IP: “Best Effort” Packet Delivery

• Routers inspect destination address, locate “next hop” in forwarding table
  • Address = ~unique identifier/locator for the receiving host

• Only provides a “I’ll give it a try” delivery service:
  • Packets may be lost
  • Packets may be corrupted
  • Packets may be delivered out of order
“Best Effort” is Lame! What to do?

• It’s the job of our Transport (layer 4) protocols to build services our apps need out of IP’s modest layer-3 service
“Best Effort” is Lame! What to do?

• #1 workhorse: TCP (Transmission Control Protocol)

• Service provided by TCP:
  • Connection oriented (explicit set-up / tear-down)
    • End hosts (processes) can have multiple concurrent long-lived communication
  • **Reliable**, in-order, *byte-stream* delivery
    • Robust detection & retransmission of lost data
TCP “Bytestream” Service

Process A on host H1

Process B on host H2

Hosts don’t ever see packet boundaries, lost or corrupted packets, retransmissions, etc.
Bidirectional communication:

There are two separate bytestreams, one in each direction.
Ports: Analogy

- Alice is pen pals with Carol. Alice’s roommate Bob is also pen pals with Carol.
- Carol’s replies are addressed to the same global (IP) address. How to tell which letters are for Bob and which are for Alice?
Ports: Analogy

- Solution: Add a room number (port) inside the letter.
- In private homes like Alice/Bob, the port numbers are meaningless.
- In a public office (server) like Cory Hall, the port numbers are constant and known.
Ports

- Ports help us distinguish between different applications on a computer or server
- Remember: TCP is built on top of IP, so the IP address is still there

IP header: send to: 1.2.3.4
TCP header: send to: port 80
I’m hungry.
## TCP Header

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence number</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Acknowledgment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HdrLen</th>
<th>Flags</th>
<th>Advertised window</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checksum</th>
<th>Urgent pointer</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Options (variable)</th>
</tr>
</thead>
</table>

Data
### TCP Header

**Ports** are associated with OS processes.

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td>Flags</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
</tbody>
</table>

**Data**
TCP Header

Ports are associated with OS processes

IP source & destination addresses plus TCP source and destination ports uniquely identifies a TCP connection

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td>Flags</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
TCP Header

Ports are associated with OS processes.

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td>Flags</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
</tbody>
</table>

Data

IP source & destination addresses plus TCP source and destination ports uniquely identifies a TCP connection.

Some port numbers are "well known" / reserved, e.g., port 80 = HTTP.
TCP Header

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sequence number**

**Acknowledgment**

<table>
<thead>
<tr>
<th>HdrLen</th>
<th>Flags</th>
<th>Advertised window</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checksum</th>
<th>Urgent pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Options (variable)**

**Data**

Starting sequence number (byte offset) of data carried in this packet.
## TCP Header

Byte streams numbered independently in each direction.

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequence number</td>
</tr>
<tr>
<td></td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>HdrLen</td>
<td>Flags</td>
</tr>
<tr>
<td></td>
<td>Advertised window</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
</tr>
<tr>
<td></td>
<td>Options (variable)</td>
</tr>
</tbody>
</table>

Data
### TCP Header

The TCP header contains various fields that are used to control the transmission of data between two hosts. Here is a breakdown of the fields included in the TCP header:

- **Source port**
- **Destination port**
- **Sequence number**
- **Acknowledgment**
- **HdrLen**
- **Flags**
- **Checksum**
- **Urgent pointer**
- **Options (variable)**
- **Data**

**Sequence number**

- Byte stream numbered independently in each direction.
- Sequence number assigned to start of byte stream is picked when connection begins; **does not start at 0**

**Other fields**

- **Source port**: Port number of the sending host.
- **Destination port**: Port number of the receiving host.
- **Sequence number**: Starting sequence number (byte offset) of data carried in this packet.
- **Acknowledgment**: Byte stream numbered independently in each direction.
- **HdrLen**: Length of header.
- **Flags**: Various flags to control the transfer.
- **Checksum**: For error checking.
- **Urgent pointer**: Points to the urgency field in the data segment.
- **Options (variable)**: Optional fields that can be included in the header.

---

**Table:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source port</td>
<td>Port number of the sending host.</td>
</tr>
<tr>
<td>Destination port</td>
<td>Port number of the receiving host.</td>
</tr>
<tr>
<td>Sequence number</td>
<td>Starting sequence number (byte offset) of data carried in this packet.</td>
</tr>
<tr>
<td>Acknowledgment</td>
<td>Byte stream numbered independently in each direction.</td>
</tr>
<tr>
<td>HdrLen</td>
<td>Length of header.</td>
</tr>
<tr>
<td>Flags</td>
<td>Various flags to control the transfer.</td>
</tr>
<tr>
<td>Checksum</td>
<td>For error checking.</td>
</tr>
<tr>
<td>Urgent pointer</td>
<td>Points to the urgency field in the data segment.</td>
</tr>
<tr>
<td>Options (variable)</td>
<td>Optional fields that can be included in the header.</td>
</tr>
</tbody>
</table>
TCP Header

Acknowledgment gives seq # just beyond highest seq. received in order.

