

Chapter 1

L12 – Network System Calls & the Internet

1.1 From End-Systems to Processes

Every computer attached to the Internet that exchanges messages is called an **end-system**.

- **Client process**: issues *requests*.
- **Server process**: generates *responses*.

Thus, the words *client* and *server* are used both for the *processes* and for the *machines* executing them.

1.1.1 Naming a Process

A process is uniquely identified by the pair

IP address : port number

IP address (e.g. 8.8.8.8) – identifies the end-system.

Port number (e.g. 53) – identifies the process on that end-system. Certain services have *well-known ports* (DNS uses port 53).

Some definitions

End-system A host connected to the Internet that can send/receive packets.

Socket A file descriptor returned by `socket()`; used with `sendto()` and `recvfrom()` instead of `read()` and `write()`.

Well-known port A globally agreed port number reserved for a specific application-level protocol (e.g. port 53 for DNS).

1.1.2 Network System Calls

The following POSIX calls allow a user process to exchange messages with a remote peer:

Server-side calls

1. `socket()` – create a network endpoint and obtain a **socket**, a special file descriptor.
2. `bind()` – register the local process name [IP, port] with the kernel (e.g. [8.8.8.8,53]).
3. `recvfrom()` – retrieve an arriving request; the kernel fills in the client’s [IP, port].
4. `sendto()` – transmit a response back to the requesting client.
5. `close()` – release the socket when no more requests should be served.

Client-side calls

1. `socket()` – create a socket (no fixed port required).
2. `sendto()` – hand the kernel the request packet and the server’s address (e.g. [8.8.8.8,53]).
3. `recvfrom()` – wait for (or poll for) the matching response.
4. `close()` – terminate communication once all responses are received.

Blocking vs. Non-blocking Each call (most notably `recvfrom()`) may be invoked in **blocking** mode (the kernel puts the process to sleep until data arrives) or **non-blocking** mode (the call returns immediately with an error code such as `EAGAIN` if no data is ready).

1.1.3 Example Network Flow

Client → Server (Request)



```

18.6.2.5: 57050
client
●

const int s = socket(...);
...bind(s, [8.8.8.8,53], ...)...

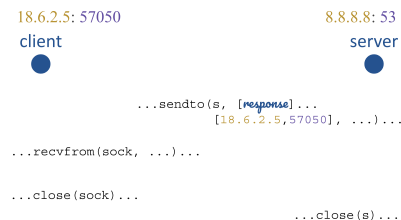
const int sock = socket(...);
...sendto(sock, [request], ...
          [8.8.8.8,53], ...)...

...recvfrom(s, ...)..

```

1. Client issues `socket()` then `sendto()` (*request*, [8.8.8.8,53]).
2. Kernel consults its routing tables and places the UDP datagram on the wire.
3. Server’s NIC receives the packet; the kernel matches [8.8.8.8,53] to the waiting socket produced by `bind()`.
4. Server process unblocks from `recvfrom()` and obtains both the request data and the client’s return address.

Server → Client (Response)



```

18.6.2.5: 57050
client
●

8.8.8.8: 53
server
●

...sendto(s, [response], ...
          [18.6.2.5,57050], ...)...

...recvfrom(sock, ...)...

...close(sock)...

...close(s)...

