An Introduction to RINA

Or

How I Learned to Stop Worrying and Love the Internet

FutureNet Tutorial Part I

John Day

May 2010

Good architecture, like good science, is maximizing the invariances and minimizing the discontinuities.

We Got Trouble!
Right Here in River City

The Internet is Facing Severe Problems: I

- Security is essentially non-existent.
  - Excuse: No one considered it in the early days
    - Security wasn't a concern for a military-funded network?
  - Actual: Systematically weak design (hacker mentality)
- Router table size is growing exponentially
  - Excuse: Memory is cheap
  - Actual: No longer on Moore's Law, it is getting expensive and caused by
- No support for multihoming
  - Excuse: not that many hosts need it, and we can kludge it
    - A military-funded network doesn't care about suvrivability?
  - Actual: Since when is 10^7 small, and the kludge doesn't scale.
    - It isn't 10^7, with Smart Grid it is more like 10^{10}.

The Internet is Facing Severe Problems: II

- Mobility is cumbersome and doesn't scale
  - Excuse: What do you mean? It works. . . . . Sort of.
  - Actual: With only physical addresses, hard to do "re-locatable" addressing
- Congestion Control keeps Utilization around 30%
  - Excuse: There is great congestion control in TCP . . . Sort of. Bandwidth is Cheap don't worry about it.
  - Actual: Any control theory book says put feedback as close to the resource as possible. TCP puts it as far away as possible.
- Quality of Service is difficult to do.
  - Excuse: Net neutrality requires that all traffic be treated equally
  - Actual: Net neutrality is political cover for their inability to find solution.
    - Notice: Running out IP addresses was not listed
      - Not a problem. A global address space is not required

And if that Weren't Bad Enough

Much of What is Believed
about the Internet is Myth

Myths of the Internet: I

- The Internet is an Engine of Innovation.
  - The Internet has in a real sense been stagnant since the late-70s
  - Living on Moore's Law and Band-aids
    - Lots of Innovation on *top* of the Internet, but even that has begun to wane.
- The Internet has decentralized administration. No one owns the Internet.
  - Actually, Same as the global PSTN, just no sexy name.
- The IETF is a Grass-Roots Democracy.
  - Actually, It most closely Resembles the Communist Party.
    - IETF meeting is the Party Congress; IESG is the Politburo
      - Designed to be dominated by an elite, just not the one they had in mind.
- The Internet is based on the ARPANET
  - Actually, It is based on the French network CYCLADES
  - The ARPANet technology was abandoned.

Myths of the Internet: II

- The Internet is not an internet, but a catenet.
  - Ceased to be an Internet on January 1, 1983.
- The Internet is a dumb network.
  - Actually, it keeps maximal state in the network, not minimal.
- The Internet has decentralized routing.
  - Actually, in most ISPs routes are statically allocated.
- IP is the Internet Protocol.
  - That is what the letters stand for but at best it is a subnet protocol.
    - More likely just a header fragment
- IP addresses name the host.
  - No, they name an interface to the host.
- TCP isn't perfect, but it is good enough.
  - Every design decision is not just wrong but makes something else worse. One thing it got right was destroyed creating IP.

The Reality is Quite Different

- If the Internet were an Operating System, it would have more in common with DOS, than UNIX.
- Imagine trying to use a modern computer with only physical memory addresses.

- That is the Internet today.

Need to Do Something About It

- For the Past Decade, DARPA, NSF, and numerous conferences, workshops, and summits have discussed and done research on a "Future InterNet Design."
  - So far they have come up dry, nothing, nada.
- Similar efforts in Europe and Asia.
  - Same result. Isn't groupthink wonderful?
- No closer to an answer now than a decade ago.
  - "Rough consensus and running code" doesn't exactly foster the kind of thought required to uncover theory.
- The field is no longer a science, but a craft.
  - They have been asking "What to build?"
  - Not asking, "What don't we understand?"

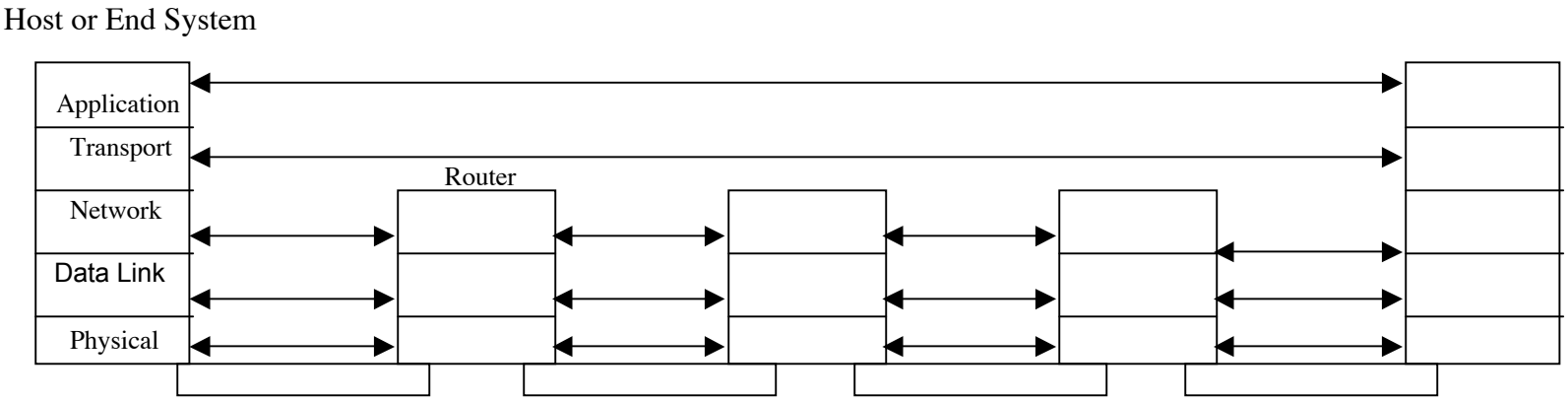
Guiding "Principles" Aren't Much Help

- Soft State/Hard State
  - All properly designed protocols are soft state; only some database operations are hard state. A specious distinction
- Loc/Id Split
  - Attempt to avoid the obvious solution. Mostly post IPng trauma.
  - Continues to route to the wrong place
- Fate Sharing
  - Mostly used as a rug . . . to sweep under.
- End-to-End Principle
  - At best a lemma. More a statement of desire, by focusing attention on the dichotomy of the middle vs the edge, it obscures the point.
  - Hence becomes an impediment to finding a path forward.

The Myths and "Principles" are *the* Major Barrier to FINDing an Answer

- The Answer has been clear since the mid-90s.
- And it is very simple:
- Networking is IPC and only IPC

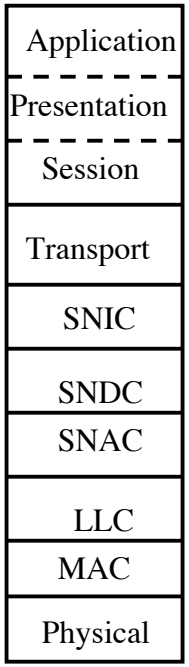
CYCLADES had Shown the Way



An Architecture with Alternating Layers of Relaying and Error Control of Increasing Scope Seemed to Make a Lot of Sense.

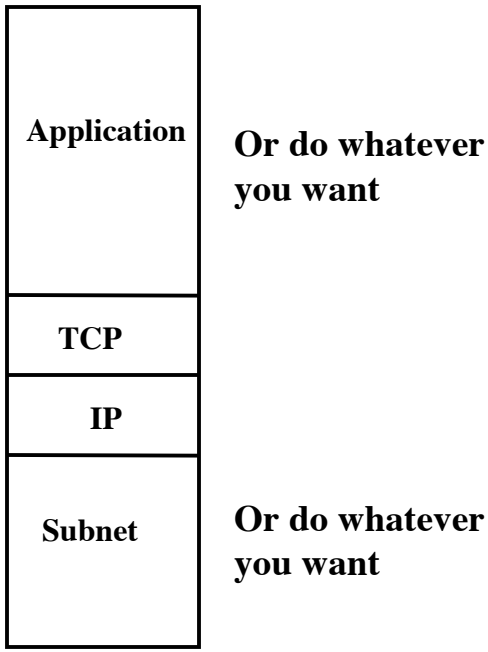
So We Went With It

But As Things Developed Things go more complex



And While Better It Wasn't The Full Answer

Meanwhile the Internet Never Got Past



We Were Missing Something, but What was It?

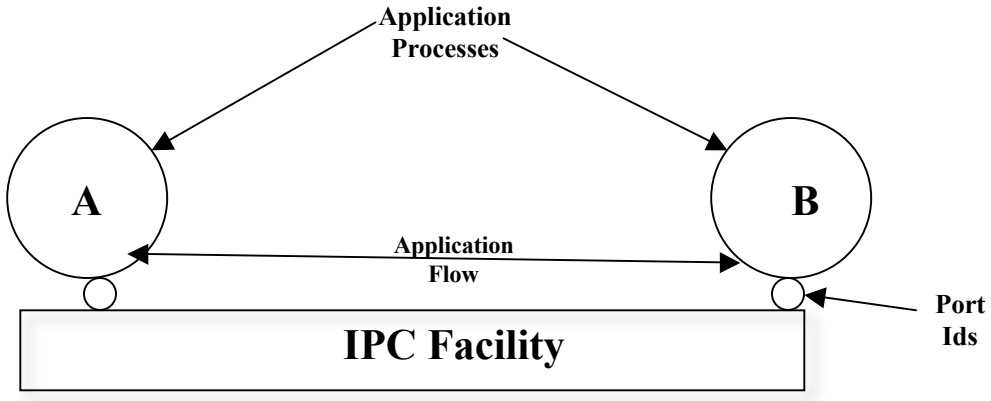
Were Layers the Wrong Model?

Probably Not, Lets Go Back to Basics

- Start simple and incrementally introduce new conditions.
  - Taking Imre Lakatos' Proofs and Refutations as a model.
- We are going to cover ground we all know.
  - Or at least we think we do. (I thought so)
- As we do, think about what is required when
  - the order is not what you might expect.
  - But the implications are even more interesting.

