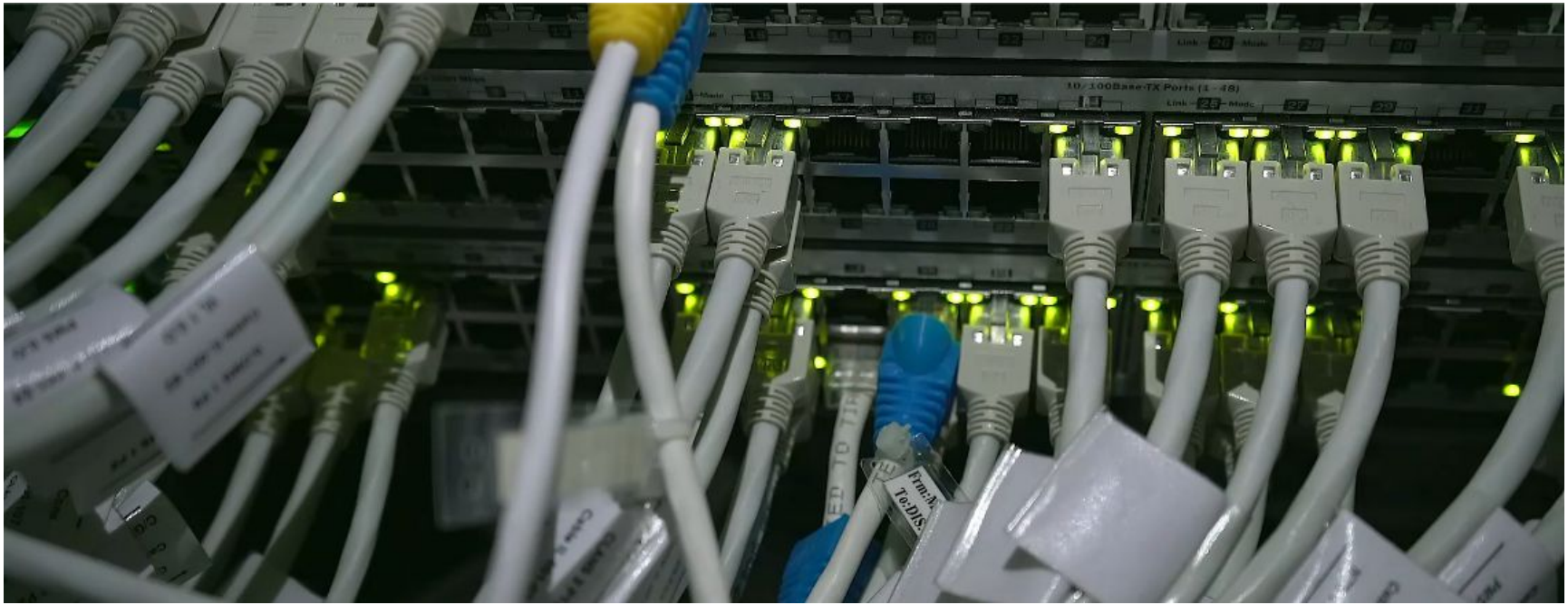
Week 8
Network Security - Protocols

THE UNIVERSITY OF SYDNEY

Dr. Suranga Seneviratne

School of Computer Science, The University of Sydney

Part I - Networking Overview



Agenda - Part I

- Networking Overview
- Attacks on Networks

Assumed Knowledge - Part I

- Computer Networking: A top-down approach – Sixth Edition by James Kurose and Keith Ross:
  - **Chapter 1** – Computer Network And The Internet
  - **Chapter 2** – Application Layer
  - **Chapter 3** – Transport Layer
  - **Chapter 4** – The Network Layer
  - **Chapter 5** – The Link Layer

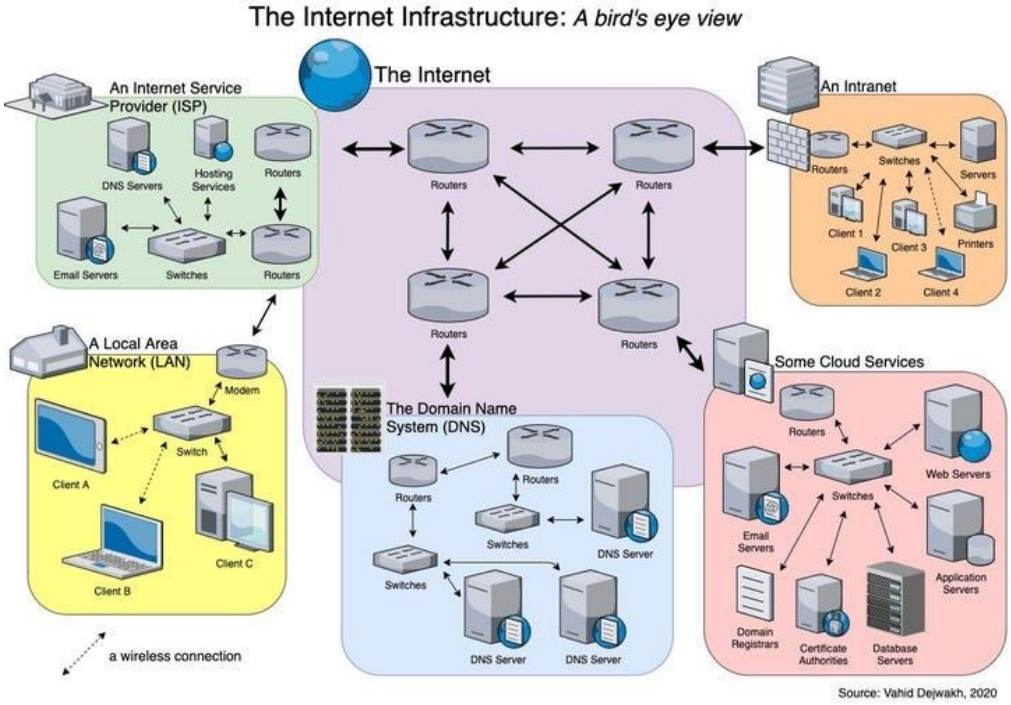
Acknowledgments

We based our slides on the latest version of the unit "Network Security" at the Technical University of Munich, Germany, and “Computer Networking: A top-down approach 8-th edition” slides and thank them for the permission to adapt their slides.

On Network Security

- We **depend** on the networks supporting our communication
  - Business
  - Government
  - Private sphere
- Network security deals with attacks and defences in the context of networked systems
- The field is huge – worth its own unit of study

The Internet



Billions of Connected Computing Devices - Commonly referred to as **hosts** or **end systems**, these are the devices at the "edge" of the network where applications actually run such as smartphones, laptops, servers, and even IoT devices.

Packet Switches - These are the traffic controllers of the digital world. They take incoming **packets** (small chunks of data) and forward them toward their ultimate destination such as routers and switches.

Communication Links - The physical or "over-the-air" paths that connect devices and switches. Examples: Fiber optic cables, copper wires, radio waves, and satellite signals. Characterised by the transmission rate of these links is known as bandwidth.

Networks - A network is more than just hardware; it is a collection of devices, routers, and links that are managed by a single organization (e.g., an ISP, a university, or a corporation). The Internet is essentially a "network of networks."

What is a Protocol?

- Communications in the internet happens using protocols. They control sending, receiving of messages. e.g., HTTP (Web), TCP, IP, DNS, SSH

- Protocols define the format, order of messages sent and received among network entities, and actions taken on message transmission, receipt

Human protocols:

- “What’s the time?”
- “I have a question”

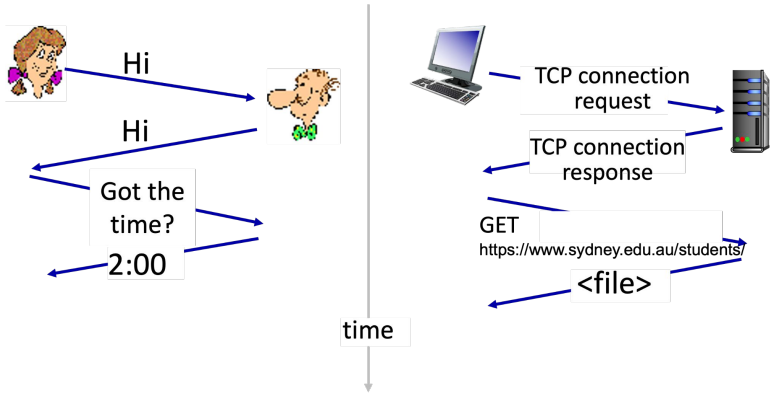
Network protocols:

- Computer (devices) rather than humans
- All communication activities on the Internet are governed by protocols

What is a Protocol?

