shown in Fig. 5, we needed to consider the following issues while porting SCTP to RTEMS:

1) Networking stack size: The reserved networking stack size of RTEMS for TCP and UDP is 4 KB. This size is not sufficient for TCP, UDP, and SCTP together and causes stack overflow problems as described in Sec. VI. Therefore, we needed to increase the size of the RTEMS networking stack to 32 KB. We used 32 KB for experimental purposes, though there seems to be room for further optimization.

2) Backward compatibility: The RTEMS networking code was ported from FreeBSD in 1998, whereas SCTP for FreeBSD was developed in 2004. The FreeBSD networking code has been significantly upgraded after 1998, and SCTP uses the upgraded TCP/IP stack in FreeBSD6.1. So, when we are porting SCTP in RTEMS, we have SCTP code assuming an upgraded IP stack but in RTEMS we have a very old IP stack leading to several incompatibility problems.

3) Lack of IPv6 support: SCTP has IPv6 support, but RTEMS does not have an IPv6 stack. Therefore, we disabled IPv6 support of SCTP in the SCTP library while porting it to RTEMS.

4) Memory management: No memory protection is available in RTEMS; the operating system and application software share the same flat memory space which is different from FreeBSD. Therefore, it is important to modify FreeBSD memory allocation/de-allocation function calls while porting SCTP to RTEMS.

5) Interprocess communication: In terms of interprocess communication, RTEMS and FreeBSD are different. These two OSes use different function calls for mutual exclusion in interprocess communication. Therefore, it is also important to modify FreeBSD interprocess communication function calls while porting SCTP to RTEMS.

Moreover, the size of SCTP source code in FreeBSD6.1 (about 55 KLOC) could be an issue for embedded systems like the SSDR running on RTEMS. Due to the scarcity of resources, embedded systems can not afford large protocol stacks. Therefore, we might need to optimize and simplify the SCTP source code further for RTEMS; possibly removing some optional features or performance enhancements.

E. Incorporating with Application Layer via System Calls

RTEMS networking system calls are located in the rtems_syscall.c source file. Therefore, we ported SCTP socket API calls such as sctp_sendmsg(), sctp_recvmsg(), sctp_bindx() into this source file. The interface and corresponding implementation of the SCTP socket functions are linked via the sctp_usrreq structure callback functions as described in Sec. III-A and through adding entries in the protocol switch table the in_proto.c source file.

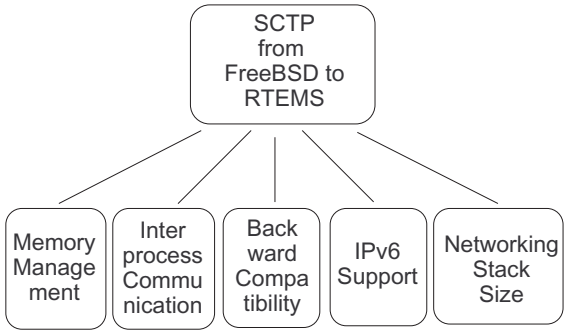

Fig. 5. Porting issues SCTP in RTEMS.

F. Incorporating with IP layer

For any outgoing data from the SCTP stack to the IP layer, SCTP calls the sctp_output() function, which directly calls ip_output() as described in Sec. III-B. But for any incoming SCTP data, a link must be made between IP and SCTP so that IP knows how to send packets from the network layer to the transport layer. This link is made by adding an entry in the protocol switch table in the in_proto.c source file. For protocol type IPPROTO_SCTP, the input method is declared as the sctp_input() callback function and this is used by IP for delivering any incoming packet to SCTP. A typical entry in the protocol switch table looks like this:

```
.pr_type = SOCK_STREAM,
.pr_domain = &inetdomain,
.pr_protocol = IPPROTO_SCTP,
.pr_flags = PR_WANTRCVD,
.pr_input = sctp_input,
.pr_ctlinput = sctp_ctlinput,
.pr_ctloutput = sctp_ctloutput,
.pr_drain = sctp_drain,
.pr_usrreqs = &sctp_usrreqs
```

V. Results of testing SCTP in RTEMS

This section describes the results of testing our ported SCTP code in RTEMS.

A. Testing scenario

For testing ported SCTP in RTEMS, we deployed two test machines over the network - one is loaded with RTEMS4.7.1 and an SCTP server application and the other is FreeBSD6.1 with an SCTP client application as shown in Fig. 6. We assume that the SCTP server is the data source and the SCTP client is the data sink in this scenario. This would be typical of the operational satellite environment, where the embedded SSDR system is the source of large imaging data files (on the order of several hundred megabytes) which are downloaded to PCs on the Earth.