```

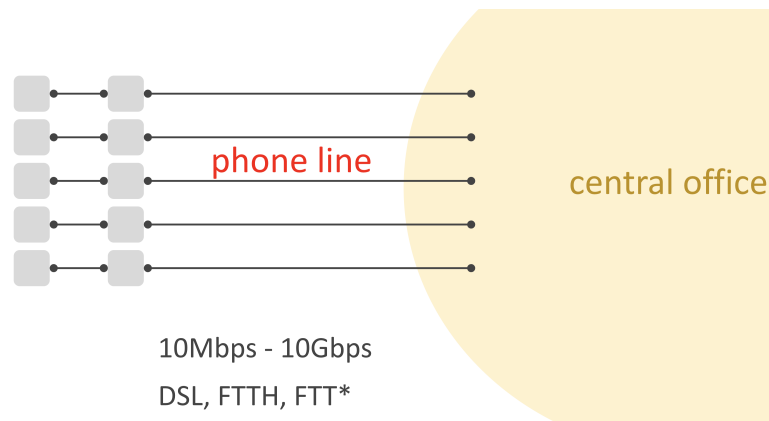
1. Server calls `sendto()` (*response*, [18.6.2.5,57050]).
2. Packet traverses the network back to the client.
3. Client’s kernel delivers the datagram to the waiting socket; `recvfrom()` returns the response.
4. Client invokes `close()`. (Whether any data is sent upon `close()` depends on the socket type—topic for a later lecture.)
5. Server eventually calls `close()` when it no longer wishes to serve further requests.

1.2 Internet Components

This section introduces the most common ways an **end-system** (e.g. your laptop) reaches the global Internet. For every access technology we list the physical path, the achievable data rate, and the main engineering trade-offs.

1.2.1 Access via the Public Switched Telephone Network

A typical home setup places the laptop behind a *DSL/Fiber modem-router* that terminates the household telephone line and relays traffic to the *central office* a few kilometres away.



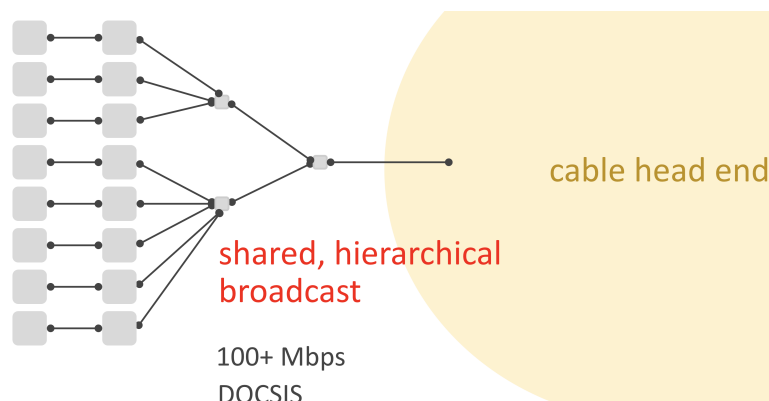
1. **Path:** Laptop → Modem/Router → Copper / Fiber local loop → Central Office → Internet.
2. **Rate:** 10–10,000 Mbps (material and DSL/Fiber technology dependent).
3. **Why legacy copper?**
 - Incumbent telcos *already own* a wire to every household — a compelling business advantage.
 - Pulling new fibre is capital-intensive; upgrades therefore proceed gradually (*FTTH*, *FTTB*, *FTTC*, ...).

End-system Any device exchanging packets over the Internet.

Modem A specialised *packet switch* that adapts IP packets to the physical characteristics of the telephone line.

1.2.2 Access via the Cable TV Network

A second option reuses the coaxial cable that once delivered only television signals.



1. **Path:** Laptop → Cable Modem/Router → Coaxial tree → *Cable Head-End* → Internet.

2. **Rate:** Typically 100–1,000 Mbps.

3. **Key differences to DSL:**

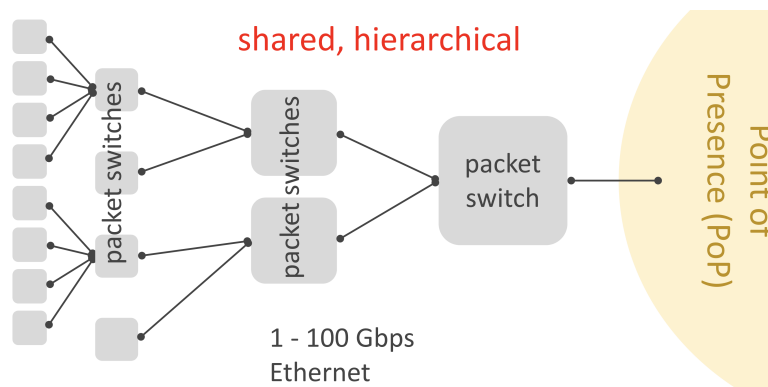
- (a) *Shared medium* — the last-mile coax is shared by all subscribers on the tree; throughput per user can drop at peak times (e.g. Saturday night).
- (b) *Broadcast medium* — frames sent by the head-end are physically delivered to *every* household on the tree.

