The Internet (pt. 2)

CSE 486/586: Distributed Systems

Ethan Blanton

Department of Computer Science and Engineering

University at Buffalo

The Transport Control Protocol

The Internet Protocol provides only best effort delivery.

TCP [2] rides on top of IP and provides reliable delivery.

TCP attempts to identify and mitigate network congestion.

“The network” does not need to be aware of TCP for either purpose.

The End-to-End Argument

If a function requires knowledge present only at the endpoints of a communication system, that function should be implemented at the endpoints.

This is a paraphrasing of the end-to-end argument. [3]

In some cases, it is possible to implement it in the network.

The end-to-end argument says that this will be:

- More difficult
- Less reliable

Applications of the E2E Argument

The argument is frequently applied to reliability, authenticity, and privacy.

When sending data to a remote system, is it better to know that:

- A local transmission succeeded, and the network is solid
- The remote system received the data

When sending encrypted data, is it better to know that:

- The data was received and decrypted by a trusted third party who will forward it
- The data was received, still encrypted, by the final recipient

Byte Streams

TCP provides a full duplex byte stream.

Full duplex means that data can travel in both directions simultaneously.

Bytes arrive in order, as they were transmitted.

The stream has no internal structure.

(The TCP standard uses octet instead of byte.)

Segments versus Datagrams

TCP data is transmitted in segments.

The TCP byte stream is broken up into these segments.

A segment occupies an IP datagram.

Segments are an artifact of implementation, invisible to the user.

Ordering Segments

IP datagrams may arrive out of order.

TCP must be able to order its segments as they were transmitted.

It does this by giving each byte a sequence number.

Segments contain a sequence number and length.

Received segments are assembled and delivered in order.

Acknowledgments

When a segment is received in order, its bytes are acknowledged.

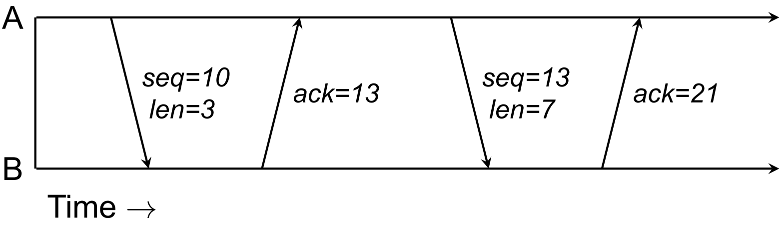
Acknowledgments (ACKs) are sent by sequence number.

An acknowledgment says:
I have received every byte up to (but not including) this sequence number.

TCP Header Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A TCP Transmission



E2E in TCP

TCP applies the E2E argument to data reliability.

ACKs track when data is processed at the remote endpoint.

Out-of-order data receipt triggers acknowledgments.

The local endpoint stores all unprocessed data.

If data is not received and processed, it is retransmitted.

Identifying Lost Data

TCP uses several algorithms [1] for identifying lost data:

- Duplicate ACKs for the same sequence number
- Selective acknowledgment (SACK) information
- Timeouts

Duplicate ACKs indicate that:

- Data is being received
- The next sequence number has not been received

We will not discuss SACK further.

Recovering from Loss

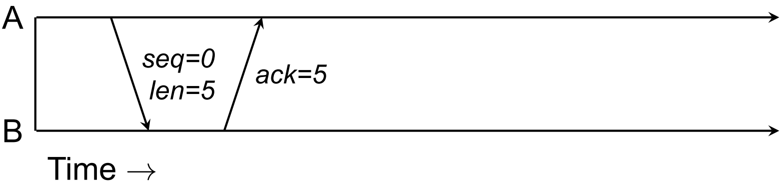
When a sequence number is identified as lost:

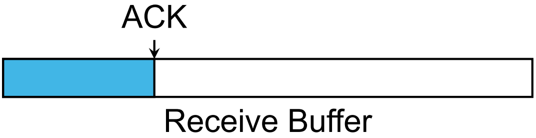
1. TCP retransmits a full segment at that sequence number
2. Resumes transmitting new data

If only one segment was lost, this normally recovers.

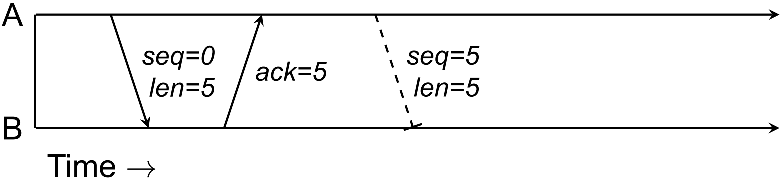
If additional segments are lost they will be detected later.

