Stalking the
Upper Layer Architecture

"You have Telnet and FTP. What else you do you need?"
- Alex McKenzie, 1975

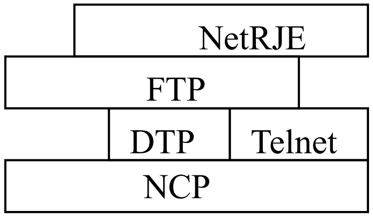
"If the Virtual Terminal is in the Presentation Layer,
where is the power supply?"
-Al Reska, 1979

John Day

October 2022

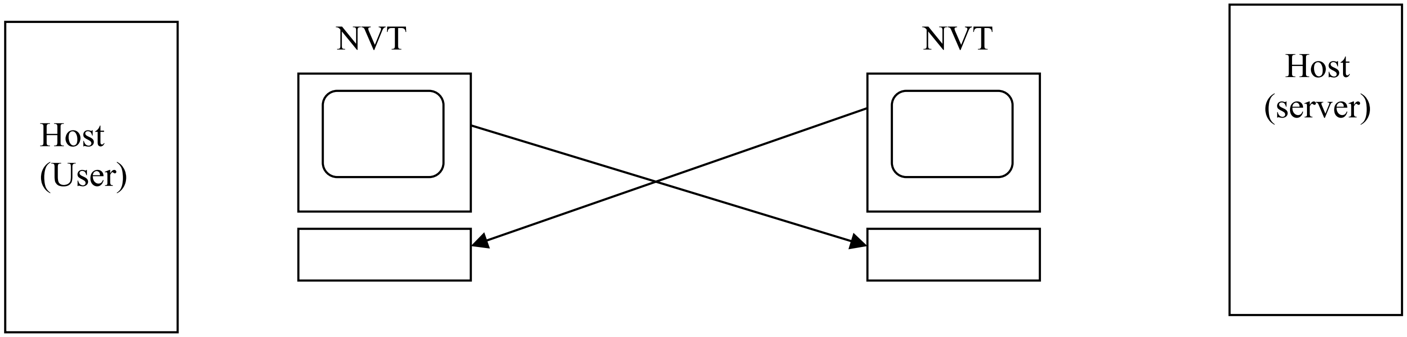
The Early Upper Layers

- Once the idea of a packet-switched data network was proposed,
  - Someone had to show that it was good for something as well (and quickly).
- The immediate answer was all the things one did with a computer:
  - Remote terminal access, Telnet
  - File Transfer, FTP
  - Remote Job Entry, RJE.
- The early model was to mimic Operating System functions in a network.
- And these were pretty good building blocks for other things.
- This led to an early upper layer architecture with some innovations and some lessons.



Telnet

An Innovative Model

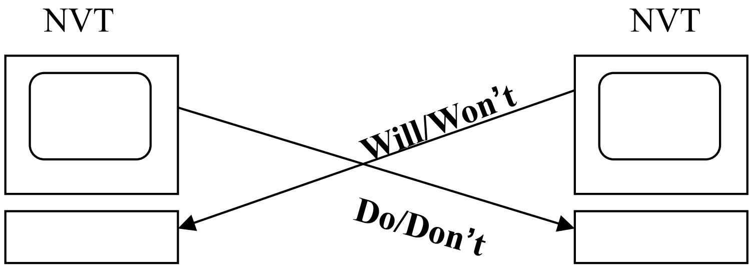


- Everyone else saw remote terminal support as remote login, asymmetrical:
  - Dumb Terminal to Smart Host
    - Skipping steps again
- But Bernie Cosell of BBN saw it as a *terminal driver* protocol and *symmetrical*.
  - Yields a character-oriented IPC facility that can be built upon.
  - It is never easy to build on something asymmetrical.
- Based not on minimal capability but on a canonical form.
  - The Network Virtual Terminal (NVT)
  - To solve the "m x n" problem.
- Stream-oriented
  - But shouldn't have been

For the Record: The design
was Bernie Cosell's idea!

Telnet: 2

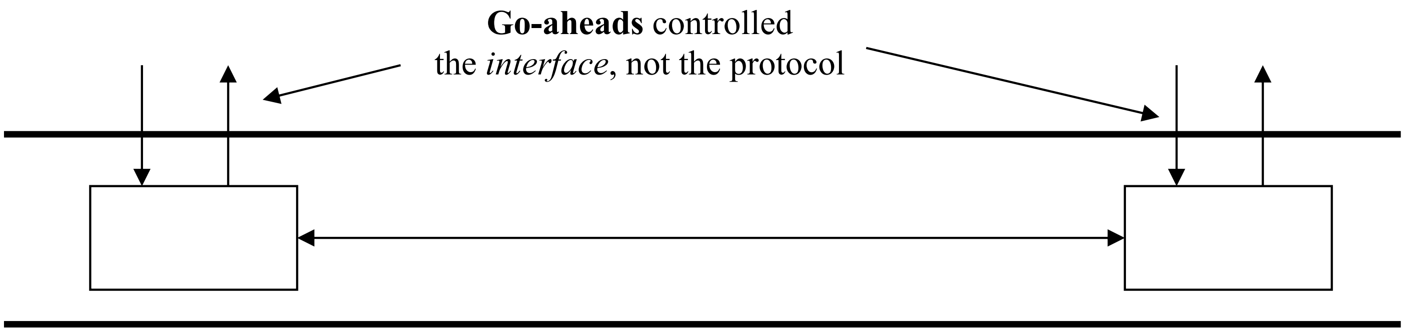
Symmetrical Negotiation



- Do/Don't, Will/Won't determine which side will perform which functions.
- A symmetrical negotiation, where each side tells the other what will or won't do and what it expects the other side to do or not do.
- One side's announcement is the other side's acknowledgement
  - with rules for resolving conflicts.
- When there is agreement, it is a Degenerate Case: the request *is* the response