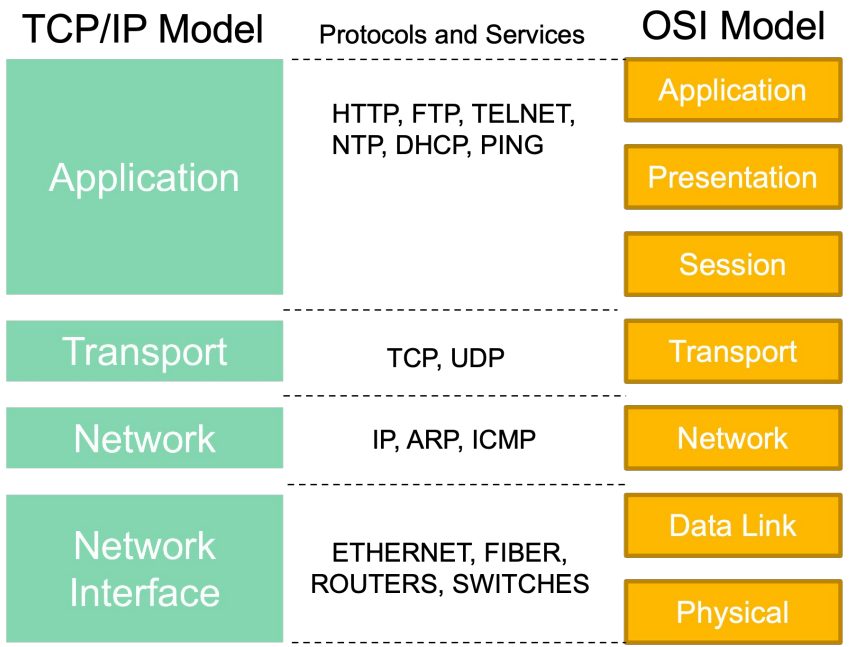
- Communications in the internet happens using protocols. They control sending, receiving of messages. e.g., HTTP (Web), TCP, IP, DNS, SSH

- Protocols define the format, order of messages sent and received among network entities, and actions taken on message transmission, receipt



The TCP/IP Model

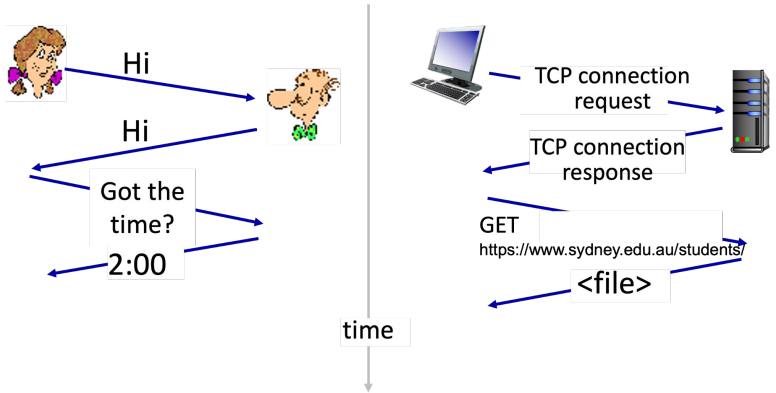
- The four layers of the TCP/IP model:
  - Application layer
  - Transport layer
  - Network layer
  - Network interface layer
    - Link layer
    - Media Access Control (MAC) Layer



What is a Protocol?

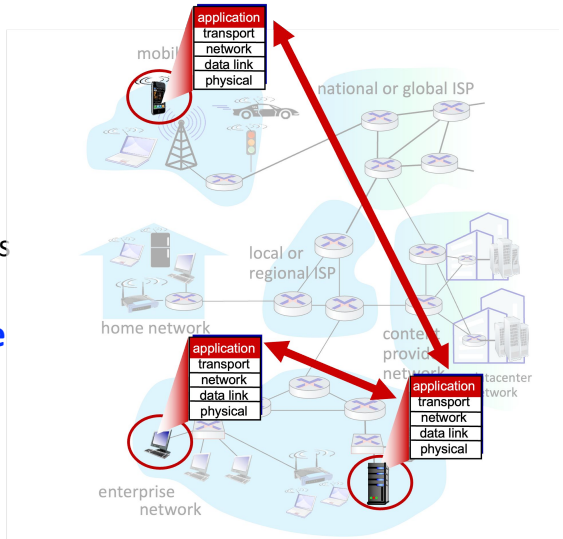
- Communications in the internet happens using protocols. They control sending, receiving of messages. e.g., HTTP (Web), TCP, IP, DNS, SSH

- Protocols define the format, order of messages sent and received among network entities, and actions taken on message transmission, receipt



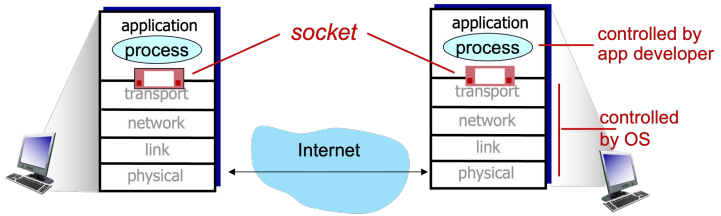
Application Layer

- Where network applications (e.g., social networking, Web, email, voice over IP, ...) and their application-layer protocols reside. For example
    - **HTTP (Hypertext Transfer Protocol):** for Web document requests and transfer
    - **SMTP (Simple Mail Transfer Protocol):** provides for the transfer of e-mail messages
    - **FTP (File Transfer Protocol):** provides for the transfer of files between two end systems
    - **DNS (Domain Name System):** hosts, DNS servers communicate to resolve names
- We refer to this packet of information at the application layer as a **message**
- But how do application processes actually send and receive these messages? **Through sockets**



Sockets - The Interface

- **Process:** program running within a host
- Process sends/receives messages to/from its socket
- Socket analogous to door
  - Sending process shoves message out door
  - Sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
  - Two sockets involved: one on each side
- Key distinction:
  - **Socket and above:** controlled by app developer (Application Layer)
  - **Below socket:** controlled by OS (Transport Layer and below)
- Sockets are the API between the application and transport layers

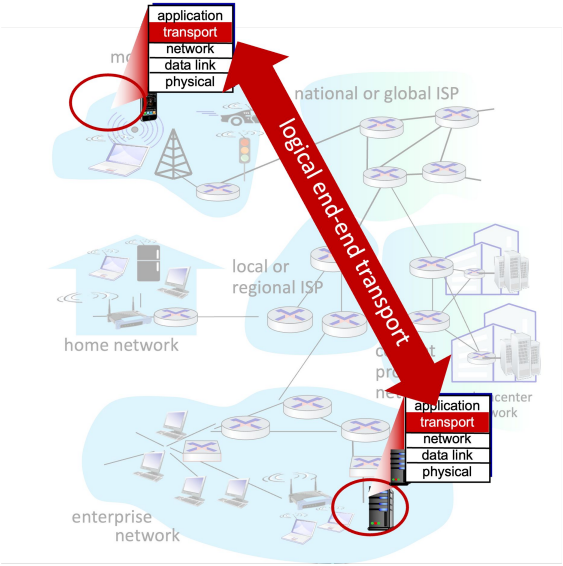


Transport Layer

- Provides **logical communication** between application processes running on different hosts Transport protocol actions in end systems:
  - **Sender:** breaks application messages into **segments**, passes to the network layer
  - **Receiver:** reassembles segments into messages, passes to the application layer

- **Two transport protocols** available to Internet applications:
  - **TCP** (Transmission Control Protocol)
  - **UDP** (User Datagram Protocol)

- Transport layer receives data from sockets and handles end-to-end delivery



TCP vs. UDP

TCP: Transmission Control Protocol

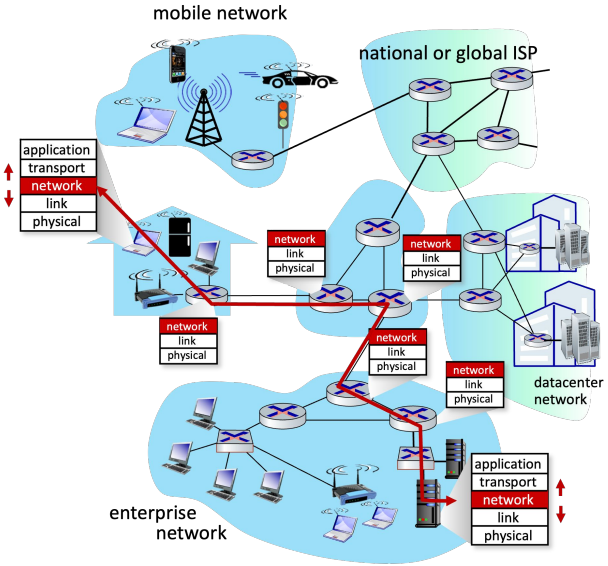
- Connection-oriented & Reliable
  - Three-way handshake establishes connection
  - Point-to-point: one sender, one receiver
  - Reliable, in-order byte stream delivery
  - Cumulative ACKs with retransmission
  - Bi-directional full duplex data flow
- Flow & Congestion Control
  - Flow control: sender won't overwhelm receiver
  - Congestion control: adjusts to network conditions
  - Pipelining with dynamic window sizing
  - MSS (Maximum Segment Size)
- **Use cases:** File transfer, email, web pages, any application requiring guaranteed delivery

