Nachos Transport Protocol Specification

LAST MODIFIED: 1 May 2004 by Amir Kamil

Overview

The Nachos Transport Protocol provides full-duplex, reliable, byte-stream communication with no message size limits. It runs on top of the nachos network abstraction, defined in `nachos.machine.NetworkLink`. User programs access the transport protocol via the nachos system calls `connect()`, `accept()`, `read()`, `write()`, and `close()` which are documented in `nachos/test/syscall.h` and in the project description.

A connection between two endpoints is uniquely identified by the tuple (*local machine address*, *local port*, *remote machine address*, *remote port*). A state machine is maintained for each unique conenction. The state transitions are covered in detail later in this document but first we give a brief overview of the life cycle of a connection. This overview omits many details related to handling error conditions -- the details are included in the full transition table below.

To establish a new connection between two endpoints, one endpoint first calls the `connect()` system call. This endpoint is referred to as the *active* endpoint. A process at the other endpoint (calledthe *passive* endpoint) must later call the `accept()` system call. As these system calls are invoked, the underlying protocol carries out a *2-way handshake*. When a user process invokes the `connect()` system call, the active endpoint send a synchronize packet (SYN). This causes a pending connection to be created. The state for this pending connection must be stored at the receiver until a user process at the passive endpoint invokes the `accept()` system call. When `accept()` is invoked, the passive end sends a SYN acknowledgement packet (SYN/ACK) back to the active endpoint and the connection is established.

Once the connection is established both sides may send data packets. Each packet is labeled with a unique 32 bit sequence number. Upon receipt of a data packet, the receiver sends an acknowledgement packet (ACK) back to the sender. Like TCP, the Nachos Transport Protocol uses a sliding window. Unlike TCP, however, the size of the window is always fixed at 16 packets. It is the responsibility of the sender to ensure that it never sends more than 16 unacknowledged packets. To guarantee that all packets are eventually received, the sender must periodically retransmit any packets that have been sent but not yet acknowledged. The frequency of the retransmission process should be once every 20,000 clock ticks.

When an endpoint closes a socket (via the `close()` system call), it sends a stop packet (STP) to the remote endpoint to indicate it has no more data to send. The sequence number in the STP packet must be one greater than the sequence number of the last data packet sent before the socket was closed. The endpoint that receives the STP packet may continue to receive retransmissions of any lost data packets until it has received all packets up to the last sequence number before the STP packet. However, no data may arrive with a sequence number larger than the STP packet sequence number. An endpoint should also not send any new data after receiving an STP message from the other endpoint.

When the second endpoint closes a connection, the two endpoints execute another handshake to complete the close. The second endpoint sends a FINish packet to which the first endpoint responds with a FINish ACKnowledgement (FIN/ACK) packet. At this point, the connection is finished and any associated state or resources may be deallocated.

Packet Format

Every packet in the nachos hardware has a 4 byte header that includes the link addresses of the sender and destination. The transport protocol adds an additional 8 byte header with the format shown below:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
DSTPORT								SRCPORT								MBZ												FIN	STP	ACK	SYN
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
SEQNO																															

The header fields are:

- **DSTPORT** -- The transport port number on the destination machine (the machine to which this packet is addresssed).
- **SRCPORT** -- The transport port number on the sending machine (the machine from which this packet was sent).
- **MBZ** -- Must Be Zero. These bits should be initialized to zero on transmission and ignored on reception.
- **FIN** -- A single bit indicating that the connection is being closed. See the state transition diagram below.
- **STP** -- A single bit indicating that the sender has closed the connection. See the state transition diagram below.
- **ACK** -- A single bit indicating that this packet is an acknowledgement of data received. If this bit is set, the packet should have no contents beyond the header and the SEQNO field indicates the packet from the sender's sequence number space that is being acknowledged.
- **SYN** -- A single bit indicating that this packet is the first packet initiating a new connection. See the state transition diagram below.

State Machine

A connection can be in one of the following seven states at one time:

- **CLOSED** -- no connection exists
- **SYN_SENT** -- sent a SYN packet, waiting for a SYN/ACK packet.
- **SYN_RCVD** -- received a SYN packet, waiting for an app to call `accept()`.
- **ESTABLISHED** -- a full-duplex connection has been established, data transfer can take place.
- **STP_RCVD** -- received a STP packet, can still receive data but cannot send data.
- **STP_SENT** -- the app called `close()`, sent an STP packet but still need to retransmit unacknowledged data.
- **CLOSING** -- send a FIN packet, waiting for a FIN/ACK packet.

