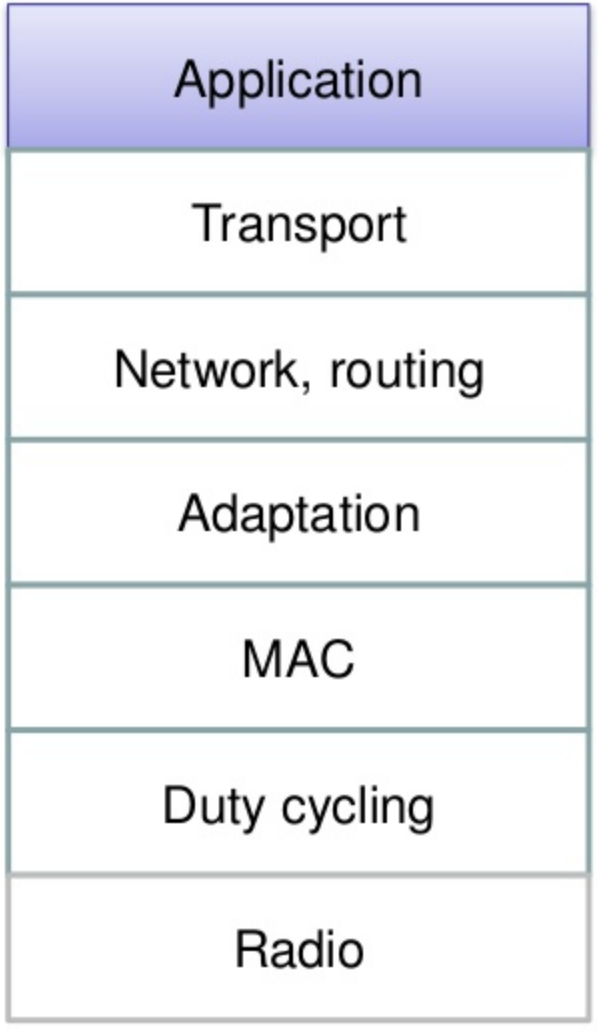
Diving into the IoT/IP stack

The IoT/IP stack

HTTP, WebSockets, CoAP	Application
TCP, UDP	Transport
IPv6, IPv4, RPL	Network, routing
6lowpan	Adaptation
CSMA/CA	MAC
ContikiMAC, CSL	Duty cycling
802.15.4	Radio

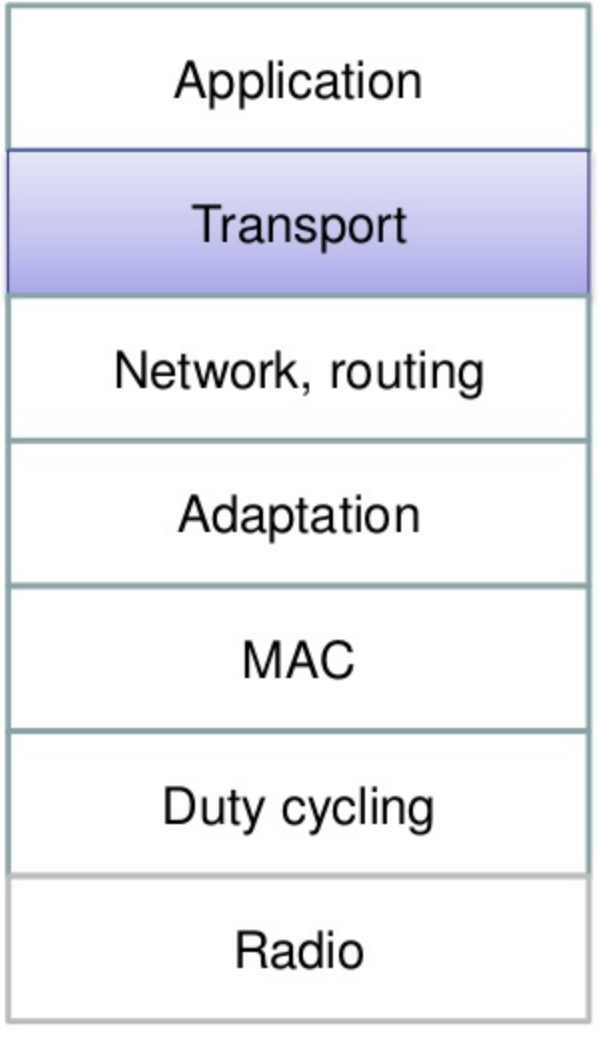
The application layer

- Do something useful with the data
- Today, applications typically run *above* the application layer
  - E.g. the Web runs on top of HTTP



