The Internet

CSE 486/586: Distributed Systems

Ethan Blanton

Department of Computer Science and Engineering
University at Buffalo

The Internet

The Internet is not a monolithic entity, or even a protocol.

It is a collection of:

- networks
- protocols
- organizations
- standards
- …

Understanding how and why it came about will help us understand it.

This isn’t a networking course, just some background.

Distributed Systems

The Internet is where we build our distributed systems today.

In many ways, the Internet is a distributed system!

It “solves” many of the problems we will explore.

…yet we have to solve some of them again.

We will explore the reasons why this is so.

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. [6]

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

A Network of Networks

> *The top level goal for the DARPA Internet Architecture was to develop an effective technique for multiplexed utilization of existing interconnected networks. [2]*

The Internet is fundamentally a network of networks.

Those networks may have varying capabilities.

This had a large influence on Internet design.

The Stateless Internet

The Internet is packet switched so that it can be stateless[1].

Packed switched networks route individual packets.

Endpoints maintain the state of connections.

The network itself sees only packets.

A protocol — the Internet Protocol [3] — carries those packets.

---

[1] Not entirely, but it doesn't (or *shouldn't*) keep connection state

Limited Guarantees

IP’s place in the network stack is on top of other networks.

It makes very few assumptions about those networks.

This is one of the reasons the Internet has dominated networking.

It is also one of the reasons the Internet is packet switched.

IP, in turn makes very few guarantees.

IP Datagrams

IP datagrams are small, self-contained packets containing:

- A source host
- A destination host
- Minimal other metadata
- Uninterpreted data

IP handles (almost) only moving those datagrams.

It does not provide:

- Protection from corruption
- Reliable transmission
- Connections of any kind

Best Effort

The Internet Protocol provides best effort delivery.

This means that datagrams are delivered if possible.

They are dropped if they cannot be delivered.

They may also be queued indefinitely awaiting delivery.

This means that individual IP datagrams may arrive:

- Late
- Out of order
- Not at all

Routing

IP datagrams are routed individually from network to network.

Routing decisions are made locally based on limited information.

Routing protocols provide some notion of global topology.

Due to local routing decisions, datagrams may experience:

- Routing loops
- Asymmetric connectivity
- Destinations that are locally unreachable

The Transport Layer

The transport layer provides additional services.

Transport protocols are data in IP datagrams.

The two most common Internet transport protocols are:

- UDP for unreliable, low-latency communication
- TCP for reliable, congestion-aware communication

Both provide:

- Multiple endpoints per host
- Protection from corruption (optional for UDP)

UDP

The User Datagram Protocol [5] provides little more than IP:

- Multiple endpoints (ports) per host
- A simple checksum over data

UDP datagrams also provide only best effort delivery.

UDP is often used for:

- Local communication
- Tasks requiring low latency
- Fixed-throughput communication
- Applications that can tolerate lost data

Advantages of UDP

Because UDP provides few extra services, it has little overhead.

Datagrams do not require connection establishment.

Best effort delivery is low latency.

The UDP data checksum provides some corruption protection.

Disadvantages of UDP

Best effort delivery permits loss.

UDP provides no recovery for lost datagrams.

Datagrams carry no connection or session information.

UDP provides no feedback on network conditions.

TCP

The Transmission Control Protocol [4] provides:

- a byte stream abstraction
- reliable delivery of data
- data ordering

TCP attempts to identify and mitigate network congestion.

“The network” does not need to be aware of TCP for any of this!

Advantages of TCP

TCP provides recovery from lost data via retransmission.

Congestion control allows effective sharing of network resources.

TCP connections provide:

- Convenient byte-oriented streaming semantics
- Persistent communication channels between endpoints

Disadvantages of TCP

TCP connection management adds significant overhead.

Loss recovery can increase latency substantially.

TCP requires bidirectional communication.

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.

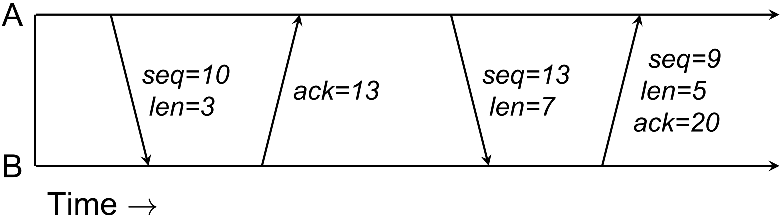
Acknowledgments

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.

A TCP Transmission



E2E in TCP

TCP applies the E2E argument to data reliability.

