Simulation of Structured Streams Transport Protocol using Network Simulator 2

Genti Daci[1], Raimena Veisllari[2]

[1]*Department of Information Engineering, Polytechnic University of Tirana*
[2]*Department of Information Engineering, Polytechnic University of Tirana*

ABSTRACT

There has been a lot of efforts from researchers lately to deploy alternative transports protocols and mechanisms on providing better packet ordering and delivery semantics. The current development of Internet and Streaming technologies suggests that there is a clear need for such alternatives to TCP Transport Protocols. Structured Streams Transport protocol(SSTP) is a new transport abstraction introduced lately on ACM and it is based on a different approach compared to other alternative transports. Main objective of SSTP is to enhance the TCP's byte stream abstraction to permit high level applications to use continuous byte streams in larger numbers easily and more efficiently. SST is still in an early experimental stage and the aim of this study is to develop a new module for this Protocol so it can be simulated and tested on the mostly used scientific network simulator in NS2. By implementing the module in NS2, we can simulate many aspects of this protocol and fine-tune it before we implement it on high level applications.

INTRODUCTION

Modern Web browsers and other transaction-oriented applications similarly challenge TCP’s simple, totally ordered stream abstraction, with their need to submit many logically independent or parallel requests to one or more servers efficiently. TCP forces applications to choose between using one stream per transaction, as in HTTP 1.0, which can be inefficient due to the costs of creating and destroying short-lived streams, and multiplexing many logical transactions onto a smaller pool of streams, as in HTTP 1.1, which creates the same head-of-line blocking problem, as in real-time media applications, where one lost packet delays delivery of all of the transactions queued behind it on the same stream [3], [4].

TCP’s limitations have been well-known for years [1], [5] and they represent the motivating force behind the development of several alternative transport protocols, such as RDP [9], SCTP [13], and DCCP [6]. While each one of the alternative transports tends to address heavily overlapping needs and problem areas, each one also takes a different approach to solving them. A common feature of all existing alternative transports is that they move away from TCP's conceptually simple byte stream semantics and toward transport abstractions. For example, DCCP provides unreliable, unordered delivery equivalent to UDP but with congestion control [6]; RDP provides reliable, optionally sequenced message delivery with congestion control [9]; SCTP provides reliable, optionally sequenced message delivery similar to RDP, but it also allows the application to associate each message with one of several logical streams within the application's transport connection, thereby permitting messages on different streams to be delivered out of order [13].

A common problem with all of these message-oriented transports is that their message abstraction does not scale arbitrarily the way that TCP's byte stream abstraction does [11]; instead, the application must break up large transactions into reasonably sized messages to avoid subtle performance problems or outright transport failures [8], [10]. SST enhances TCP's byte stream abstraction to permit applications to use streams in larger numbers easily and efficiently.

This paper is organized in three parts. The first part introduces the basic idea behind the new transport abstraction and the motive that attracted us toward this work. The second part of the paper represents our current work related to the simulator and the implementation of the module. The third and last part of the paper ends with the conclusions and the related future work.

MOTIVATION

The network simulator NS2 is evolving as the most used simulator by the progressive network research departments in universities and it is becoming more and more sophisticated and its testing results of different protocols and different network scenarios have been proven very important for their development [2], [7], and [12]. Also, another important point in favor of spending intellectual energy and valuable time of all these scholars worldwide over this simulator is the fact that although we live in a century when personal computers and their connections costs has decreased hundreds of times compared to when they were first implemented, it is not always possible to have all the needed resources for setting up different testing scenarios.

Actually in Albania it is very difficult to have all the required hardware resources to test or deploy different network scenarios through laboratories that is why a network simulator is very helpful to be developed and used by the students. The fact that NS2 has been recently introduced as a network simulator in laboratories at our university, results in a desire not just to use this tool as a simple or let it be a more well trained user, but also to add to this tool new modules to widen and broaden its range of operability. Also, as we displayed in the introduction part of this article, ourselves being overwhelmed by the emerging and new concepts of transport protocols and the need to really find and develop an efficient one, concluded in joining this two interests in a common endeavor.

The goal of this project is to develop an efficient module of the SST protocol for NS2 while always aiming to be faithful to its original deployment of the user space prototype [3], [4].