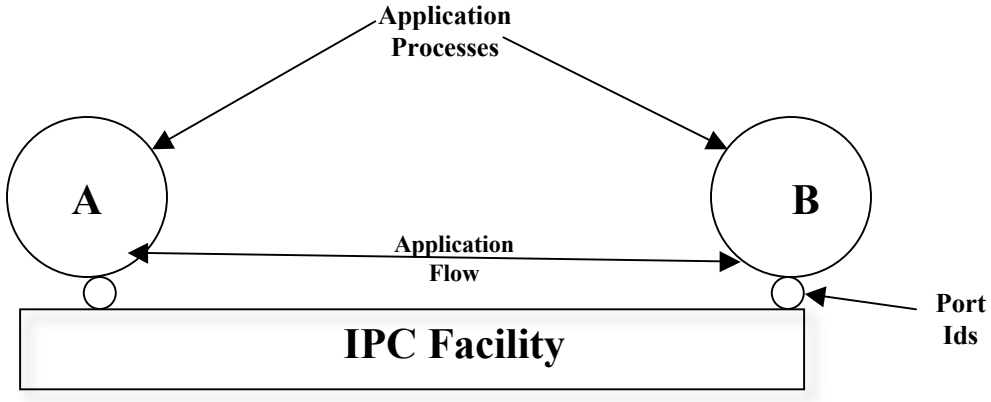
1: Start with the Basics

Two applications communicating in the same system.



Communication within
a Single Processing System

How Does It Work?



1) **A** invokes IPC to create a channel to **B**; **a = Allocate (B, QoS);**

2) IPC determines whether it has the resources to honor the request.

3) If so, IPC allocates port-id **a** and uses "search rules" to find **B** and determine whether A has access to B**.**

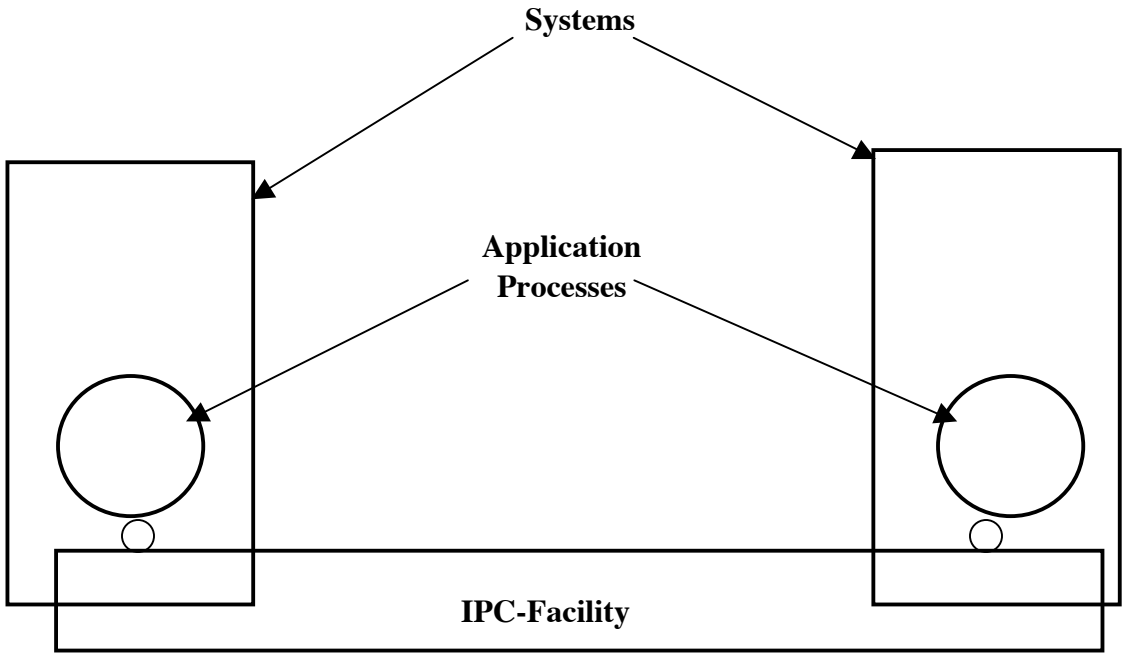
4) IPC may cause an instance of **B** to be created. **B** is notified of the IPC request from **A** and given a port-id, **b**.

5) If **B** responds positively, and IPC notifies **A** (the API could be blocking in which case the assignment of the port-id, **a** would be done now).

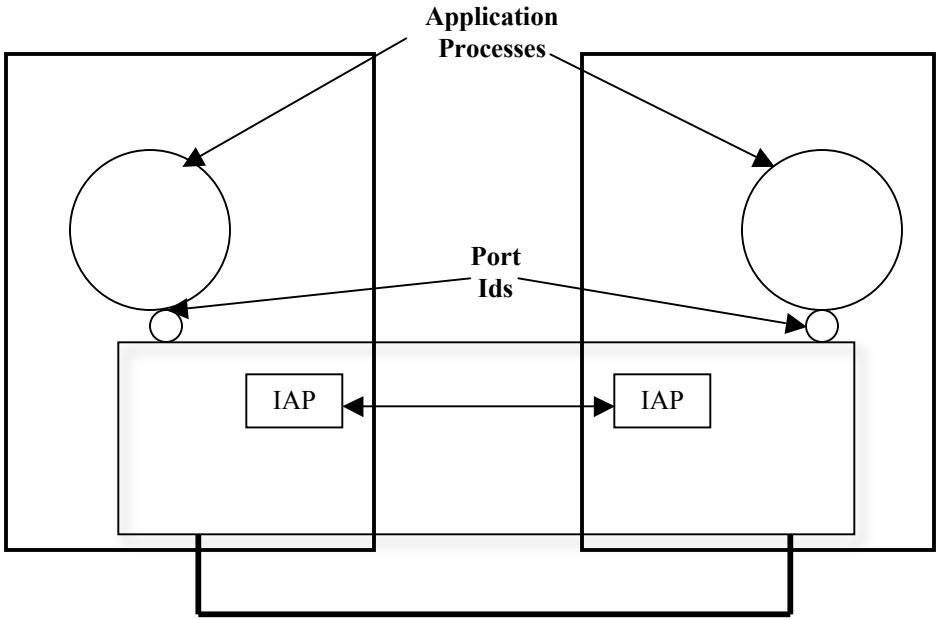
6) Thru n) Then using system calls **A** may send PDUs to **B** by calling **Write(a, buf),** which B receives by invoking **Read(b, read_buffer)**

7) When they are done one or both invoke **Deallocate** with the appropriate parameters

2: Two Application Communicating in Distinct Systems



How Does It Work Now?



1) A invokes IPC to create a channel to **B**; a = **Allocate (B, QoS);**

2) IPC determines whether it has the resources to honor the request.

3) If so, IPC uses "search rules" to find **B.** and determine if A has access to B.

- Management of name space is no longer under the control of a single system.
  - Each system no longer knows all available applications.
- Local Access Control can no longer be relied on to provide adequate authorization and authentication.

- Need a protocol to carry application names and access control information.
  - Lets call it the IPC Access Protocol

What Does "IAP" Look Like?

- Simple Request/Response Protocol
  - IAP-Req(Dest-Appl-name, Src-Appl-name, QoS params, Src-Capability)
  - IAP-Resp(Dest-Appl-name, Src-Appl-name, QoS params, result)

Op	Dest Appl Name	Src Appl Name	QoS & Policies	Capability

- How do we know when to use it?
  - If the application isn't here, it must be there!
- But we have a problem. How do we get it there?

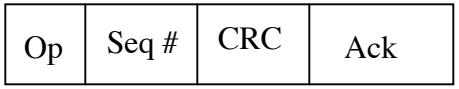
Getting from A to B

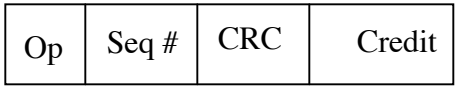
- We knew this was going to be a problem sooner or later.
- We need to be able to send information from A to B.
- And we know:
  - Bad things can happen to messages in transit. Protection against lost or corrupted messages
    - This can be expensive
  - Receiver must be able to tell sender, it is going too fast. We need Flow Control, and
  - We have lost our means of synchronization:
    - No common test and set means shared memory can no longer be used
    - Must create shared state between two systems. An explicit synchronization mechanism is required.
- We need some kind of Protocol for Error and Flow Control.

What Would an EFCP Look Like?

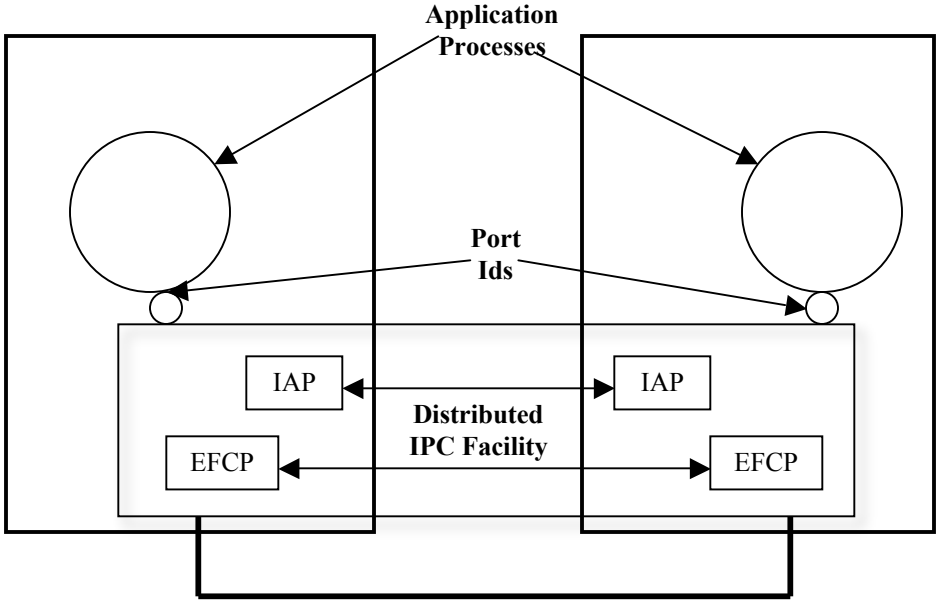
- Symmetric Protocol to establish error and flow control
  - Transfer PDU

  - Ack and Flow Control PDU types





How Does It Work Now?