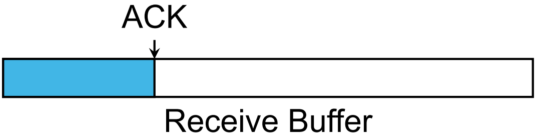
Lost Data Example



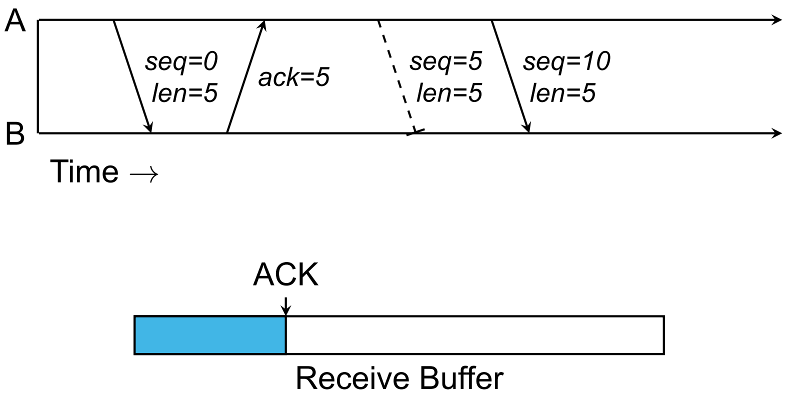


Lost Data Example

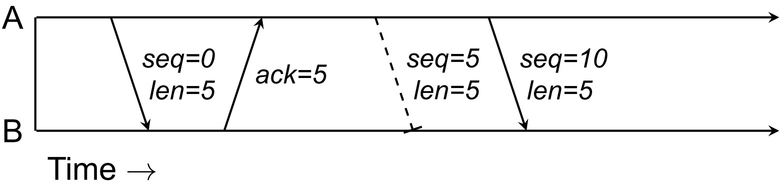


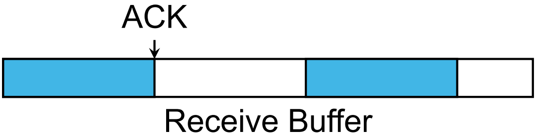


Lost Data Example

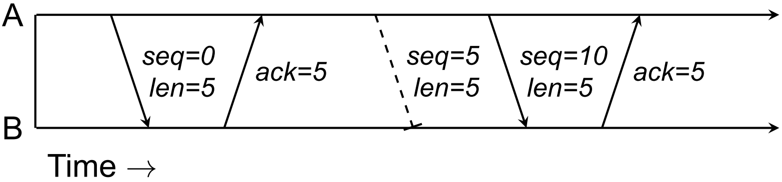


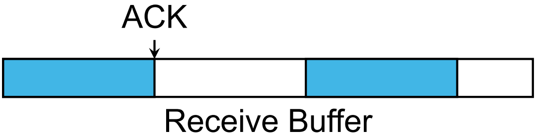
Lost Data Example





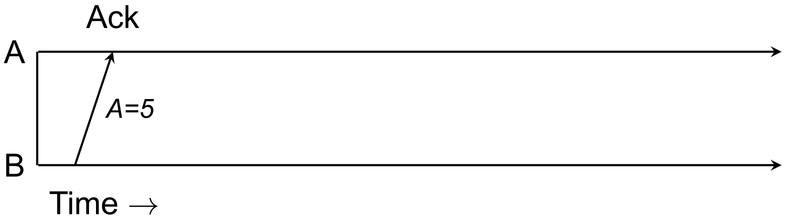
Lost Data Example





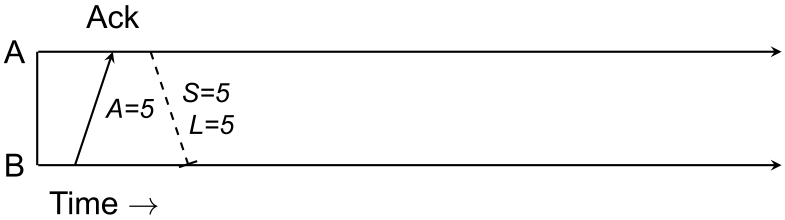
Retransmission

Three duplicate ACKs normally trigger retransmission.



