Documentation for Satellite Transport Protocol

Tom Henderson

August 2000

1 Introduction

This document describes the Satellite Transport Protocol (STP), an experimental transport protocol for use over IP-compatible networks that was developed under the University of California, Berkeley's BARWAN research project. STP was originally developed for the BSDi Unix operating system and was subsequently ported to FreeBSD Unix, and is freely available at `ftp://daedalus.cs.berkeley.edu/pub/stp/`.

STP was specifically designed to overcome the following three problems encountered with TCP over satellite links (particularly satellite links that are asymmetric in bandwidth). These problems persist even in state-of-the-art TCP implementations that have satellite-friendly options such as "SACK" and "window-scale":

- **Reverse channel usage** For unidirectional data transfers, STP uses less of the reverse channel bandwidth than does TCP, commonly achieving an average reduction of 20% for object-laden Web pages (such as `www.cnn.com`), and a reduction of up to an order of magnitude for large images and file transfers.
- **Performance with high BERs** By using selective negative acknowledgments and a robust data acknowledgment system, STP operates efficiently under BERs that can range as low as 10^{-5}.
- **Sensitivity to RTT variations** STP is relatively immune to problems related to highly-varying round trip times (such as burstiness due to ACK clumping in TCP). In a satellite environment with multiple access delays, the round trip time can be highly variable.

STP provides TCP-like reliable byte-streaming data service. An example application that could benefit from using STP is a protocol conversion gateway in a satellite system. Such an application, as well as some further background and performance results, are described in:

[HK97] T. Henderson and R. Katz, Satellite Transport Protocol (STP): An SSCOP-based Transport Protocol for Datagram Satellite Networks, 2nd International Workshop on Satellite-based Information Services (WOSBIS '97), Budapest, Hungary, Oct. 1, 1997.

[HK99] T. Henderson and R. Katz, Transport Protocols for Internet-Compatible Satellite Networks, IEEE Journal on Selected Areas in Communications, Vol. 17, No. 2, pp. 345-359, February 1999.

2 Operation

STP has four basic packet types for data transfer (we ignore, for now, the additional packet types needed for connection setup and release). The *Sequenced Data (SD)* packet is simply a variable length segment of user data, together with a 24 bit sequence number and a checksum. SD packets which have not yet been acknowledged are stored in a buffer, along with a timestamp indicating the last time that they were sent to the receiver. No control data is included in the SD packets; instead, the transmitter and receiver exchange *POLL* and *STAT(us)* messages. Periodically, the transmitter sends a POLL packet to the receiver. This POLL packet contains a timestamp and the sequence number of the next in-sequence SD packet to be sent. The receiver responds to the POLL by issuing a STAT message which echoes the timestamp, includes the highest in-sequence packet to have been successfully received, and contains a list of all gaps in the sequence number space. The STAT message is similar in concept to a TCP selective acknowledgment, except that the STAT message reports the entire state of the receiver buffer (rather than the three most recent gaps in a SACK). Since each STAT message is a complete report of the state of the receiver, STP is robust to the loss of POLLs or STATs.

The fourth basic packet type is called a *USTAT (unsolicited STAT)* packet. USTATs are data-driven explicit negative acknowledgments, and are used by the receiver to immediately report gaps in the received sequence of packets without waiting for a POLL message to arrive. This allows the POLL and STAT exchange to be run at a low frequency (typically two or three per RTT when the RTT is large). In a network in which sequence integrity is guaranteed or highly likely, a USTAT can be sent upon any reception of a packet numbered beyond the next expected. If resequencing by the network is possible, USTATs can be delayed until there is a high probability that the missing packet was not reordered by the network. However, if the

USTAT is sent too early there is only the small penalty of a redundant retransmission. USTATs are the primary form of negative ack**n**owledgment, and STATs recover all second-order losses.

