CSE 486/586: Distributed Systems

The Internet in Two Lectures (Part 2)

Ethan Blanton

Department of Computer Science and Engineering
University at Buffalo

Last Time …

- A protocol is an agreement on how to communicate.
  - Syntax & Semantics
- The Internet is, primarily, a network of networks.
- Its durability and longevity are due to good layering.
  - Routing and addressing are handled in the network.

Administrivia

Remember:

- AI Quiz is due today!
- Programming Assignment 1 is due on Monday!

Please join Piazza and check it often.

The Other Layers

We talked about the network layer.
What are the other layers?

Just as importantly, where are they handled?

Remember:

- the network: routers, gateways, *etc.*
- endpoints: communicating entities

And why?

Above the Network Layer

The network layer handles:

- Addressing
- Routing
- Fragmentation

Great; we can name hosts and move packets over dissimilar underlying networks!

But what about:

- Differentiating applications
- Reliability
- Flow control (we won't talk much about this; see [1])

Reliability and Survivability

We said on Wednesday that IP is best effort.

Packets are forwarded when:

- There is a valid path from here to there
- Bandwidth is available
- Nothing gets corrupted/damaged

They might be duplicated, lost, or delivered out of order.

This isn't reliable, but it may be survivable.

- Reliability: Ensures either delivery, or notice to the sender of failed delivery.
  - (It's more complicated than that.)
- Survivability: Ensures continued function of the network in the face of localized failures.

Network Layer Reliability

Why doesn't IP provide reliability?

Network Layer Reliability

Why doesn't IP provide reliability?

- Not all services need it
- Simplify the network layer

Note that some protocols have provided this at the network layer.

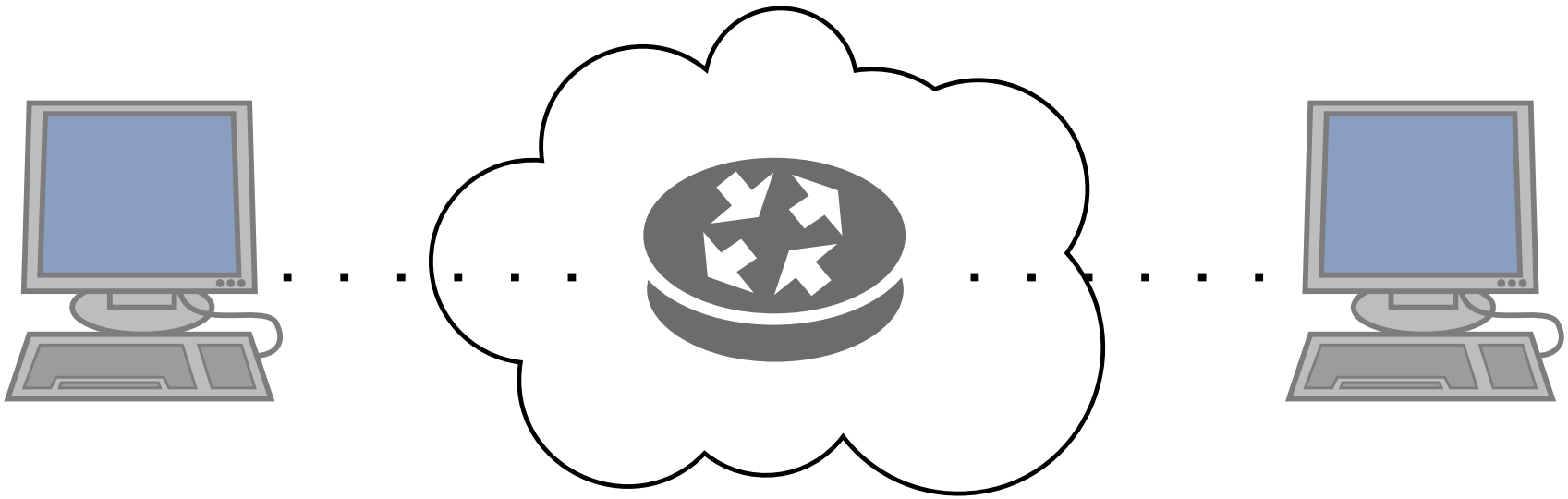
- X.25
- ATM
- *etc.*

Locating Reliability

Where *should* reliability be handled?