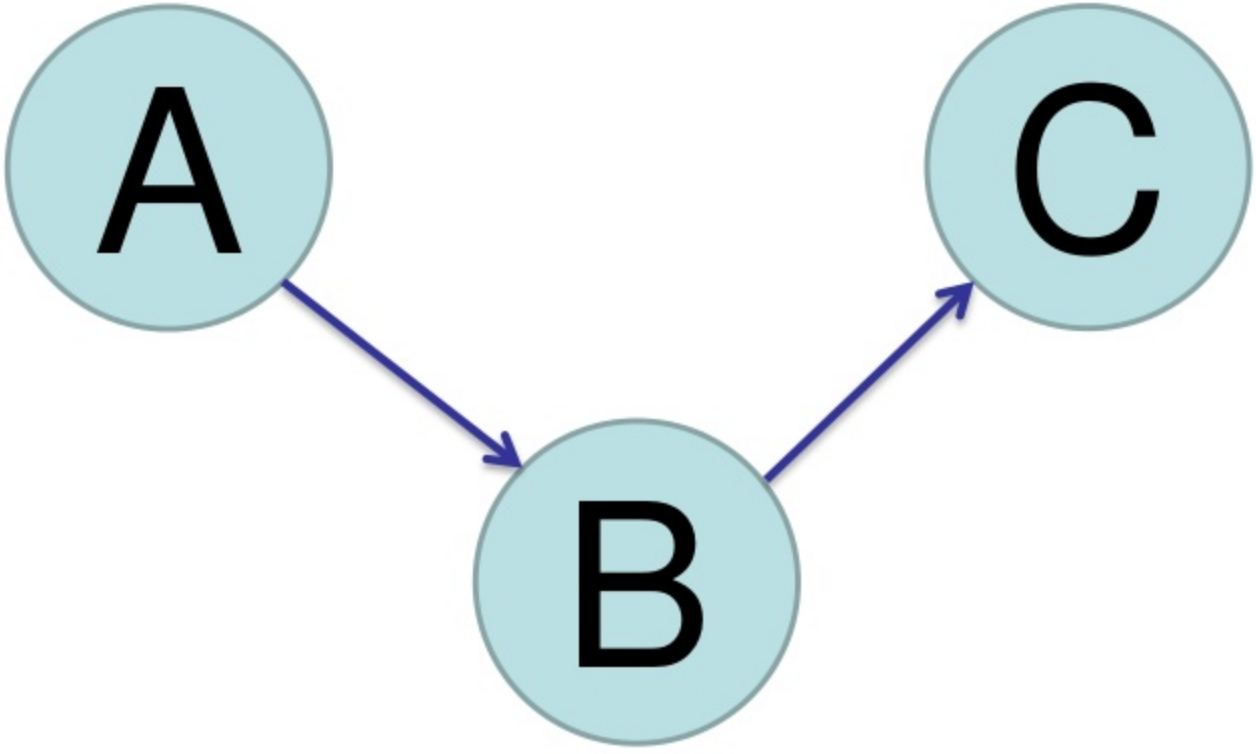
The transport layer

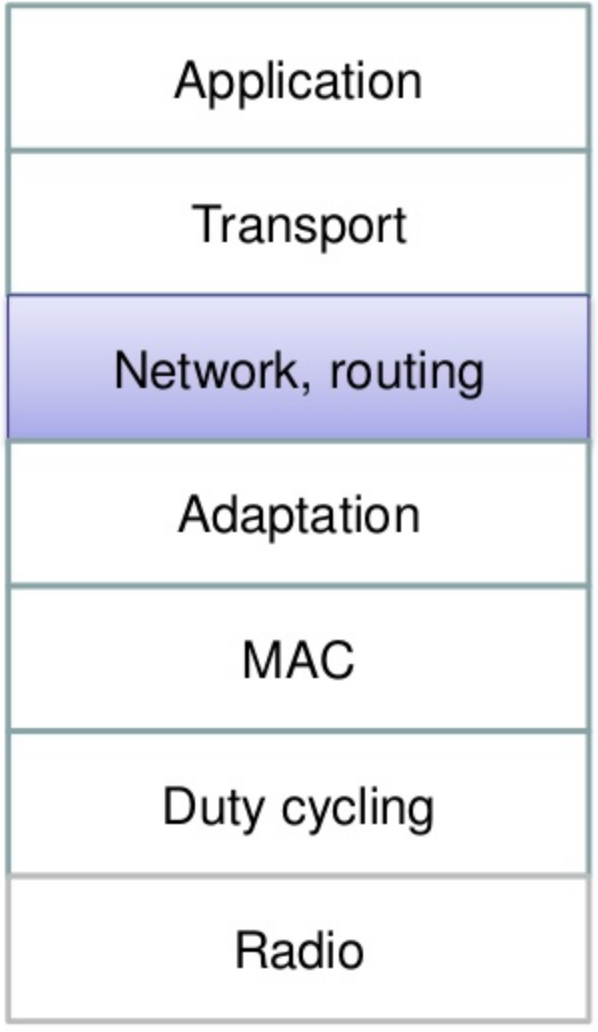
- Allow program A to send data to program B
  - Even though the network may drop packets
  - Get packets in order, despite them being reordered on the way



The network layer

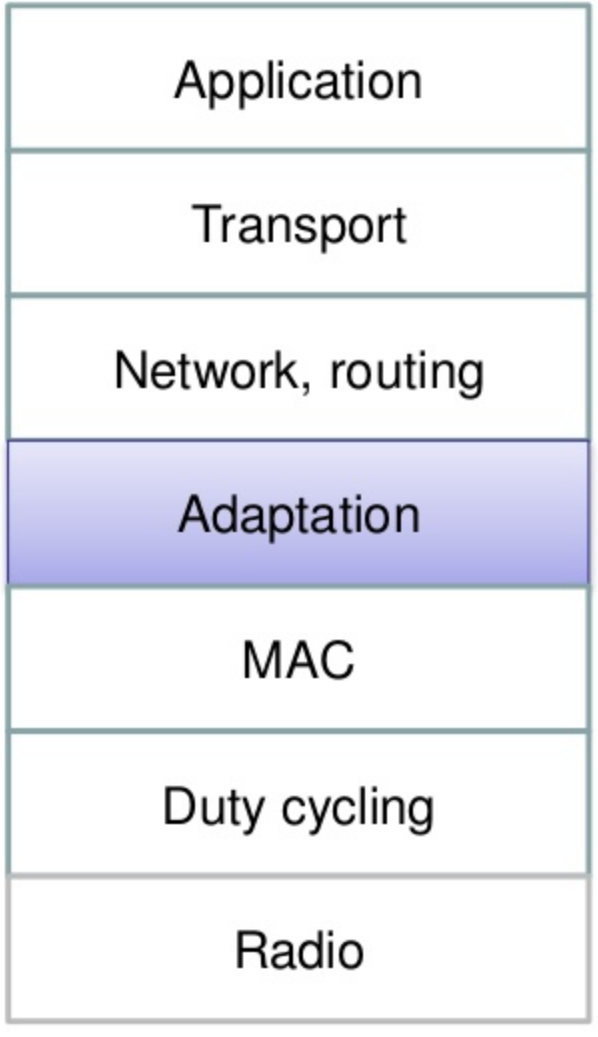
- Figure out how to send a packet from A to C, through node B