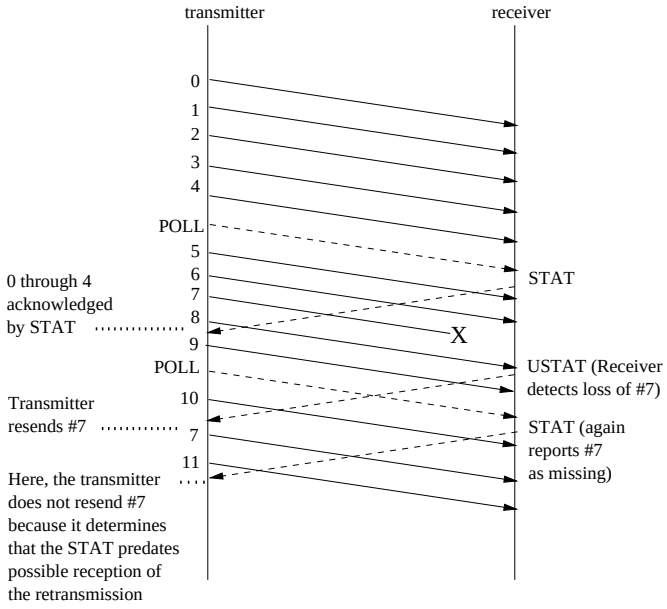

Figure 1: Example of STP bulk data transfer.

The basic operation of STP can best be illustrated by an example. For simplicity, Figure 1 only illustrates one direction of data transfer, assumes that sequence integrity of transmissions is preserved, and does not contain packets involved with beginning and closing a connection. In the example, the transmitter sends a series of consecutively numbered packets. After packet (SD) #4 is sent, a POLL packet is sent (due to either the expiration of a POLL timer or a threshold on the number of new packets sent). The POLL tells the receiver that the next message to be sent is #5, so the receiver knows that it should have received packets 0 through 4. In this case, since they have all been received, the receiver returns a STAT packet acknowledging all data up to and including packet #4. After sending the POLL, the transmitter continues with packets 5 through 9. However, packet #7 is lost. The receiver detects this loss upon receipt of packet #8 and immediately requests retransmission of #7 with a USTAT packet. Before this USTAT is received at the transmitter, the transmitter agains sends a POLL packet. Upon reception of the USTAT, the transmitter immediately resends #7, continues on with new data transmission, and then receives a STAT packet again reporting #7 as missing. However, the timestamp in the STAT packet allows the transmitter to determine that the retransmission has not yet had an opportunity to reach the receiver, thereby avoiding an unnecessary retransmission. If #7 had again been lost, the next STAT message would have stimulated a second retransmission.

Figure 2 illustrates a hypothetical data transaction in which the client (the initiator of the connection) writes 400 bytes of data to a server and receives an 8000 byte response. The example illustrates typical system calls that would be used by the application. The connection uses TCP-like window control, so that the congestion window builds by one packet for each packet acknowledged. The server is listening on a particular port, and the client connects to that port by issuing a `connect()` system call, which causes STP to send a BGN packet. This exchange of BGN and BGNAK coordinates the sequence numbers to be used by both sides and establishes the window sizes in each direction. The client then writes 400 bytes to the socket, which stimulates an SD to be sent to the other side. In this case, the client is configured to POLL with the first burst of data, so the actual packet sent is a concatenation of an SD with a POLL (see Appendix A about concatenation of packet types). The server responds to the POLL by issuing a STAT, reporting the next sequence number (#2) that the server expects to receive. The server then writes 8000 bytes of data to the socket, and it requires six packets to complete the transfer. Upon successful completion of the data transfer, both sides exchange END messages to close down their respective halves of the connection.