Telnet: 3

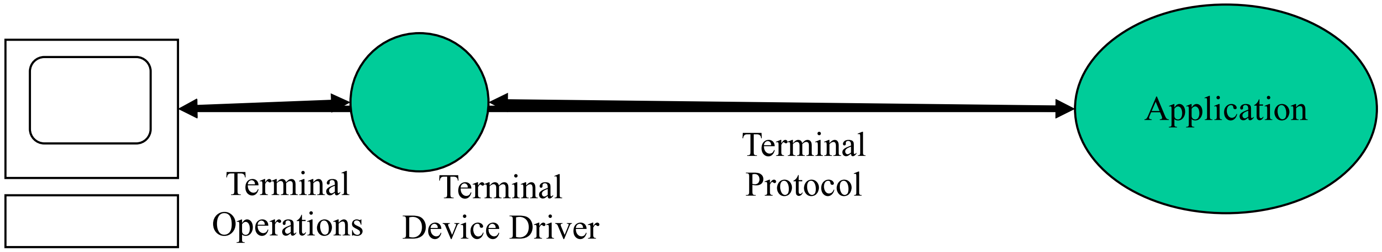
Half/Full Duplex



Go-aheads controlled the *interface*, not the protocol

- Most saw supporting half-duplex terminals as requiring the *protocol* to be half-duplex.
- Telnet designers realized that it only required blocking at the *interface*.
  - This allowed full and half duplex interworking, both are degenerate cases.
  - Go-aheads are inserted by the host and ignored if you don’t need them.
- While not important today, the insights are important and can be used elsewhere.

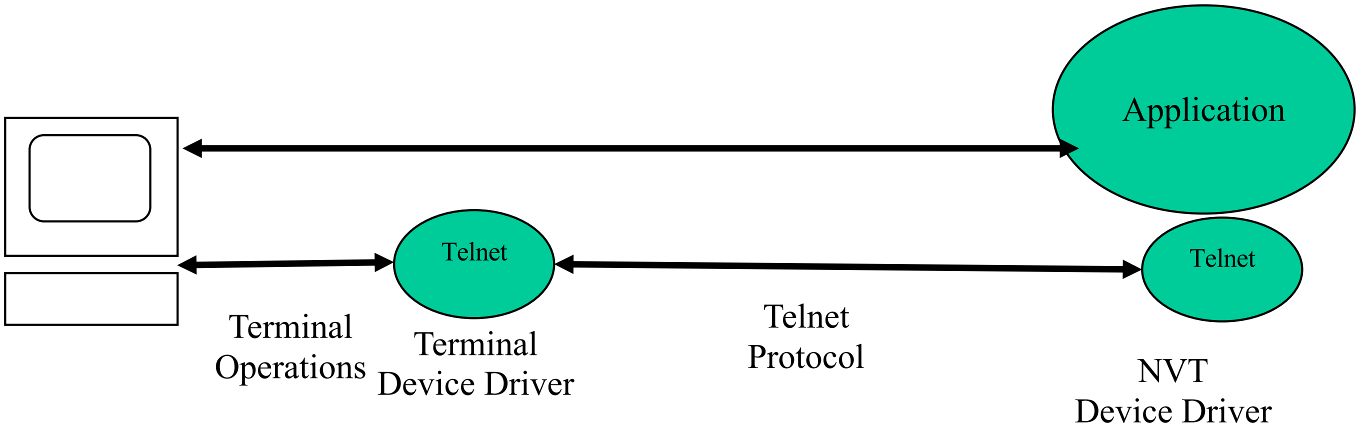
Telnet as a Case Study in Design: I



- Notice how most everyone saw this as terminal talking to a computer or at most an application.
- This couldn't be the case: The terminal didn't have a processor. It couldn't process a terminal protocol.
- There had to be a process in between that controlled the terminal.
- They knew that but they continued to think about it as if it wasn't there.
  - (Skipping steps again.)

Telnet as a Case Study in Design: II

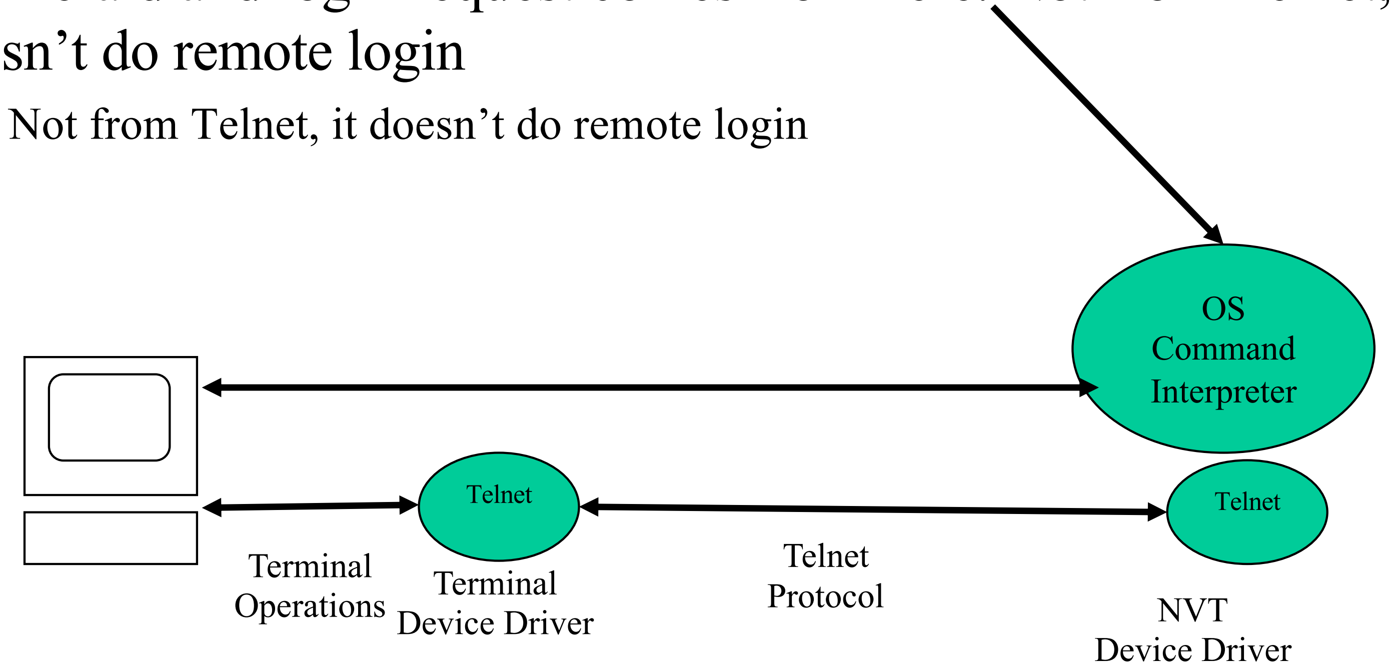
- What Bernie saw was that the picture was really this:



- This was much more versatile and much to his surprise was immediately adopted as New Telnet and put on Socket 23 for testing.
  - At the time, some systems tied terminals to specific applications. This worked for that too.
- Most people were too focused on the original picture and didn't realize that in this case the detail was important.
- Consequently,

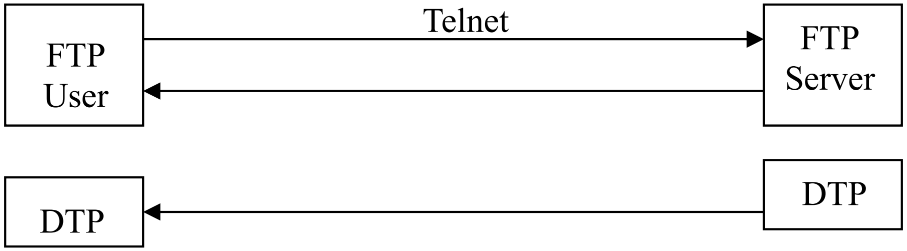
Telnet as a Case Study in Design: III

- When you use Telnet on Socket 23, you are doing this
- The herald and login request comes from here. Not from Telnet, it doesn’t do remote login
  - Not from Telnet, it doesn’t do remote login



FTP

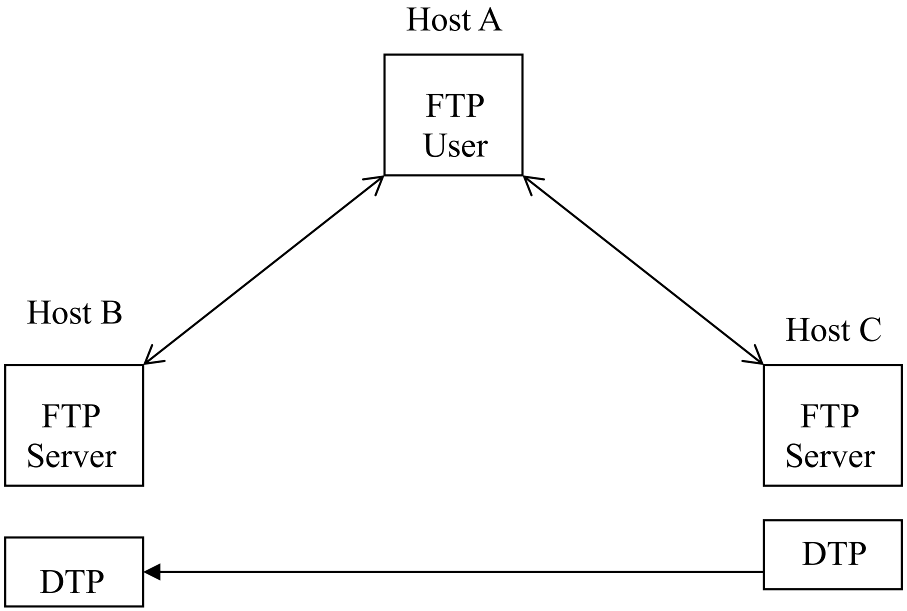
Not everything is Symmetrical



- Built on Telnet with a rudimentary virtual file system model
  - This allows *You* to be the User FTP Process*
- Separate data and control connections
- Defined a Network Virtual File System
- Supported checkpoints and 3-way transfers.
- Mail starts out as two FTP commands

*This is the beginning of the insanity that everything should be encoded in readable characters.

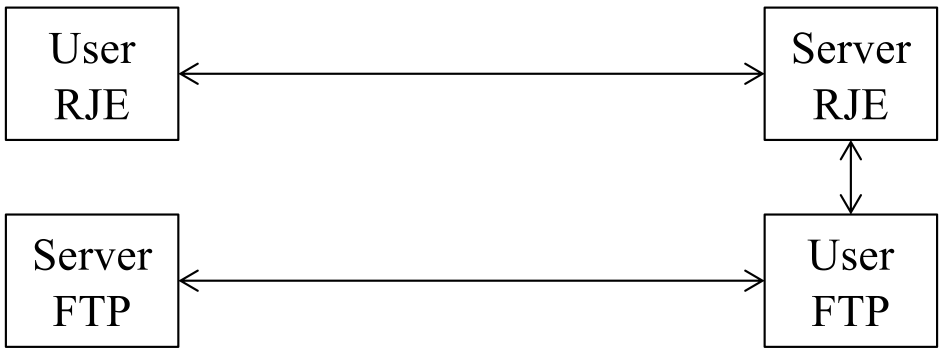
How a 3-Way Transfer Works



- Host A telnets to hosts B and C.
- Host A sends B, C’s transfer SOCKet; and C, B’s transfer SOCKet.
- Host A tells B to listen, to be PASiVe.
- Then A tells C to RETRieve and tells B to STORe.