ACKs reveal 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

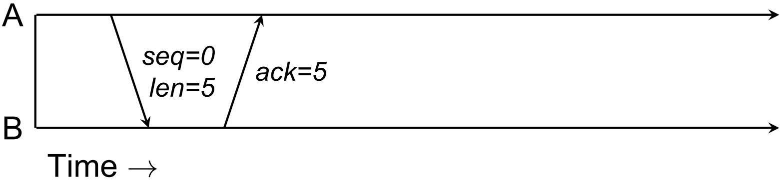
When a sequence number is identified as lost:

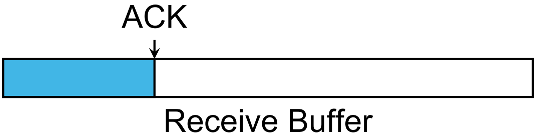
1. TCP retransmits a full segment at that sequence number
2. Resumes transmitting new data

If only one segment was lost, this normally recovers.

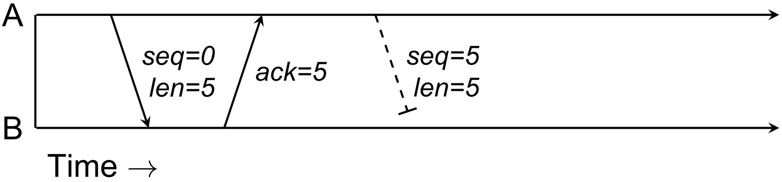
If additional segments are lost they will be detected later.