3 Use

An application that normally uses TCP can be easily converted to one that uses STP by changing the protocol number used to invoke the TCP socket. For example, the following example line of C code opens a socket and returns a file descriptor pointer which is assigned to `sockfd`:

```
sockfd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

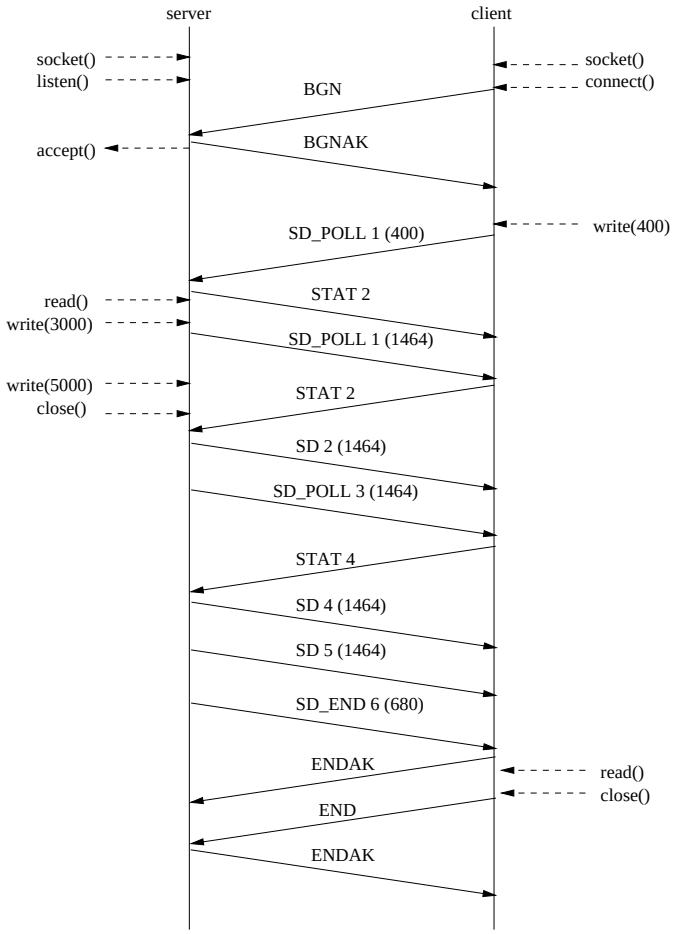

Figure 2: Example of a STP transaction.

STP can instead be invoked by defining a constant `IPPROTO_STP`, with the value equal to the STP protocol number, and by replacing the use of `IPPROTO_TCP` above. All subsequent operations on the descriptor `sockfd` will then use an instance of the STP protocol. We have selected protocol number 11 (allocated to the now-defunct Network Voice Protocol) as the STP protocol number, although it would be fine to allocate a number from the free space.

3.1 Socket buffer sizes

In order to sustain high throughput over satellite channels, the socket buffers must be set to a value larger than the kernel defaults. In BSD systems, the `SO_SNDBUF` and `SO_RCVBUF` socket options typically default to 64 KB. The `setsockopt()` system call can raise this limit on a per-socket basis; the maximum socket buffer size is implementation dependent.

3.2 Configuration options

There are a number of configuration options that affect the performance and operation of the protocol. All configuration options are listed in the file `stp.h`, but we summarize here a few of the main options and their implications. If any of these options are desired to be accessible and changeable without recompiling the kernel, they can be added as `ioctls`.

3.2.1 Flow control

- `WINDOW_CONTROL` Set to 1 if TCP-like window control is to be used. Mutually exclusive with RATE_CONTROL.
  - `INITIAL_CWND` Initial value (in packets) of the congestion window.

  - SDP_THRESHOLD When opening up the window, this constant tells the transmitter to POLL at the end of each burst of data.
  - IDLE_RESTART Number of seconds of idle time after which the sender should reset the congestion window to its initial value.

- RATE_CONTROL Set to 1 if an explicit sending RATE is to be used. Mutually exclusive with WINDOW_CONTROL.
  - RATE The maximum sending rate, in bits per second. If set to -1, will send at an unlimited rate (subject to the constraints of MAXBURST listed below).

- MAXBURST The maximum number of packets that can be sent in any burst. Overrides any burst setting that may have been set by WINDOW or RATE control.

- INITIAL_WINDOW Initial value of receiver window to use when sending data before the BGNAK has been received (for fast connection opening).

3.2.2 Polling