UDP: User Datagram Protocol

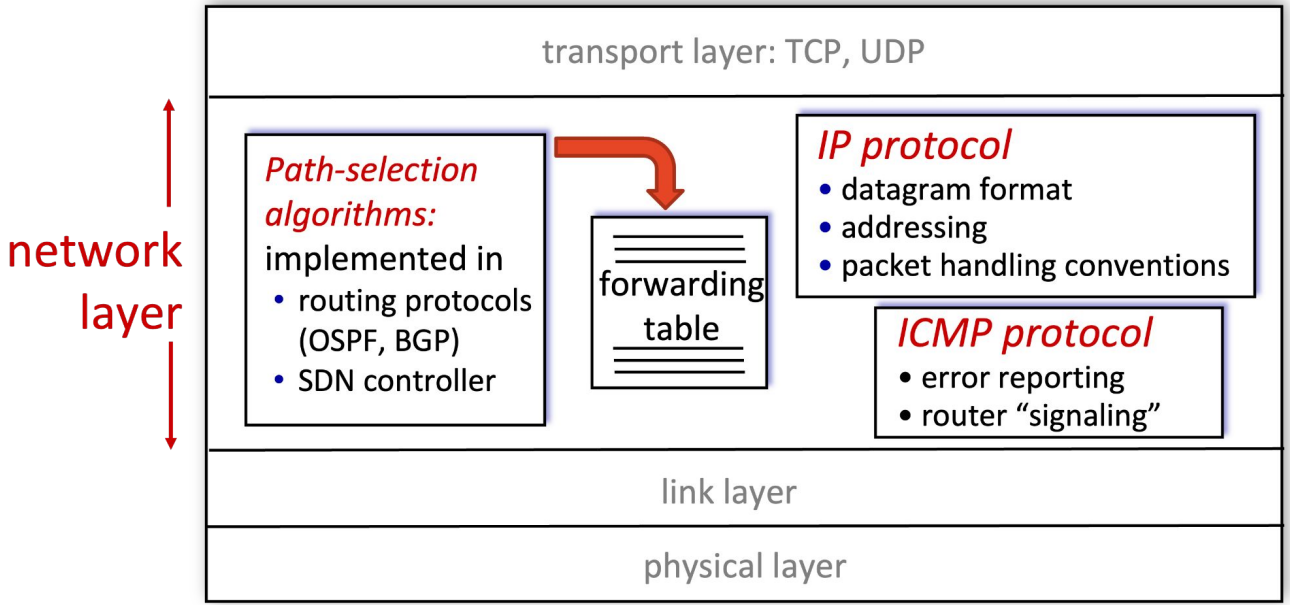
- Connectionless & Lightweight
  - No handshake (no RTT overhead)
  - "No frills" - best effort delivery
  - Unreliable, unordered delivery
  - Segments may be lost or out of order
- Advantages
  - Fast: no connection setup overhead
  - Works when network service is compromised
  - Simple: minimal protocol overhead
  - Checksum for basic error detection
  - Application layer can add custom reliability
- **Use cases:** Streaming media (loss tolerant), DNS, SNMP, HTTP/3, real-time apps (rate sensitive)

Network Layer

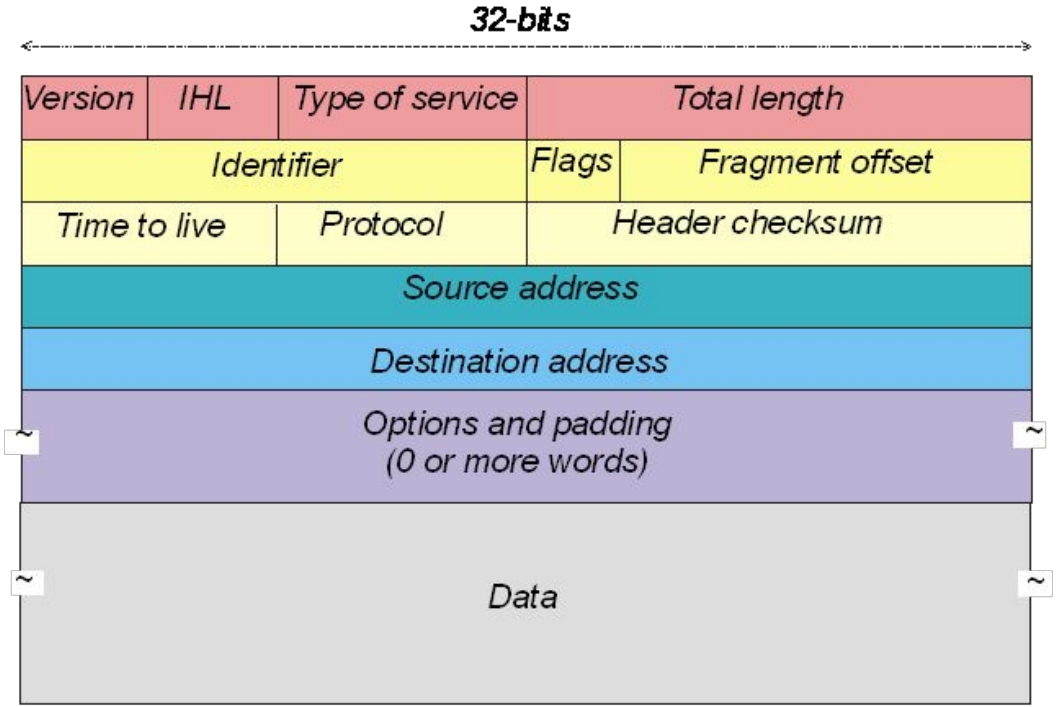
- Also called **Internet layer**
- Transport segment from sending to receiving host
  - **Sender**: encapsulates segments into datagrams, passes to link layer
  - **Receiver**: delivers segments to transport layer
- Network layer protocols are in **every Internet device**: hosts, routers
- **Routers**:
  - Examines header fields in all IP datagrams passing through it
  - Moves datagrams from input ports to output ports to transfer datagrams along end-end path



Network Layer - Protocols and Functions

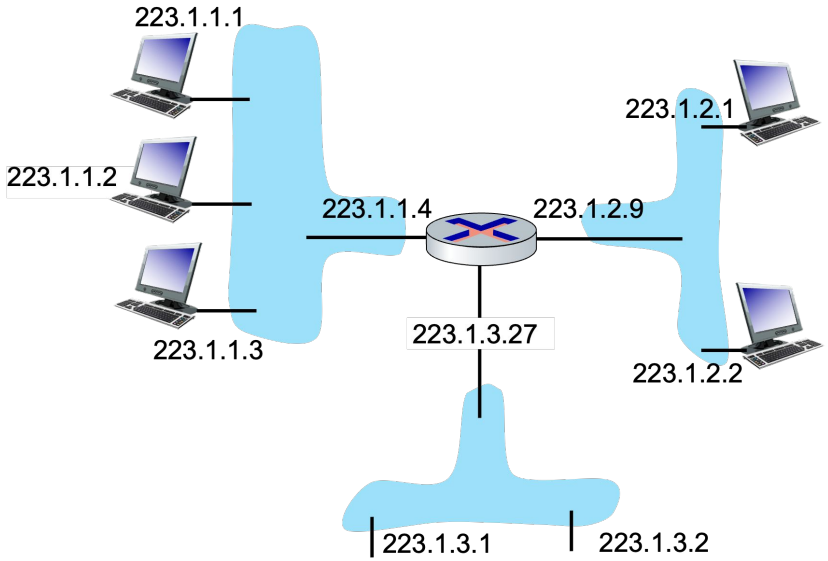


IP Datagram Format



IP Addressing

- IPv4 address: 32-bit identifier associated with each host or router **interface**
- Interface: connection between host/router and physical link
  - Router's typically have multiple interfaces
  - Host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)

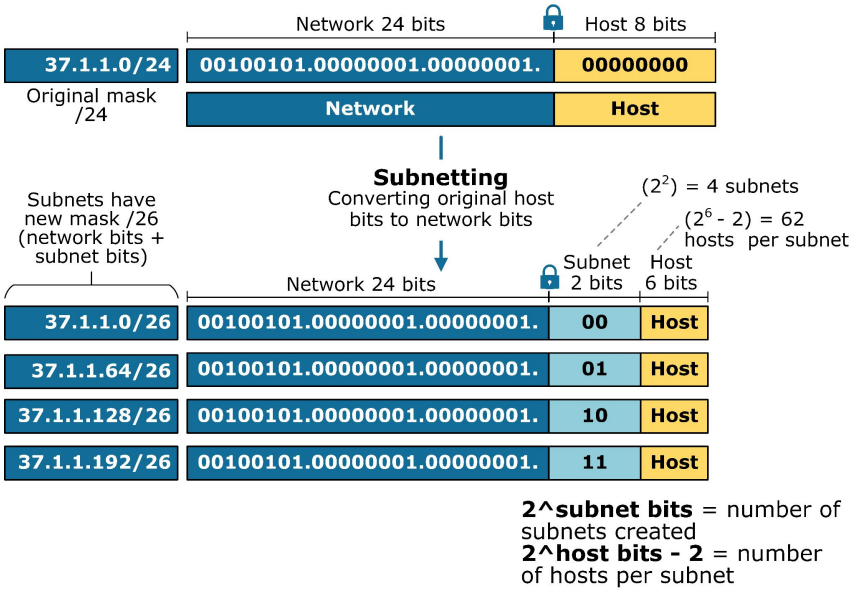


223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

Subnets

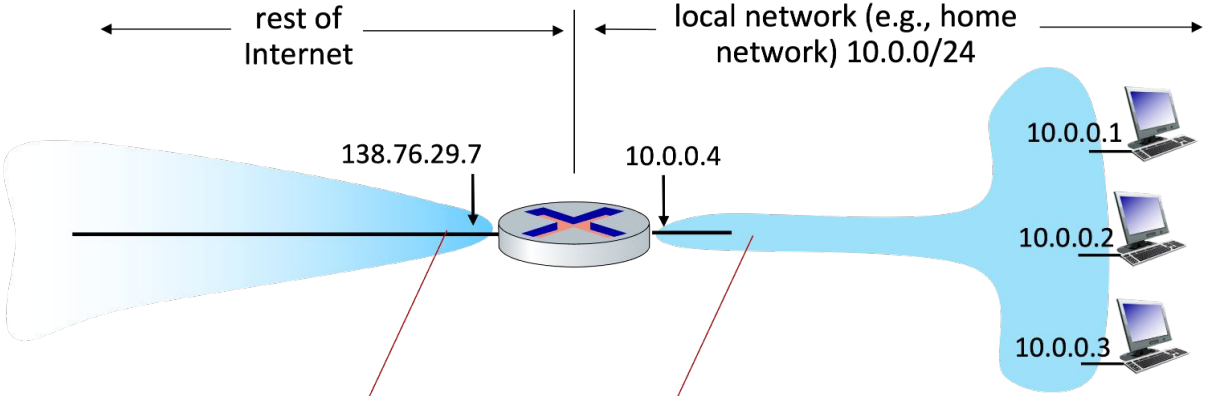
- What's a subnet ?
  - Device interfaces that can physically reach each other **without passing through an intervening router**
- IP addresses have structure:
  - Subnet part: Devices in same subnet have common high order bits
  - Host part: Remaining low order bits
- Recipe for defining subnets:
  - Detach each interface from its host or router, creating "islands" of isolated networks
  - Each isolated network is called a subnet



https://www.networkacademy.io/ccna/ip-subnetting/what-is-subnetting

NAT: Network Address Translation

- NAT: all devices in local network share just one IPv4 address as far as outside world is concerned