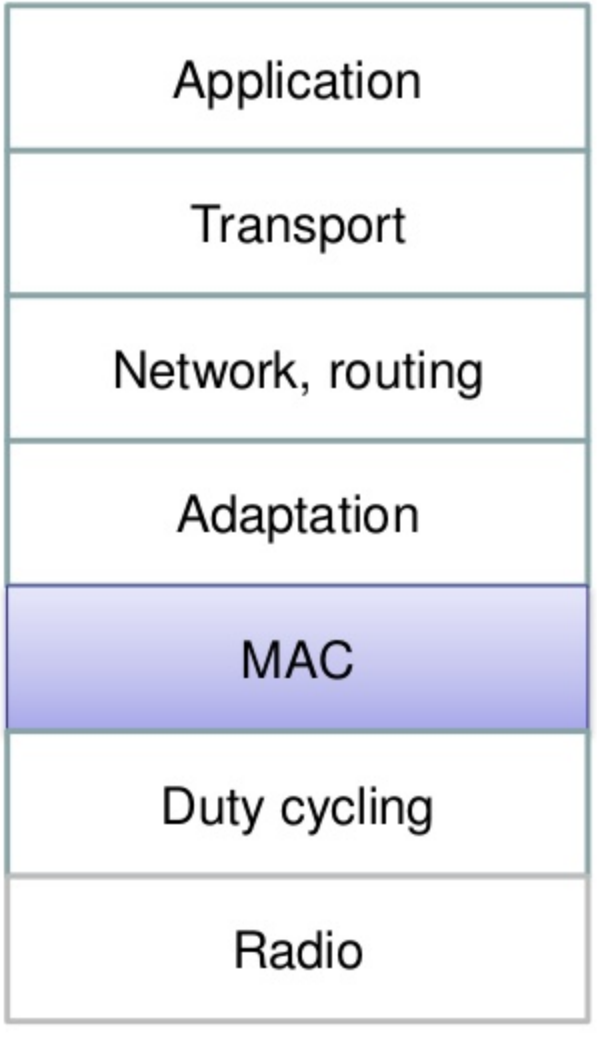
The adaptation layer

- Send network layer packets over the radio
- Compress headers, fragment packets



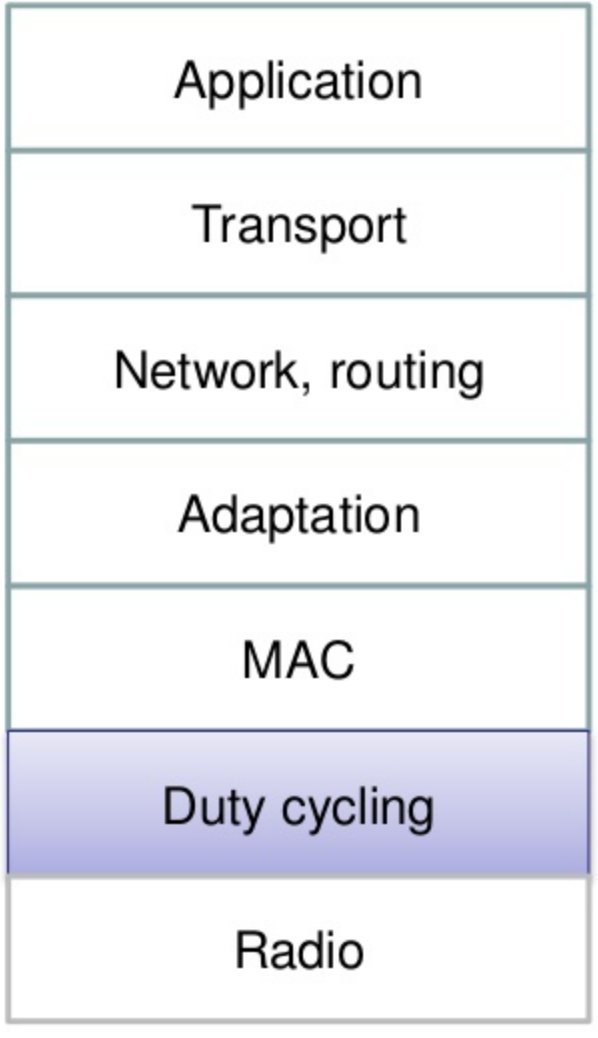
The MAC layer

- Make sure to not send when others are sending
- Back off exponentially when there is too much traffic