If sender sends N bytestream bytes starting at seq S then “ack” for it will be S+N.
Sequence Numbers

Host A

ISN (initial sequence number)

Sequence number from A = 1st byte of data

TCP HDR

TCP Data

Host B

TCP HDR

TCP Data

ACK sequence number from B = next expected byte
TCP Header

<table>
<thead>
<tr>
<th>Uses include:</th>
</tr>
</thead>
<tbody>
<tr>
<td>acknowledging data (&quot;ACK&quot;)</td>
</tr>
<tr>
<td>setting up (&quot;SYN&quot;) and closing</td>
</tr>
<tr>
<td>connections (&quot;FIN&quot; and &quot;RST&quot;)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence number</th>
<th>Acknowledgment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advertised window</th>
<th>Checksum</th>
<th>Urgent pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HdrLen</th>
<th>Flags</th>
<th>Options (variable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
</table>
Establishing a TCP Connection

- Three-way handshake to establish connection
  - Host A sends a **SYN** (open; “synchronize sequence numbers”) to host B
  - Host B returns a SYN acknowledgment (**SYN+ACK**)
  - Host A sends an **ACK** to acknowledge the **SYN+ACK**

Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on local clock)
Timing Diagram: 3-Way Handshaking

Different starting initial sequence numbers (ISNs) in each direction

Client (initiator)
- `connect()`
- `SYN, SeqNum = x`
- `SYN + ACK, SeqNum = y, Ack = x + 1`
- `ACK, Ack = y + 1`

Server
- `listen()`
- `Passive Open`
- Active Open

Different starting initial sequence numbers (ISNs) in each direction
UDP

- UDP (User Datagram Protocol) is an alternative to TCP
- At the transport layer (layer 4), you have to choose TCP or UDP
UDP

- UDP offers no reliability guarantees (still best-effort), but it adds ports
- Benefit: much faster than TCP (no handshake required)
- UDP header:

<table>
<thead>
<tr>
<th></th>
<th>16-bit source port</th>
<th>16-bit destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>16-bit length field</td>
<td>16-bit checksum</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>Payload: arbitrary data</td>
</tr>
</tbody>
</table>
TCP Conn. Setup & Data Exchange

Client (initiator)
IP address 1.2.1.2, port 3344

Server
IP address 9.8.7.6, port 80

SrcA=1.2.1.2, SrcP=3344,
DstA=9.8.7.6, DstP=80, SYN, Seq = x

SrcA=9.8.7.6, SrcP=80,
DstA=1.2.1.2, DstP=3344, SYN+ACK, Seq = y, Ack = x+1

SrcA=1.2.1.2, SrcP=3344,
DstA=9.8.7.6, DstP=80, ACK, Seq = x+1, Ack = y+1

SrcA=1.2.1.2, SrcP=3344, DstA=9.8.7.6, DstP=80,
ACK, Seq=x+1, Ack = y+1, Data="GET /login.html"

SrcA=9.8.7.6, SrcP=80, DstA=1.2.1.2, DstP=3344,
ACK, Seq = y+1, Ack = x+16, Data="200 OK … <html> …"
Abrupt Termination

• If A sends a TCP packet with RST flag to B and sequence number fits, connection is terminated
  • Unilateral, and takes effect immediately
TCP Threat: Disruption aka RST injection

- The attacker can inject RST packets and block connection
  - TCP clients must respect RST packets and stop all communication

- Who uses this?
  - China: The Great Firewall does this to TCP requests
  - A long time ago: Comcast, to block BitTorrent uploads
  - Some intrusion detection systems: To hopefully mitigate an attack in progress

**Discuss with a partner:** Who can do RST injection? (a) off-path attacker, (b) on-path attacker, (c) man-in-the-middle
TCP Threat: Data Injection

- If attacker knows **ports & sequence numbers** (e.g., on-path attacker), attacker can inject data into any TCP connection
  - Receiver B is *none the wiser!*

- Termed TCP **connection hijacking** (or “*session hijacking*”)
  - A general means to take over an already-established connection!

