

# Distributed Systems

## 2a. Networking – Part 2

Paul Krzyzanowski  
pxk@cs.rutgers.edu

# Networks

---

- Packet versus Circuit switching
- Baseband versus Broadband
  - Ethernet CSMA/CD
- Layers of protocols
  - Data Link (2) versus Network (3) versus Transport (4)
- Communication protocols
  - Connection-oriented (virtual circuit)
  - Connectionless (datagram)

# IP Networking

---

- IP addressing
  - Network-Host partitioning
  - Class-based addressing
  - Classless Inter-Domain Routing (CIDR)
- Transport layer protocols: TCP & UDP
  - Port numbers
- Packet encapsulation
  - Ethernet – IP – TCP/UDP
- Why would you ever use UDP?

# Routing

---

- Packets take a series of **hops** to get to their destination
  - Figure out the path
- Generate/receive packet at machine
  - check destination
    - If destination = local address, deliver locally
  - else
    - Increment hop count (discard if hop # = TTL)
    - Use destination address to search routing table
    - Each entry has address and netmask. Match returns interface
    - Transmit to destination interface
- **Static routing**

# Routing

---

- **Dynamic routing**
  - Class of protocols by which machines can adjust routing tables to benefit from load changes and failures
  - E.g., RIP (routed), OSPF
- **Router**
  - Switching element that connects two or more network interfaces
  - Operates at layer 3 (network) – does not extend a LAN

# Transport-layer protocols over IP

---

- IP sends packets to machine
  - No mechanism for identifying sending or receiving application
- Transport layer uses **port number** to identify application
- TCP – Transmission Control Protocol
- UDP – User Datagram Protocol

# TCP – Transmission Control Protocol

---

- Virtual circuit service  
(connection-oriented)
- Send acknowledgement for each received packet
- Checksum to validate data
- Data may be transmitted simultaneously in both directions

# UDP – User Datagram Protocol

---

- **Datagram service** (connectionless)
- Data may be lost
- Data may arrive out of sequence
- Checksum for data but no retransmission
  - Bad packets dropped

# Programming Interfaces

# Sockets

---

- IP lets us send data between machines
- TCP & UDP are *transport layer* protocols
  - Contain **port number** to identify transport endpoint (application)
- The most popular abstraction for transport layer connectivity: **sockets**
  - Developed at UC Berkeley

# Sockets

---

- Attempt at generalized IPC model
- Goals:
  - communication between processes should not depend on whether they are on the same machine
  - efficiency
  - compatibility
  - support different protocols and naming conventions

# Socket

---

Abstract object from which messages are sent and received

- Looks like a file descriptor
- Application can select particular style of communication
  - Virtual circuit, datagram, message-based, in-order delivery
- Unrelated processes should be able to locate communication endpoints
  - Sockets can have a *name*
  - Name should be meaningful in the communications domain

# The OS Interface to Sockets

# Step 1

---

## Create a socket

```
int s = socket(domain, type, protocol)
```

**AF\_INET**

**SOCK\_STREAM**  
**SOCK\_DGRAM**

useful if some families  
have more than one  
protocol to support a  
given service

Conceptually similar to open BUT

- *open* creates a new reference to a possibly existing object
- *socket* creates a new instance of an object

# Step 2

---

Name the socket (assign address, port)

```
int error = bind(s, addr, addrlen)
```

**socket**

**Address structure**  
`struct sockaddr*`

**length of address structure**

# Step 3a (server)

---

Set socket to be able to accept connections

```
int error = listen(s, backlog)
```

socket

queue length for pending  
connections

# Step 3b (server)

---

Wait for a connection from client

```
int snew = accept(s, cIntraddr, &cIntralen)
```

socket

pointer to address structure

length of address structure

new socket  
for this session

s is only used for managing the queue of connection requests

# Step 3 (client)

---

Connect to server

```
int error = connect(s, svraddr, svraddrlen)
```

socket

address structure  
struct sockaddr\*

length of address  
structure

# Step 4

---

## Exchange data

### Connection-oriented

**read/write**

**recv/send** (*extra flags*)

### Connectionless

**sendto/recvfrom**

**sendmsg/recvmsg**

# Step 5

---

## Close connection

**shutdown** (*s*, *how*)

*how*:

0: can send but not receive

1: cannot send more data

2: cannot send or receive (=0+1)