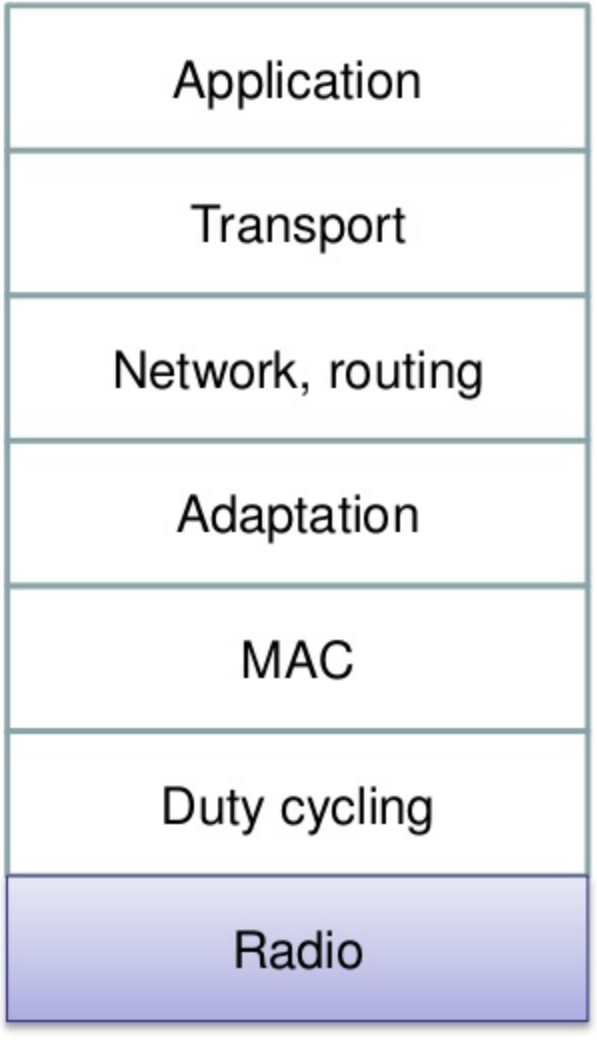
The radio duty cycling layer

- Turn off the radio when not needed, to save power
  - The radio consumes more power than other components
- Make sure nodes are awake to hear messages from others

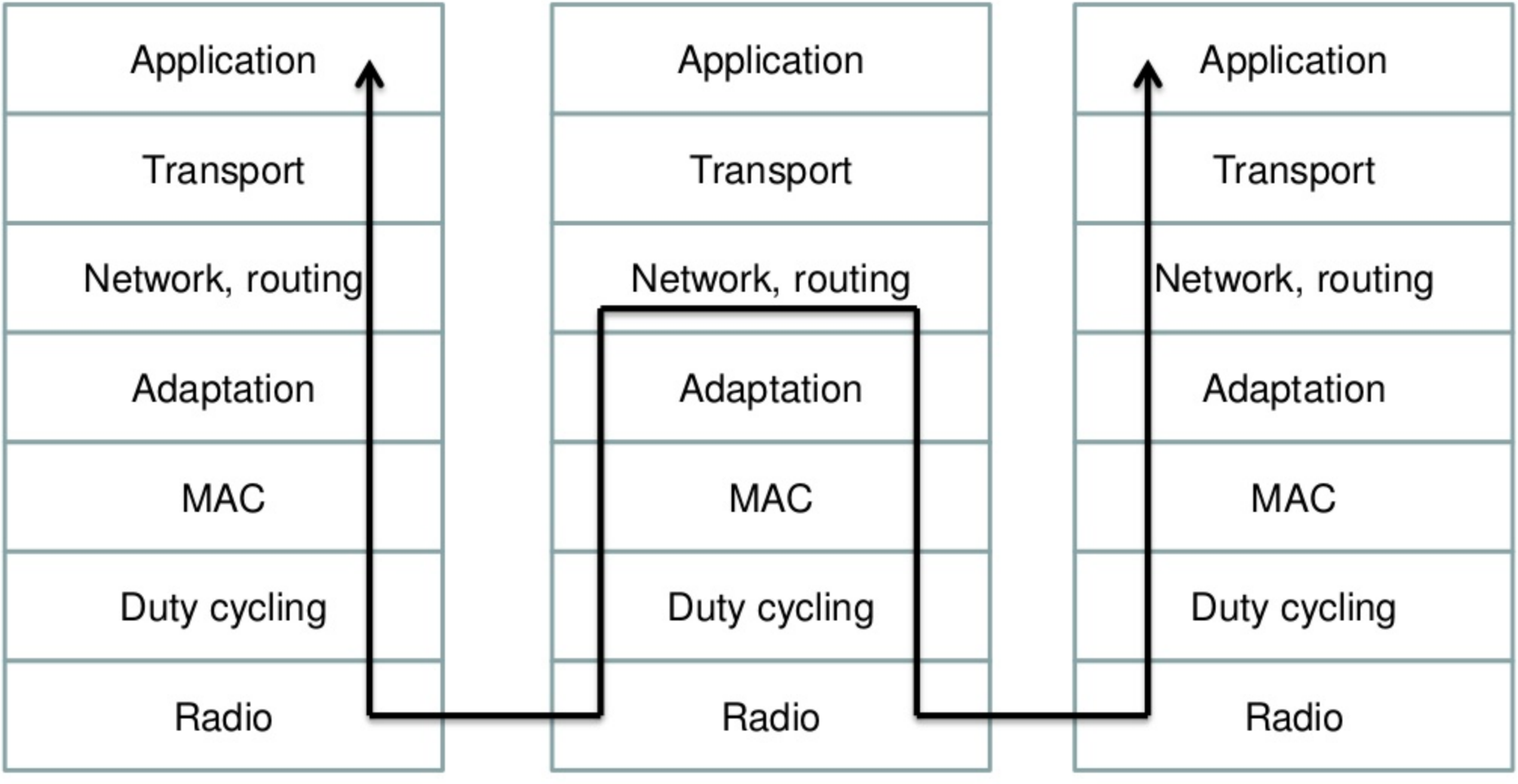


The radio link layer

- Frequency bands
  - 2.4 GHz and sub-GHz
- Modulations and encodings
- Bit rates and bit errors
- Standards and non-standards