The frequency with which a STP connection POLLs has implications on both how much reverse-channel bandwidth is used (to return STATs) and how fast the transmitter can acknowledge data. In general, if either the BER is non-negligible or the transmitter and receiver are operating with small windows, more than one POLL per round trip time is warranted. Otherwise, it is possible to operate with high performance with more infrequent POLLs.

- POLLS_PER_RTT Number of POLLs to be sent each RTT.

- MIN_POLL_INTERVAL Minimum POLLing interval (in ms).

- STP_POLL_LIMIT If running POLLs at very low duty cycle, it is useful to trigger a poll if a certain fraction of the maximum send buffer is outstanding. This triggers a POLL if (1/STP_POLL_LIMIT)th of the buffer has been sent since the last POLL.

4 Performance

This section describes some of the performance results that we have observed with the newest STP implementation. These experiments were performed in the summer of 1999. Figure 3 illustrates our experimental testbed. We used STP between two machines, each running an instance of the (freely available) Squid Web cache software. Between the machines, we used a satellite channel delay simulator that inserted approximately 300ms delay (each way) between the machines. The cache on the terminal side was configured as a proxy only (i.e., it did not cache any content), and it was directed to use the cache on the other side of the simulated satellite link as its parent cache for all Web requests. Likewise, we configured our Web browsers to not cache any content, so therefore, all page requests were handled by either the parent cache or servers in the Internet. We then wrote scripting software to trace and log the total number of bytes (TCP/IP or STP/IP) used over the satellite channel, in each direction. For STP, we used unlimited rate flow control, and configured the sender to send one POLL per round trip time.

We experimented with image transfers and Web page transfers. For typical Web page transfers, we repeatedly downloaded (five times each) the front page of ten of the most popular sites on the Internet.[1] On average, when looking at popular, object-laden web pages, STP was able to reduce the number of bytes on the backchannel by approximately 20%. STP also used approximately 2% less bytes on the forward channel (from server to client). We then repeated our tests by downloading a large image (320 KB) ten times. In this case, STP reduced the number of bytes on the reverse channel by approximately 85%.

Our previous results using the FreeBSD implementation of STP, when driven by a Web traffic generator based on HTTP traffic traces, indicated that STP should result in a reduction of roughly 25% (when compared with TCP) in the number of packets exchanged between server and client. These experiments were conducted with STP making use of the T/TCP socket calls described above, resulting in a slightly higher efficiency than we obtained in our recent experiments. When we conducted a large number of large file transfers (10 MB files) and compared TCP performance to that of STP, we observed that STP's reverse bandwidth usage was an order of magnitude less than that of TCP.