RJE

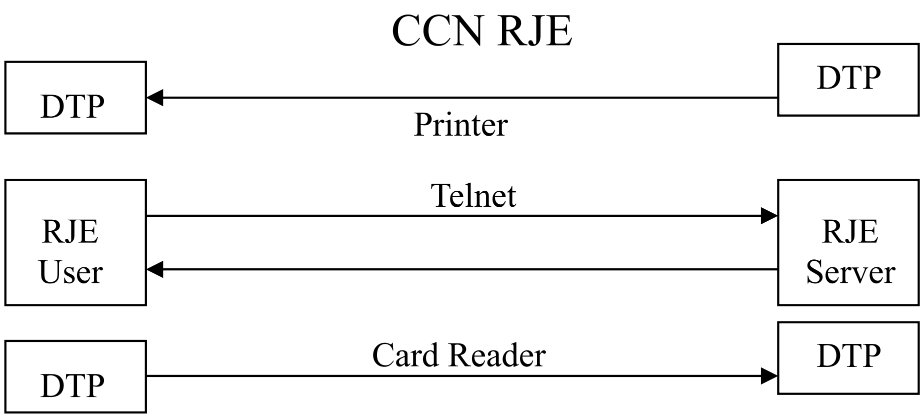
(A step too far)



- The Problem Remotely Submit a Job for Execution.
- The Solution Looks Obvious.
  - Need to retrieve the Input for the job (a file) and store the output of the job (a file)
  - Use FTP!
- Bad Idea!
  - Elegance lost sight of the purpose: provide processing resources to very processing constrained clients.
  - Greatest burden is put on the client.
  - If it could handle the load, it wouldn’t need RJE.

RJE

(A More Pragmatic Solution)



- CCN RJE did a better job of addressing the need by keeping the user side resource requirements small.

As with Any First Attempt

- Some innovative and truly insightful developments. Important Lessons learned:
  - The need for Application Names, not Well-Known Sockets (now ports)
  - A single canonical syntax is too restrictive and leads to unnecessary inefficiency
  - Syntax translation must be aware of semantic invariants.
  - Importance of separating control and data
  - Importance of decoupled symmetrical models of protocols
  - Functions tend to partition "vertically," not horizontally.
- There were also a few mistakes and compromises that should not have had a long life. Some died quickly; others are still around.
  - The limitations of the TIP created less than proper conventions that we still live with in FTP and SMTP: 4 character commands and replies in text, the PORT command, etc.
- We thought Telnet was good, but it took 2 tries. We needed another round on the others, and there was much more to do. (Contrary to what Alex thought!)
  - Graphics Directory Resource Sharing Editors
  - Distributed Data Bases Voice etc.

But It Worked and Worked Well!

- Beyond our wildest imaginations! And quickly became indispensable.
- As with any new idea, the success of these early applications lead to many things being tried and considered almost immediately.
  - Voice, wireless, remote database, file access, cross-net debugging, remote development, telecommuting, personal computers with a mouse and keyset, a network editor, file sharing, even a variation on the web.
  - All before 1974.
- Plans were being made for much more elaborate applications for resource sharing. The upper layers were unexplored territory, but there were several intriguing projects in the works on distributed databases, distributed operating systems, etc.
- We knew we didn't have an upper layer architecture, but we were confident that developing some of these projects would lead to one.

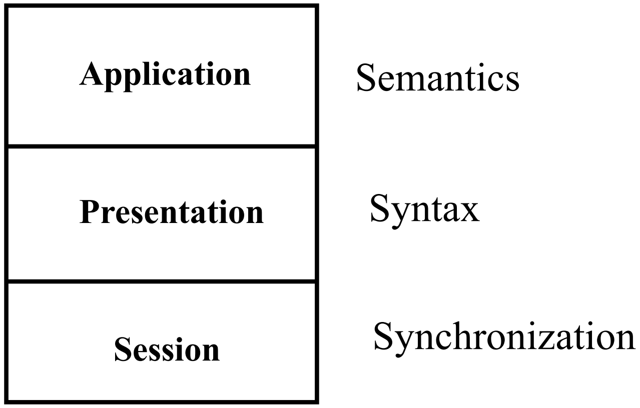
A Screeching Halt

- In the early 1974, ARPA stopped all upper layer development. And for the Internet, that is pretty much where it stayed for 20 years.
  - Hafner says that ARPA feared the “activist” approach taken by the participants would lead them to “losing control of the network.”
- The failure to develop applications removed the driver that would have forced finishing the addressing architecture, creating a true directory system, better security, richer mechanisms for distributed applications, etc. and many of today’s problems, mobility, multihoming, spam, viruses, etc. would be non-existent or nearly so.
  - Also, the emphasis on Resource Sharing disappeared
- By never re-visiting a decision, it was ensured that the system that the world now relies on was never productized and is an uncompleted demo.

Upper Layer Development Shifts to OSI

- After consideration of the upper layers stopped in the Internet, the only major new investigation occurred in the OSI debates beginning in 1978. Upper Layer work starts around 1981.
- Charles Bachman, creator of the Entity-Relation Database Model, had a keen interest in the upper layers when he proposed the model that became the OSI 7 Layer Model
  - There was a general feeling that the lower layers were reasonably well understood, but there were few ideas about the upper layers.
    - The 'official' reason given was Bachman's model was chosen because it looked reasonable and it was felt that it had enough flexibility to accommodate whatever might be discovered.
      - And it didn't have the same number of layers as IBM's SNA!
      - The Real Reason: It was the CYCLADES model
    - Although it did seem to have some disturbing asymmetries embedded in it.
  - OSI was a highly charged political environment between the computer and PTT camps.
    - Very quickly the Session Layer was hijacked for CCITT's Videotex, e.g., the French Minitel.
  - Initially, OSI had no new applications beyond more elaborate versions of the common 3.
    - This seems to be as much because the 80's were a period of consolidation as the newness of the ideas and the hardware limitations conspired that no new applications were proposed anywhere.