The packet path



The IoT/IP Protocols

The application layer

The application layer

- The layer where applications supposedly run
- But many applications actually run on top of an application layer protocol
- Application layer protocols for the IoT
  - HTTP (TCP)
  - WebSockets (TCP)
  - CoAP (UDP)

HTTP

- HyperText Transfer Protocol
- The same HTTP which is used on the web
- HTTP is often used for automated mechanisms
- Useful for mesh-to-Internet communication

REST

- Representational State Transfer
- A conceptualization of the way HTTP works
  - Resource: an URL
  - Resources are manipulated with methods
    - GET http://device-25.local/sensor/light
    - POST http://device-28.local/configuration sensor=123&batt=3300
    - DELETE, UPDATE, HEAD, ...
- There is a lot of confusion around what REST is and is not

WebSockets

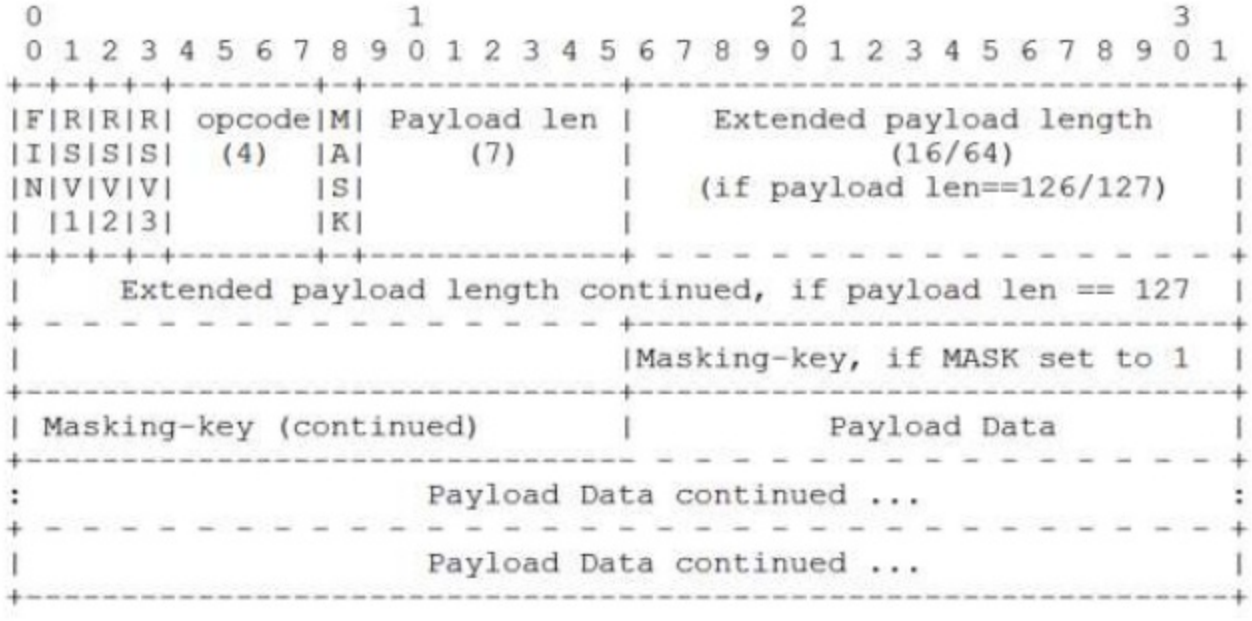
- A recently defined standard
  - RFC6455
- Provides a TCP-like service on top of HTTP with very little overhead
- Goes through firewalls, proxies (TCP port 80)
- Compatible with HTTP
  - A WebSocket connection starts with a handshake over HTTP
- Useful for mesh-to-Internet communication

Websocket exchange

```
GET /path HTTP/1.1
Host: api.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key:
x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: example
Sec-WebSocket-Version: 12
Origin: http://example.com
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept:
HSmrc0sM1YUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: example
```

Websocket message header

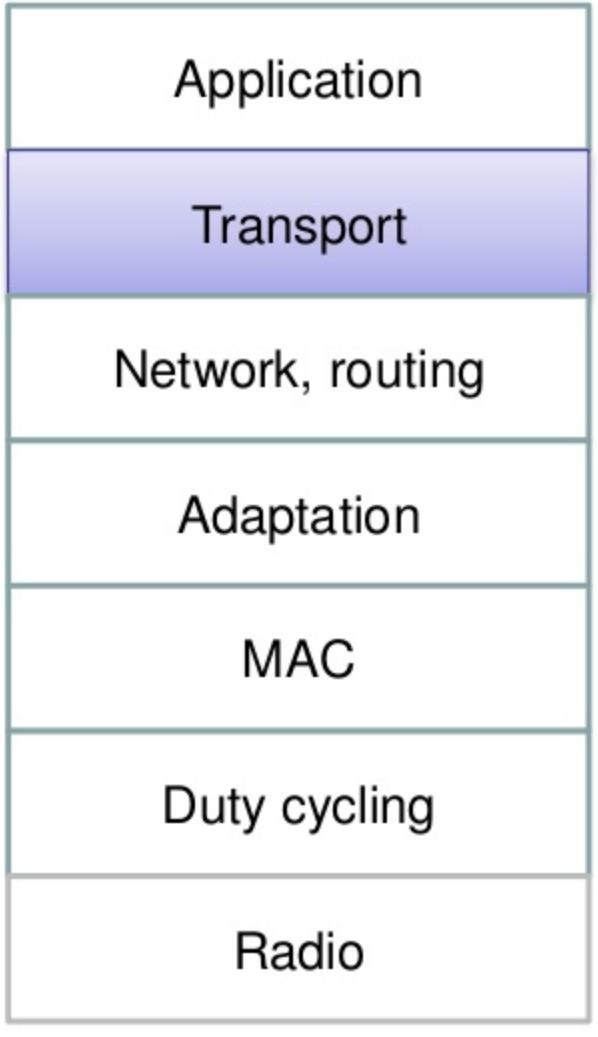


CoAP

- Message protocol over UDP
- Started as a bit-optimized REST mechanism
- Has grown into a fully fledged transport/application solution
  - Retransmissions
  - Confirmable/non-confirmable transmission modes
  - Support for caching
  - Support for sleeping nodes
  - Subscription support
  - Resource discovery
- Useful for in-mesh communication
  - May have problems with firewalls