Shared medium Multiple end-systems contend for the same link.

Broadcast medium A single transmission is received by all nodes on the medium.

1.2.3 Access via a Point of Presence (PoP)

Enterprise campuses and data-centres aggregate thousands of machines through a switched hierarchy that ultimately connects to an **ISP Point of Presence (PoP)**.



1. **Path:** Workstation → Access switch → Aggregation/Core switch → PoP → Internet.
2. **Rate:** 1–100 Gbps per link inside the site.
3. **Hierarchy:** Many small switches feed fewer, higher-capacity switches, reducing cost while maintaining performance.

PoP The interface between a customer network (campus, enterprise, data-centre) and an Internet Service Provider.

Summary

- **Access technologies:** DSL/Fiber over phone lines, Cable-TV coax, cellular, satellite, campus PoP, ...
- **Ownership:** Each path segment is operated by an *Internet Service Provider (ISP)* such as a telco (Swisscom), a cable operator (UPC/Sunrise), or a research network (SWITCH).
- **Design heuristic:** When extending connectivity, always ask:
 1. What infrastructure already exists?
 2. Which media are users accustomed to paying for?

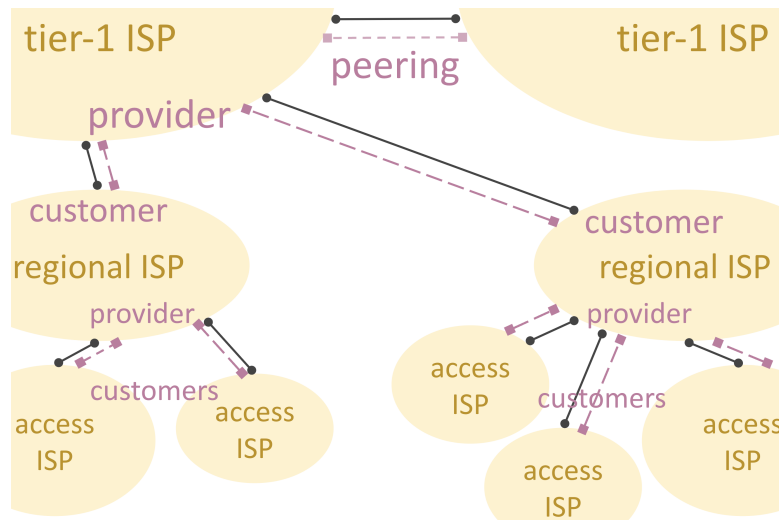
Leveraging existing assets often yields the most economical solution.

1.3 Internet Service Providers (ISPs)

Every end-system ultimately reaches the global Internet through one or more **Internet Service Providers (ISPs)**. To scale worldwide connectivity, ISPs organise themselves in a *three-tier hierarchy* linked by specific business agreements.

1.3.1 Why a Hierarchy?

Directly wiring every pair of the world's access networks is infeasible (thousands of telcos, cable operators, universities, enterprises, ...). Instead, ISPs form a multi-level structure:



1. **Access ISPs** — interface with end-systems (e.g. Swisscom, Sunrise).
2. **Regional ISPs** — aggregate many access ISPs within a country or region.
3. **Tier-1 ISPs** — few, globe-spanning backbones that can reach every network without paying another provider.

Definition. *Customer-provider relationship:* the lower-tier ISP (customer) pays the higher-tier ISP (provider) for global reachability.