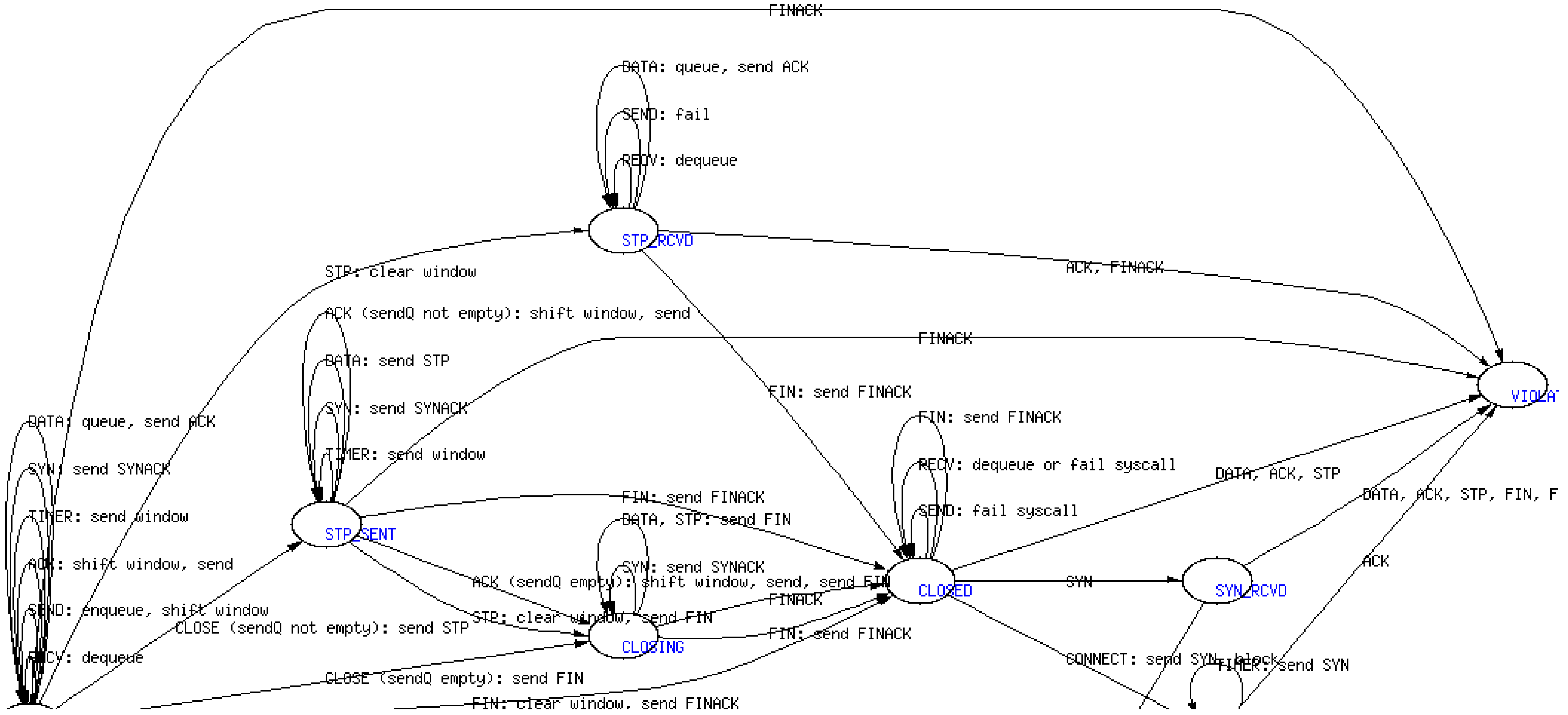
The following events may occur to cause state transitions:

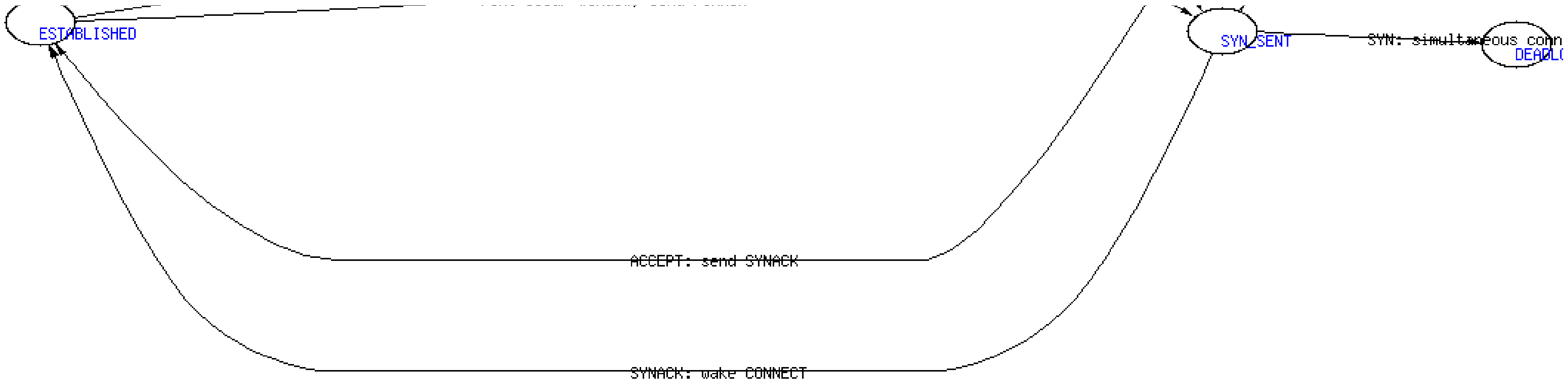
- **CONNECT** -- an app called `connect()`.
- **ACCEPT** -- an app called `accept()`.
- **RECV** -- an app called `read()`.
- **SEND** -- an app called `write()`.

- **CLOSE** -- an app called `close()`.
- **TIMER** -- the retransmission timer ticked.
- **SYN** -- a SYN packet is received (a packet with the SYN bit set).
- **SYNACK** -- a SYN/ACK packet is received (a packet with the SYN and ACK bits set).
- **DATA** -- a data packet is received (a packet with none of the SYN, ACK, STP, or FIN bits set).
- **ACK** -- an ACK packet is received (a packet with the ACK bit set).
- **STP** -- a STP packet is received (a packet with the STP bit set).
- **FIN** -- a FIN packet is received (a packet with the FIN bit set).
- **FINACK** -- a FIN/ACK packet is received (a packet with the FIN and ACK bits set).

The state transitions are summarized in the following diagram and detailed in the following table.

NOTE: There is a small **bug** in the diagram. The **TIMER** event in the **CLOSING** state should result in retransmission of the **FIN** packet.





State	Event	Action
CLOSED	CONNECT	send SYN, goto SYN_SENT, block
	RECV	dequue data, or fail syscall if none available
	SEND	fail syscall
	SYN	goto SYN_RCVD
	DATA, ACK, STP	protocol error
	FIN	send FINACK
SYN_SENT	TIMER	send SYN
	SYN	protocol deadlock!
	SYNACK	goto ESTABLISHED, wake thread waiting in `connect()`
	DATA, STP, FIN	send SYN
	ACK	protocol error
SYN_RCVD	ACCEPT	send SYNACK, goto ESTABLISHED
	DATA, ACK, STP, FIN, FINACK	protocol error
ESTABLISHED	RECV	dequeue data
	SEND	queue data, shift send window
	CLOSE	if send queue is empty, send FIN, goto CLOSING. else send STOP, goto STP_SENT

	TIMER	resend unacknowledged packets
	SYN	send SYN/ACK
	DATA	queue data, send ACK
	ACK	shift send window, send data
	STP	clear send window, goto STP_RCVD
	FIN	clear send window, send FIN/ACK, goto CLOSED
	FINACK	protocol error
STP_SENT	TIMER	retransmit unacknowledged packet
	SYN	send SYNACK
	DATA	send STP
	ACK	shift send window, send data. if send queue is empty, send FIN and goto CLOSING
	STP	clear send window, send FIN, goto CLOSING
	FIN	send FIN/ACK, goto CLOSED
	FINACK	protocol error
STP_RCVD	RECV	dequeue data
	SEND	fail syscall
	CLOSE	send FIN, goto CLOSING
	DATA	queue data, send ACK
	ACK, FINACK	protocol error
	FIN	send FIN/ACK, goto CLOSED
CLOSING	TIMER	send FIN
	SYN	send SYN/ACK
	DATA, STP	send FIN
	FIN	send FIN/ACK, goto CLOSED
	FINACK	goto CLOSED