We said above the network layer … but by whom?

Thought Experiment: reliability in the network:

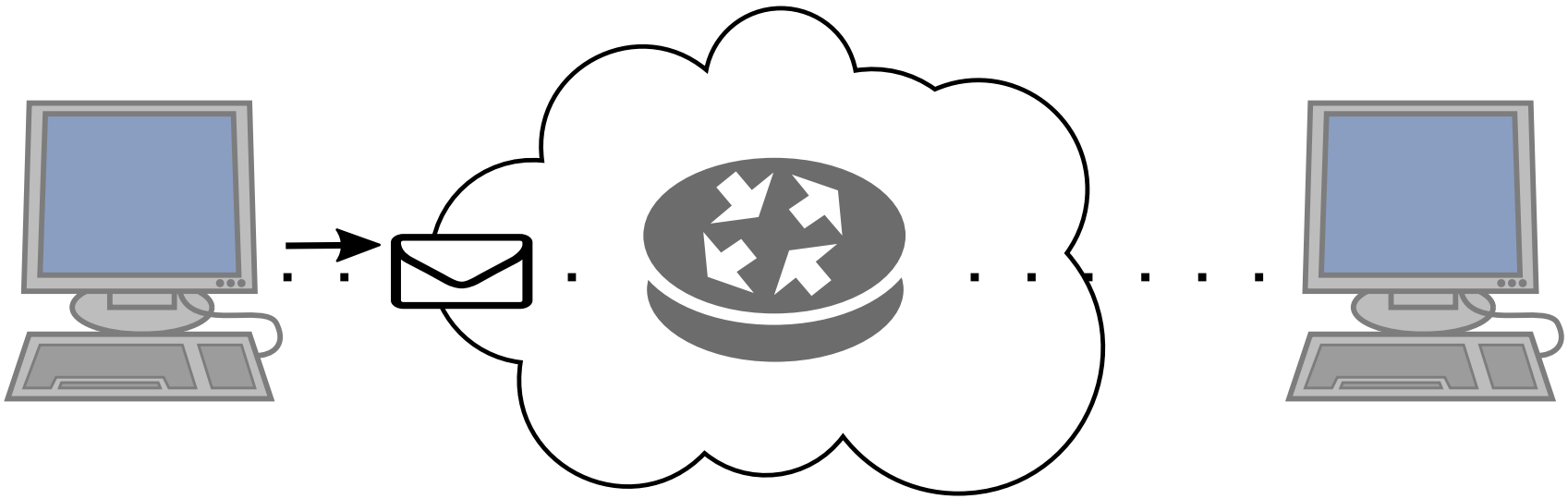


Locating Reliability

Where *should* reliability be handled?

We said above the network layer … but by whom?

Thought Experiment: reliability in the network:

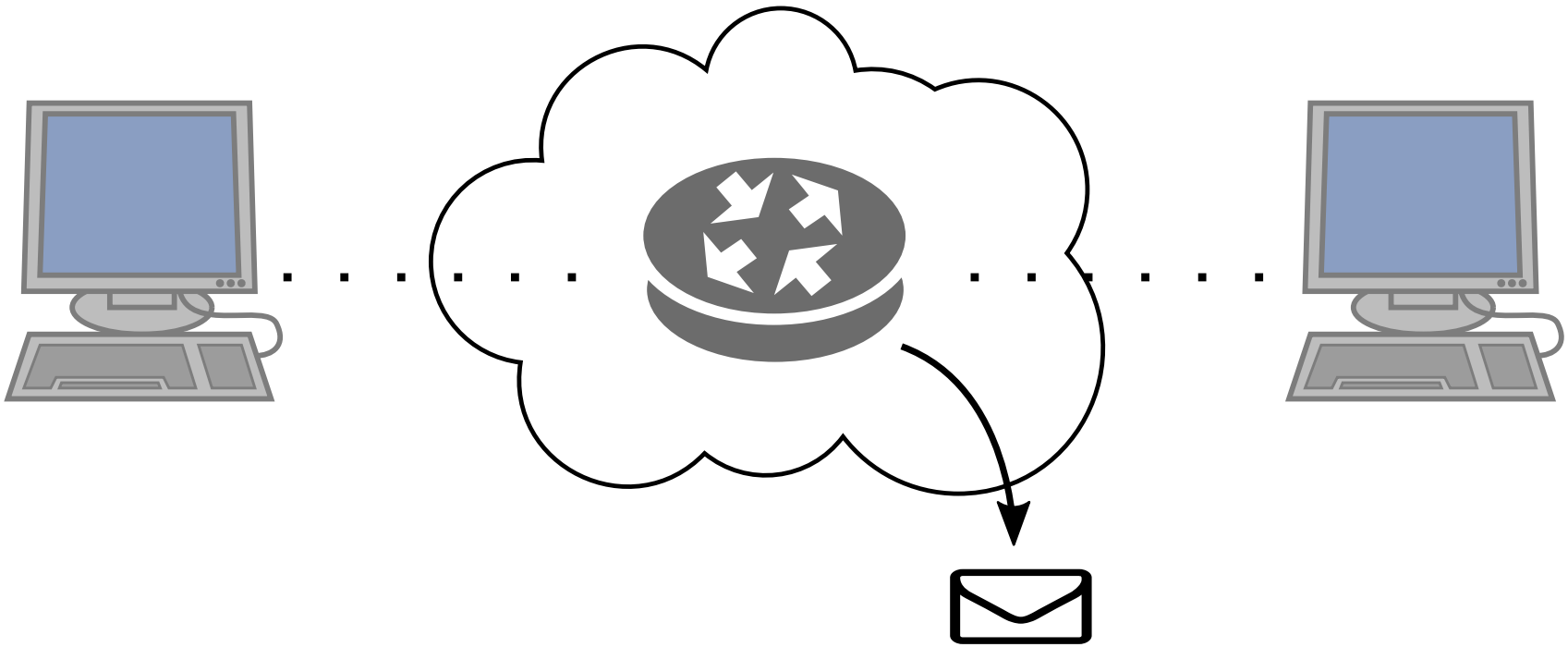


Locating Reliability

Where *should* reliability be handled?

We said above the network layer … but by whom?

Thought Experiment: reliability in the network:

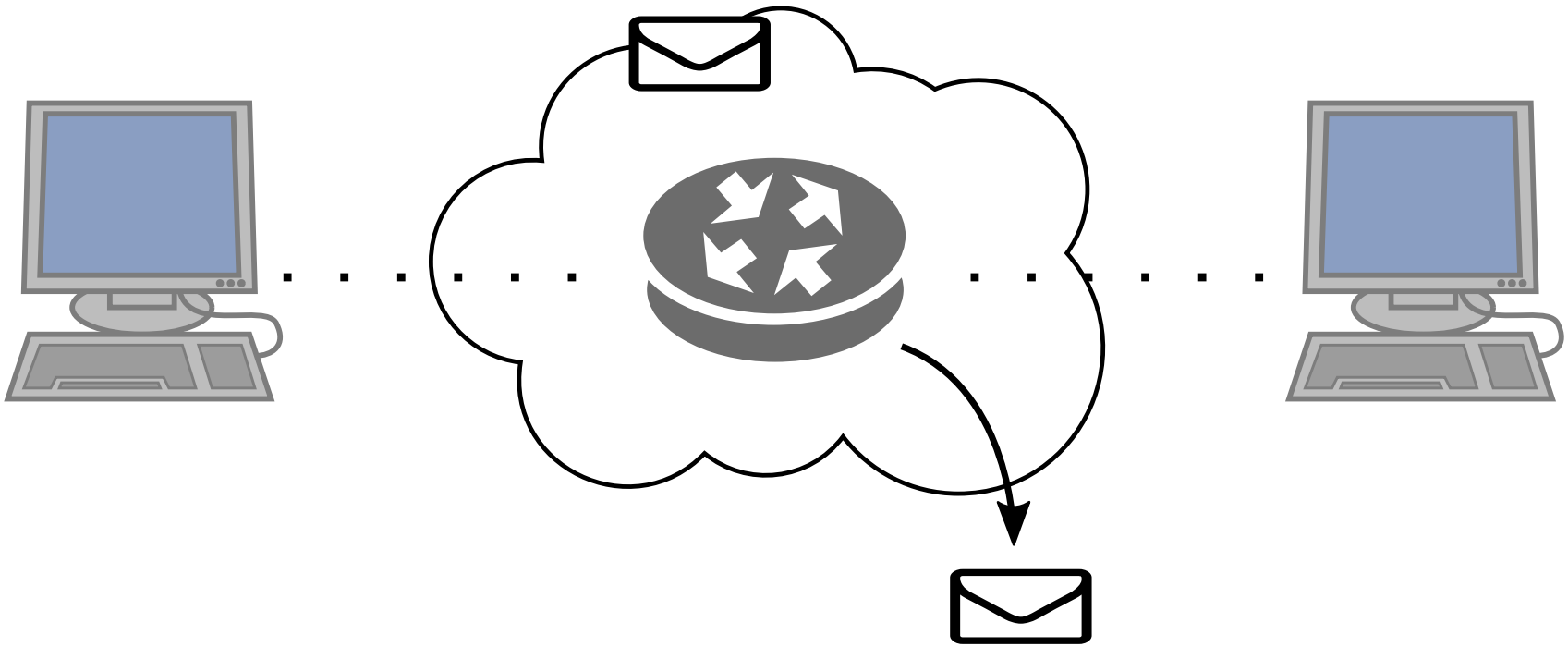


Locating Reliability

Where *should* reliability be handled?

We said above the network layer … but by whom?

Thought Experiment: reliability in the network:

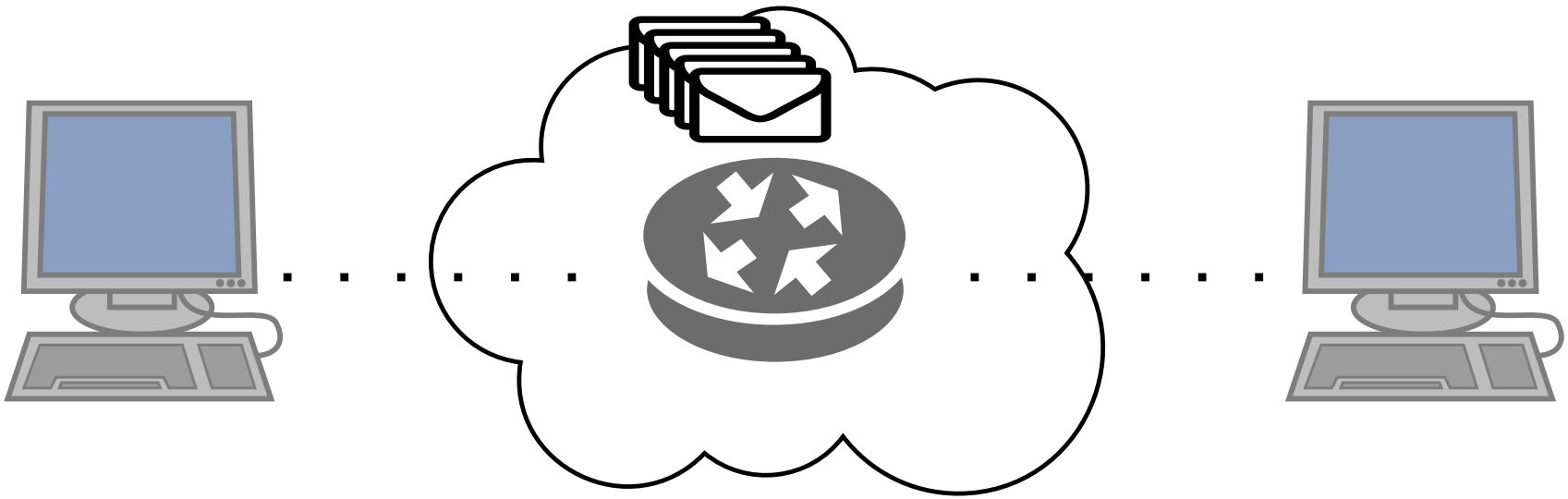


Locating Reliability

Where *should* reliability be handled?