[1] Yahoo, AOL, Microsoft, Excite, Altavista, CNN, Xoom, Go.com, Lycos, and Amazon.

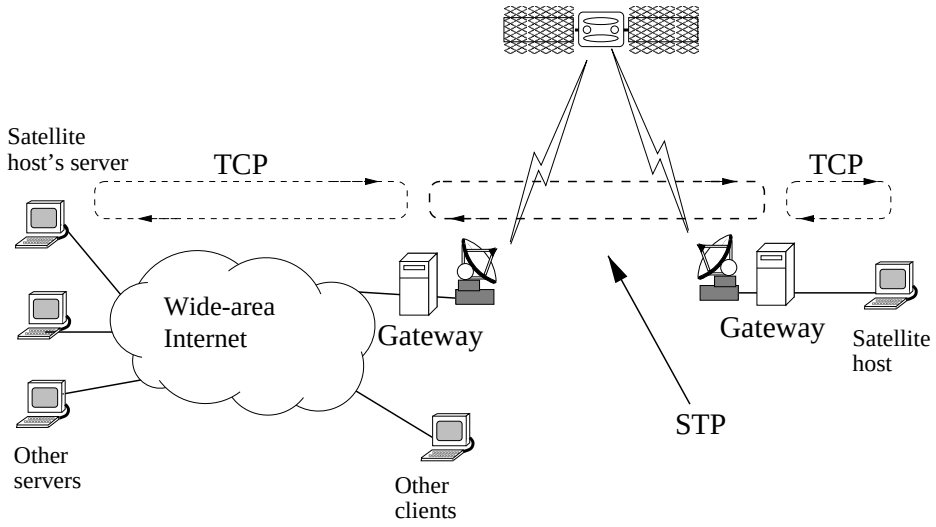

Figure 3: Illustration of the environment used for performance experiments.

5 Summary

The Satellite Transport Protocol is specifically designed to outperform TCP in environments characterized by either high bandwidth-delay products, high bit error rates, or bandwidth asymmetry (or a combination of the above). The main difference between STP and TCP is how the two protocols acknowledge data– STP attempts to acknowledge data as infrequently as possible while still offering rapid recovery from losses in the data transfer. The tradeoff between the two (rapid recovery from errors and reverse channel efficiency) can be tuned by optimizing the frequency and conditions under which the data transmitter sends POLL packets to the receiver.

5.1 Lessons learned

Based on experience with this protocol, the following recommendations are offered on using STP:

1. If the objective is to only perform a protocol conversion between TCP and STP, operating in split mode (i.e., not going through any application), it would be better to modify this implementation to perform the split entirely in-kernel. Essentially what must be done is to allow a TCP and a STP connection to share a single send/receive buffer and coordinate access between the two. For example, in passing a packet from a TCP connection to a STP connection, pass a pointer to the received packet. STP can then overwrite the header information and now "owns" the packet, but no memory-to-memory copies are needed.

2. There does not appear to be much of an advantage over TCP when using STP to tie Web caches together, because the nature of data transfer between proxies for most object-laden Web pages is that packets between the caches dribble through at the rate at which they are received from the Internet, and more frequent ACKing (POLLing) is therefore required to sustain the connection. In such a case, it would be worthwhile to consider a different type of application that gathered up whole pages, perhaps compressed them, and sent the whole page as a unit to the cache; i.e., turn the data into more of a bulk transfer, which is what STP excels at. In the extreme, if the channel were not very lossy it may be worthwhile to consider running an application-level protocol sending such complete pages over UDP, thereby avoiding the socket buffer kernel resources that reliable streaming protocols like TCP and STP require.

A Appendix: Additional implementation information