1) **A** invokes IPC to create a channel to **B**; a = **Allocate (B, QoS);**
2) IPC determines whether it has the resources to honor the request.
3) Send IAP Request to access B**,** creating an EFCP connection and determines if A has access to B.
4) IPC may cause **B** to be instantiated. **B** is notified of the IPC request from **A** and given a port-id, **b**.
5) If **B** responds positively, and IPC notifies **A** with a different port-id, **a**.
6) Thru n) Then using system calls **A** may send PDUs to **B** by calling **Write(a, buf),** which B receives by invoking **Read(b, read_buffer)** over the EFCP connection
7) When they are done one or both invoke Deallocate with the appropriate parameters

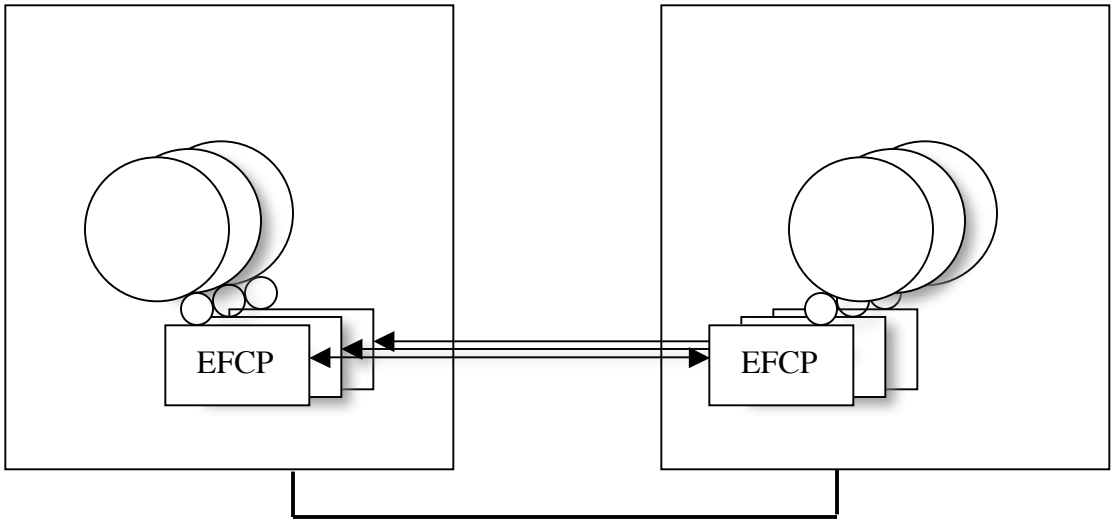
Two Applications in Two Systems Required Three New Concepts

- An Application Name Space that spans both systems. (not really new)
  - Should be location-independent in general so that applications can move.
- A Protocol to carry Application Names and access control info
  - Applications need to know with whom they are talking
  - IPC must know what Application is being requested to be able to find it.
    - For now, if the requested Application isn't local, it must in the other system.
- A Protocol that provides the IPC Mechanism and does Error and Flow Control.
  - To maintain shared state about the communication, i.e. synchronization
  - To detect errors and ensure order
  - To provide flow control
- Resource allocation can be handled for now by either end refusing service.

3: Simultaneous Communication Between Two Systems

i.e. multiple applications at the same time

- To support this we have multiple instances of the EFCP.



Will have to add the ability in EFCP to distinguish one flow from another.

Typically use the port-ids of the source and destination.

Connection-id

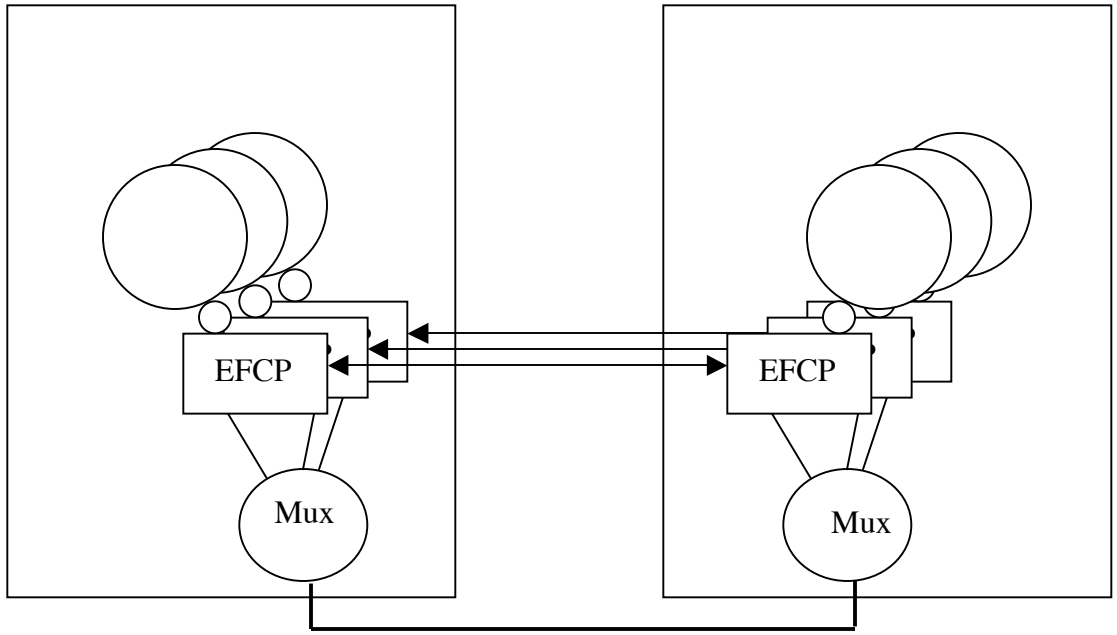
Dest-port	Src-port	Op	Seq #	CRC	Data

Also include the port-ids in the information sent in IAP to be used in EFCP synchronization (establishment).

Simultaneous Communication Between Two Systems

i.e. multiple applications at the same time

- In addition to multiple instances of the EFCP.

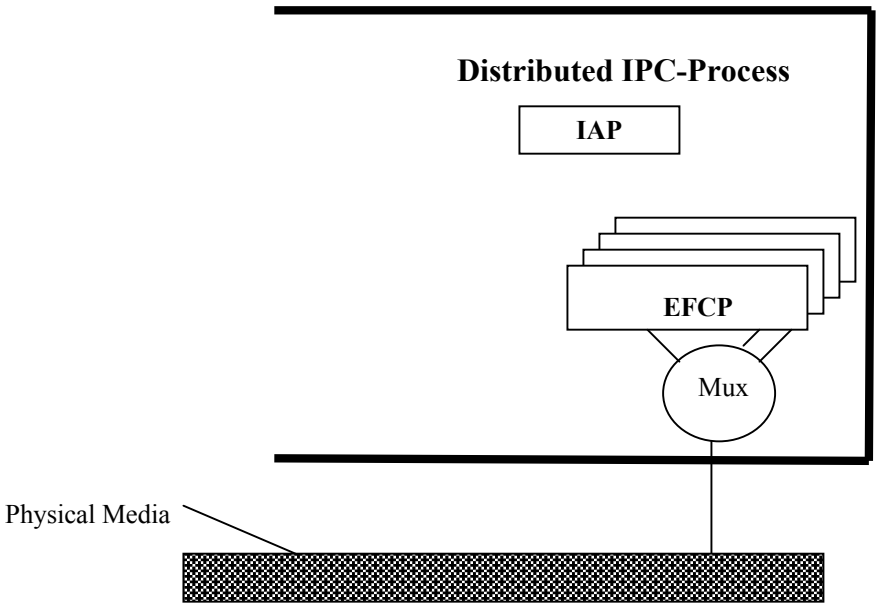


Will also need an application to manage multiple users of a single resource.

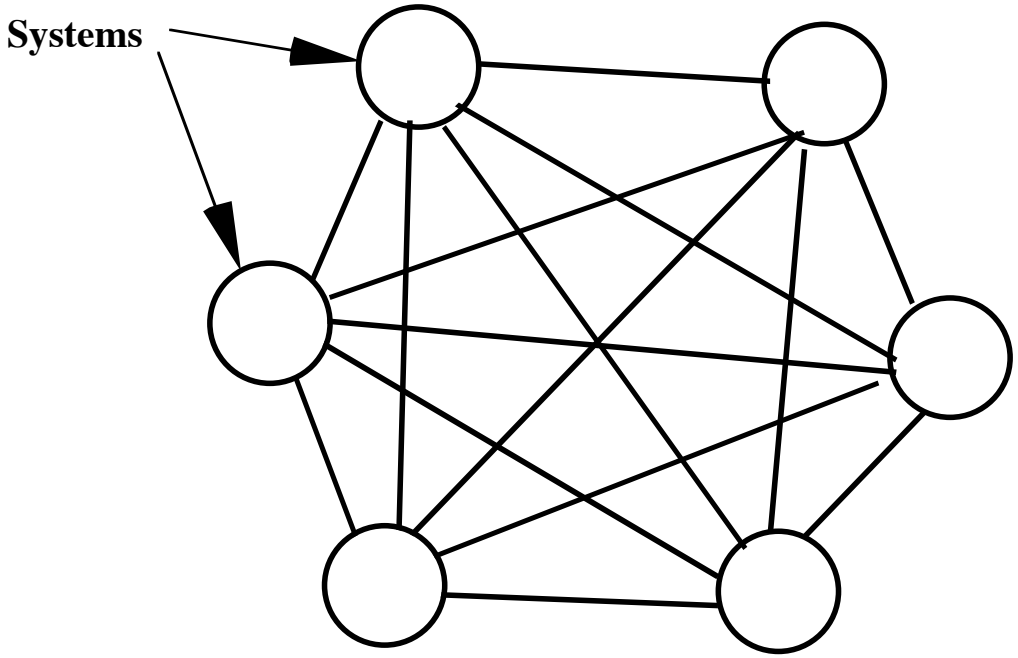
Multiple Instances of IPC

- New Concept: a multiplexing application to manage the single resource, the physical media.
- The multiplexing application will need to be fast, its functionality should be minimized, i.e. just the scheduling of messages to send.
  - To provide QoS, we use the EFCP and scheduling by the Mux.
- To do resource allocation, we will just let the other side refuse if it can't satisfy the request.
- Application naming gets a bit more complicated than just multiple application-names.
  - Must allow multiple instances of the same process