The Early View of Upper Layers



Application — Semantics

Presentation — Syntax

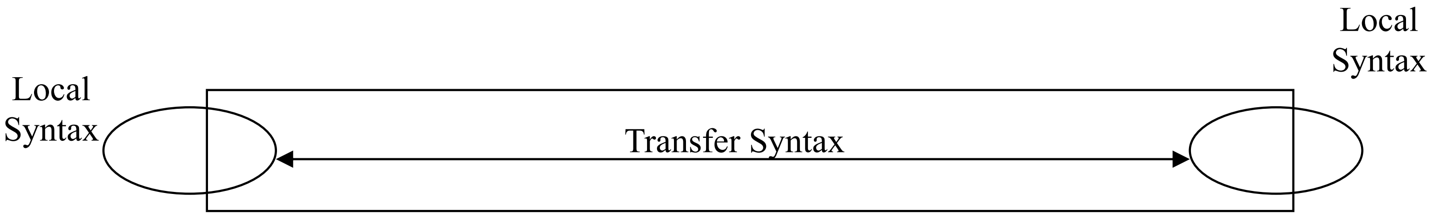
Session — Synchronization

- Seems plausible. But separating Syntax and Semantics has always proved difficult.

OSI Session Layer

- Really has nothing to do with Sessions
  - Stolen by the PTTs for Videotex and Teletex
- Supposedly a tool-kit of primitives:
  - Major/Minor Synch
  - Resynch
  - Activity Management
  - Half/Full Duplex (Europeans didn't realize it didn't need to be in the protocol.)
- Sessions were really established at the Application Layer.
- This is going to cause trouble.
  - Not doing what the problem says always does.
    - You can pay the piper now or pay a lot more later! ;-)

The Presentation Layer
Negotiates the Transfer Syntax



- The Presentation Layer negotiates the Abstract Syntax and Concrete Syntax (Encoding Rules) for the transfer.
  - Avoids requiring a canonical form with a technique with far greater implications
  - By Making Protocols invariant with respect to syntax.
- The actual translation must be done in the Application Layer.
    - Not much of a layer.
- Presentation does NOT do compression or encryption.

Abstract and Concrete Syntax

- An Abstract Syntax Language can compile to
- Potentially, many Concrete Syntaxes or Encoding Rules as the actual bits generated
- Analogous to a Programming Language (abstract) with different code generators (concrete).
- Creates a protocol specification that is *invariant with respect to syntax.*

Programming Language		**Abstract Syntax**
< integer > ::= INTEGER <identifier>;	**Language Definition**	GeneralizedTime ::= [Universal 24] Implicit VisibleString

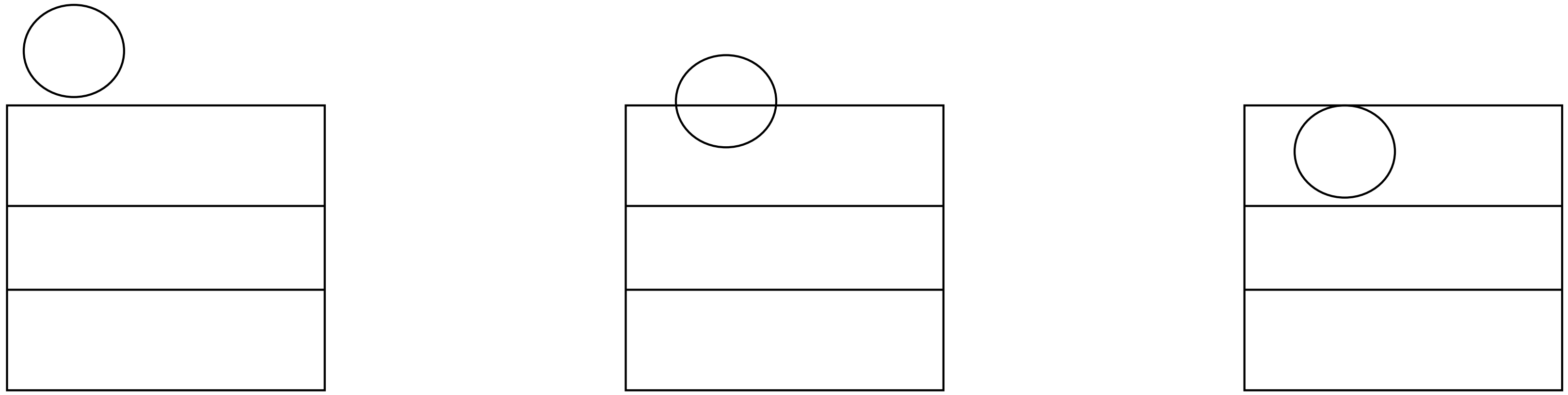
INTEGER X;	**Statement Definition**	EventTime ::= Set {
		[0] IMPLICIT GeneralizedTime Optional
		[1] IMPLICIT LocalTime Optional}

(32-bit word)	**Encoding**	I | L | GeneralizedTime
$012A_{16}$	**Value**	0203000142_{16}

Applications and Communication: I

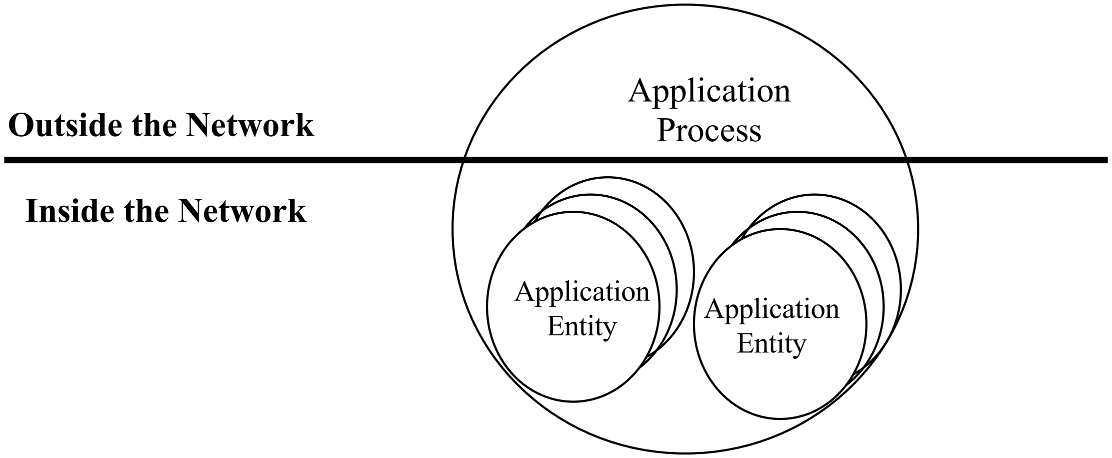
Is the Application in or out of the IPC environment?