This appendix provides additional information, although the code is the best source. Figure 4 illustrates the STP states and the main state transitions between them. Not all possible state transitions are shown, only the main ones that indicate in general how a state is reached. Typical stimuli and responses that occur between state changes are illustrated. In addition, one of four main timers may be running in a state (BEGIN, END, KEEPALIVE (KA), or POLL). In italics next to the state bubble is the timer (if any) that is running in that state.

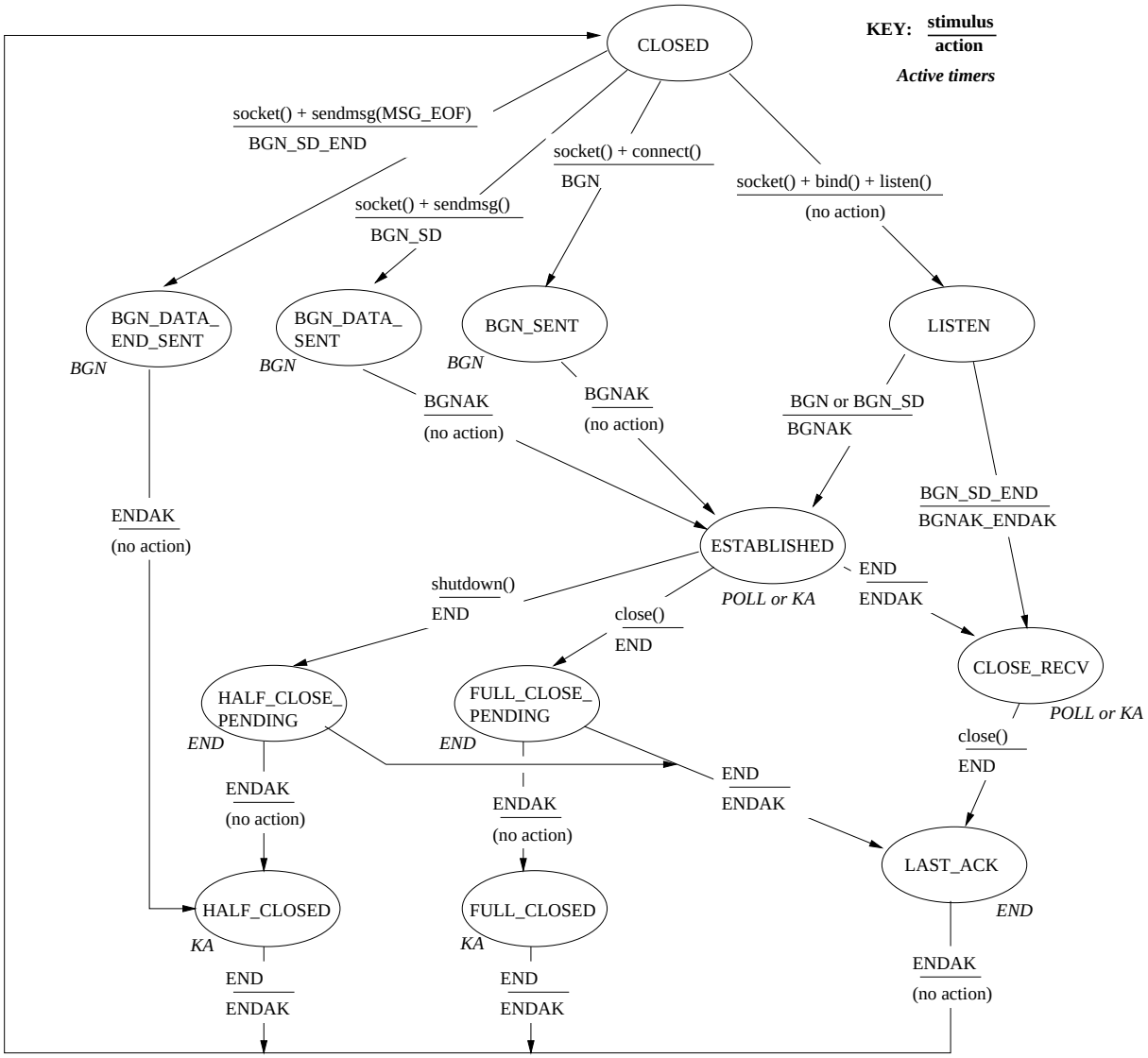

Figure 4: Illustration of the major state transitions in STP.

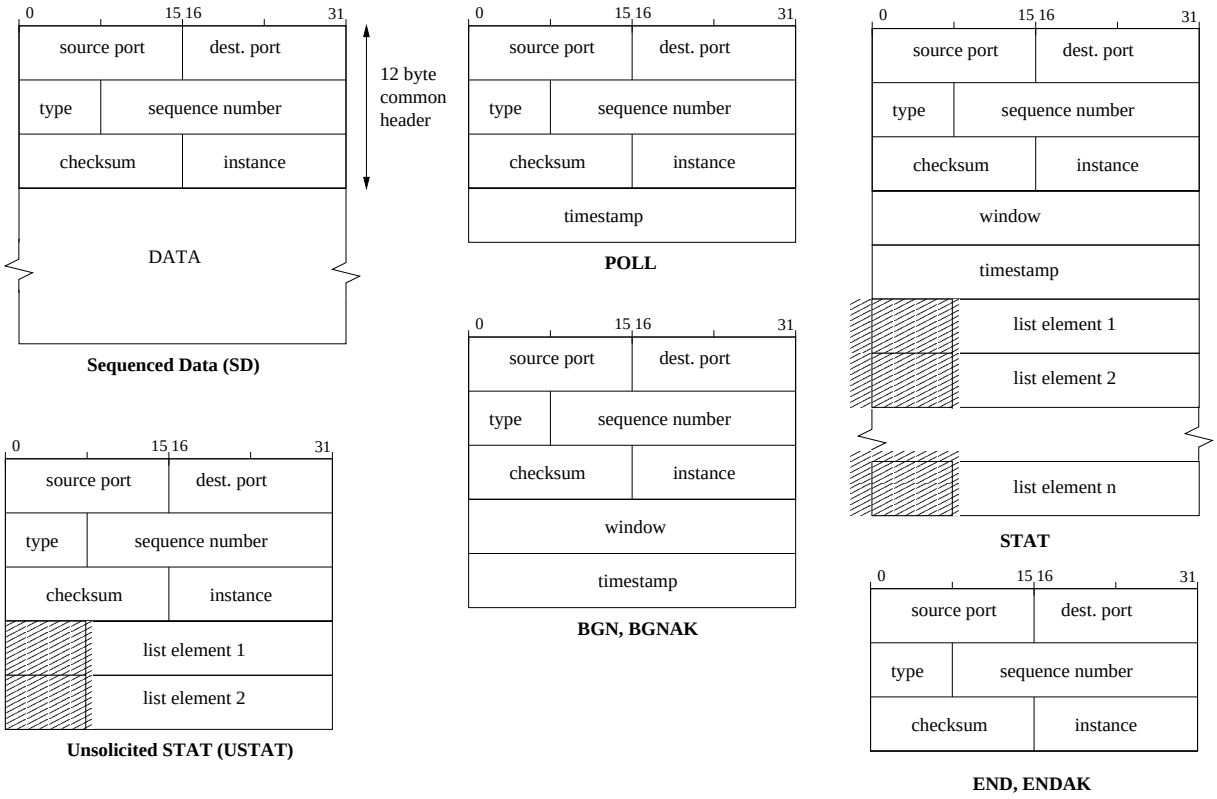

Figure 5: Illustration of the key packet types in STP.

Figure 5 illustrates the key packet types used by STP. The fields are byte-aligned and the data portion of packets is 32-bit aligned. Above, we discussed the use of the SD, POLL, STAT, and USTAT packets. The BGN and BGNAK are used to open a connection, and the END and ENDAK are used to close a connection. Each packet contains a 12-byte common header that includes the source and destination port numbers (as in TCP), a type field, a 24-bit sequence number, a 16-bit checksum, and an instance number (to distinguish between different connections that may happen to use the same port numbers). Certain packets are permitted to be concatenated for the purpose of conserving the number of packets transmitted; for example, an SD and a POLL packet may be concatenated, in which case the type field is the logical “OR” of the SD and POLL values, and the POLL timestamp precedes the data.

B Appendix: Relation of STP to other protocols

STP is derived from the ATM-based link layer protocol known as SSCOP (Service Specific Connection Oriented Protocol) [ITU94]. SSCOP was intended to become the end-to-end native transport protocol for ATM, but ATM has never been deployed end-to-end. SSCOP is currently used as the signaling link layer protocol in the Signaling AAL (SAAL), and as a link layer protocol for wideband CDMA networks. SSCOP itself is derived from two convergent research efforts in the late 1980s: light-weight transport protocol research at AT&T Bell Laboratories by Sabnani, Netravali, and Roome (the so-called SNR transport protocol [Net90]), and satellite protocol research at COMSAT Laboratories [Mil86].

Many of the features in satellite-optimized TCP can be found in STP. For example, the concept of selective negative acknowledgments are implemented in SCPS-TP (TCP for space communications) [Tra97]. "Rate-pacing" TCP implementations and reductions in the amount of ACK traffic were investigated by [Bal97]. The expansion of TCP protocol syntax to handle large delay-bandwidth product links was standardized in Internet RFC 1323 [Jac92]. The fast retransmit mechanism in TCP [Ste97] makes all first order loss recovery data-driven (although repeat losses in TCP are always handled by a timeout). The main difference between TCP and these satellite-optimized TCPs is that STP implements the desirable features as default protocol behavior. In fact, such TCP extensions can be viewed as TCP evolving towards the basic STP design philosophy. As a result, the performance of suitably modified TCPs can approach that of STP over satellite channels.