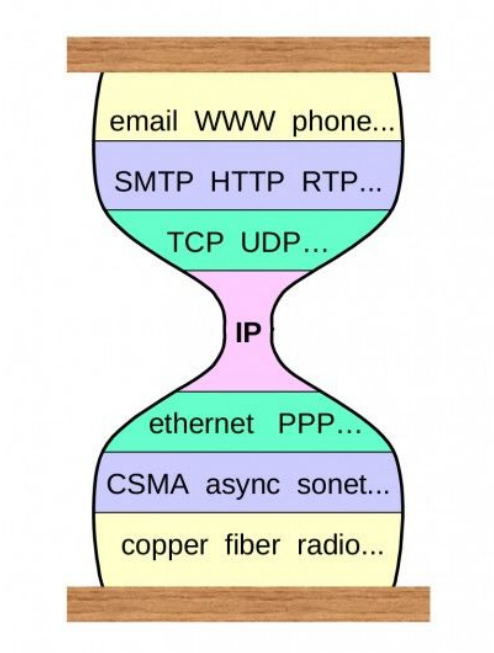


IPv6

- Expanded Address Capacity
  - Moves from 32-bit to 128-bit addresses, providing trillions of IPs for every person on Earth.
  - Eliminates the need for NAT (Network Address Translation), allowing every device a unique public IP.
- Streamlined Header Design and Modern Traffic Handling
  - Uses a 40-byte fixed header to simplify and accelerate router processing.
  - Removed the header checksum and hop-by-hop fragmentation, offloading error-checking and packet-sizing tasks to the network "edges."
  - A 20-bit field allows routers to identify and prioritize specific traffic streams
  - Native Security: Designed with IPsec support as a core requirement, providing a standardized framework for encryption and authentication.
- The Migration Gap (Why it’s not finished)
  - Compatibility, Financial ROI, NAT versions

IP Hourglass

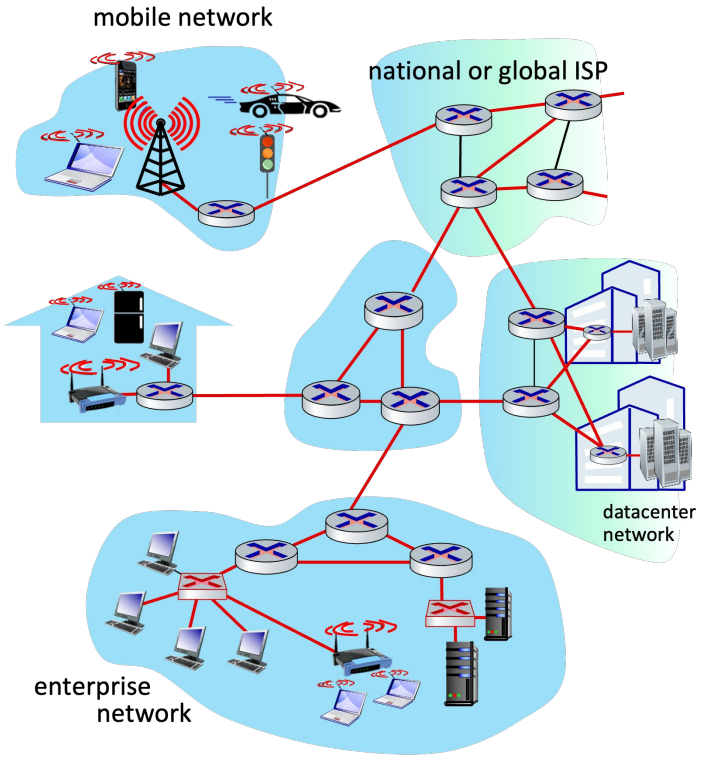
- Internet's "thin waist":
  - One network layer protocol: IP
  - Must be implemented by every (billions) of Internet-connected devices



Link Layer

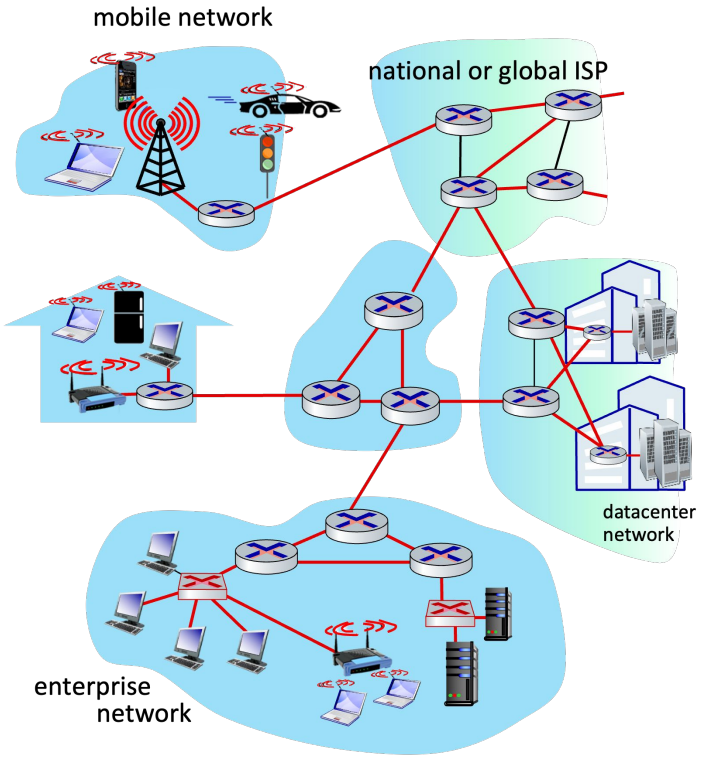
Terminology

- Hosts and routers: nodes
- Communication channels that connect adjacent nodes along communication path: links
  - Wired
  - Wireless
  - LANs
- Layer-1 (or 2) packet: frame, encapsulates datagram
- Link layer has responsibility of transferring datagram from one node to physically adjacent node over a link



Link Layer Services

- Framing, link access: encapsulate datagram into frame, MAC address in frame headers identify source, destination (not same as IP address)
- Reliable delivery between adjacent nodes
  - Flow control
  - Error correction
  - Error detection
  - Half-duplex and full-duplex

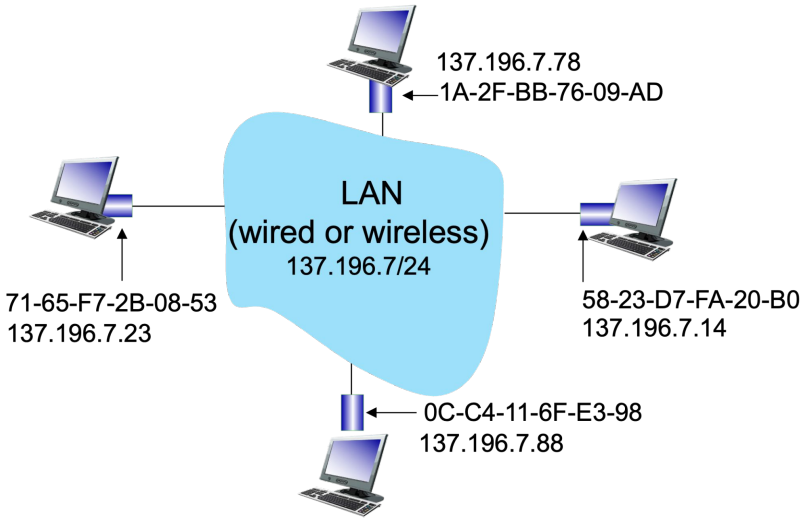


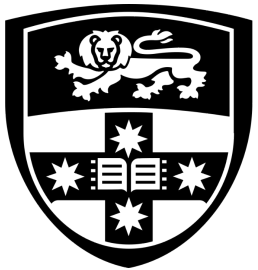
MAC Addresses

- Function: used “locally” to get frame from one interface to another physically-connected interface

- Each interface on Local Area Network (LAN):
  - Has unique 48-bit MAC address
  - Has a locally unique 32-bit IP address (as we’ve seen)

- Use Address Resolution Protocol (ARP) to find MAC address based on given IP address.