We said above the network layer … but by whom?

Thought Experiment: reliability in the network:



The End-to-End Principle

The end-to-end principle [5][1] is used widely in Internet architecture.

Over-simplified:

> *If an operation requires knowledge known only at the endpoint, it should be handled entirely by the endpoint.*

Entirely is a strong claim ... why?

- Often the end system has to do it anyway
- Doing it twice may carry extra effort
- … sometimes it really is too strong

[1] *End-to-end arguments in system design* [5] is required reading

End-to-end examples

- File transfer
  - Network consistency checks
  - E2E consistency checks
- Packet validation
  - *When the CRC and TCP checksum disagree* [6]
- Privacy and authentication
  - WPA *vs.* VPN or IPsec *vs.* SSL

Yet:

- Wireless FEC
- Ethernet and Wireless retries

Fate Sharing

Handling services in the endpoints provides fate sharing:

State relating to an endpoint's communication is lost *only if the endpoint itself is lost*.

The advantages of this are:

- "Stateless" network
- Protection against intermediate network failures
- Reduced implementation complexity
  - In the network
  - *In the endpoints*

Transport Protocols

Back to protocols …

Reliability and application differentiation are left to the transport layer.

You'll recognize: TCP, UDP

UDP provides basically just application differentiation.

TCP provides application differentiation, plus reliability and flow control.

TCP

TCP is the Transmission Control Protocol [3].

It builds on top of IP datagrams, and provides:

- Application differentiation (port numbers)
- A data stream, rather than discrete packets
  - An individual TCP "packet" is called a segment
  - Segments are invisible to the application
- Reliable transmission of data
  - Retries
  - Acknowledgments
- Flow control

It handles reliability in the endpoints.

TCP Reliability

How does TCP provide reliability?

Segments are acknowledged, unacknowledged segments are retransmitted.

Unacknowledged segments are detected by:

- A timer
- Evidence of *later segments* arriving

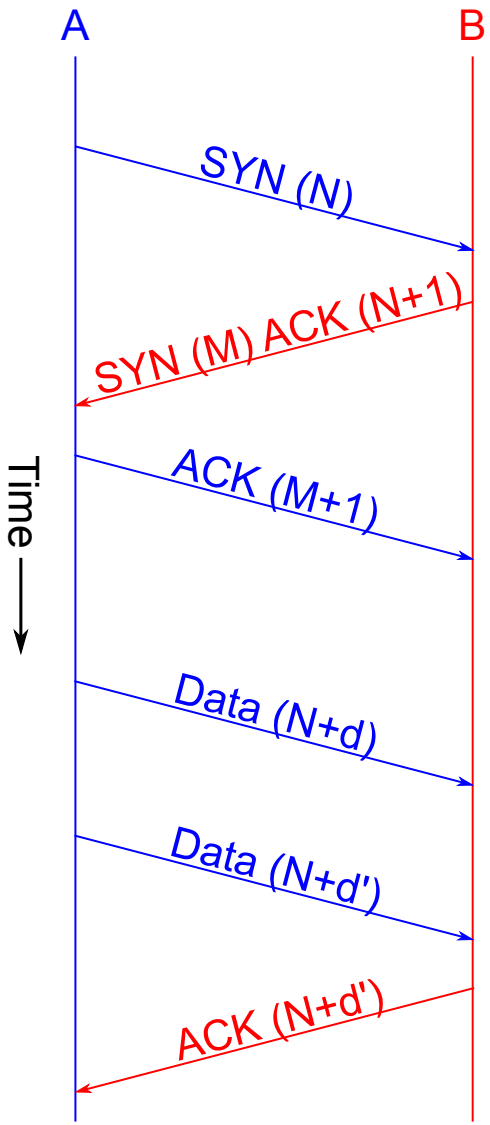
Duplicate and reordered segments are corrected by:

- Sequence numbers

Packet corruption is detected by:

- A checksum (inverted 16-bit one's complement)

TCP Connection Setup

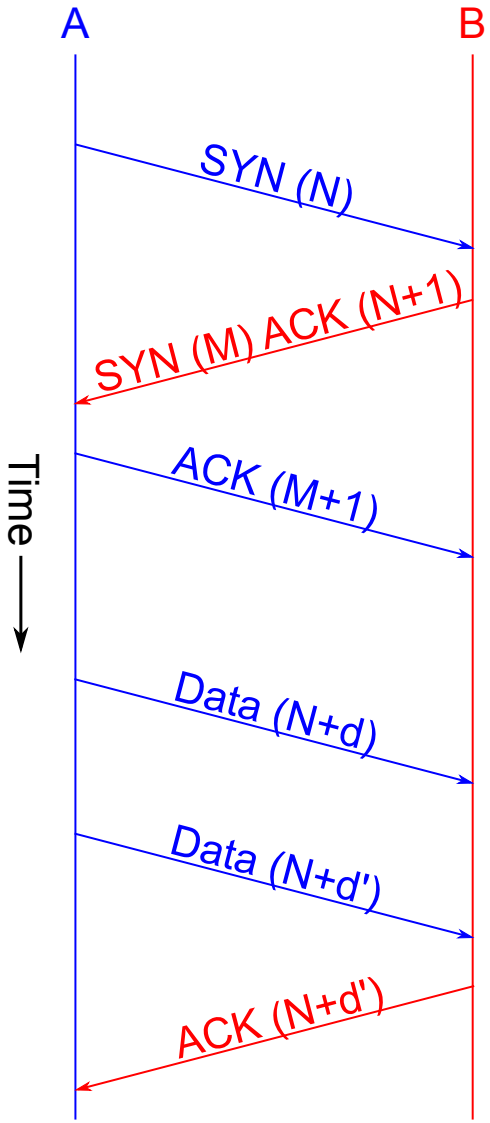


Begins with a three-way handshake:

- Host A sends SYN
- Host B sends SYN+ACK
- Host A sends ACK
- Established!

Why 3-way?

TCP Connection Setup



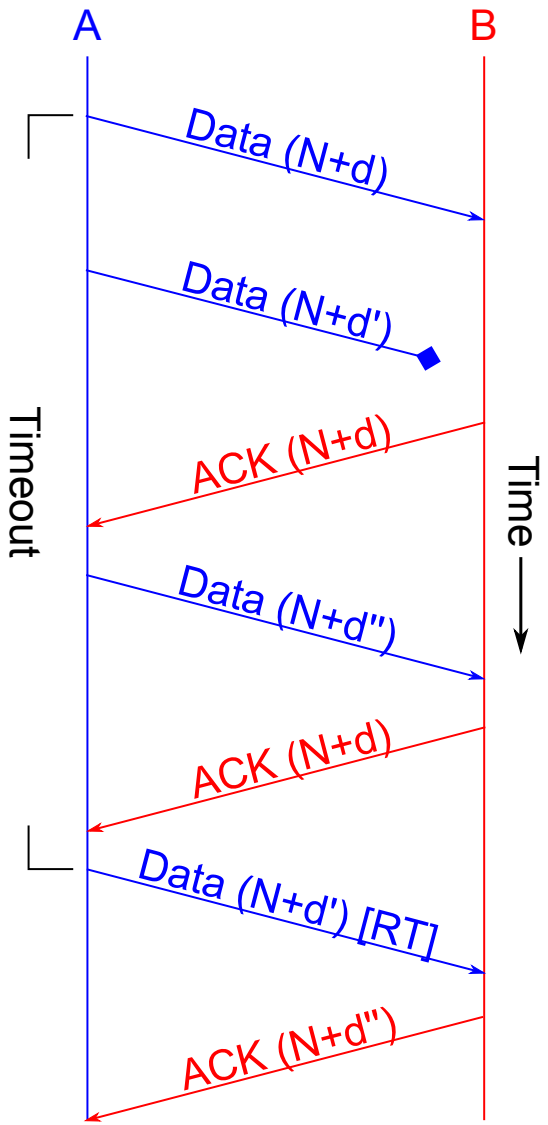
Begins with a three-way handshake:

- Host A sends SYN
- Host B sends SYN+ACK
- Host A sends ACK
- Established!