There have been many other transport protocols invented over the years. Rather than exhaustively comparing STP with all of them, I will briefly discuss STP's relation to two protocols explicitly designed for the satellite or related environments: the Network Block Transfer (NETBLT) protocol [Cla87] and the Xpress Transport Protocol (XTP) [Wea94]. NETBLT is designed for large file transfers in which the application submits data to the network in large, fixed block sizes called "buffers." While NETBLT uses selective acknowledgments and supports rate controls in addition to window control, STP provides a byte streaming data service (which is more closely aligned with applications), has a looser coupling between protocol logic and timers (reducing reliance on accurate values of the RTT), offers reduced latency in protocol handshaking (making it suitable for small transfers), and incorporates dynamic congestion control. As for XTP, many of the design principles of STP are also reflected in XTP (such as transmitter driven acknowlegments, USTATs and "fastnaks," selective acknowledgments). The main differences are that XTP has a much richer API (including multicast support) and exposes much more protocol policy to the applications, whereas STP was designed to meet the standard sockets API for unicast traffic and is more optimized for bit-efficiency (such as smaller protocol headers).

References:

[Bal97] H. Balakrishnan, V. Padmanabhan, and R. Katz, "The Effects of Asymmetry on TCP Performance," Proceedings of Third ACM/IEEE MobiCom Conference, pp. 77-89, Sept. 1997.

[Cla87] D. Clark, M. Lambert, and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol," Internet RFC 998, 1987.

[ITU94] "B-ISDN Signaling ATM Adaptation Layer– Service Specific Connection Oriented Protocol," ITU-T Recommendation Q.2110, 1994.

[Jac92] Jacobson, V., R. Braden, and D. Borman, "TCP Extensions for High Performance," Internet RFC 1323, 1992.

[Mil86] K. Mills, D. Chitre, H. Chong, and A. Agarwal, "A Joint COMSAT/NBS Experiment on Transport Protocol," Proceedings of 7th International Conference on Digital Satellite Communications (ICDSC), May 1986.

[Net90] A. Netravali, W. Roome, and K. Sabnani, "Design and Implementation of a High-Speed Transport Protocol," IEEE Transactions on Communications, vol. 38, no. 11, pp. 2025-38, Nov. 1990.

[Ste97] Stevens, W. R., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," Internet RFC 2001, 1997.

[Tra97] E. Travis, W. Durst, and G. Miller, " TCP Extensions for Space Communications," Wireless Networks, vol. 3, no. 5, pp. 389-403, 1997.

[Wea94] A. Weaver, "Xpress Transport Protocol, Version 4.0, " http://www.cs.virginia.edu/ netlab/xtp_stuff/xtp4_tutorial.ps, 1994.