The IoT/IP Protocols

The transport layer



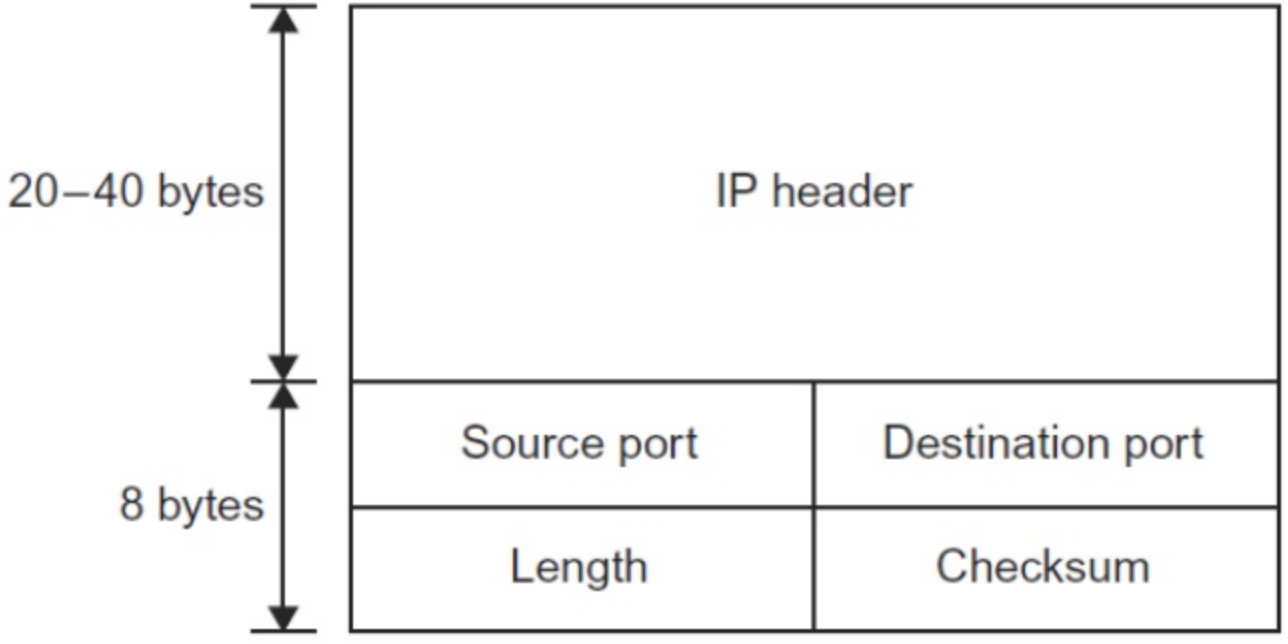
The problem

- Getting data from application A to application B
- UDP
  - Best-effort delivery, no ordering
- TCP
  - Reliable stream delivery: in order, reliably
    - We'll know whether it was received or not

UDP

- User Datagram Protocol
- Provides 16-bit port numbers
- Payload checksum
- Length field
- No protocol logic