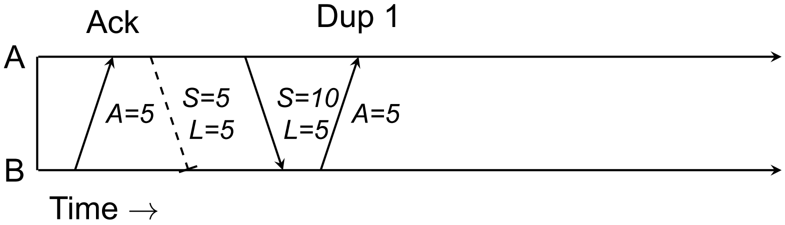
Retransmission

Three duplicate ACKs normally trigger retransmission.



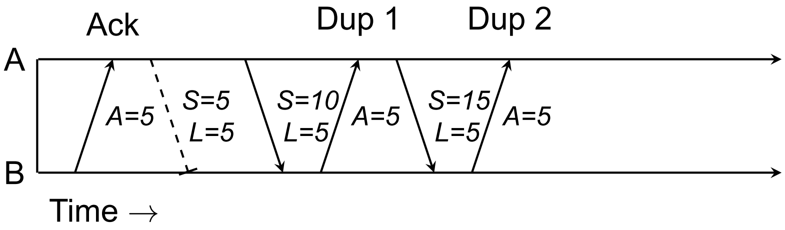
Retransmission

Three duplicate ACKs normally trigger retransmission.



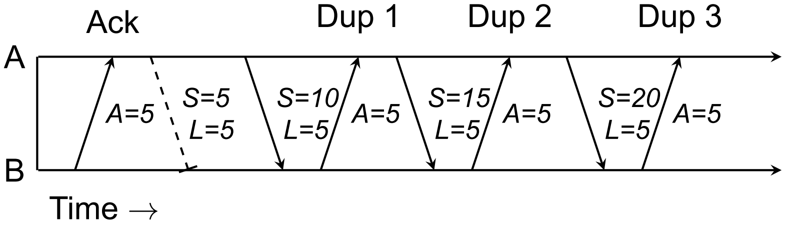
Retransmission

Three duplicate ACKs normally trigger retransmission.



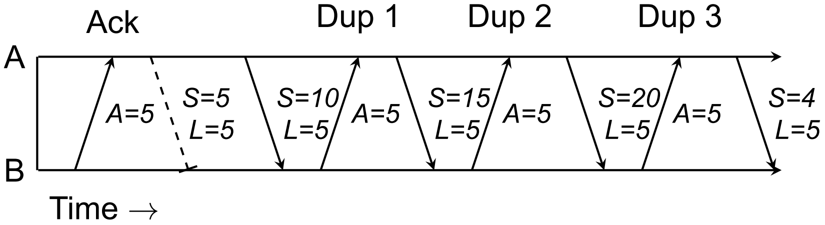
Retransmission

Three duplicate ACKs normally trigger retransmission.



Retransmission

Three duplicate ACKs normally trigger retransmission.



Other Considerations

TCP handles many other situations cleanly.

- What if acknowledgments are lost?
- What if acknowledgments are duplicated?
- What if multiple segments are lost?
- What if segments are duplicated?
- How does it know which segments to retransmit?
- What if retransmissions are lost?
- How are segments in the reverse direction handled?

A Distributed State Machine

TCP loss recovery is a distributed state machine.

Each endpoint keeps track of:

- What it has transmitted
- What it has received
- What the other endpoint has received

The endpoints cooperatively recover lost data.

Summary

- The end-to-end argument provides guidance on where to implement functionality.
- TCP provides services that IP does not.
- The TCP model is an full duplex in-order byte stream.
- TCP loss recovery is effected by a distributed state machine.

Next Time . . .

- Some thoughts on Go

References I

Required Readings

[3] Jerome H. Saltzer, David P. Reed, and David D. Clark. "End-to-End Arguments in System Design". In: *ACM Transactions on Computer Systems* 2.4 (Nov. 1984), pp. 277–288. URL: https://groups.csail.mit.edu/ana/Publications/PubPDFs/End-to-End%20Arguments%20in%20System%20Design.pdf.

Optional Readings

[1] Ethan Blanton et al. *A Conservative Loss Recovery Algorithm based on Selective Acknowledgment (SACK) for TCP*. Aug. 2012. URL: https://tools.ietf.org/rfc/rfc6675.txt.

References II

[2] Information Sciences Institute. *Transmission Control Protocol*. Ed. by Jon Postel. Sept. 1981. URL: https://tools.ietf.org/rfc/rfc793.txt.