SST MODULE IMPLEMENTATION

SST is implemented as a user-space prototype so that it can be easily used by application layer developers in their applications by just adding and calling its functions in a SST library [3], [4]. It is implemented originally by running on top of UDP, although, the transport abstract allows its implementation on top of the network layer IP protocol, which is the way that we planned to develop the module. Also it is to be noticed that as SSTs original implementation is yet to be finished, the ns2 module too is not yet complete. It is still to be tested under different test cases and it is to be applied as a patch and to be distributed as a document for ns2 users. The implementation passes through three different phases that comprise the analysis of the architecture of the protocol, the design of the simplified module while specifying the collaboration and inheritance diagrams of the sst ns2 agent and finally the implementation phase which is robustly based on the first two.

Analysis and Architectural Overview of the Protocol

SST’s main application-visible enhancement to TCP’s stream abstraction is what amounts to a fork operation, meaning that given any existing SST stream, either endpoint can initiate a new stream as a child of that existing stream as it can be seen in figure 1.

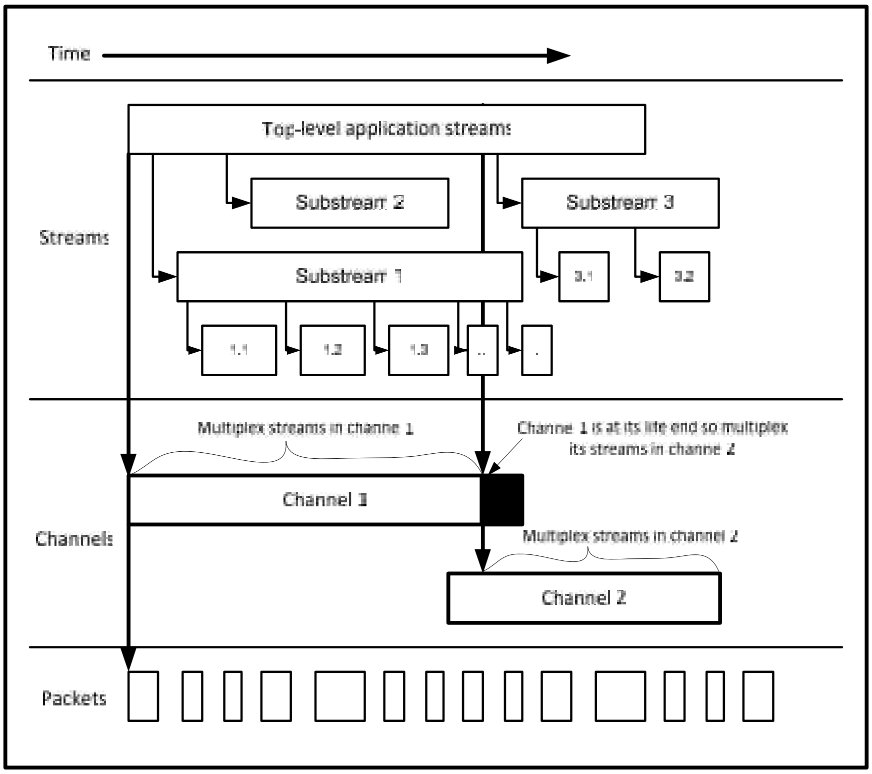
Time
Streams
Top-level application streams
Substream 2
Substream 3
Substream 1
3.1
3.2
1.1
1.2
1.3
..
.
Multiplex streams in channel 1
Channel 1 is at its life end so multiplex its streams in channel 2
Channels
Channel 1
Multiplex streams in channel 2
Channel 2
Packets

Fig.1: Structured Streams Transport Protocol fork process of the streams.

SST consists of three related sub-protocols, organized as shown in figure 2. The channel protocol is a connection-oriented best-effort delivery service that provides packet sequencing, integrity and privacy protection, selective acknowledgment, and congestion control [3], [4].

