Presentation layer

the **Presentation Layer** has nothing to do with networking and is ignored by most.

However, presentation has a major impact on the commercial success of application–layer software, so the Presentation Layer is very important, but not from the point of view of data transmission.

Check David Hill's paper for a propagandist's view.

Networking professionals tend to shun the term "presentation" and use **Layout engine** instead. Check wikipedia for a general overview or mozilla'a page for technical details.

Session layer

A **session** is a **persistent** virtual[a] link ("connection") of two application–layer processes.

The Session Layer is needed only if the transport layer is connection–oriented. If the transport layer is connectionless, there is no notion of a session (no circuit needed).

In Internet protocols, the session layer is not a separate layer, but is incorporated into an **API** (Application Program Interface), the software interface between the application layer and the transport layer.

[a] As opposed to **physical** which is dealt with in the PL.

Simple session

Suppose that an application $\mathcal{A}_A$ running on host $\mathcal{A}$ wishes to have a session with an application $\mathcal{B}_B$ on host $\mathcal{B}$.

The Session Layer software manages this session. It is invoked at least twice:

- When the session is to be established. SL creates a virtual circuit between $\mathcal{A} : A$ and $\mathcal{B} : B$ (many virtual circuits connecting $\mathcal{A}$ and $\mathcal{B}$ may coexist simultaneously).
  The creation of the virtual circuit, followed by a handshake between $\mathcal{A} : A$ and $\mathcal{B} : B$, starts the session.
- When an application wants to close a session ("**tear down**"). SL ensures that the resources allocated to the session are released.
- Additionally, SL may be invoked for management tasks, which include checkpointing the session flow, resynchronising a session, and many other.

Typically, SL is available as a software library made of of tools (methods, functions) that are invoked directly by the application software.

Among the many APIs, **Socket API**, **RPC API**, and **NetBIOS** are most popular, with sockets (UNIX and WinSOCK) dominating the software industry.

Conceptually an API is an interface between a user process and the OS kernel; applications have no other means to communicate with the Transport Layer because it is inside the kernel and is interrupt/signal/timer driven (user processes cannot use these resources).

Firewalls

A network firewall is system used to control data traffic flowing between two networks: a trusted network (owning the firewall) and an untrusted network (usually the rest of the world). Its purpose is to provide a barrier preventing unwanted or unauthorised traffic; this is accomplished by applying a set of dynamically–configured filters (or rules).

Firewalls can be placed within the Application layer, the Session layer or deeper within the stack (Transport, Network or even Data Link layer). **NAT** will be covered in due time.

Application–layer firewalls

An application gateway or proxy acts on behalf of a protected node: when an untrusted network sends a connection request to the protected node, the gateway/proxy captures the connection preventing the untrusted connection from reaching its destination directly.

Then the gateway opens another connection between itself and the protected node. Traffic from the untrusted network is filtered by the gateway in an application–dependent way.

A very similar method is used at the NL, protecting a whole subnet as opposed to a single application.

Session–layer firewalls

A circuit–level gateway monitors every virtual circuit passing through it, attempting to detect (and destroy) unauthorised packets coming inside the connection stream (these gateways are called stateful filters to distinguish them from packet–level filtering).