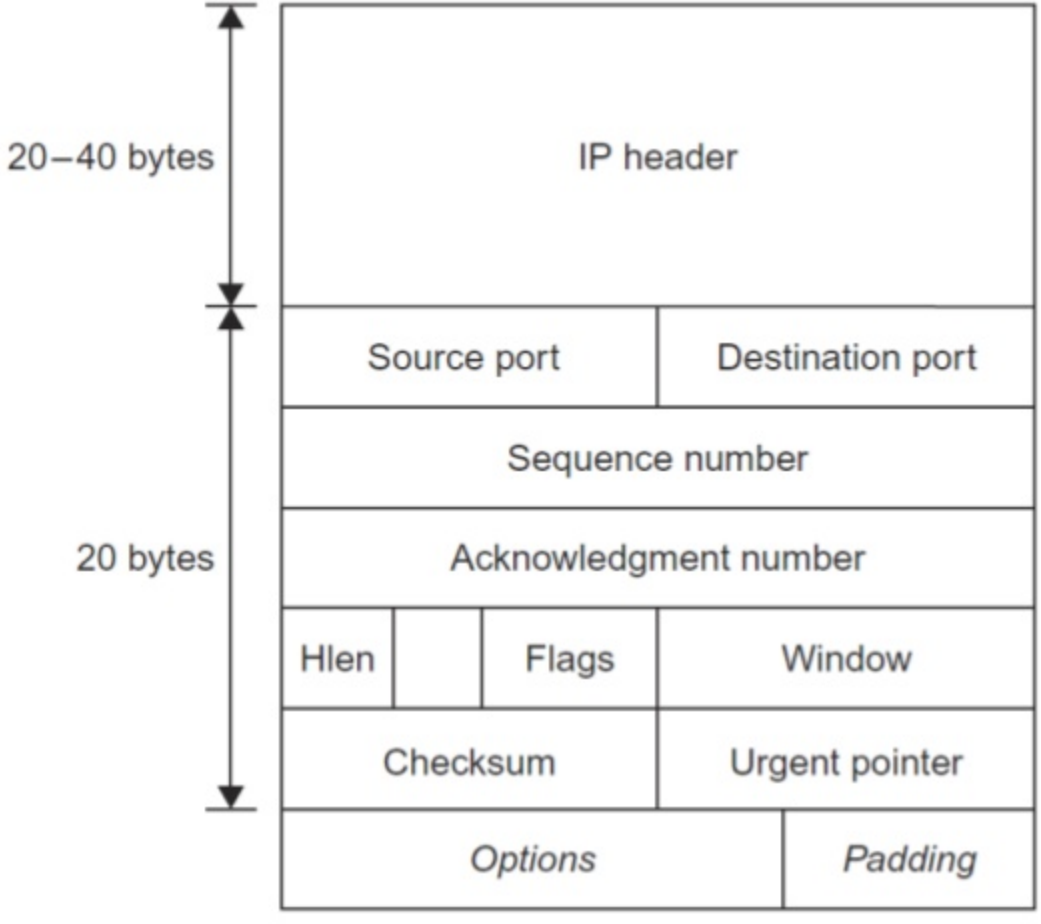
The UDP Header



TCP

- Transport Control Protocol
- Significantly more complex than UDP
- Reliable, in-order stream delivery
  - Byte sequence
  - End-to-end acknowledgements
  - Flow control
    - Avoid overloading the receiver
  - Congestion control
    - Avoid overloading the network