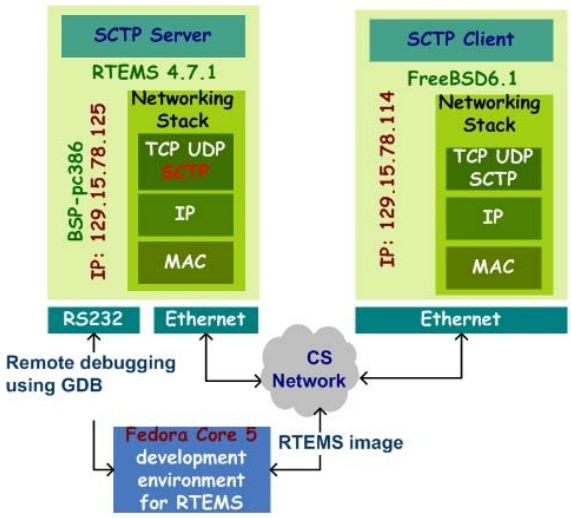

Fig. 6. Test Architecture of SCTP in RTEMS.

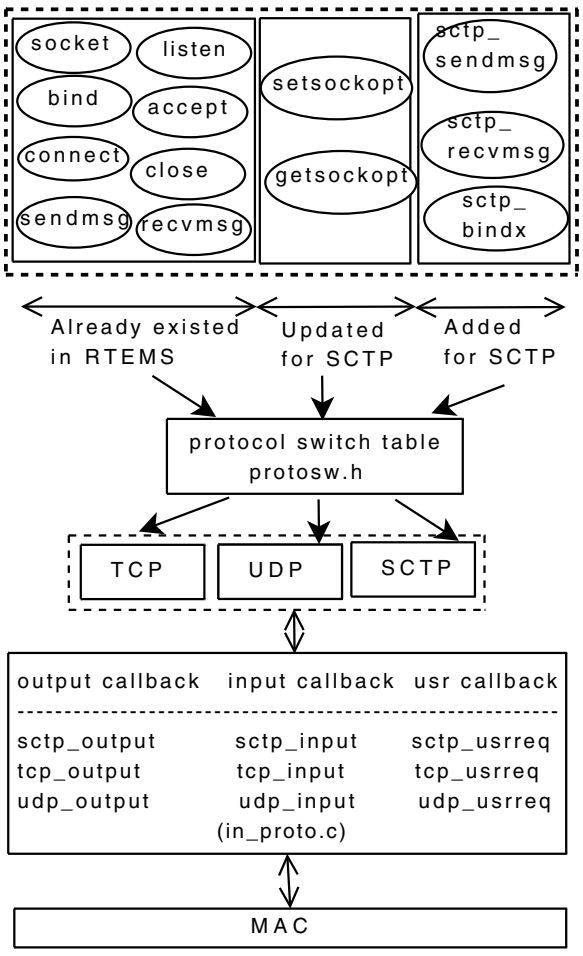

Fig. 7. Overview of SCTP integration with different layers in RTEMS.

the SCTP server executes the following socket API functions in order to serve SCTP-based requests to any client in the Internet:

1) Create an SCTP socket through the standard socket() system call.
2) Bind that socket in a specified port by calling bind() from the socket API.
3) Listen on that socket by calling listen() from the socket API.
4) Finally call accept() which blocks until an SCTP association is created.
5) Call connect() from client machines, using the server's IP address and bound port.
6) When any SCTP client connects to the server and its accept() returns, the server then calls sctp_sendmsg() to send application messages to the client.
7) After finishing data transfer, the SCTP server calls close() from the socket API to shutdown its side of the association.

The SCTP client executes the following process in order to communicate with an SCTP based server in the Internet.

1) Create a SCTP socket by calling socket().
2) Connects to the server with a known IP address and bound port number by calling connect().
3) After successful creation of an SCTP association, the SCTP client starts receiving data from the server by calling the sctp_recvmsg() function.

B. Testing SCTP socket APIs

Fig. 7 shows the integration of SCTP with the application layer by the socket API. For each SCTP socket API function, we examined the status code returned by the API after execution and found that they reported success. Moreover, we also examined the trace of SCTP packets using tcpdump, a common network protocol analyzer tool. We found that the SCTP client server handshake was successful along with verifying the transferred data between client and server.