Why 3-way?

Bi-directional connection!

TCP Loss Recovery



Retransmissions:

- When data is sent a, a timer is started
- If it expires without ACK, RT
- There is also fast retransmit [1]

A form of flow control called congestion control [1, 2] is tightly coupled with loss recovery.

TCP Use Cases

When should you use TCP?

TCP Use Cases

When should you use TCP?

- You want reliable, in-order delivery of a byte stream
- Moderate overhead (~30+% by time!) is acceptable
- You want to be “network friendly”
- You can stand variable throughput

TCP Use Cases

When should you use TCP?

- You want reliable, in-order delivery of a byte stream
- Moderate overhead (~30+% by time!) is acceptable
- You want to be “network friendly”
- You can stand variable throughput

When might you not want this?

TCP Use Cases

When should you use TCP?

- You want reliable, in-order delivery of a byte stream
- Moderate overhead (~30+% by time!) is acceptable
- You want to be “network friendly”
- You can stand variable throughput

When might you not want this?

- Loss is tolerable
- Constant bitrate stream
- Data is truly discrete packets
- …

UDP

User Datagram Protocol [4], provides little more than IP:

- Application differentiation (port numbers)
- (Optional) data checksum (same as TCP)

Notably:

- No flow control
- No byte stream
- No reliability

Benefits of UDP

- No connection establishment (~2 RTT for TCP!)
- No throughput variation from flow / congestion control
- Lesser state requirements
- Little packet data overhead

Used for:

- Streaming multimedia
- Voice over IP
- DNS

So … Why Reinvent the Wheel?

If TCP provides reliability, why do we need to care?

What exactly is TCP guaranteeing?

So … Why Reinvent the Wheel?

If TCP provides reliability, why do we need to care?

What exactly is TCP guaranteeing?

- The data arrived at the remote host (not application!)
- It probably wasn't very corrupted
- The host wasn't too slow
- …

So … Why Reinvent the Wheel?

If TCP provides reliability, why do we need to care?

What exactly is TCP guaranteeing?

- The data arrived at the remote host (not application!)
- It probably wasn't very corrupted
- The host wasn't too slow
- …

Sometimes, we may want greater or lesser guarantees!

These things are only pairwise … and only when data is flowing.

Summary

The end-to-end principle is powerful.

The transport layer provides additional functionality:

- TCP: App. differentiation, flow control, reliability
- UDP: Application differentiation

TCP guarantees are end-to-end at the host level.

Next time: Android architecture and API

References I

Required Readings

[5] 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: http://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf.

Optional Readings

[1] Mark Allman, Vern Paxson, and Ethan Blanton. *TCP Congestion Control*. RFC 5681. Sept. 2009. 18 pp. URL: https://www.rfc-editor.org/rfc/rfc5681.txt.

[2] Van Jacobson and Michael J. Karels. “Congestion Avoidance and Control”. In: *Proceedings of ACM SIGCOMM*. Association for Computing Machinery, Aug. 1988. URL: http://ee.lbl.gov/papers/congavoid.pdf.

[3] Jon Postel. *Transmission Control Protocol*. RFC 793. Sept. 1981. 85 pp. URL: https://www.rfc-editor.org/rfc/rfc793.txt.

[4] Jon Postel. *User Datagram Protocol*. RFC 768. Aug. 1980. 3 pp. URL: https://www.rfc-editor.org/rfc/rfc768.txt.

References II

[6] Jonathan Stone and Craig Partridge. “When the CRC and TCP checksum disagree”. In: *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. Association for Computing Machinery, Aug. 2000, pp. 309–319.

Acknowledgements

These slides are based on slides from Steve Ko, used with permission.

Those slides contain the following attribution:

- These slides contain material developed and copyrighted by
  - Indranil Gupta ant UIUC
  - Mike Freedman and Jen Rexford at Princeton