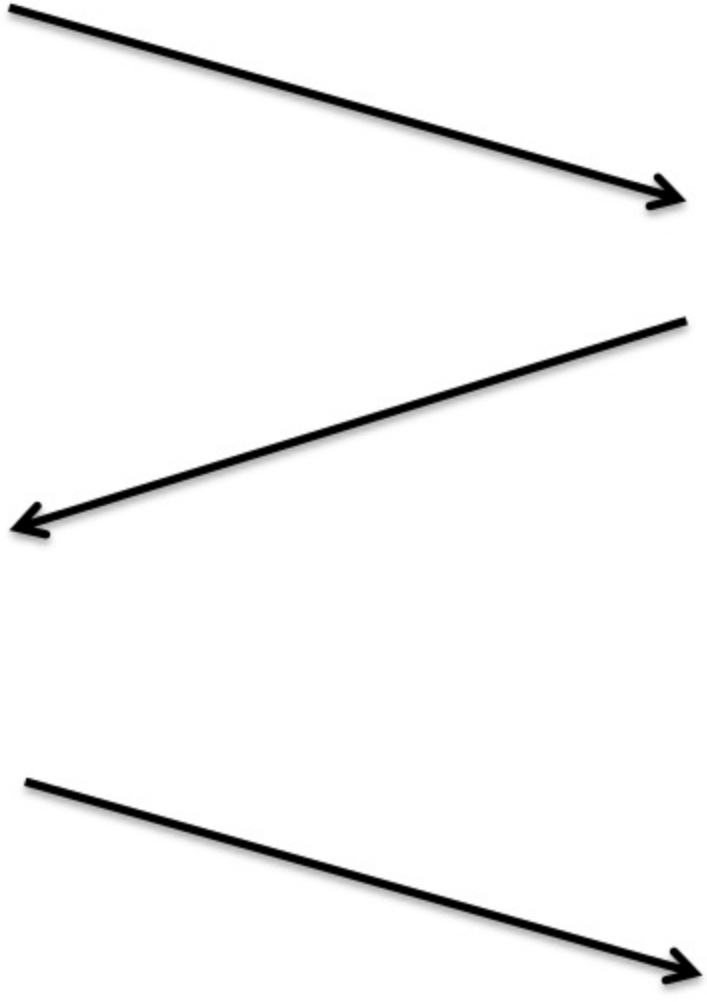
The TCP header



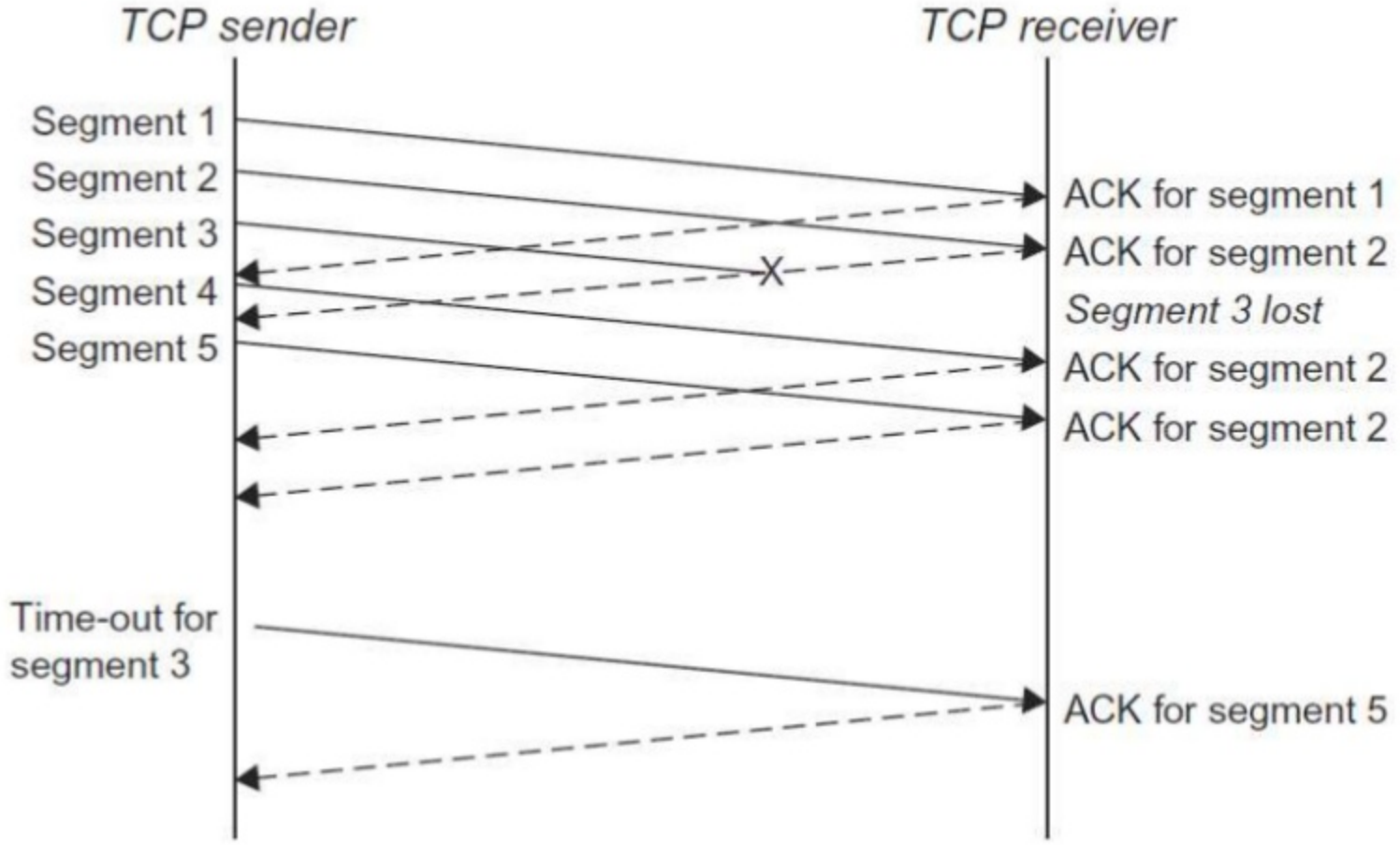
The TCP byte stream

TCP 3-way handshake, opening

- SYN
  - Includes maximum segment size option
- SYN + ACK
  - Also includes maximum segment size option
- ACK

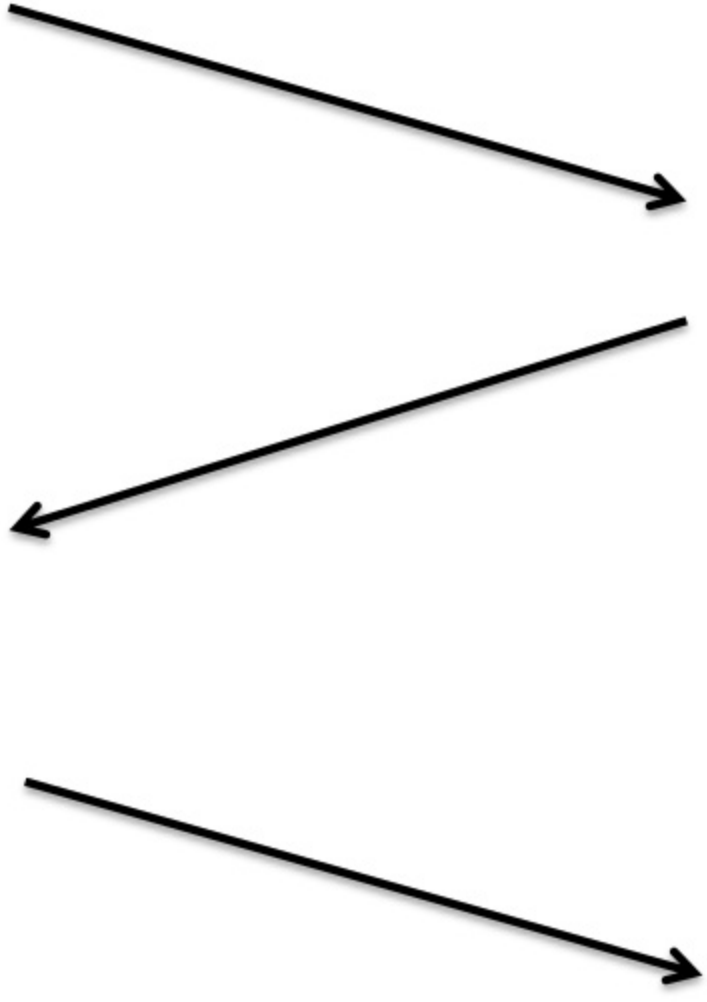


TCP retransmissions



TCP 3-way handshake, closing

- FIN
- FIN+ ACK
- ACK



TCP problems for wireless

- TCP reacts strongly to packet loss
  - Reduces sending rate significantly
- TCP does not like reordering
  - May reduce sending rate
- But TCP can be made efficient for low-power wireless
  - E.g. Duquennoy, Österlind, Dunkels, SenSys 2011

TCP for IoT applications

- May be overkill for some applications
  - Not all applications need a reliable data stream
- Does not support multicast/broadcast operation
  - TCP is strictly point-to-point
- But TCP works well with the Internet

In Contiki

- Full support for UDP and TCP
- TCP implementation is simple
  - Uses stop-and-wait, resulting in low throughput
  - TCP-segment splitting can improve throughput
- UDP and TCP callback-based sockets (Contiki 3.x)
  - Register a socket with a callback function that gets called when something happens

Takeaway transport layer

- UDP and TCP
  - UDP: simple, datagram delivery
  - TCP: reliable byte stream

More



http://thingsquare.com