IPC to Support Simultaneous Communication between Two Systems



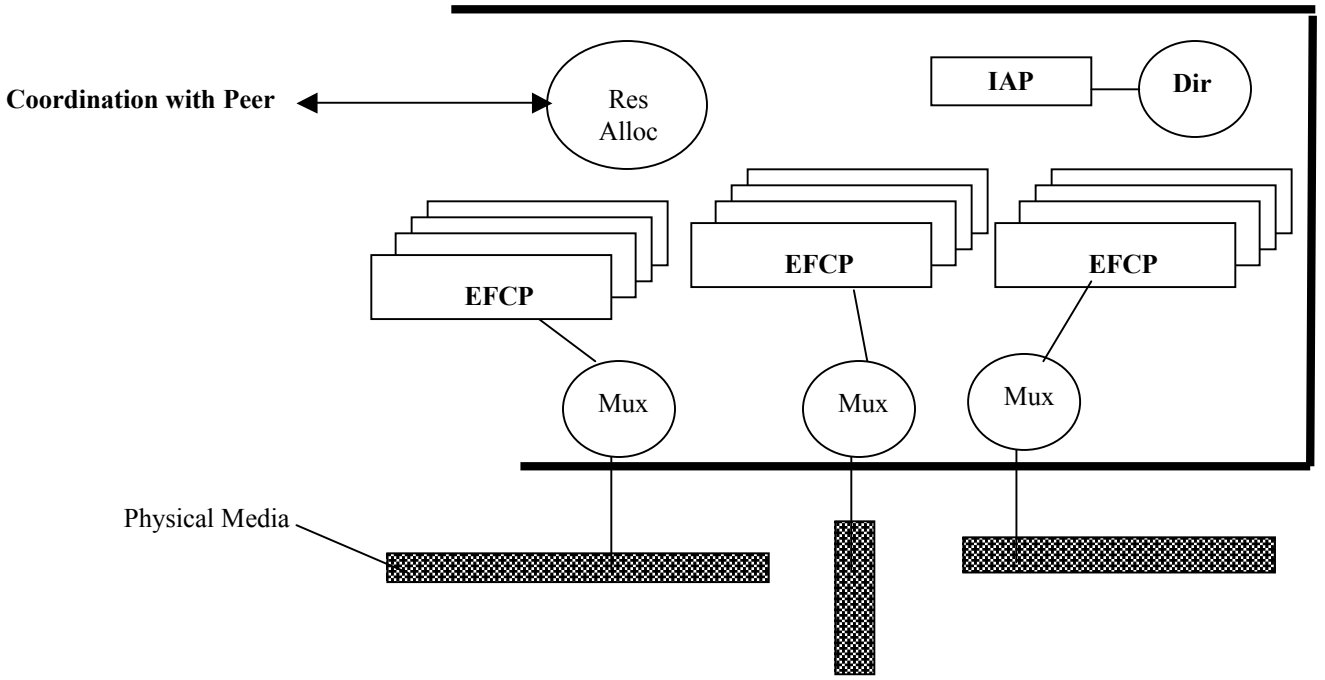
4: Communication with N Systems



Replication Entails More Management

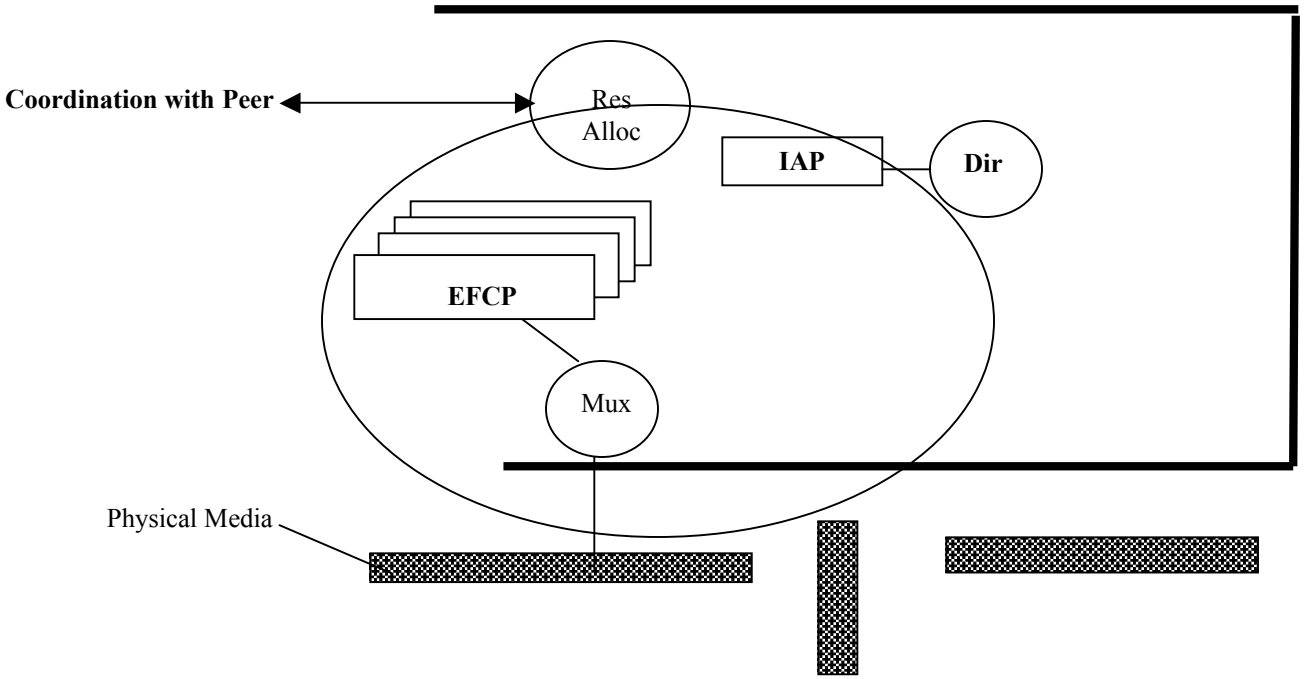
- Separate Multiplexing Application for each physical media interface.
- IPC can find the destination by choosing the appropriate interface.
- If enough applications, create a Directory to remember what is where, i.e. what application names are at the other end of which interfaces.
- Same local names can be used to keep track of which EFCP-instances (port-ids) are bound to which Multiplexing Application.
- With many destinations, may need to coordinate resource allocation information within our distributed IPC Facility.

With Multiple Interfaces It Gets Messy



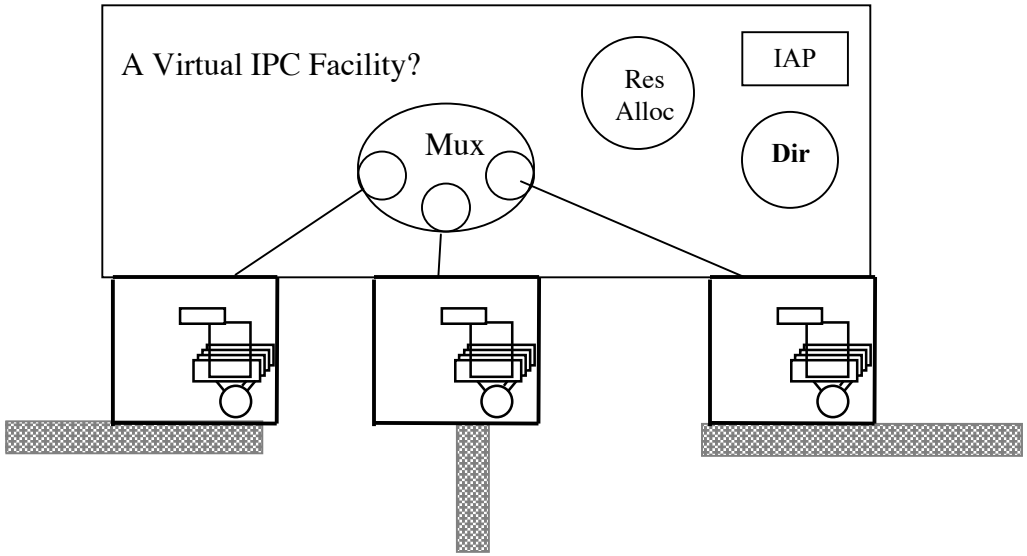
- So a task to manage the use of the interfaces and mask any differences.
  - A little organizing will help.

There is Some Common Structure



- We can organize interface IPC into modules of similar elements.
- Each one constitutes a Distributed IPC Facility of its own.
  - As required, consists of IAP, EFCP, Multiplexing Application, Directory, Per-Interface Resource Allocation
- Then we just need an application to manage their use and moderate user requests.

A Little Re-organizing

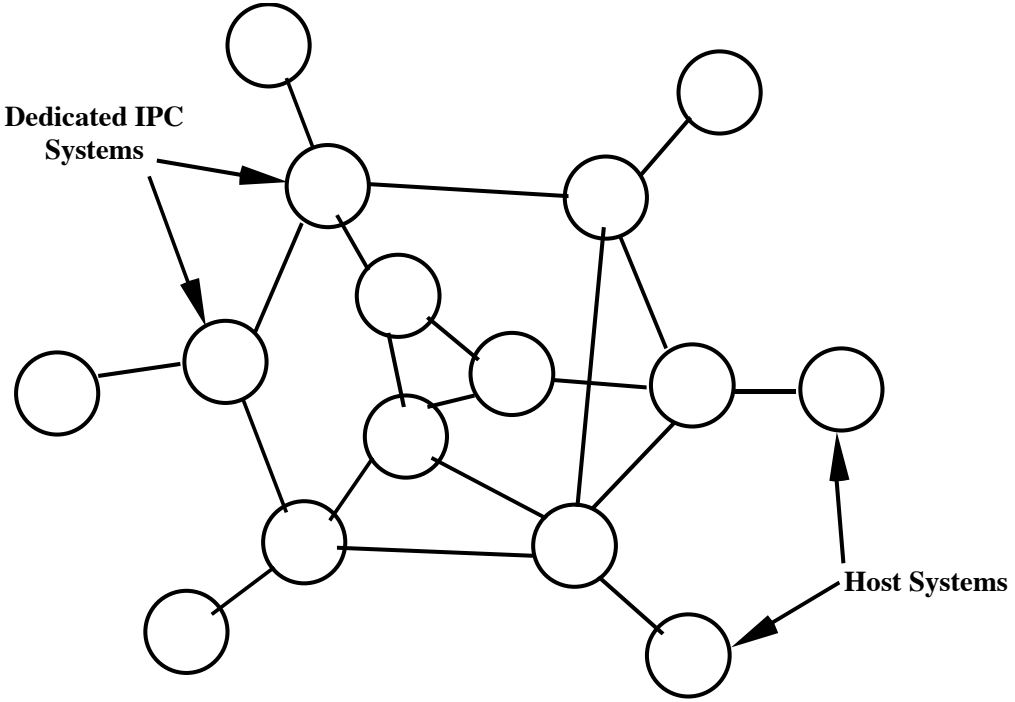


So we have a Distributed IPC Facility for each Interface and an application over all of them to manage their use and to give the user a common interface, a Virtual IPC Facility?