# Programming: sockets

---

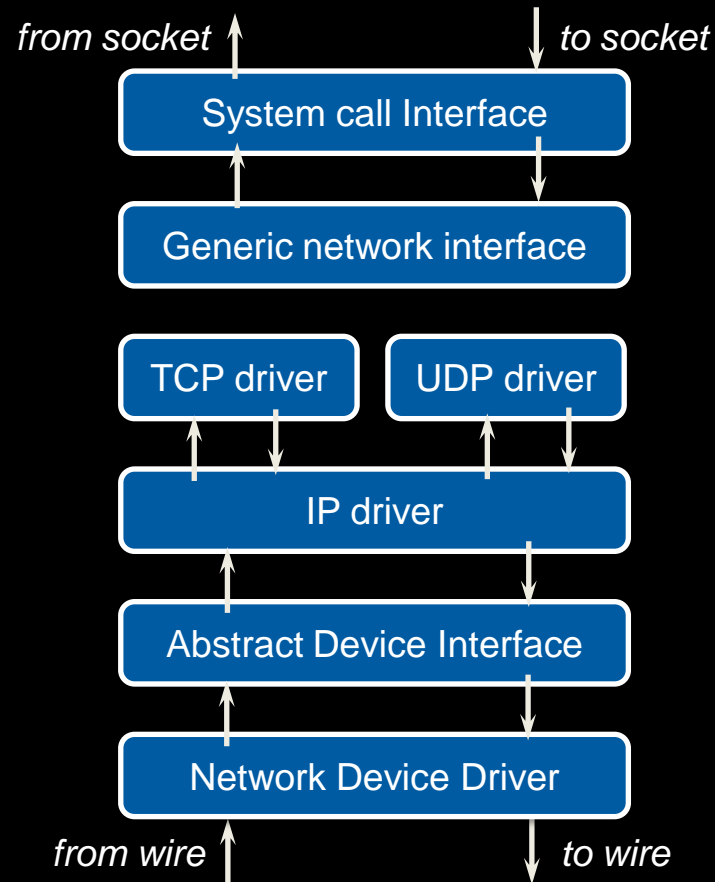
## Connection-oriented

- socket
- bind
- listen
- accept
- connect
- read/write
- shutdown/close

## Connectionless

- socket
- bind
- sendmsg/recvmmsg
- close

# Data Flow Between the Wire & App



# IP driver responsibilities

---

- Get operating parameters from device driver
  - Maximum packet size (MTU)
  - Functions to initialize HW headers
  - Length of HW header
- Routing packets
  - From one physical network to another
- Fragmenting packets
- Send operations from higher-layers
- Receiving data from device driver
- Dropping bad/expired data

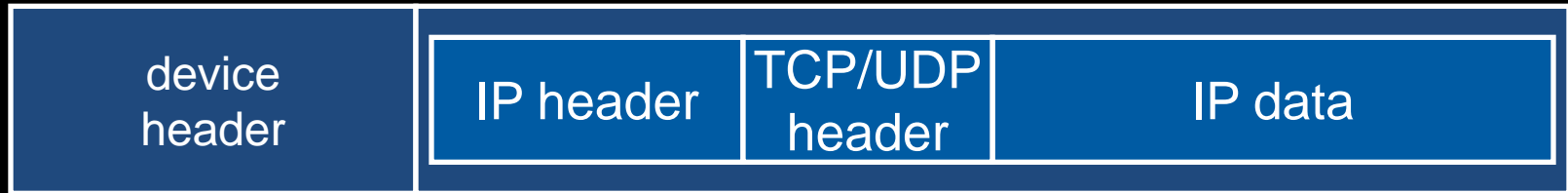
# Device driver (e.g., ethernet) responsibilities

---

- Controls network interface card
  - Comparable to character driver
- Processes interrupts from network interface
  - Receive packets
  - Send them to IP driver
- Get packets from IP driver
  - Send them to hardware
  - Ensure packet goes out without collision
    - Follow the rules of Ethernet's CSMA/CD