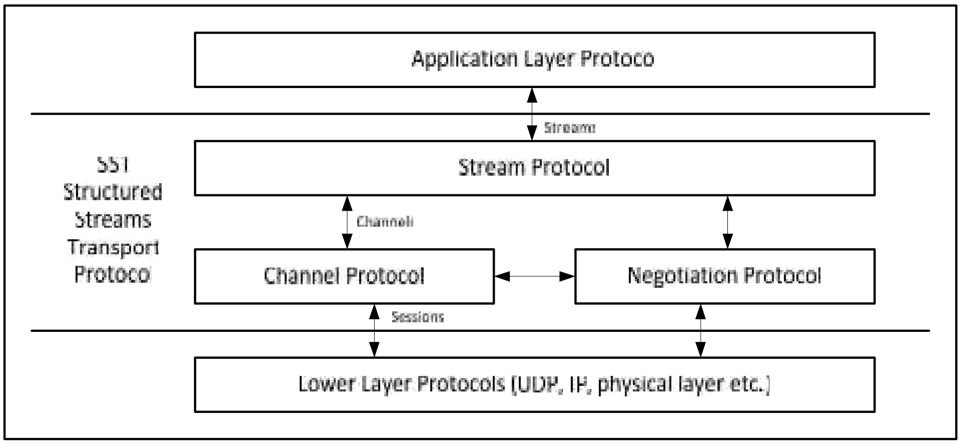
Application Layer Protocol
Streams
SST Structured Streams Transport Protocol
Stream Protocol
Channels
Channel Protocol
Negotiation Protocol
Sessions
Lower Layer Protocols (UDP, IP, physical layer etc.)

Fig.2: Structured Streams Transport protocol layered architecture.

The negotiation protocol sets up the channel protocol's state, negotiating shared security keys and optional features. Finally, the stream protocol builds on the channel and negotiation protocols to implement the reliable structured stream abstraction SST presents to the application. From the layered structure of the protocol we extract the basic functions needed for implementing the new abstraction for ns2.

SST Module Design

The process of designing the module has been based on the user-space prototype available for use by the application developers [1] while simplifying its structure. This simplification was made because it seemed to us trivial to implement such features as packet encryptation or XDR streams encoding that as well as being new features might also add complexity to the module and might distract us from the main goal of a functional and faithful module for the simulator.

Ns2 is an event driven simulator and is based on C++ for its coding and OTcl for its testing. Since the implementation code will be in an object oriented language firstly we defined and collected the needed classes and objects that will build its structure. Figure 3 shows the inheritance diagram of the SST agent.

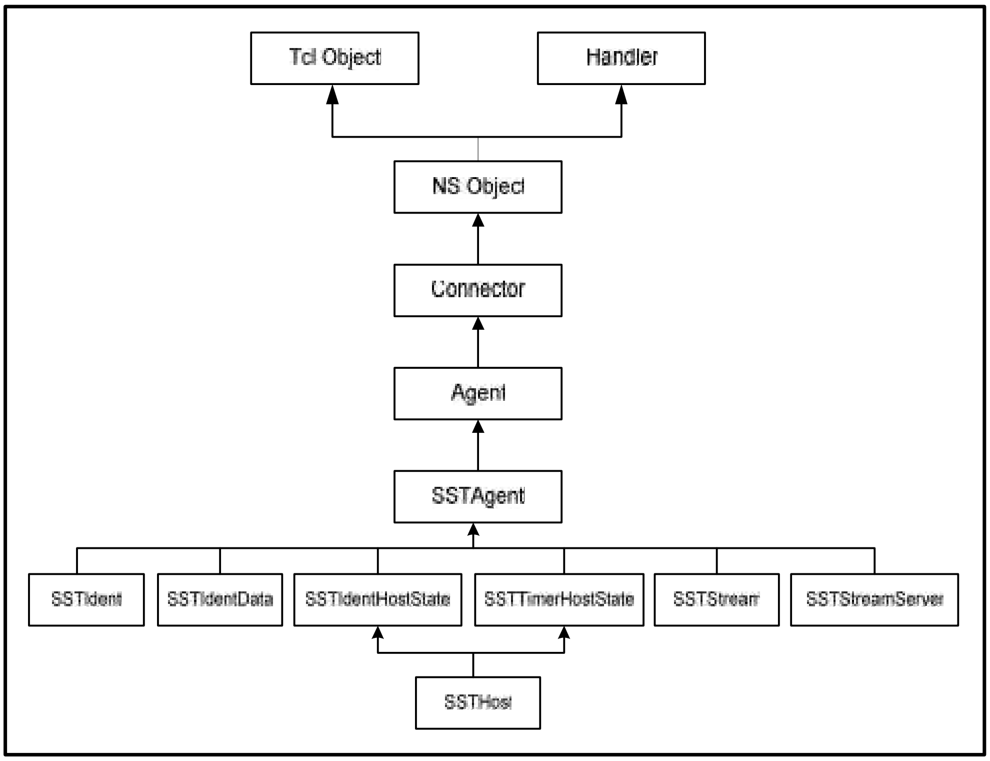

Fig.3: SSTAgent inheritance diagram.