- The early ARPANet/Internet didn’t worry too much about it. They didn’t need to. Only one FTP per system, only one remote login per system, etc.
- By 1985, OSI had tackled the problem, partly due to turf. Was the Application process inside or outside OSI?



- It wasn’t until the web came along that we had an example that in general an application protocol might be part of many applications and an application might have many application protocols.
- The only known case of a major technical insight coming from a political turf battle!

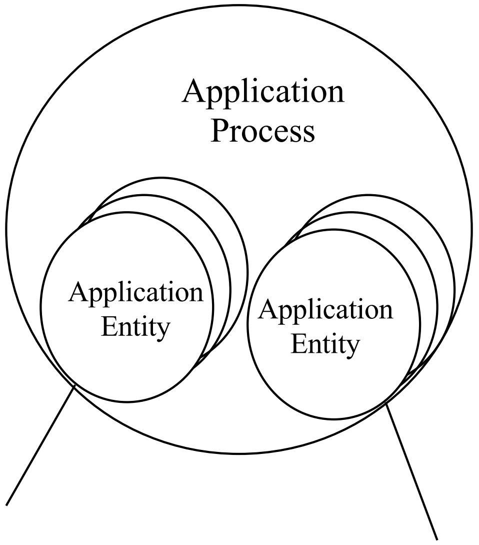
Applications and Communication: II



- The Application-Entity (AE) is that part of the application concerned with communication, i.e., shared state with its peer.
- The rest of the Application Process is concerned with the reason for the application in the first place.
- An Application Process may have multiple AEs, they assumed, for different application protocols.

Structure of an Application Process

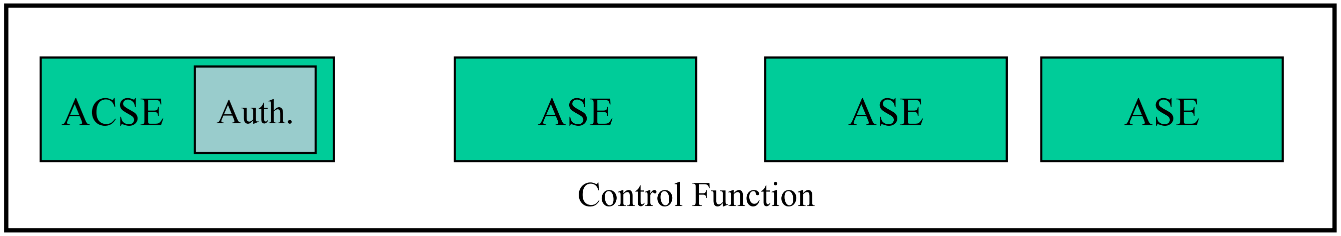
Important Insight



- An AP can have multiple AEs and multiple instances of multiple AEs as well as multiple instances of APs.
  - The web was the first example that used this distinction.
  - Consider a web application using both HTTP and a remote database protocol.
    - Two AEs
  - Multiple users of a social networking site: Multiple instances an AE, etc.

Application Entity Structure

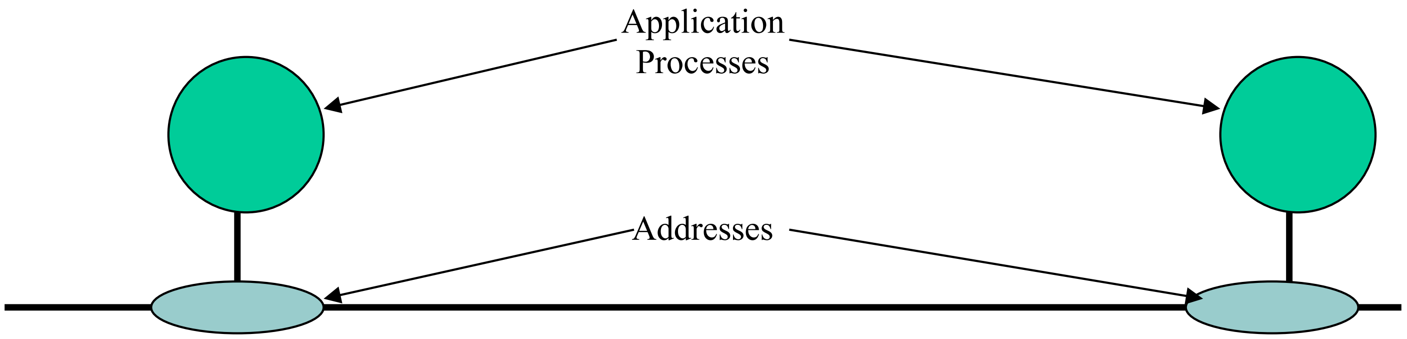
Modularity Is Good



- OSI took a modular approach to applications. An AE is composed of modules coordinated by a control function.
- One combined Application Service Elements, e.g., file transfer, checkpoint/recovery, etc. to form an application protocol.
- A common ASE was necessary to create the Application connection.
  - Application Control Service Element (ACSE) created the application connection.
  - It also had a plug-in place holder for authentication.