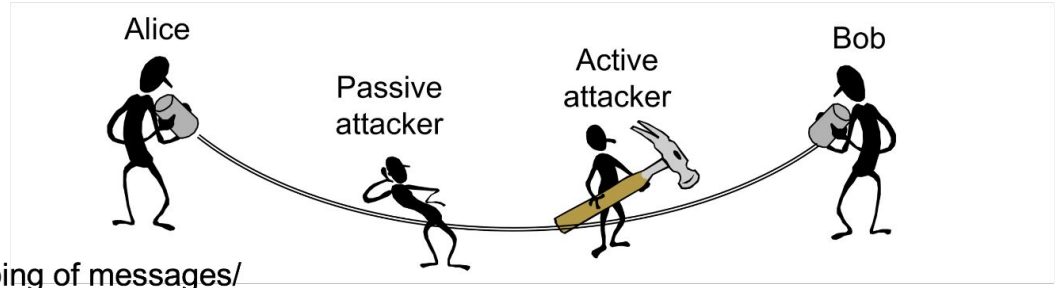


THE UNIVERSITY OF
SYDNEY

What is an Attack on a Network

- An event or sequence of actions that might lead to a violation of one or more security goals
  - Decisive difference to "simple" software security: the input to our programs (e.g. web server, mail server) comes from remote, and there is no a priori control who can send us input.
  - Further attacks become possible simply by the remote side being able to choose any kind or any amount of traffic to our own network.
  - Successful intrusion into our network gives attacker plenty of opportunity to explore, breach, and exploits

Attacks on Communication Networks



Eavesdropping of messages/
Recognition of communication parties

Message forging

Replay/delay messages

Passive attacks

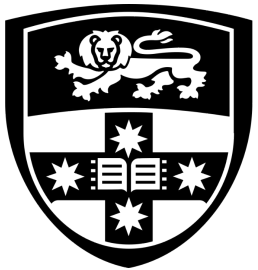
Active attacks

Traffic analysis

Masquerade

Insert / delete messages

Sabotage
(“Denial of Service“ attacks)



THE UNIVERSITY OF
SYDNEY

Part II - Networking Security: Protocols

Agenda - Part II

- Network security – protocols
  - Application Layer
  - Transport Layer
  - IP Layer

Recommended Reading - Part II

- Computer Networking: A top-down approach – Sixth Edition by James Kurose and Keith Ross:
  - **Chapter 8** – Security In Computer Networks

- Cryptography and Network Security - Seventh Edition by William Stallings:
  - **Chapter 17** – Transport-Level Security
  - **Chapter 20** – IP Security

Common Cryptographic Protocols by Layer

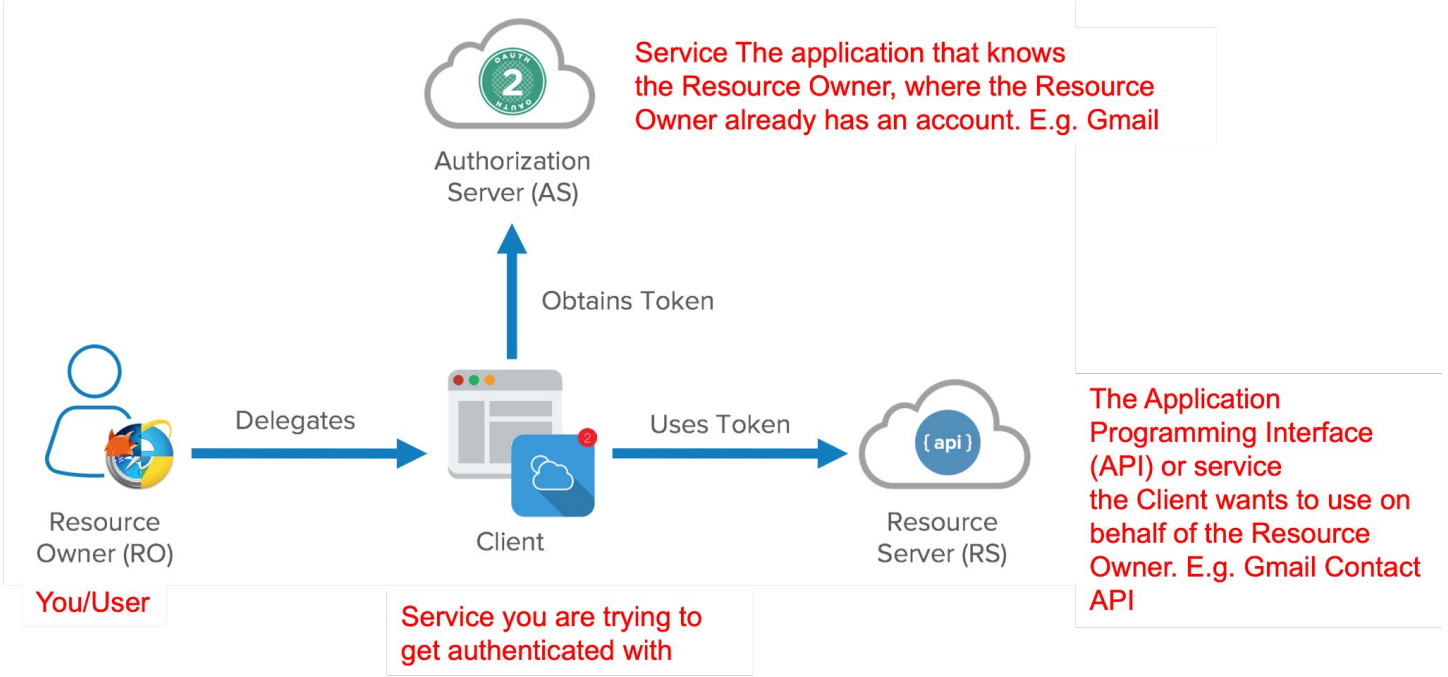
- **Link layer**
  - WPA3, WPA2 (broken: WPA, WEP)
- **Network layer**
  - IPSec and IKEv2
- **Transport Layer**
  - TLS (for TCP), DTLS (for UDP)
- **Application Layer**
  - OpenID, OAuth, DNSSEC, …

As of today, TLS, WPA2/3, OpenID, OAuth, and IPSec are the ones you are most likely to encounter in a professional role (in that order). We focus on the standards above Layer 2, working top-down.

Application Layer: OAuth and OpenID

- OAuth is a **standard for authorization**
  - Allows users of an Internet service to grant another service access to (parts of) their account
  - Some ideas remind of Kerberos: *e.g.,* an authorisation server creates a ‘ticket’ that allows access
- However, OAuth is designed for use with HTTP; websites can use the defined flows
  - Results in highly complex protocol flows
  - Several possible flows have been standardized
- OpenID is an authentication protocol that uses OAuth flows, but is complementary
- Relatively complex protocols; but for cloud services, they have become standard (Google, Microsoft Azure, Facebook, Twitter, ...)

Application Layer: OAuth



https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc

https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth

Transport Layer: Transport Layer Security (TLS)

- Originally developed as Secure Socket Layer (SSL) by Netscape (later known as Mozilla)
  - SSLv1 proved to be (very) flawed, never real deployment
  - SSLv2 saw deployment but was also flawed!
  - SSLv3 finally robust, standardised as Transport Layer Security by the IETF
- Out of old habit, TLS is still sometimes referred to as SSL
  - However, SSL3 has been phased out (marginal usage)
- Latest version of TLS is 1.3
  - Standardised in 2018
  - We discuss TLS 1.2 first, then TLS 1.3

TLS Design Rationale

- Original case for SSL/TLS: protect HTTP on the application layer

- **Design goals:**
  - Generic security layer (layer ‘4.5’) between TCP and application layer
  - TLS makes use of TCP underneath
  - Reuse of ‘socket abstraction’: applications read/write to/from TLS sockets instead of TCP sockets
  - Use of **encryption, integrity, origin authentication** as transparent to the application as possible

- Why not placing TLS on the application layer? Because:
  - Every application layer protocol would have to be extended to use it
  - Application-layer protocols are often text-based (very inefficient)

Recap: Socket Abstraction

- Application requests network communication from kernel by defining requirements, e.g., reliable, stream-based
- Socket is an object into which the application can write, and from which it can read—very much like a file descriptor
- Kernel sends/receives over network
  - ‘Streaming communication’: TCP does reliable, connection-oriented communication
  - ‘Datagram communication’: UDP does for unreliable, message-oriented communication
- Socket is abstraction over source IP, destination IP, source port, destination port, and protocol used
- Raw socket is a socket without any layer-specific formatting done by the kernel—all left to application

TLS Sockets

- Sockets are created with system calls, using a standardised API
- TLS sockets are implemented in libraries
  - Library implements ciphers, MACs, etc.
  - Library creates socket with the help of the kernel
  - Library provides TLS socket to application, which behaves similarly to normal TCP socket
- Example: Python TCP (client initating the connection)

```
socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
s.connect((HOST, PORT))
```

- Common design: TLS socket ‘wraps’ around normal socket

```
t = ssl.wrap_socket(sock, …)
```

- Wrapping has many parameters - safe defaults are important

Conceptual view of TLS

- TLS assumes the existence of an X.509 PKI: authenticating party must have an X.509 certificate identifying it.
- Other variants (e.g., pre-shared key) exist, but are rare
- TLS authentication can be mutual; but commonly only the server authenticates to the client.
- Client application often authenticates to server application, *e.g.,* using passwords

- TLS 1.3 always uses **Authenticated Diffie-Hellman** (hence has forward security)
  - TLS 1.2 only strongly suggested it
  - TLS 1.2 also allowed classic hybrid cryptography, with key created together by client and server with complex key creation between the two parties

Conceptual Protocol Flow

- We show the idea of TLS for server-only authentication, with Diffie-Hellman. Let the actors be client **C** and server **S**. Secret Diffie-Hellman values are c and s, respectively

$$C \rightarrow S : N_C$$
$$S \rightarrow C : N_S, Cert_S, g^s, Sig_S(N_C, N_S, g^s)$$

$$C \rightarrow S : g^c, MAC_{K_{C,S}}(X)$$
$$(Let\ X := N_C, N_S, Cert_S, g^s, g^c, Sig_S(N_C, N_S, g^s)$$

$$S \rightarrow C : MAC_{K_{C,S}}(Y)$$
$$(Let\ Y := (X, MAC_{K_{C,S}}(X))$$

Rationale of the Protocol Flow

- $C \rightarrow S : N_C$

  Client sends nonce to identify new session

- $S \rightarrow C : N_S, Cert_S, g^s, Sig_S(N_C, N_S, g^s)$

  Server picks nonce of its own to refer to new session; sends certificate; picks Diffie- Hellman value; signs everything

This first half of the TLS protocol flow is mainly the negotiation of the cryptography and declaration of nonces that will be used for freshness.