To implement a new protocol in NS2 it is needed to build a new SST agent that inherits from class Agent [2], [12] from which inherit all the other classes needed to implement the streamserver, the host states and the other classes as shown in figure 3. The timer, which is not shown in the above figure, is always needed in NS2 to simulate a virtual scheduler for testing our module and being able to trace it with NS2 trace tools.

Coding and integrating the module

After creating the new packet type to exchange information between objects in the simulation, we add our new struct hdr_sst_pkt so that our control packets will be able to be sent and received by nodes in the simulation. Also we created the new protocol agent to link the application layer with the network layer, and proceeded with the following steps for completing the implementation:

4. specify the position of every class in the class hierarchy;
5. creating a new header file for every respective class;
6. creating every class and filling it with methods;
7. defining the connection with OTcl;
8. Finally we build and debug the module.

Finally, there are a few steps needed to integrate our new module in ns2 [2], [7], [12] for further simulations and testing.

CONCLUSION

The need for finding new techniques and new transport algorithms that manage to stay up-to-date with the emerging new requirements of applications, while staying faithful to the final aim of improving the overall performance of the communication is tending to be overwhelming and also it is proven to be the main goal of many research and development teams dealing with network communications. The new structured streams transport protocol aims to address some of TCPs main problems for efficiently using all the available resources and addresses the problems of transaction size and instance association by augmenting traditional streams with an explicit hereditary structure.

The primary way of testing and evaluating a protocol and the benefits deriving from its application is to create a real world implementation in an operating system and building a testing environment with expense routers, switches and end point hosts. However, in countries like Albania, where there might not have enough budget for projects to take this approach, it is simplest and costless to implement the protocol in a simulation environment like ns2 that stays faithful to a real world implementation and is not limited by hardware resources. Also another important characteristic from which we can benefit is that it is easier to modify the testing environment, the protocol implementation and also to monitor its performance in a wide range of scenarios.

We note that the real world module of SST is still not yet fully released [2], [3] and our actual ns2 module has not been implemented with all the specified SST functionalism, leaving outside the integrated IPsec like encryption and XDR streams. Also our future work is aiming toward creating testing scenarios to compare SST with TCP and UDP native transport protocols in terms of performance and ability to deal with different application transaction size.

REFERENCES

[1] Balakrishnan, H., Padmanabhan, V.N., Seshan, S., Stemm, M., Katz, R.H., (1998) TCP behavior of a busy Internet server: Analysis and improvements. IEEE INFOCOM 98 1, 252-262.

[2] Fall, K., Varadhan, K., (2008) The ns Manual. Retrived 2008, from http://www.isi.edu/nsnam/ns/ns-documentation.html.

[3] Ford, B., (2007) Structured Streams: a New Transport Abstraction. Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications, 361-372.

[4] Ford, B., (2007) Structured Stream Transport Preliminary Protocol Specification. Retrived 2008, from http://pdos.csail.mit.edu/uia/sst/.

[5] Information Sciences Institute, University of Southern California, (1981) Transmission Control Protocol Darpa Internet Program Protocol Specification. RFC 793, 9-52.

[6] Kohler, E., Handley, M., Floyd, S., (2006) Datagram congestion control protocol DCCP. RFC 4340.

[7] Mattsson, N., (2004) Master's Thesis a DCCP module for ns-2. Lulea University of Technology, Sweden. Retrieved 2008, from http://epubl.luth.se/1402-1617/2004/175/.

[8] Medina, A., Allman, M., Floyd, S., (2005) Measuring the Evolution of Transport Protocols in the Internet. ACM SIGCOMM Computer Communication 35, 37-52.

[9] Partridge, C. and Hinden, R., (1990) Version 2 of the reliable data protocol (RDP). RFC 1151.

[10] Postel, J., (1980) User datagram protocol. RFC 768.

[11] Rajamani, R., Kumar, S., Gupta, N., (2002) SCTP versus TCP: Comparing the performance of transport protocols for web traffic. Technical report, University of Wisconsin-Madison.

[12] Ros, F. J., Ruiz, P.M., (2004) Implementing a New Manet Unicast Routing Protocol in NS2. Retrieved 2008, from http://masimum.dif.um.es/nsrt-howto/.

[13] Stewart, R., Stream control transmission protocol, October 2000. RFC 2960.