Why ACSE was Necessary

(a flaw in the architecture)



- Addresses were exposed at the layer boundary (bellhead think).
- To avoid the well-known port kludge. You have a choice:
  - Something on the other side has to recognize the first message for every application, or
  - A common message format that carries the name of the application.
- Why is it a flaw?
  - Go back to the IPC Model: Never Exposes addresses to the application.
    - (Can you find out the memory address of a variable in a Python program?)
  - The Internet has the same problem

New Results from the OSI Work

Despite the politics there were new insights

- The capability to negotiate the syntax of the application.
- The distinction between abstract and concrete syntax.
  - Makes protocols invariant with respect to syntax
- That a common mechanism was required for establishing application associations, e.g., common header, application names, authentication, etc.
  - But this is an artifact of a bug in the architecture
- That application protocols should be composed of components some of which were re-usable, e.g., the common establishment mechanism.
- Determining the nature of the boundary at the top of the architecture.
  - Distinguishing that part of the application process (AP) that dealt with communications, the application entity (AE), from the other aspects of the AP, e.g., processing, database access, etc.

But Problems Begin to Arise: I

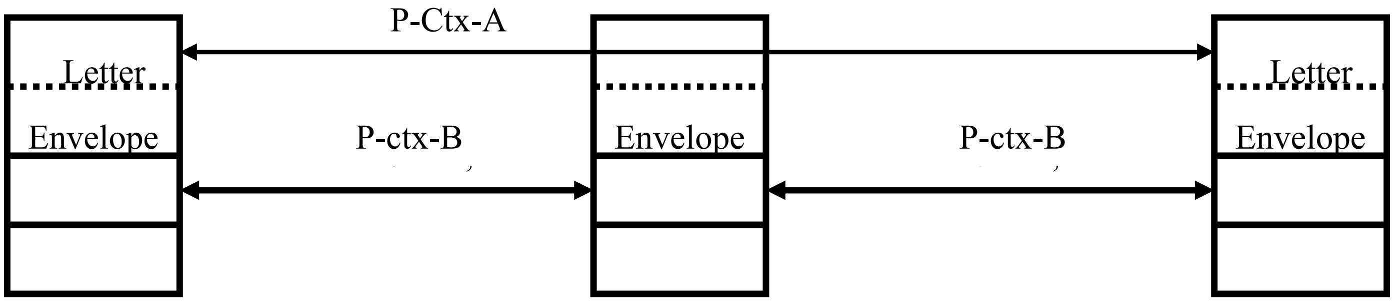
- Before OSI, no one had dug deeply into the nature of the upper layers.
- There were the 3 applications, and some others being proposed, but the sense was that there should be a general structure.
- Bachman's 7 layers had seemed reasonable, (and met political requirements).
  - There was a general consensus that there would be something called a Session Layer (see DECNET for example).
  - Presentation seemed to connote display on terminals,
    - Although, that wasn't Bachman's view.
  - On the other, Process-to-process did not seem to have anything to "present."
    - There was an asymmetry here that was troubling.
    - Perhaps a red-herring, but abstract-syntax/encoding rules avoids any problem.

But Problems Begin to Arise: II

- By 1983, it was realized that the upper 3 layers were not really layers in the same sense as the lower four.
  - We should have recognized at this point that scope was the difference.
- Steps were taken to align the 3 protocol specs, making judicious choices of defaults to make it possible to implement the upper three layers as a single layer.
  - The insightful implementor would see that the best strategy would be to implement it as one. (This became known as the “OSI clueless test”)
- However, by the late 80’s, the consequences of hijacking Session, in pursuit of short-term political agendas, and not “following the problem” began to show.

But Problems Began to Arise: III

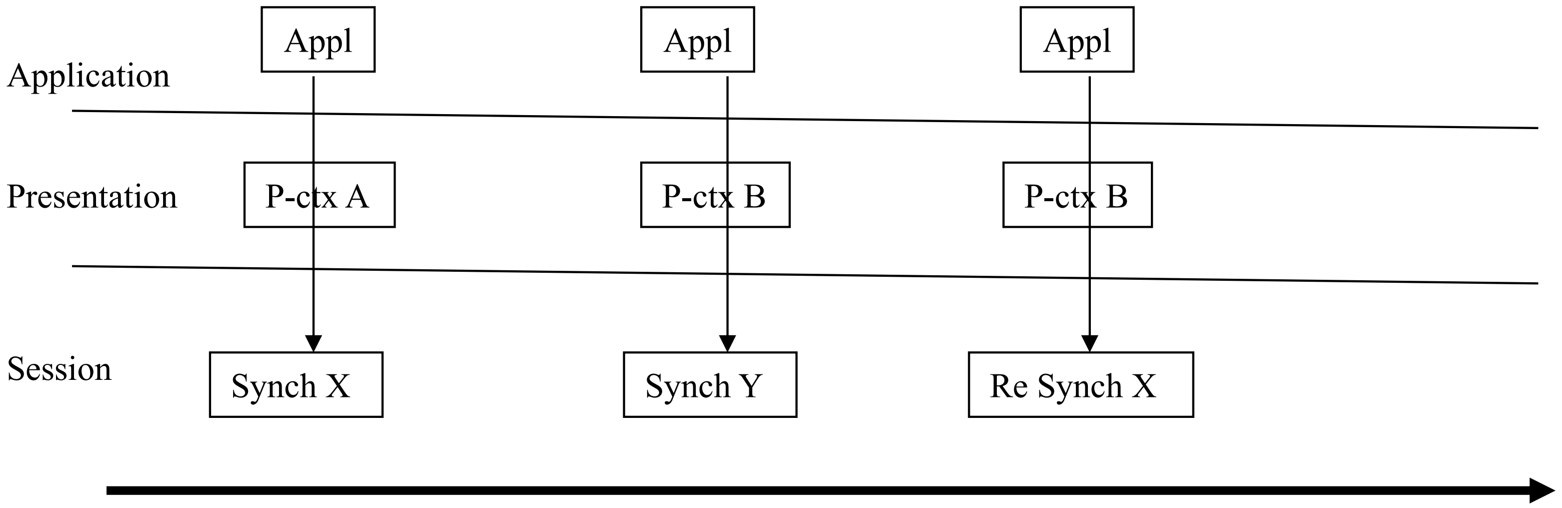
- A similar problem appeared in Presentation with the syntax of Mail.
- They had not foreseen that one would build applications on top of applications of different scope.
  - Because of the simplistic (mainframe) view of Applications, in mail the envelope and the letter had to have the same syntax. This meant that every mail relay would have to understand every syntax that a letter might use, regardless of whether the relay would ever use the syntax. Clearly a non-starter.