Rationale of the Protocol Flow

- $C \rightarrow S: g^c, MAC_{K_{C,S}}(X)$

  $(Let\ X := N_C, N_S, Cert_S, g^s, g^c, Sig_S(N_C, N_S, g^s)$

  Client can authenticate S thanks to the certificate and the signed nonce. It can derive the symmetric key and use MACs from this point on. Client sends Diffie-Hellman value of its own

- $S \rightarrow C: MAC_{K_{C,S}}(Y)$

  $(Let\ Y := (X, MAC_{K_{C,S}}(X))$

  Correctness of Diffie-Hellman can be checked via MAC. Server has symmetric key now, too. Server creates final MAC on everything. When receiving the final MAC, client can be assured that no parameters have been tampered with.

The second half is the actual authentication and key establishment. The pattern of ‘two messages for negotiation, two for AKE’ is quite common in protocols.

Negotiations in TLS

- Negotiates concrete ciphers, hash functions, and MACs to use between the principals
  - This is very common in many cryptographic protocols
  - TLS standard contains codepoints for supported mechanism
  - Client proposes suite of mechanisms; server picks its choice or rejects connection

- TLS supports extensions:
  - Server Name Indication: domain name the connection is meant for
  - Identification of protocol that is being protected
  - Several other, useful mechanisms, like use of Certificate Transparency and indication of application-layer protocol, unusual crypto etc.

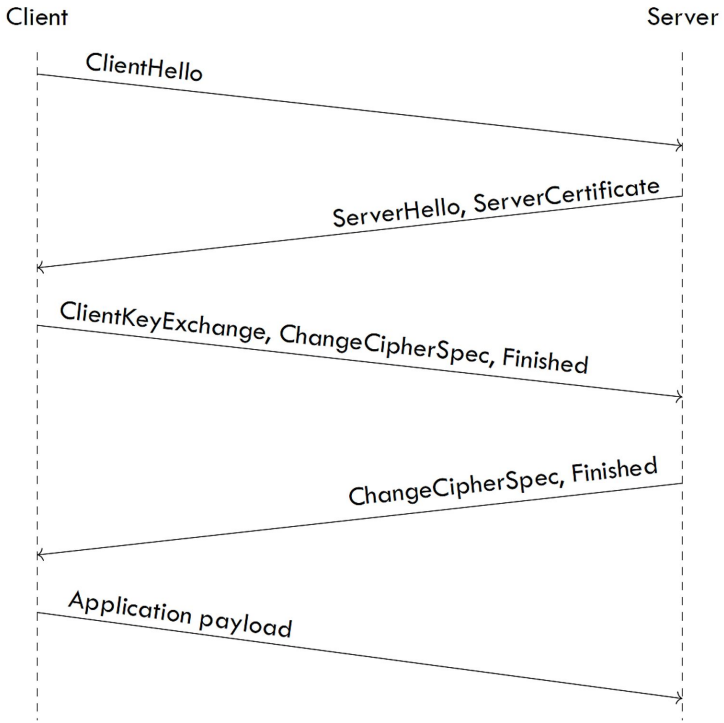
TLS Subprotocols

- TLS is message-based: subprotocols define message exchanges
- Record Protocol embeds messages and maps them onto TCP
- Handshaking Protocols: group of protocols consisting of
  - Handshake Protocol (protocol name, not category)
  - Change Cipherspec: signal transitions in use of cryptography
  - Alert: signal errors
- TLS Application Data: secured payloads

TLS Handshake Protocol

- Our ‘conceptual protocol flow’ is really the Handshake Protocol
- Its messages map to our abstract notation. **The most important ones are:**
  - `ClientHello:` Version number, nonce, cipher suites
  - `ServerHello:` Version number, nonce, cipher suites, session ID
  - `ServerCertificate:` certificate
  - `ServerKeyExchange:` Diffie-Hellman parameter
  - `ClientKeyExchange:` Diffie-Hellman value
  - `Finished:` MACs
- Messages of the Change CipherSpec protocol may appear before or after messages of the Handshake Protocol; they signal that the next messages use (different) encryption

Protocol Flow - TLS 1.2



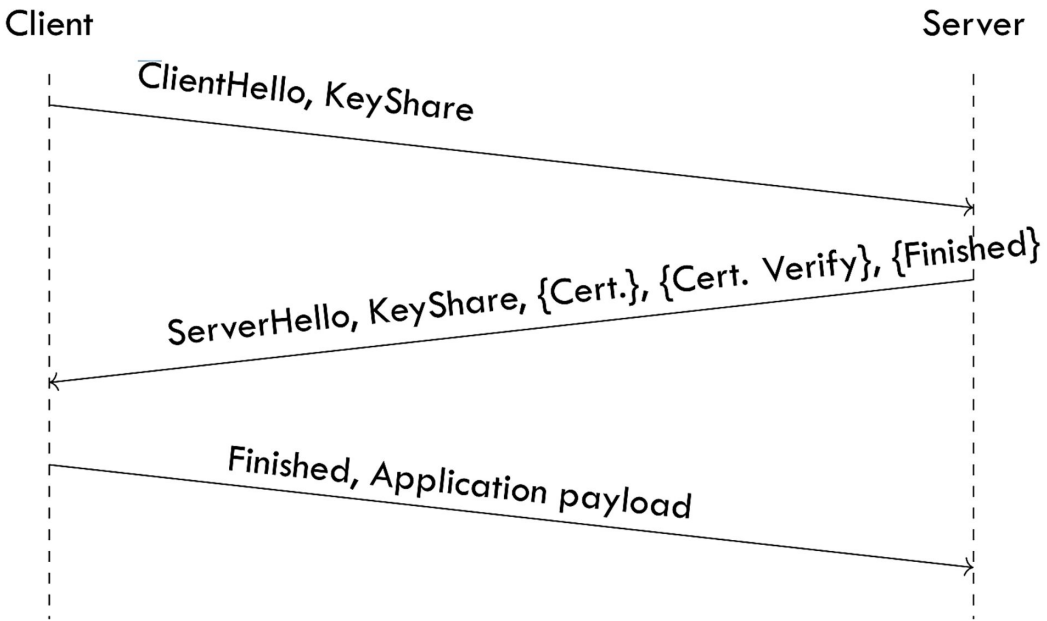
TLS 1.3

- TLS 1.2 was already the **result of many improvements** to TLS 1.0 and then TLS 1.1
- Still contained weaknesses due to its use of relatively old cryptography, MAC-then-encrypt instead of Encrypt-then-MAC, use of compression
- From about 2008, serious pressure on the protocol - redesign was needed

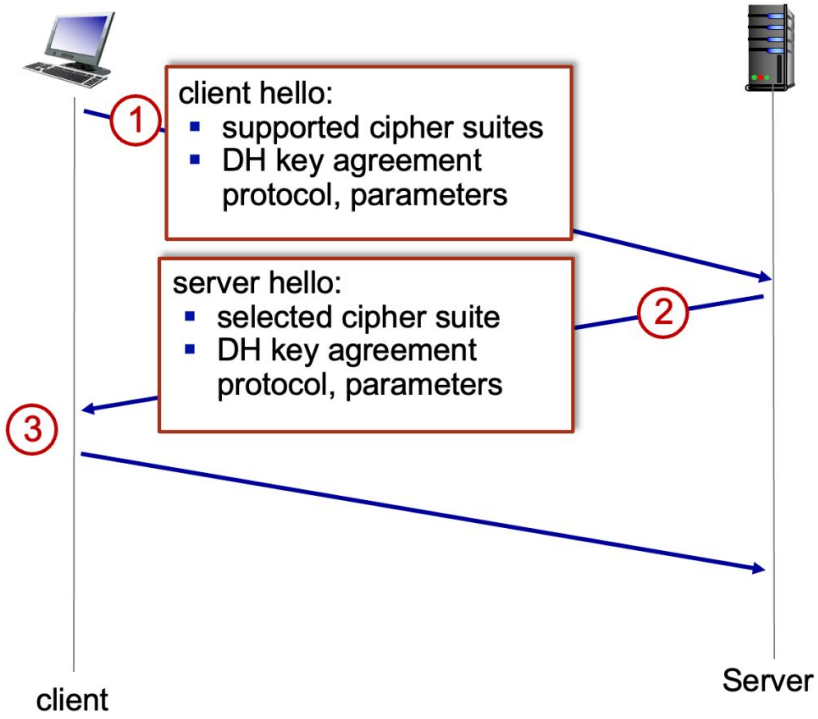
Major Differences in TLS 1.3

- TLS 1.2 allowed null cipher (authenticate-only); disallowed now
- TLS 1.3 allows only **AEAD ciphers** (e.g., AES-GCM, ChaCha20-Poly1305)
- Diffie-Hellman now default except for one pre-shared key mode
- Encryption starts immediately after ServerHello
- Certificates are encrypted and integrity-protected
- Same for Diffie-Hellman values (KeyShare)
- **1-RTT:** client can send application payload data after just one RTT.