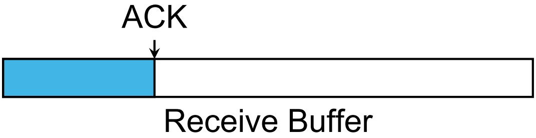
Lost Data Example



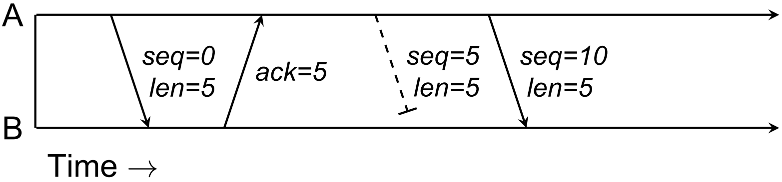


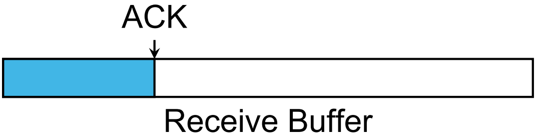
Lost Data Example



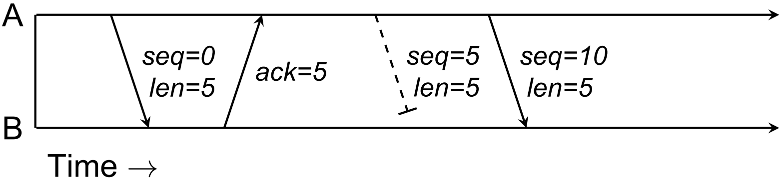


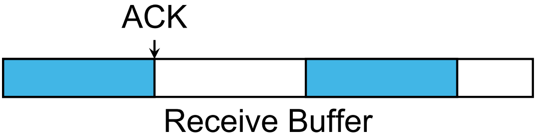
Lost Data Example



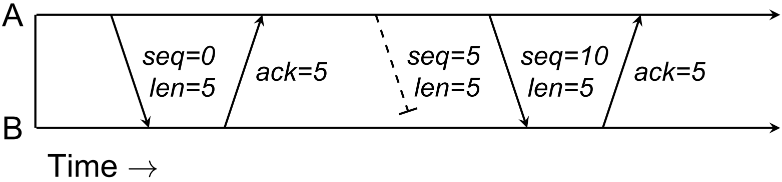


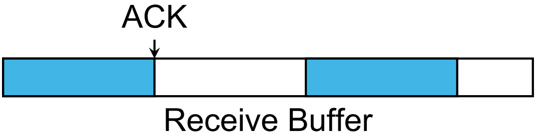
Lost Data Example





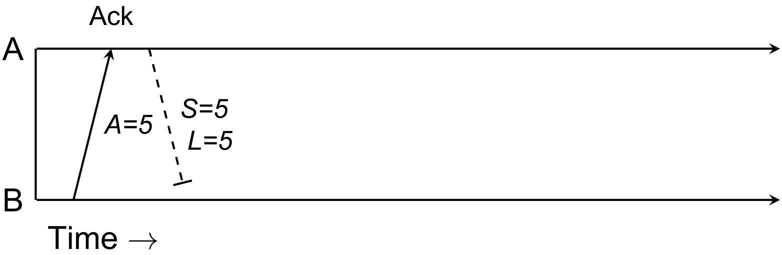
Lost Data Example





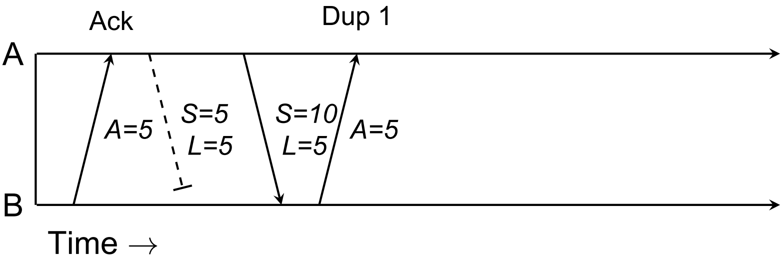
Retransmission

Three duplicate ACKs normally trigger retransmission.



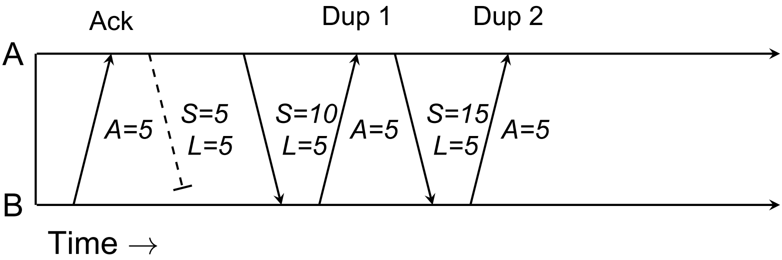
Retransmission

Three duplicate ACKs normally trigger retransmission.



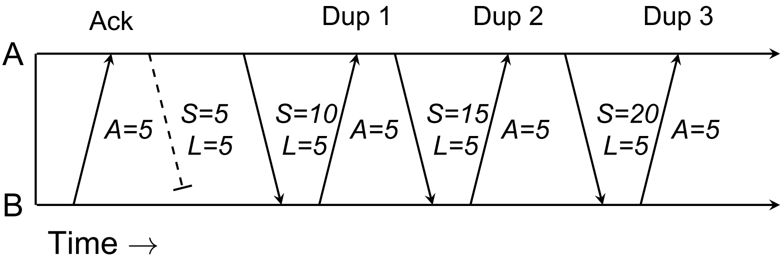
Retransmission

Three duplicate ACKs normally trigger retransmission.



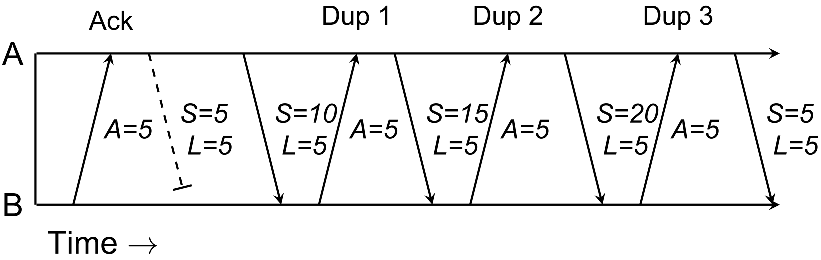
Retransmission

Three duplicate ACKs normally trigger retransmission.



Retransmission

Three duplicate ACKs normally trigger retransmission.



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 Internet is a network of networks
- IP will run over many networks because it makes few assumptions
- IP provides very limited service
- Transport protocols ride on top of IP
- UDP is connectionless datagrams
- TCP is connected byte streams
- The end-to-end argument provides guidance on where to implement functionality.
- TCP provides services that IP does not.
- TCP loss recovery is effected by a distributed state machine.

Next Time …

- Some thoughts on Go

References I

Required Readings

[6] 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] David D. Clark. “The Design Philosophy of the DARPA Internet Protocols”. In: *Computer Communication Review* 18.4 (Aug. 1988), pp. 106–114. URL: `http://ccr.sigcomm.org/archive/1995/jan95/ccr-9501-clark.pdf`.

[3] Information Sciences Institute. *The Internet Protocol*. Ed. by Jon Postel. Sept. 1981. URL: `https://tools.ietf.org/rfc/rfc791.txt`.

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

[5] Jon Postel. *User Datagram Protocol*. Aug. 1980. URL: `https://tools.ietf.org/rfc/rfc768.txt`.

To retrieve a copy of this material, or related materials, see `https://www.cse.buffalo.edu/~eblanton/`.