1.3.2 Peering Agreements

When two ISPs exchange large, roughly balanced traffic volumes, they may bypass providers and interconnect as **peers**:

- *Settlement-free peering:* each party carries the other's traffic at no cost.
- *Paid peering:* one ISP compensates the other if traffic is highly asymmetric.

Peering can occur at *any* tier (access ↔ access, regional ↔ regional, tier-1 ↔ tier-1) whenever it is economically attractive.

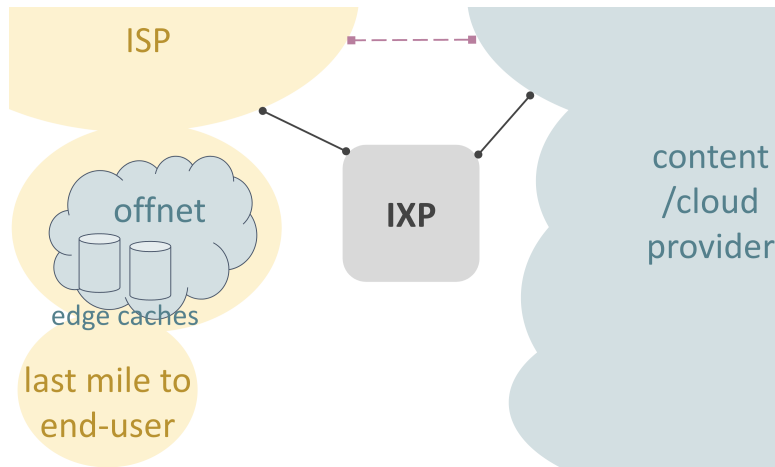
1.3.3 Internet eXchange Points (IXPs)

ISPs rarely drag a private cable to every partner. Instead, they each attach *one* high-speed link to a neutral facility called an **IXP**—essentially a massive Ethernet switch where multiple ISPs exchange traffic.

IXP Neutral switching fabric that enables many bilateral or multilateral interconnections through a single physical port.

1.3.4 Content/Cloud Providers as Networks

Companies such as Google, Microsoft, Amazon, and Meta evolved from regular customers of ISPs into operators of near-global backbones:



1. Directly connect their data-centres to large ISPs and IXPs.
2. Peer with ISPs to minimise transit cost and latency.
3. Deploy **edge caches** *inside* ISP networks to store popular content close to users.
4. Interconnect those caches via an **off-net**—a private overlay that is *logically* theirs but *physically* located within partner ISPs.

1.3.5 Edge Caches and Off-nets

Definition. *Edge cache:* server cluster owned and managed by the content provider but installed within the ISP's premises; stores the most popular objects for that ISP's subscribers.

Definition. *Off-net:* the private backbone interconnecting multiple edge caches that reside outside the content provider's core network.

This arrangement improves user experience while reducing upstream traffic for the ISP.

Recap

1. The Internet scales through a three-tier ISP hierarchy (access–regional–tier-1) governed by customer–provider contracts.
2. Peering offers a cost-effective shortcut whenever traffic volumes justify it.
3. IXPs simplify physical connectivity by acting as neutral switching hubs.
4. Large content/cloud providers now run quasi-global networks, peer with ISPs, and deploy edge caches/off-nets for performance.