BUT

- This fully connected graph approach isn't going to scale very well and is going to get very expensive.
  - And not everyone needs to talk to everyone else all the time.
- Need to do something better.

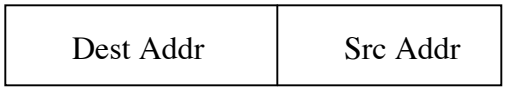
5: Communicating with N Systems (On the Cheap)



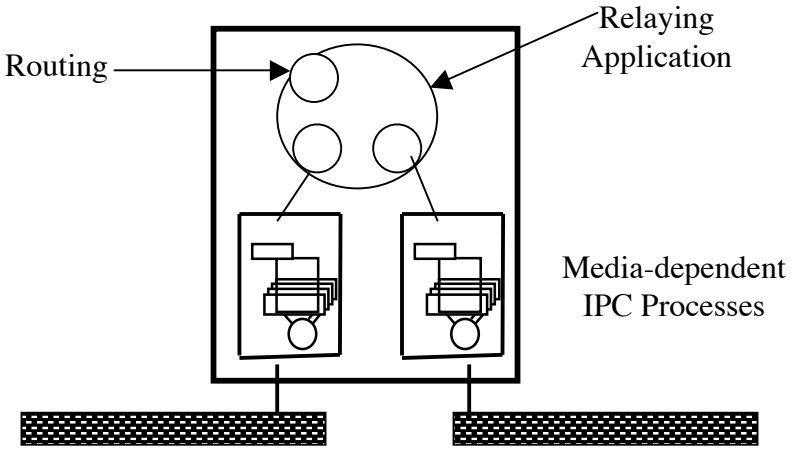
By dedicating systems to IPC, reduce the number of lines required and even out usage by recognizing that not everyone talks to everyone else the same amount.

Communications on the Cheap

- We will need systems dedicated relaying and multiplexing.
- That requires some new elements:
  - Globally accepted names for source and destination muxing apps.
  - And also for the relays. Relays require names for routing. Have to know where you are to determine where to go next.
  - Need routing applications too, which will need to exchange information on connectivity.
- Will need a header on all PDUs to carry the names for relaying and multiplexing.
  - Interface IPC Facilities will need one too if they are multiple access.

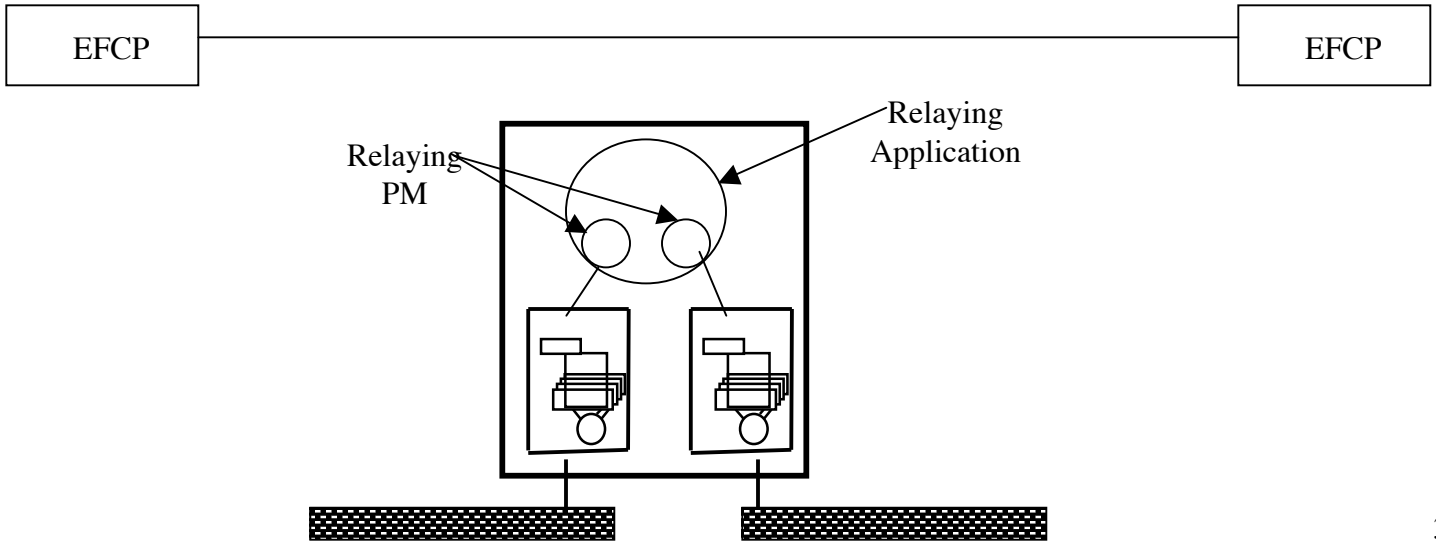


Common Relaying and Multiplexing Application Header



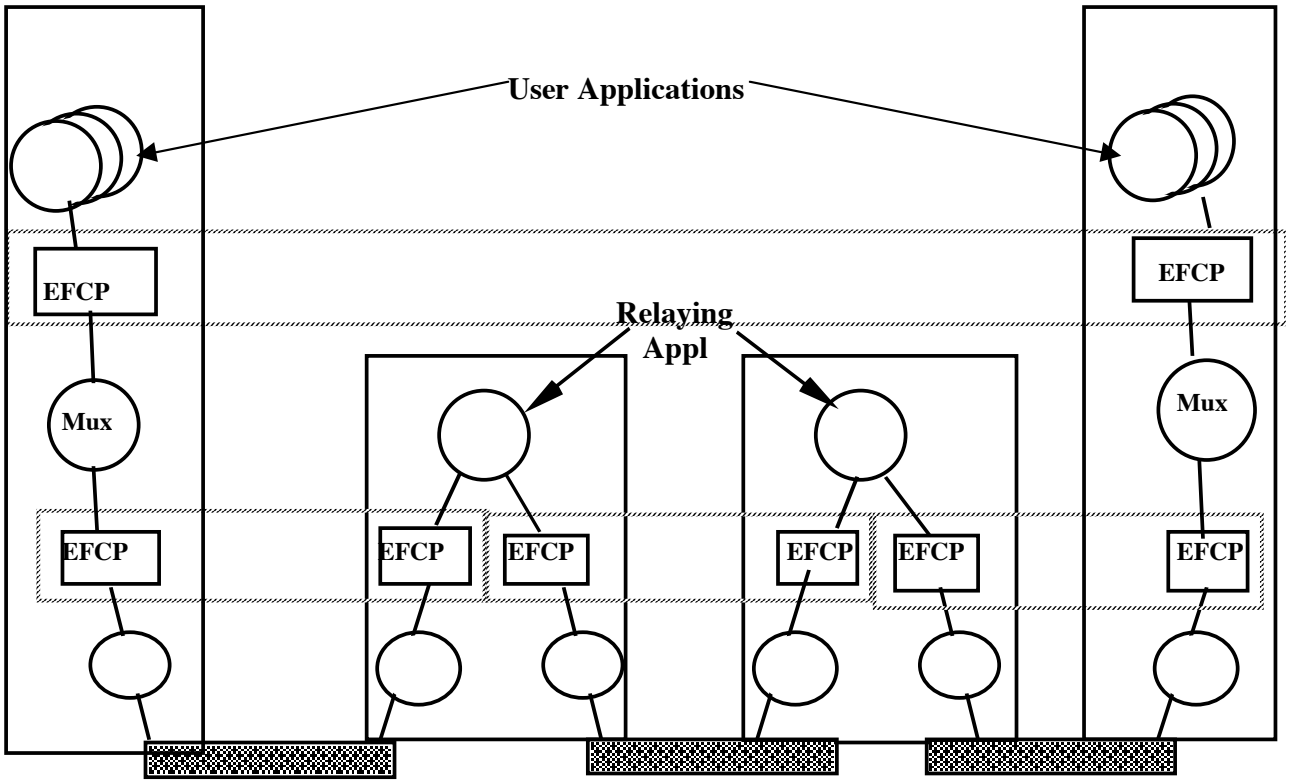
Communications on the Cheap

- But relaying systems create problems too.
  - Can't avoid momentary congestion from time-to-time.
  - Annoying bit errors can occur in their memories.
- Will have to have an EFCP operating over the relays to ensure required QoS reliability parameters.
  - Our virtual IPC Facility isn't very virtual.