- **We are toast if an attacker can see our TCP traffic!**
  - Because then they immediately know the *port & sequence numbers*

![Diagram showing TCP connection hijacking](image-url)
TCP Data Injection

Client (initiator)
IP address 1.2.1.2, port 3344

Server
IP address 9.8.7.6, port 80

Attacker (AirPwn, QUANTUM, etc)
IP address 6.6.6.6, port N/A

SrcA=1.2.1.2, SrcP=3344, DstA=9.8.7.6, DstP=80,
ACK, Seq=x+1, Ack = y+1, Data="GET /login.html

Client dutifully processes as server’s response

SrcA=9.8.7.6, SrcP=80,
DstA=1.2.1.2, DstP=3344,
ACK, Seq = y+1, Ack = x+16
Data="200 OK … <poison> …"
TCP Data Injection

Client (initiator)
IP address 1.2.1.2, port 3344

Server
IP address 9.8.7.6, port 80

Attacker
IP address 6.6.6.6, port N/A

Client ignores since already processed that part of bytestream: the network can duplicate packets so only pay attention to the first version in sequence

SrcA=1.2.1.2, SrcP=3344, DstA=9.8.7.6, DstP=80, ACK, Seq=x+1, Ack = y+1, Data="GET /login.html"

SrcA=9.8.7.6, SrcP=80, DstA=1.2.1.2, DstP=3344, ACK, Seq = y+1, Ack = x+16, Data="200 OK … <poison> …"

SrcA=9.8.7.6, SrcP=80, DstA=1.2.1.2, DstP=3344, ACK, Seq = y+1, Ack = x+16, Data="200 OK … <html> …"
TCP Threat: Blind Hijacking

• Is it possible for an off-path attacker to inject into a TCP connection even if they can’t see our traffic?
• YES: if somehow they can infer or guess the port and sequence numbers
TCP Threat: Blind Spoofing

- Is it possible for an off-path attacker to create a fake TCP connection, even if they can’t see responses?
- Yes if somehow they can infer or guess the TCP initial sequence numbers
- Why would an attacker want to do this?
  - Perhaps to leverage a server’s trust of a given client as identified by its IP address
  - Perhaps to frame a given client so the attacker’s actions during the connections can’t be traced back to the attacker
Blind Spoofing on TCP Handshake

Alleged Client (not actual)
IP address 1.2.1.2, port N/A

Server
IP address 9.8.7.6, port 80

Blind Attacker
SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, SYN, Seq = z

Attacker’s goal:
SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, ACK, Seq = z+1, ACK = y+1

SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, ACK, Seq = z+1, ACK = y+1, Data = “GET /transfer-money.html”
Blind Spoofing on TCP Handshake

**Alleged Client (not actual)**
- IP address 1.2.1.2, port NA

**Server**
- IP address 9.8.7.6, port 80

**Blind Attacker**
- SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, SYN, Seq = z
- SrcA=9.8.7.6, SrcP=80, DstA=1.2.1.2, DstP=5566, SYN+ACK, Seq = y, Ack = x+1

Small Note #1: if alleged client receives this, will be confused ⇒ send a RST back to server … … So attacker may need to hurry!

But firewalls may inadvertently stop this reply to the alleged client so it never sends the RST 😐
Blind Spoofing on TCP Handshake

Alleged Client (not actual)
IP address 1.2.1.2, port NA

Server
IP address 9.8.7.6, port 80

Blind Attacker
SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, SYN, Seq = z
SrcA=9.8.7.6, SrcP=80, DstA=1.2.1.2, DstP=5566, SYN+ACK, Seq = y, Ack = z+1

Big Note #2: attacker doesn’t get to see this packet!
Blind Spoofing on TCP Handshake

Alleged Client (not actual)
IP address 1.2.1.2, port N/A

Server
IP address 9.8.7.6, port 80

Blind Attacker
SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, SYN, Seq = z

SrcA=9.8.7.6, SrcP=80, DstA=1.2.1.2, DstP=5566, SYN+ACK, Seq = y, Ack = z+1

So how can the attacker figure out what value of y to use for their ACK?

SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, ACK, Seq = z+1, ACK = y+1

SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, ACK, Seq = z+1, ACK = y+1, Data = “GET /transfer-money.html"
**Reminder: Establishing a TCP Connection**

Each host tells its *Initial Sequence Number* (ISN) to the other host. (Spec says to pick based on local clock)

Hmm, any way for the attacker to know this?

Sure – make a non-spoofed connection first, and see what server used for ISN y then!

How Do We Fix This?

Use a (Pseudo)-Random ISN
Summary of TCP Security Issues

• An attacker who can observe your TCP connection can manipulate it:
  • Forcefully terminate by forging a RST packet
  • Inject (spoof) data into either direction by forging data packets
  • Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  • Remains a major threat today

• Blind spoofing no longer a threat
  • Due to randomization of TCP initial sequence numbers
Ghost of blind spoofing...

- CVE-2016-5696
  - "Off-Path TCP Exploits: Global Rate Limit Considered Dangerous" Usenix Security 2016 https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cao

- Key idea:
  - RFC 5961 added some global rate limits that acted as an information leak:
    - Could determine if two hosts were communicating on a given port
    - Could determine if your guess at the sequence number is “in window”
  - Once you get the sequence #s, you can then inject arbitrary content into the TCP stream
- Fixed today