Protocol Flow - TLS 1.3



TLS 1.3 Handshake: 1-RTT



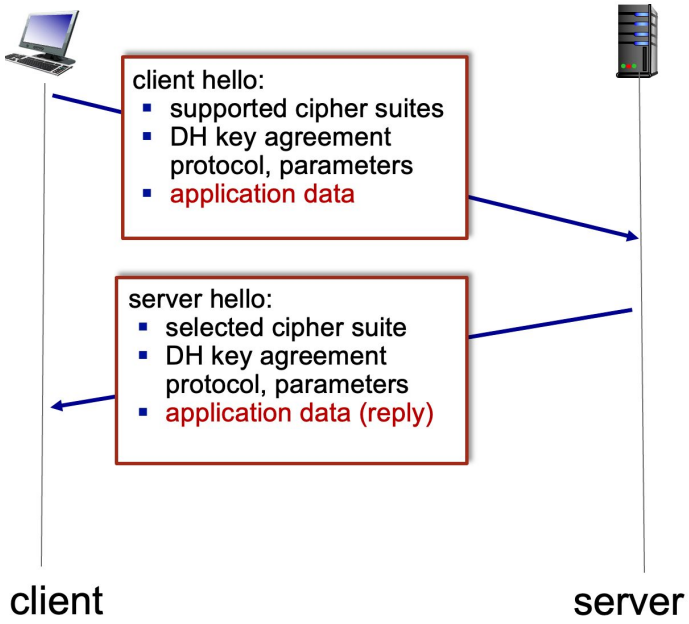
1. client TLS hello msg:
   - guesses key agreement protocol, parameters
   - indicates cipher suites it supports

2. server TLS hello msg chooses
   - key agreement protocol, parameters
   - cipher suite
   - server-signed certificate

3. client:
   - checks server certificate
   - generates key
   - can now make application request (e.g.., HTTPS GET)

TLS 1.3: 0-RTT

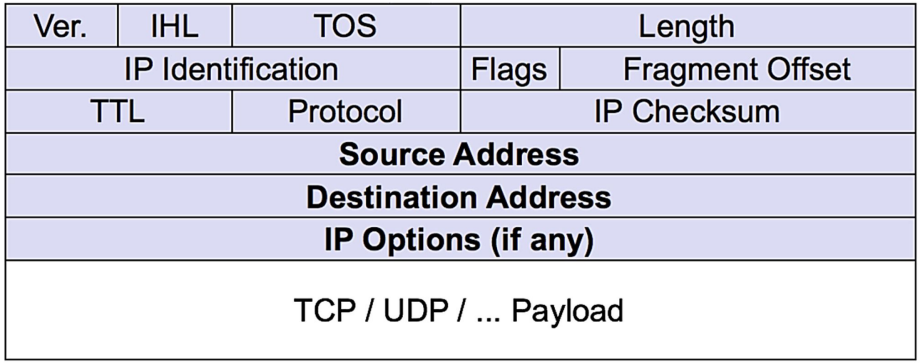
- TLS 1.2 has two ‘session resumption features’ (just one RTT)
  - Session identifier - server can look up previous key
  - Session ticket - client holds token that it can send server to resume connection
- TLS 1.3 deprecates this; replaces it with ‘pre-shared key’ mode
  - Conceptually similar to a session ticket
  - Use extension Early Data to signal 0-RTT will be used
  - Intentional design choice - acknowledges weaknesses
- Problem: session key not forward-secure
  - If compromised, all encrypted data is known to attacker
- Problem: no replay protection
  - Must only use for idempotent messages on application layer
  - E.g., HTTP GET
  - Problem: unknown if operators will implement this correctly



Internet Layer: IPSec

- **IPSec** is a set of protocols that, similar to TLS, provide:
  - Authentication and Key establishment
  - Confidentiality
  - Integrity
- IPSec embeds its protective mechanisms as IP payloads
  - Recall: TLS does it as TCP payloads
- **Difference to TLS:**
  - TLS does not protect any part of the TCP packet information (i.e., no integrity for port number, segment number etc.)
  - IPSec authenticates/integrity-protects the actual IP packet
  - TLS can work in user space. IPSec needs to be run in kernel space: it can only be configured by admins.

IPv4 Packet Format



- When an entity receives an IP packet, it has no assurance of
  - Data origin
  - Data integrity
  - Confidentiality

- The idea of IPSec is to configure the protection for IP packets between two IP address/address ranges - done with a policy

Applications of IPSec

- IPsec provides the capability to secure communications across a L A N, private and public WAN s, and the Internet

- Examples include:
  - Secure branch office connectivity over the Internet
  - Secure remote access over the Internet
  - Establishing extranet and intranet connectivity with partners
  - Enhancing electronic commerce security

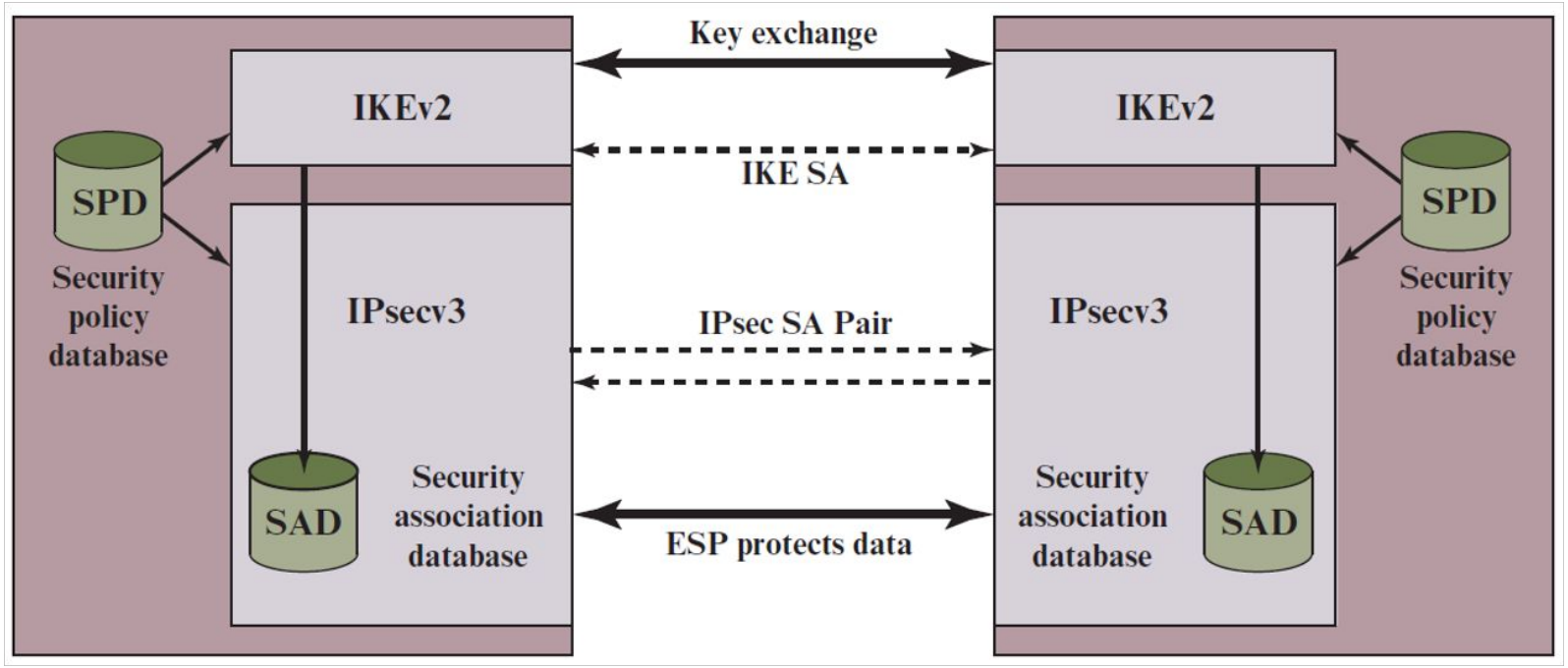
- Principal feature of IPSec is that it can encrypt and/or authenticate all traffic at the IP level
  - Thus, all distributed applications (remote logon, client/server, e-mail, file transfer, Web access) can be secured

IPSec Services

- IPsec provides security services at the IP layer by enabling a system to:
  - Select required security protocols
  - Determine the algorithm(s) to use for the service(s)
  - Put in place any cryptographic keys required to provide the requested services

- RFC 4301 lists the following services:
  - Access control
  - Connectionless integrity
  - Data origin authentication
  - Rejection of replayed packets (a form of partial sequence integrity)
  - Confidentiality (encryption)
  - Limited traffic flow confidentiality

IPSec Architecture Overview



Components of IPSec

i) Security Association Database

ii) Security Policy Database

iii) ESP: Encapsulating Security Payload

- Authentication, integrity, confidentiality

iv) AH: Authentication Header

- Authentication and integrity for IP header (not payload)

v) IKEv2: Internet Key Exchange

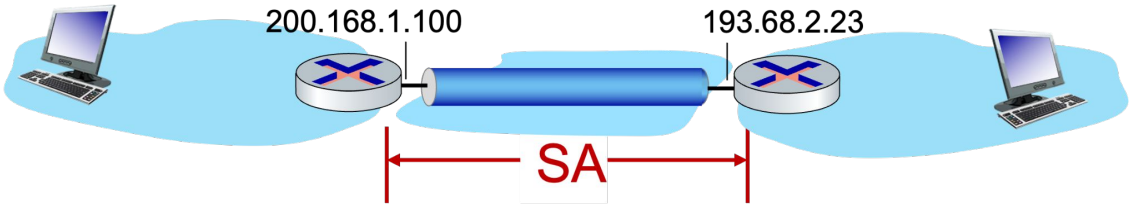
- AKE protocol

vi) Two different modes: tunnel and transport

- IPSec is highly complex!
- Some functionality is even duplicated

i) Security Association (SA)

- **Security Association** describes the concrete way how incoming/outgoing packets are going to be protected
- An SA is a negotiated outcome, with the negotiation based on the definition of the Security Policy
- SA describes simplex connection: one direction
  - Need two SAs to describe bi-directional communication
  - Total of four SAs: two per party and two per mode (ESP and AH)
- Stored in the Security Association Database