We then swapped the client/server applications between RTEMS4.7.1 and FreeBSD6.1, repeated the experiment, and found that socket API's were similarly successful. Thus all ported SCTP socket funtions in RTEMS were initially tested.

VI. Runtime problems and debugging of SCTP in RTEMS

After porting SCTP in RTEMS and adding necessary socket APIs, we found that RTEMS networking stack overflows during use of SCTP/IP in the Internet. To debug the RTEMS network stack, we used RTEMS with a remote GDB debugging tool. Remote debugging over TCP/IP is not possible for debuging the TCP/IP stack. Therefore, we used remote GDB over serial communication to fix the overflow

problem. Fig. 6 shows that the RTEMS machine was connected with development machine via RS232 serial port and we ran a GDB remote debugger in the development machine to trace the problem. We found that the RTEMS network stack was created with fixed 4 KB available memory size in the rtems_queue.c source file.

networkDaemonTid = rtems_bsdnet_newproc ("ntwk", 4096, networkDaemon, NULL);

This size is enough for TCP and UDP in the networking stack. But for TCP, UDP and SCTP together, the stack size is not sufficient because of the large volume of SCTP source code (55 KLOC) and this explains the RTEMS network stack overflow during SCTP communication. We resolved the network stack overflow problem by increasing the size from 4 KB to 32 KB (which leaves room for optimization). We could then successfully run client/server communication over SCTP/IP in Internet tests.

VII. Conclusion

It may be advantageous to deploy the SCTP transport layer protocol in space due to the unique features that it offers above and beyond typical TCP and UDP transport protocols. In this paper, we have briefly discussed the system architecture to deploy SCTP in a model of the UK-DMC satellite and test the SIGMA mobility management protocol. We have also described the progress and various steps in implementing of SCTP in RTEMS, a real time operating system used in the UK-DMC satellite. Our future work will be evaluation of SCTP and SIGMA after deployment in space.

References

[1] R. S. et el., "Stream control transmission protocol," RFC 2960, October 2000.

[2] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP over satellite channels using standard mechanisms." RFC 2488, January 1999.

[3] S.Fu, M. Atiquzzaman, and W. Ivancic, "Evaluation of SCTP for space networks," *EEE Wireless Communications*, vol. 12, no. 5, pp. 54–62, Oct 2005.

[4] M. Atiquzzaman and W. Ivancic, "Evaluation of SCTP multi-streaming over satellite links," in *12th International Conference on Computer Communications and Networks*, Dallas, TX, USA, Oct 20-22, 2003, pp. 591–594.

[5] S. Fu and M. Atiquzzaman, "SIGMA: A transport layer handover protocol for mobile terrestrial and space networks," in *e-Business and Telecommunincation Networks*, J. Ascenso, L. Vasiu, and C. Belo, Eds. Springer, 2006, pp. 41–52.

[6] P. K. Chowdhury, M. Atiquzzaman, and W. Ivancic, "SINEMO: An IP-diversity based approach for network mobility in space," in *Second International Conference on Space Mission Challenges for Information Technology*, Pasadena, CA, July 17-21, 2006.

[7] "AlSAT-1 DMC satellite working well in orbit with first use of IP,," SSTL press release, 6 December 2002.

[8] "RTEMS - real time operating system," http://rtems.com/.

[9] "List of SCTP implementations-," http://www.sctp.org/.

[10] T. Straumann, "Open source real-time operating system overview," in *Proceedings of the 8th International Conference, ICALEPCS 2001*, San Jose, California, Nov 27-30, 2001, pp. 235–237.

[11] "Surrey satellite technology limited SSTL," http://www.sstl.co.uk/.

[12] http://roland.grc.nasa.gov/ivancic/papers_presentations/2008/IETF71_DTN_Tests_UK-DMC.pdf, 2008.

[13] L. Wood, W. Eddy, W. Ivancic, J. McKim, and C. Jackson, "Saratoga: a delay-tolerant networking convergence layer with efficient link utilization," in *2007 International Workshop on Satellite and Space Communications*, Piscataway, NJ, USA, Sep 13-14, 2007, pp. 168–172.

[14] M. A. P. Chowdhury and W. Ivancic, "Performance of end-to-end mobility management in satellite ip networks," in *IEEE Globecom 2006*, San Francisco, CA, USA, Nov 27 - Dec 1, 2006, pp. 1–5.