The Big Picture

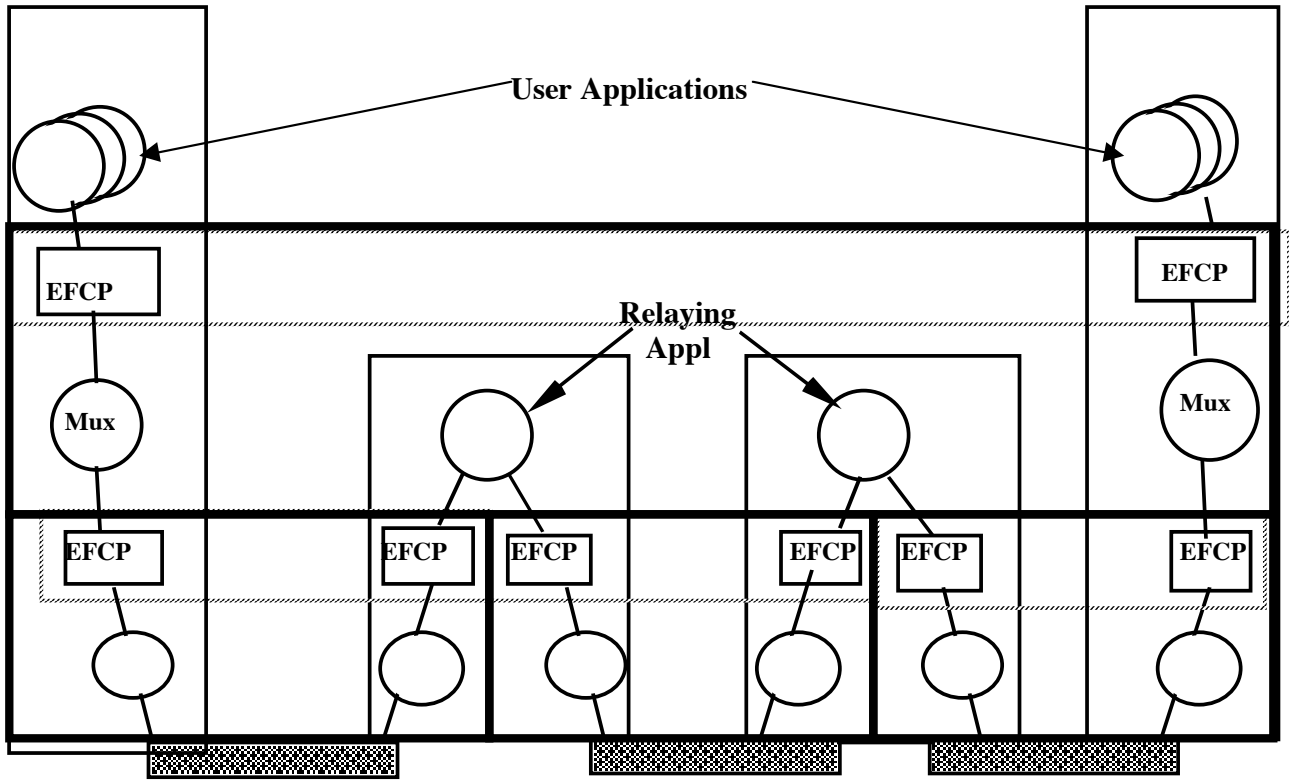
But this is half way between a bead-on-a-string model and a layered model



This should look familiar.

The IPC Model

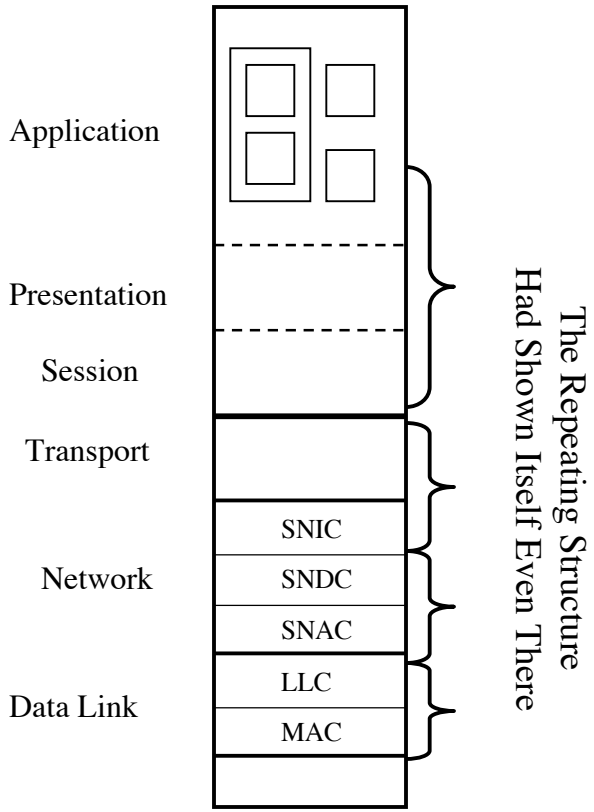
(A Purely CS View)



Summary

- "Networking is InterProcess Communication"
  - . . . . and only IPC.
    - The quote is Bob Metcalfe, 1972. (The rest is mine.)
- A layer is a distributed application that manges IPC consisting of a collection of SDU protection, muxing, EFCP, and their associated routing and resource management tasks.
    - This is a distributed computing problem, not a "telecom" problem.
- And this Distributed IPC Facility repeats.
- This constitutes a basis for a general theory of networking.

OSI Should Have Seen the Pattern



From This We Can Construct What We Need

But First We Need a Few Tools

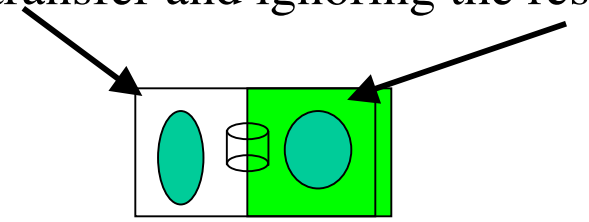
- Resolving the Connection vs Connectionless Debate
  - The center of the 30 Years War
- The Nature of Applications and their Protocols.
  - A Couple of Important Insights
- Watson's Fundamental Results on Synchronization
  - Probably the most important insight in networking since datagrams
- Separating Mechanism and Policy
  - Pulling out the invariances

The Connection Connectionless War

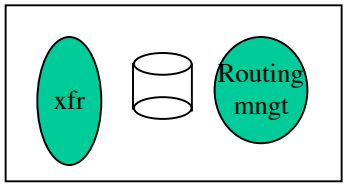
- The technical side of what was really an economic war.
  - The Layered Model invalidated both the PTT and IBM business models.
  - Connectionless removed the security blanket of determinism.
  - The war created a bunker mentality that made understanding hard.
    - All or nothing.
- For years, we saw it as the amount of shared state.
  - Connections had lots of shared state; connectionless very little.
- A function of the layer, not a service. Not related to reliability
  - Not visible over the layer boundary.
  - Ports must be allocated even for a connectionless flow.
- Later it became clear that
  - As traffic becomes more deterministic, connections are preferred
    - Down in the layers and in toward the backbone
  - As traffic becomes more stochastic, connectionless is preferred
    - As one moves up and toward the periphery

Resolving the CO/CL Problem

- Lets look at this very carefully
- What makes connection-oriented so brittle to failure?
  - When a failure occurs, no one knows what to do.
  - Have to go back to the edge to find out how to recover.
- What makes connectionless so resilient to failure?
  - Everyone knows how to route everything!
- Just a minute! That means!
  - Yes, connectionless isn't minimal state, but maximal state.
    - The dumb network ain't so dumb.
  - Where did we go wrong?
- We were focusing on the data transfer and ignoring the rest:



We Need to Look at the Whole Thing



(A bit like doing a conservation of energy problem and getting the boundaries on the system wrong.)

- The amount of state is about the same, although the amount of replication is different.
- We have been distributing connectivity information to every Node in a layer, but
- We have insisted in distributing resource allocation information only on a need to know basis, i.e. connection-like.
  - Even if we aren't too sure who needs to know.
- Now we have to work out how to do resource allocation more like how we do routing. (Left as an exercise.)

So What Do We Know About CO/CL

- It is a function of the layer. Should not be visible to applications.
- Connectionless is characterized by the maximal dissemination of state information and dynamic resource allocation.
- Connection-oriented mechanisms attempt to limit dissemination of state information and tends toward static resource allocation.
- Applications request the allocation of comm resources.
  - The layer determines what mechanisms and policies to use.
  - Tends toward CO when traffic density is high and deterministic.
  - CL when traffic density is low and stochastic.

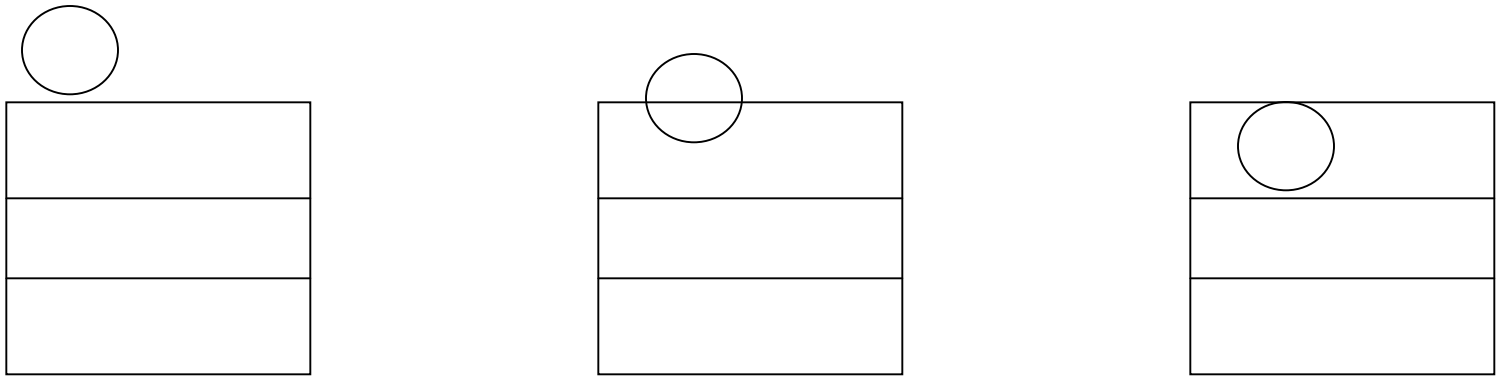
The Upper Layers

- After the initial success, this was one of the big unknowns.
  - Operating Systems had been a good initial guide:
    - NCP - Interprocess Communication
    - Telnet - terminal device driver
    - FTP - the file system
    - NETRJE - RJE
- But what was the general structure?
    - There was early work that indicated it wasn't layered.
  - OSI made a stab at it
  - By 1983, had realized that there were no upper layers.T
    - There were common functions.
    - But the nature of the Application itself was interesting.

Applications and Communication: I

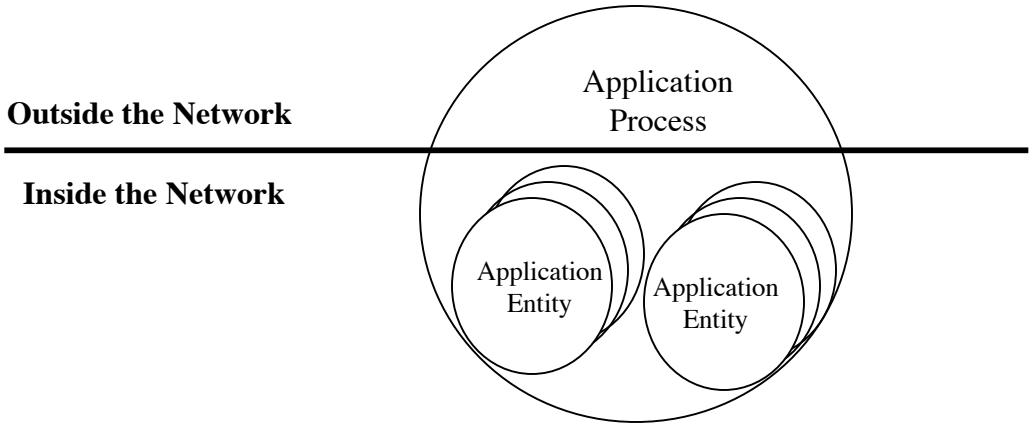
Is the Application in or out of the IPC environment?