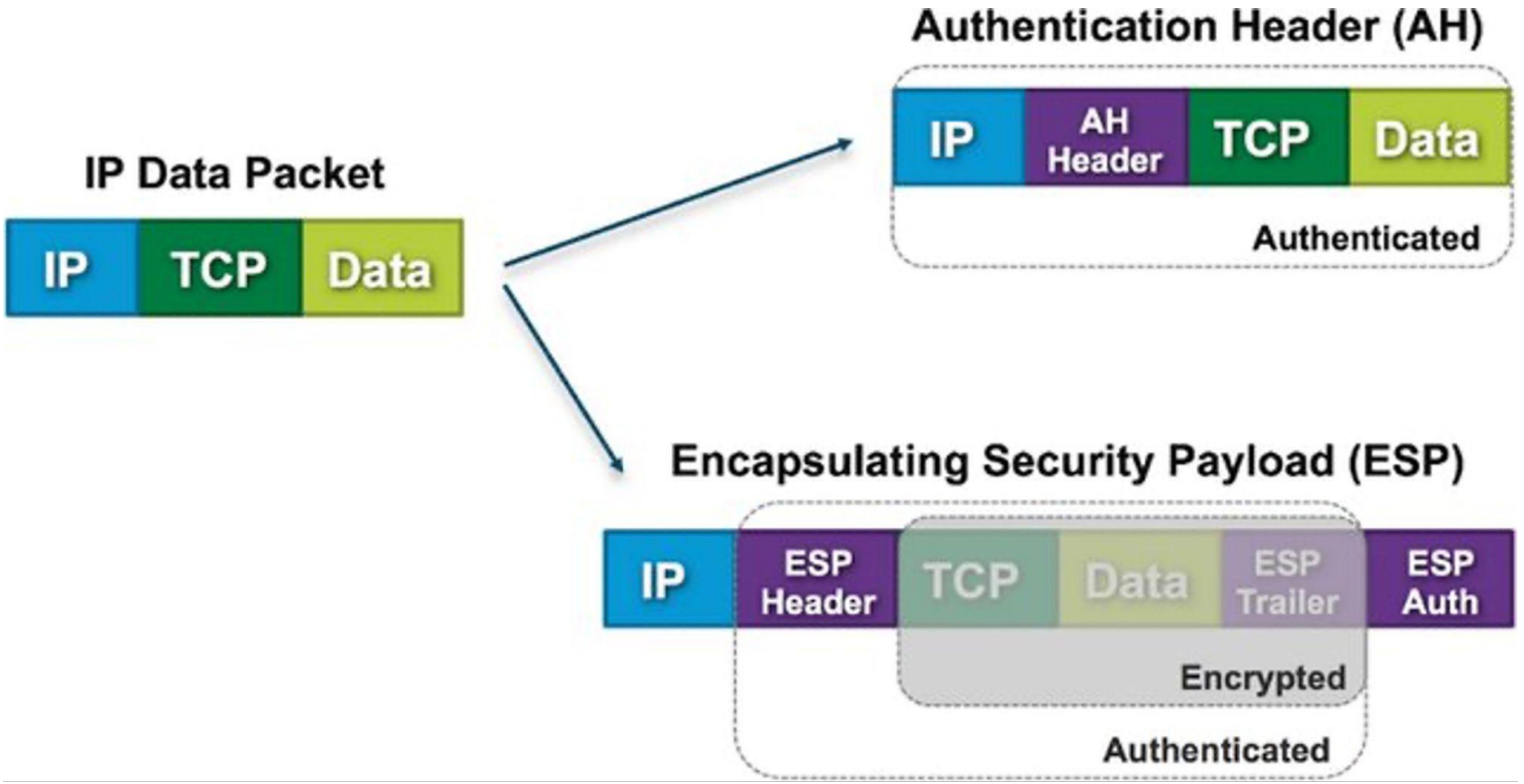
i) Security Policy Database

- Defines the parameters associated with each SA
- Normally defined by the following parameters in a SAD entry:
  - Security parameter index
  - Sequence number counter
  - Sequence counter overflow
  - Anti-replay window
  - AH information
  - ESP information
  - Lifetime of this security association
  - IPsec protocol mode
  - Path MTU

Host SPD Example

Protocol	Local IP	Port	Remote IP	Port	Action	Comment
UDP	1.2.3.101	500	*	500	BYPASS	IKE
ICMP	1.2.3.101	*	*	*	BYPASS	Error messages
*	1.2.3.101	*	1.2.3.0/24	*	PROTECT: ESP intransport-mode	Encrypt intranet traffic
TCP	1.2.3.101	*	1.2.4.10	80	PROTECT: ESP intransport-mode	Encrypt to server
TCP	1.2.3.101	*	1.2.4.10	443	BYPASS	TLS: avoid double encryption
*	1.2.3.101	*	1.2.4.0/24	*	DISCARD	Others in DMZ
*	1.2.3.101	*	*	*	BYPASS	Internet

ESP vs. AH


Authentication Header (AH)
IP
AH Header
TCP
Data
Authenticated
IP Data Packet
IP
TCP
Data
Encapsulating Security Payload (ESP)
IP
ESP Header
TCP
Data
ESP Trailer
ESP Auth
Encrypted
Authenticated

iii) Authentication Header (AH)

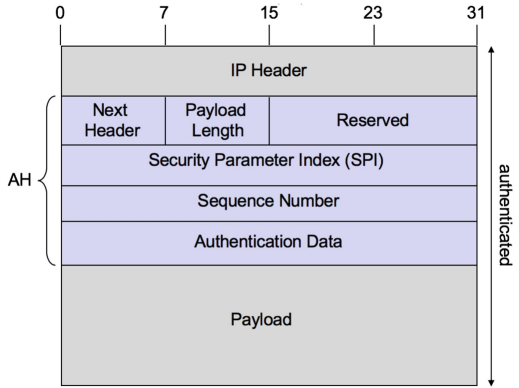


Figure: Use of Authentication header.

- AH **authenticates** parts of the IP header and the payload.
  - Reason: parts of the IP header change as the packet is being forwarded (e.g., Time-to-live)
  - Authenticated parts are the ‘normally immutable’ ones: source and destination IP, version, length, etc

iv) Encapsulating Security Payload (ESP)

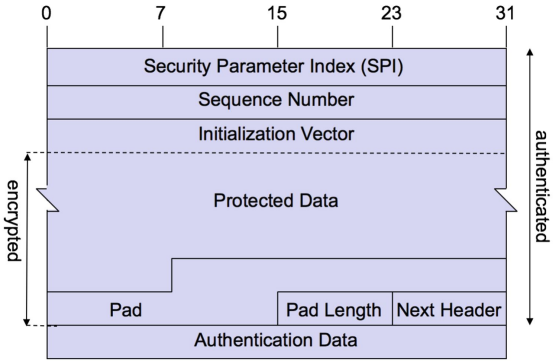


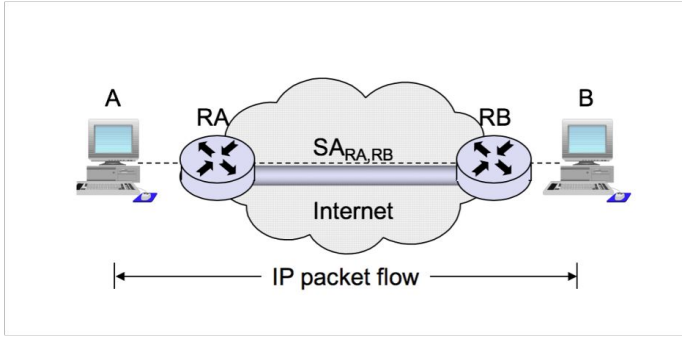
Figure: ESP header, protected data, trailer

- **ESP** can be used stand-alone or combined with AH
  - ESP adds another header (after IP header and possibly AH header), and a trailer after the protected data
  - We omit discussion of the particular fields

v) IKEv2

- Provides mutual authentication and key establishment between two parties
  - Negotiates cryptographic parameters: ciphers, hash functions, etc.
- Results in a total of four so-called **Security Associations (SA)**
  - One SA per party and direction
- The SAs created with IKEv2 are then used by AH and/or ESP
- The SAs are stored in the Security Association Database

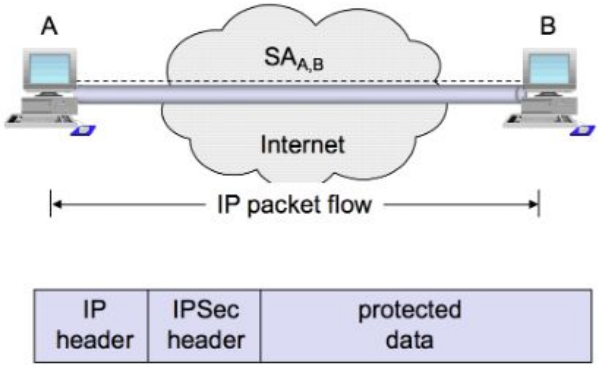
vi) a. IPSec Tunnel Mode



Protocol \ Mode	Transport	Tunnel
AH	IP AH Data	IP AH IP Data
ESP	IP ESP Data ESP-T	IP ESP IP Data ESP-T
AH-ESP	IP AH ESP Data ESP-T	IP AH ESP IP Data ESP-T

- **Tunnel mode** is used when the **gateways** (routers of a network) are responsible for adding the cryptographic protection of IPSec to the IP packets their hosts are sending
- Tunnel mode encapsulates IP packets: the entire IP packet is encrypted and authenticated and placed inside a new IP packet plus IPSec header.

vii) b. IPSec Transport Mode



- **Transport Mode** is used when the endpoint of IP communication is the same principal that also adds the protection
- Transport mode adds a header (and trailer in case of ESP) that protects the (normal, non-encapsulated) payload
- Transport Mode authenticates and encrypts only the payload. **The IP header is only authenticated, not encrypted.**

Recap

- **We discussed:**
  - Networking basics
  - OAuth
  - Transport Layer Security (TLS)
  - IPSec