# Device header (Ethernet II)

---



# IP header

---

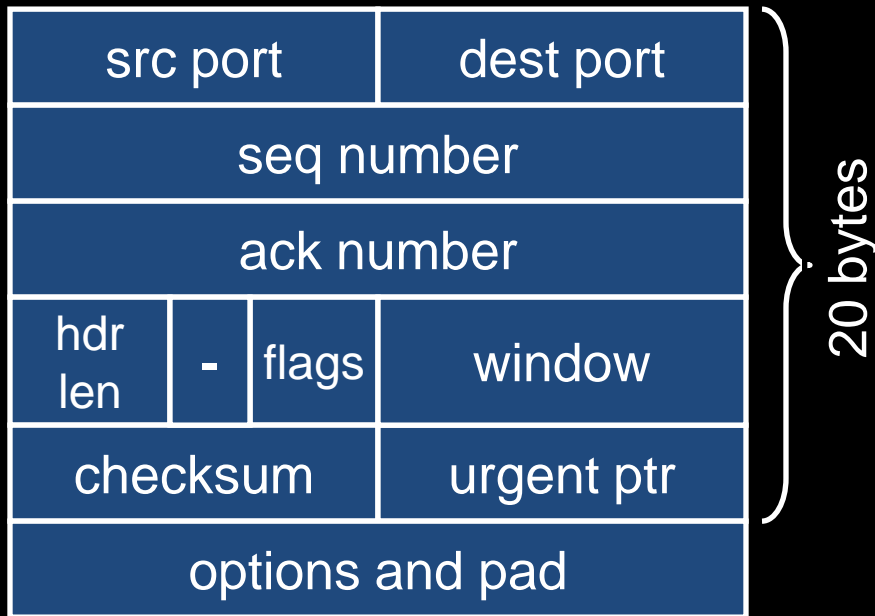


vers	hlen	svc type (TOS)	total length	
fragment identification		flags	fragment offset	
TTL	protocol		header checksum	
source IP address				
destination IP address				
options and pad				

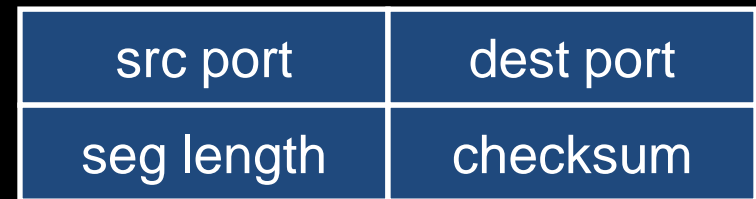
# Headers: TCP & UDP

---

## TCP header



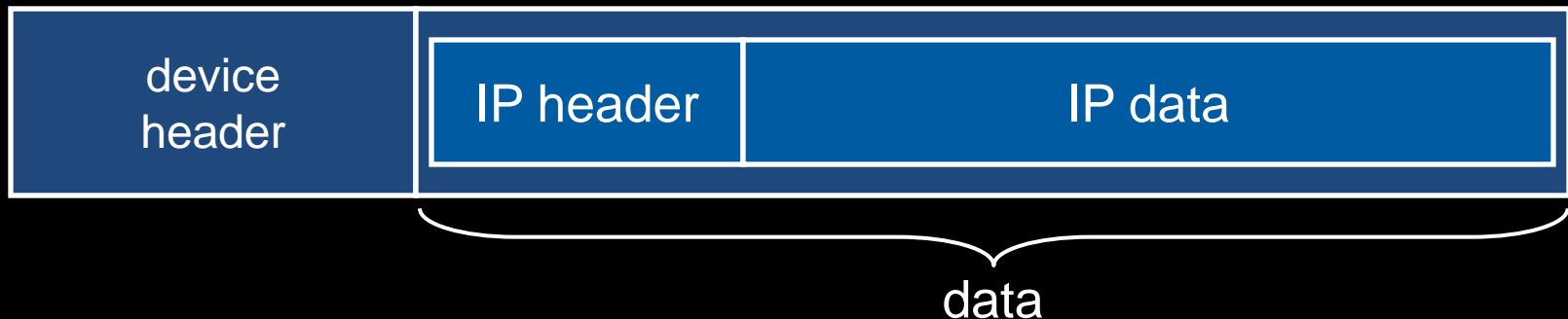
## UDP header



# Network device

---

- Network card examines packets on wire
  - Compares destination addresses
- Before packet is sent, it must be **enveloped** (**encapsulated**) for the physical network



# Device addressing

---

- We need to use a packet address that makes sense on the network we're using: IP addresses mean nothing to an ethernet NIC
- IP address → ethernet address
- Address Resolution Protocol (**ARP**)
  - Check local ARP cache
  - Send broadcast message requesting ethernet address of machine with certain IP address
  - Wait for response (with timeout)

The End