- 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.

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.

BUT

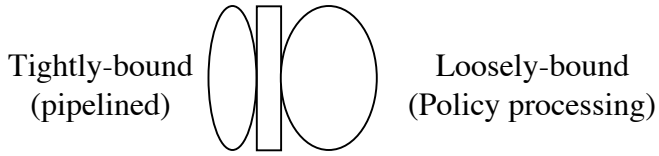
There is only one application protocol

- Huh!? Think about it. What can you do remotely?
  - Read/Write – Create/Delete – Start/Stop
  - On various objects. Everything is just an object outside the protocol.
    - Application protocols modify state outside the protocol.
- In that case, why do we need to name this application-entity stuff?
  - A particular collection of objects are required for an activity.
  - May be shared with others, but has its own access control.
  - One protocol, potentially shared objects, different state machines
  - Hence, all application protocols are stateless, the state is in the application.

Delta-t 1980

- Richard Watson develops delta-t, a unique approach.
  - Assumes all connections exist all the time.
  - TCBs are simply caches of state on ones with recent activity
- Watson proves that the conditions for distributed synchronization are met *if and only if* 3 timers are bounded:
    - Maximum Packet Lifetime
    - Maximum number of Retries
    - Maximum time before Ack
  - That no explicit state synchronization, i.e. hard state, is necessary.
    - SYNs, FINs are unnecessary
- IOW, *all* properly designed data transfer protocols are soft-state.
    - Including protocols like HDLC
- 1981 paper, Watson shows that TCP has all three timers and more.
- And PNA figures out that . . . .

The Structure of Protocols



- If we separate mechanism and policy, we find there are
- Two kinds of mechanisms:
  - Tightly-Bound: Those that must be associated with the Transfer PDU
    - policy is imposed by the sender.
  - Loosely-Bound: Those that don't have to be.
    - policy is imposed by the receiver.
  - Furthermore, the two are only loosely coupled through a state vector.
    - Implies a very different structure for protocols and their implementations
      - Right, we split TCP in the wrong direction
- Noting that syntactic differences are minimal, we can conclude that
- There is one data transfer protocol with a small number of encodings.

Implications: Protocols I

- Data Transfer Protocols modify state *internal* to the Protocol. Application Protocols modify state *external* to the protocol.
- There are only two protocols (full stop):
  - A data transfer protocol, based on delta-t
  - An Application protocol that can perform 6 operations on objects:
  - There is no distinct protocol like IP.
    - Was just a common header fragment anyway.
- A Layer provides IPC to either another layer or to a Distributed Application using a programming model. The application protocol is the "assembly language" for distributed computing.
  - As we shall see, we have now made network architecture independent of protocols.

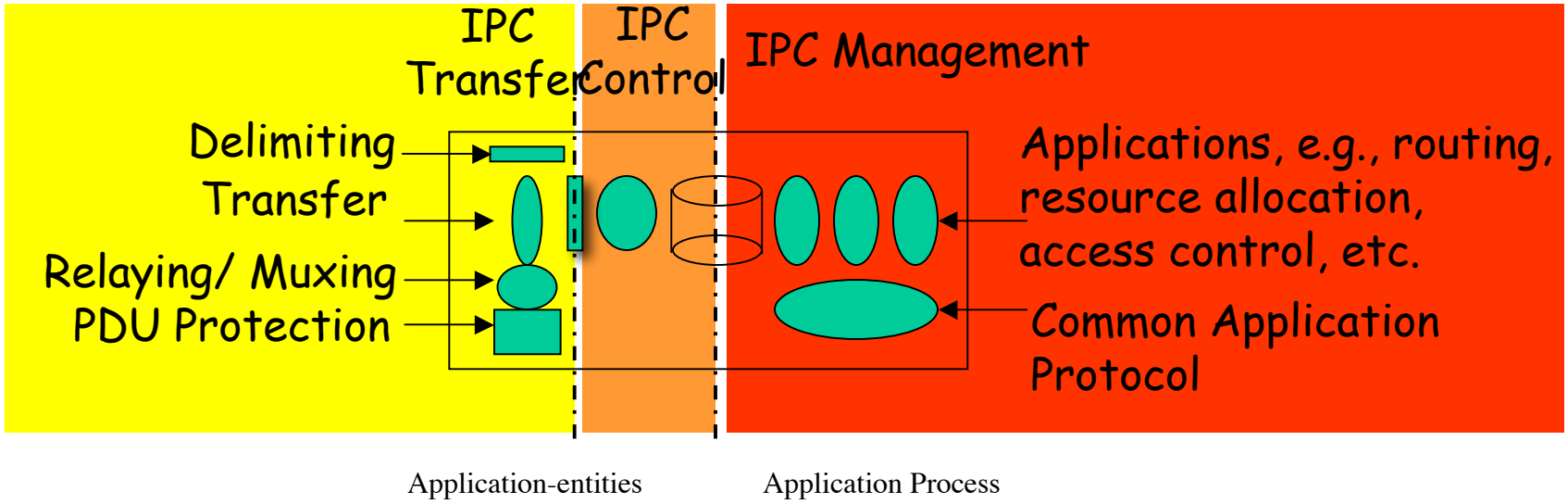
Implications: Protocols II

- "Hard state" only occurs for some uses of application protocols
  - Storing in a database *may* be hard-state. Everything else is soft-state.
  - Hence the "hard-state/soft-state" distinction at best states the obvious.
- Separating mechanism and policy in a delta-t like protocol will yield the entire range from UDP-like to TCP-like.
- Watson implies decoupling "port allocation" from synchronization.
  - Greatly simplifying and improving security, enabling multi-flow allocations of IPC, etc.
- And One Other Thing:

Fundamental Result

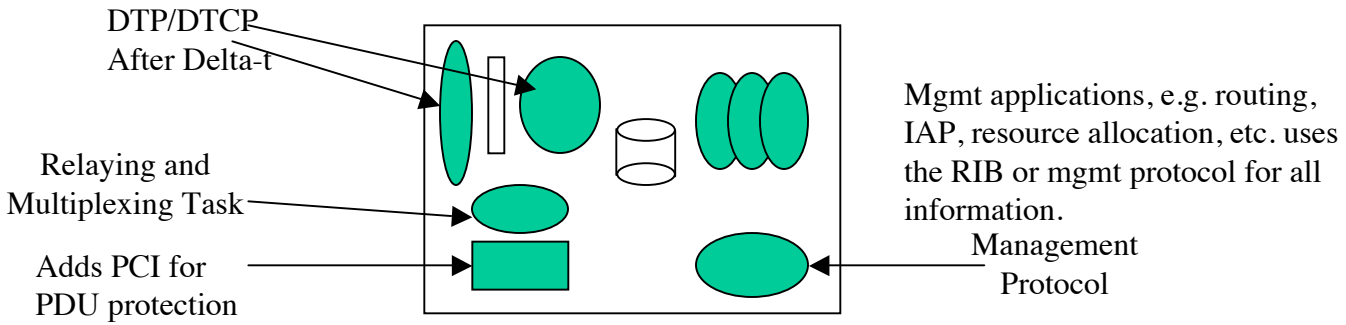
- Watson's result also defines the bounds of networking or IPC:
  - It is IPC if and only if Maximum Packet Lifetime can be bounded.
  - If MPL can't be bounded, it is remote storage.

What a Layer Looks Like



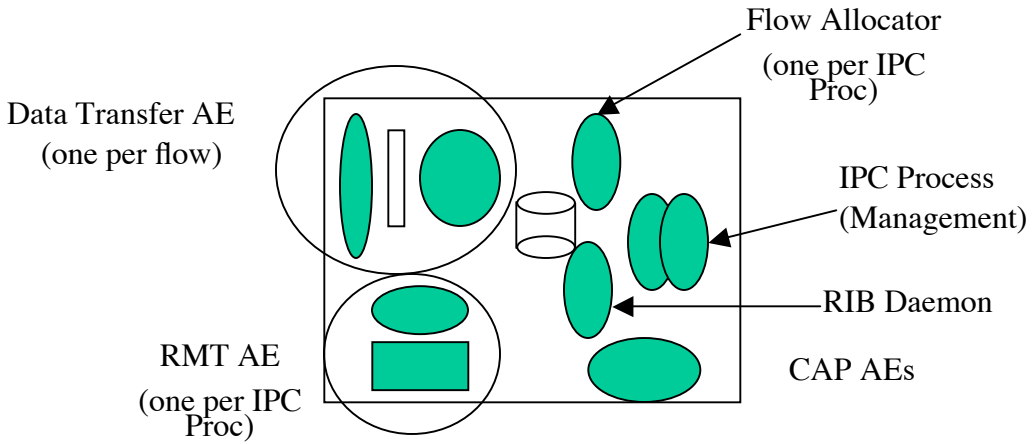
- Processing at 3 timescales, decoupled by either a State Vector or a Resource Information Base
  - IPC Transfer actually moves the data (≈ IP + UDP)
  - IPC Control (optional) for retransmission (ack) and flow control, etc.
  - IPC Layer Management for routing, resource allocation, locating applications, access control, monitoring lower layer, etc.

What are the Protocols?



- Only two
  - A data transfer protocol, based on delta-t with mechanism and policy separated. This provides both unreliable and reliable flows.
  - A management protocol based on CMIP leveraging the constrained form of map/reduce in CMIP.