- The limitations of Bachman's mainframe model of computing manifested itself. The world of the mid-70's was not the world of even the late 80's.
- Lesson: Do what the problem tells you. Not just short-term requirements. You can pay the piper now or later (when it is a lot more expensive)

But Problems Began to Arise: IV

- There were applications that required more than one ASE to use the Session mechanisms on the same connection at the same time.
  - The Session mechanisms were not “re-entrant.”
    - A Transaction Processing protocol would require that ability.
- Furthermore, if the Application did a re-synch to a previous point (that might be using a different concrete syntax), Presentation would have to track what the Application was invoking in Session! (talk about messy!)

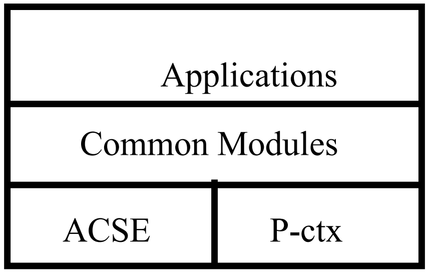


A Recursive Application Layer Structure

- Take the basic structure outlined above:
  - A common establishment ASE (ACSE) (and a variation for connectionless), a collection of ASEs, a method for combining them. Presentation becomes a function of ACSE. Session mechanisms become ASEs.
- And just define the structure to recurse:
  - Allow Application protocols to be defined that were encapsulated in other Application protocols with multiplexing.
- In this structure, the mechanisms from Session become ASEs and are used like any other ASE and can be re-entrant.
- The mail problem is accommodated by realizing that the letter is a connectionless application protocol embedded in a connection-mode relaying protocol.
  - The syntax of the connectionless protocol need only be understood by the sender and receiver of the mail. The syntax of the relay protocol is understood only by the relays.
- The standards for this were completed in the early 90's.
  - Interesting but not entirely right.

Green Side Up![*]

- This makes it clear that
  - Session is a collection of common mechanisms for managing the application’s data flow should have been ASEs, application modules
  - Presentation was not really a layer, but a common function of the application machinery for specifying the syntax of the data transfer phase, *and so*
- At best the upper three layers are upside down!



Applications

Common Modules

ACSE

P-ctx

  - ACSE should sit directly on top of Transport, Presentation negotiated the syntax of the data transfer phase, and the Session mechanisms were ASEs.
  - Also, once the transition was made to a networked environment, applications were designed to be used in a network, Changing transfer syntax became less important.

[*] Obtuse reference to a definitely politically incorrect ethnic joke about laying sod. ;-)

But the Conclusion is Inescapable:
There is no Upper Layer Architecture

- Not as we thought. There is an Application Layer on top of Transport
- There *is* an Application Layer Structure
  - Application Processes with one or more Application-Entities
  - AEs are built of ASEs, including ACSE for setting up Application connections.
  - Coordinated by a Control Function
    - This is primarily a design and implementation methodology
  - And more complex distributed applications can be built on top existing AEs.
- But you need a complete architecture to do it.
- Meanwhile, back in the Internet new applications struggle to develop with 70s technology.

Then Events Shifted back to the Internet

- But not to the old source, that had grown cold in the intervening 15 years.
  - The old source was dead. For evidence, see SNMP.
- The development of the Web was done by Berners-Lee at CERN
  - But it was “just another Gopher” until Andressen at NCSA made most anything “click-able” and that proved to be the turning point.
- HTTP was another application protocol rather than development of upper layer architecture, but it pointed out some of the problems in the Internet ULA:
  - The one-to-one correspondence between protocol and application fails
  - The tools to handle proxies and caches were non-existent.
  - The mice and elephant problem, (one size fits all congestion “avoidance”)
  - The lack of the AE/AP distinction becomes important.
- Things are not really simpler yet.

Is the AE the Application Protocol?

- At first blush, yes.
- But consider the following:
  - There are only six operations on objects that can be done remotely:
    - Read/Write
    - Create/Delete
    - Start/Stop
  - That implies that there is only one Application Protocol!
  - The *objects* being manipulated are always part of the *application*, not the AE
    - (Application protocols affect state external to the protocol)
- So there is no AE or only one AE?!
  - No, The AE defines the collection of objects available to a given activity.
    - It defines the objects available for this communication.
  - Thus, we transition from an IPC model to a programming model.
    - Distributed Applications manipulate objects in a programming language that is translated into sequences of the six operations on the objects.
    - There is more but that can wait.

What Does the One Application Protocol Look Like?

- Connect(Application-Process-Name, etc; Authentication)
- Disconnect()
- <Operation>(Invoke-Id, filter, object-list)
  - Where <Operation> may be:
    - Create/Delete
    - Read/Write
    - Start/Stop
  - Filter ::= <relational expression>
- Transition from an IPC model to a programming language model.
  - Greatly simplifies the development and deployment of distributed applications.

What Have We Learned?

- Don’t Skip Steps!
- Collect Models so that you don’t fall victim to
  - “When All You have is a Hammer, Everything looks like your *Thumb*!”
- Look for solutions that are symmetrical
  - But keep in mind not all are problems are symmetrical.
- Sometimes what looks logical technically, isn’t right.
  - (Importance of knowing when to stop.)
- In general, follow good design principles even though there is no technical reason to. (A very good chance the problem will bite you later.)
  - “Good enough” is seldom good enough in the long term and the longer the more expensive to fix.
  - Always look for common structures, but don’t go too far. ;-)
- The importance of the AE/AP distinction.
- Making protocols independent of syntax.
- Applications are all about the object models not protocols.
- A bit of doubt is always healthy. ;-)

That’s the Theory!

(later we look at current applications)