1.4 The Network Interface and the CPU

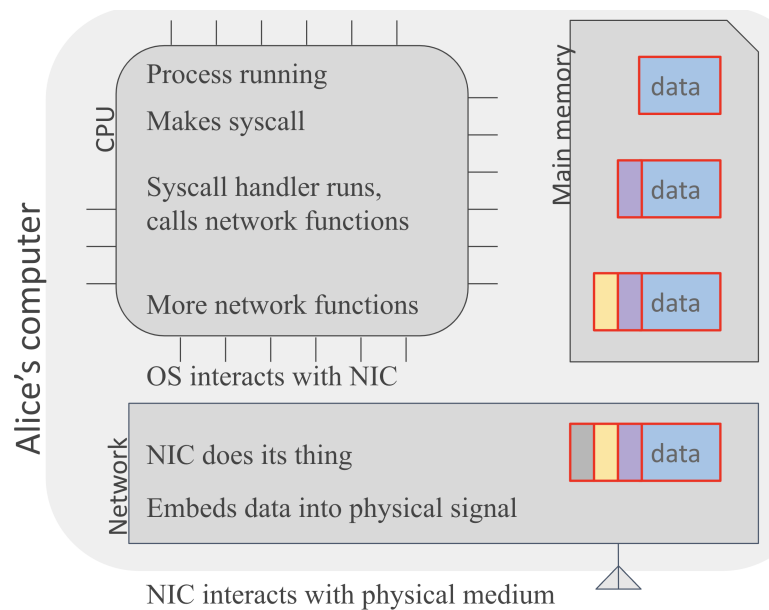
Every Internet message created by a user process must travel from *main memory* to the *Network Interface Card (NIC)*, descend through the five protocol layers, and finally leave the machine as a physical signal. This section shows how the operating system, the NIC, and the layered “network stack” cooperate to make that happen.

1.4.1 Hardware Overview

NIC controller On-board processor that handles packet queues and offloads work from the CPU.

I/O controller CPU’s gateway to peripheral devices.

DMA controller Copies data between main memory and device memory *without* CPU intervention.



Send path (user process → wire):

1. User process issues a `sendto()` *network syscall*; CPU traps into *kernel mode*.
2. Kernel prepends *transport* and *network* metadata to the user data.
3. Kernel programs the I/O controller; DMA copies the buffer to NIC memory.
4. NIC adds a *link-layer* header, forming a **packet** = *payload* + *all headers*.
5. NIC converts the packet bits into an electrical/optical/radio signal and transmits it onto the medium.

1.4.2 The Five-Layer Internet Stack

Application User programs (web, chat, file-transfer).

Transport TCP and UDP.

Network IP (Internet Protocol).

Link Ethernet, Wi-Fi, DOCSIS, DSL, ...

Physical Copper wire, fibre optics, radio, free-space optics.

Layer Properties

application	web	BitTorrent	DNS	...
transport		TCP	UDP	
network			IP	
link	DSL	FTTH	DOCSIS	Ethernet ...
physical	copper	fiber optics	radio waves	...

- **Abstraction:** each layer hides lower-level details from the layer above.
- **Encapsulation:** a layer *prepends* its own header.
- **Decapsulation:** a layer *removes* its header on the receiving host.
- **Interfaces:** adjacent layers communicate through a well-defined API (e.g. a *syscall* between Application and Transport).

Layers reduce complexity and increase flexibility; they do not directly improve raw performance.

Example 1.4.2.1 (EPFL Web Server). Consider the machine that answers HTTP requests for *www.epfl.ch*. Important naming facts:

EPFL web server	DNS name <i>www.epfl.ch</i> <i>www.epfl.ch</i>	
	process name <i>www.epfl.ch, 80</i> <i>104.20.228.42, 80</i>	
	port number <i>80</i>	
	local intf name <i>en0</i>	IP address <i>104.20.228.42</i> <i>104.20.229.42</i>
		MAC address <i>5c:f9:38:a4:00:76</i>
NIC		

1. **Application layer** - Runs a web-server process.

2. **Transport layer**

- Identifies that process with **port 80**.
- Other processes on the same host listen on different port numbers.

3. **Network layer**

- Sees the host's network interface as *en0*.
- Associates one or more **IP addresses** with *en0* (e.g. *104.20.228.42*).

4. **Link layer**

- Uses a globally-unique **MAC address** (e.g. *5c:f9:38:a4:00:76*) for the same interface.

Name Identifiers

- **Interface identifiers**

DNS name Human-readable alias (*www.epfl.ch*).

IP address Network-layer locator (*104.20.228.42*).

MAC address Link-layer locator (*5c:f9:38:a4:00:76*).

OS handle Local label (e.g. *en0*).

- **Process identifier** *interface-name*, *port*

A two-tuple: first choose the interface by DNS/IP, then the process by its port number.