The Application Entities of an IPC Process



- The Application Process of an IPC Process is management.
  - Flow Allocator manages flows, finds the destination and does access control, manages the binding of connection-endpoint-ids to port-ids.
  - Data Transfer AE is the error and flow control protocol
  - RMT AE consists of the relaying and multiplexing task and SDU Protection
  - RIB Daemon maintains the local RIB information.
  - CAP AEs are management flows with other members of the DIF and NMS

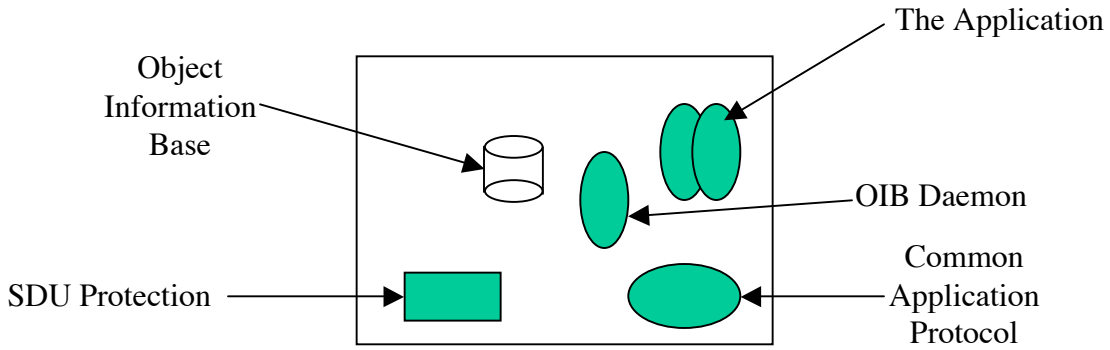
But Wait a Minute!

- If a Layer is a Distributed Application that Does IPC,
- What is a Distributed Application?
  - a Distributed Application Facility (DAF).

- Good Question!

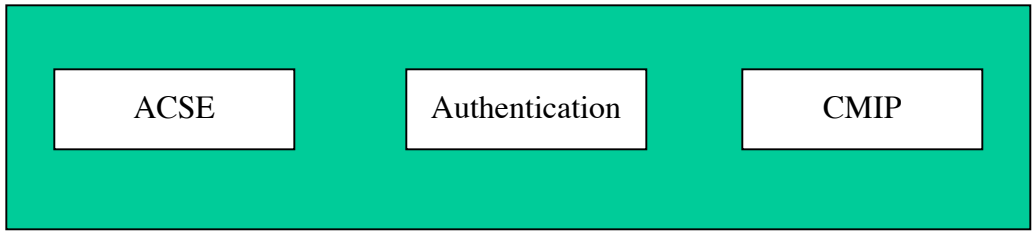
- What are the elements that would be common to distributed applications that don't do manage IPC.
    - Notice that a DIF is primarily a DAF that manages IPC.

Distributed Application Facility



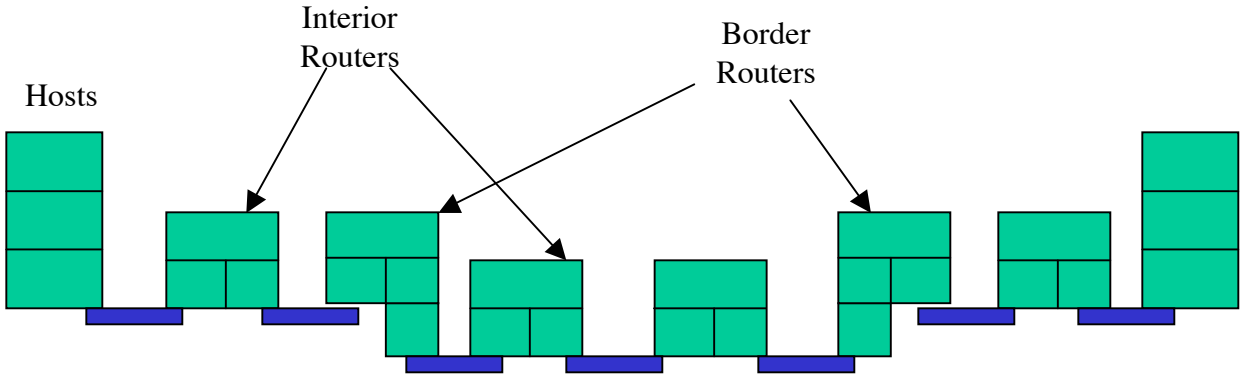
- A DAF consists of SDU protection, the Common Application Protocol (CMIP + ACSE), an Object Information Base, and the OIB Daemon.
- The transition from a DIF to a DAF is a transition from an IPC model to a programming language model.
  - The task of the OIB Daemon is to populate the OIB with whatever the Application needs on whatever schedule it needs it: a sort of “schema pager”

Common Distributed Application Protocol



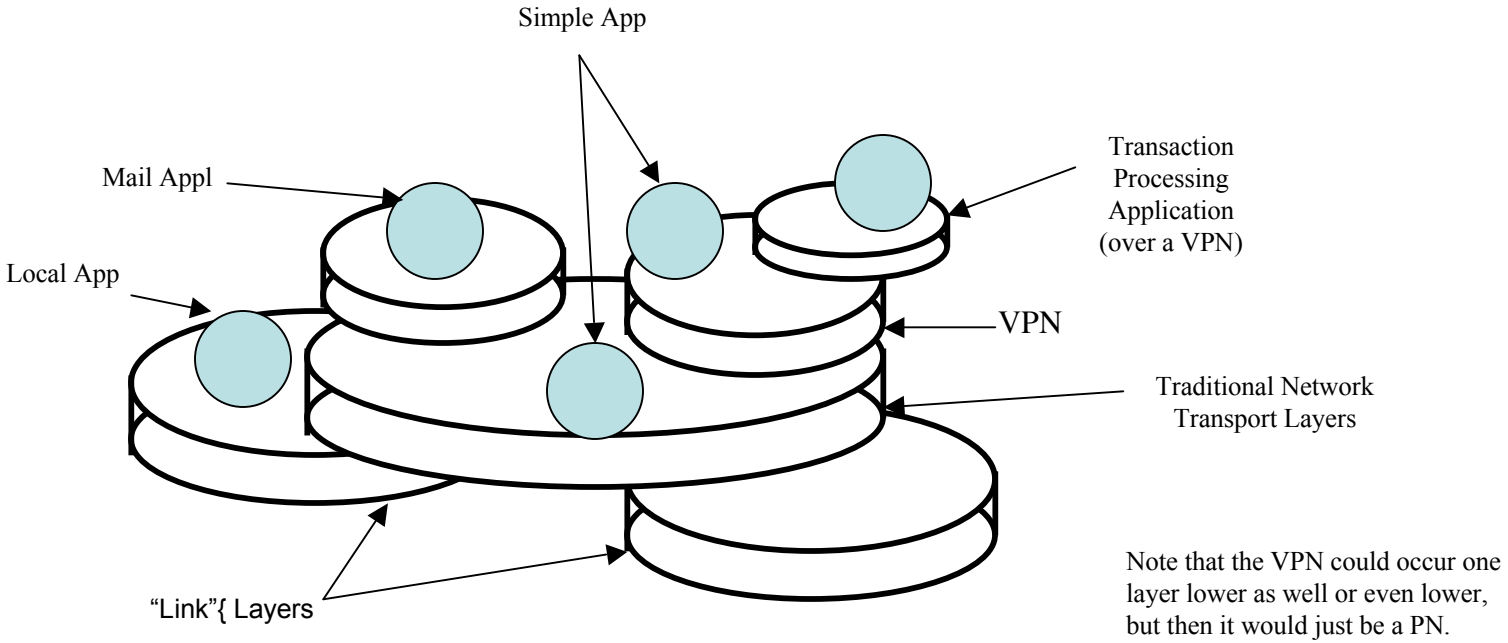
- ACSE is the common protocol for establishing application connections. It ensure that there is a known first exchange.
  - ACSE was defined to be used recursively and has hooks for. . .
- An authentication module is policy of whatever strength required.
- CMIP provides the minimal six operations and a basic object-oriented functionality (scope and filter).
  - Would other programming paradigms lead to different functions?

Only Three Kinds of Systems



- Middleboxes? We don't need no stinking middleboxes!
- NATs: either no where or everywhere,
  - NATs only break broken architectures
- The *Architecture* may have more layers, but no *box* need have more than the usual complement.
  - Hosts may have more layers, depending on what they do.

Hosts Might Have More DIFs



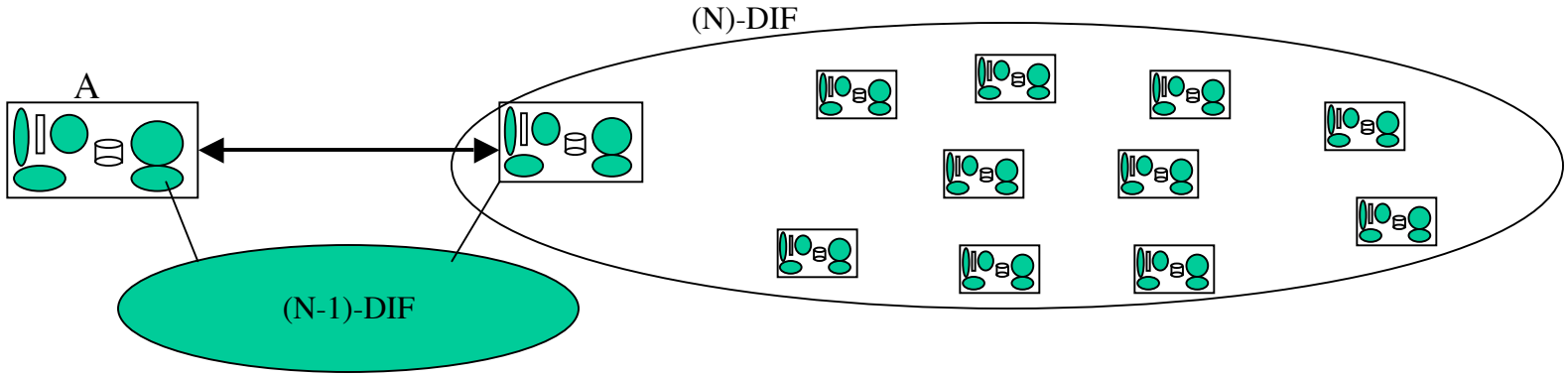
User Applications use whatever layer has sufficient scope to communicate with their apposite.

All Communication goes through Three Phases

- Enrollment
  - Operations to create sufficient state within the network to allow an instance of communication to be created.
- Allocation (also known as Establishment)
  - Operations required to allocate an instance of communication creating sufficient shared state among instances to support the functions of the data transfer phase.
- Data Transfer
  - Operations to provide the actual transfer of data and functions which support it.
- Most of our attention has been on the last two. The first has often been ignored and is usually seen as necessarily ad-hoc. But enrollment turns out to be key.

How Does It Work?

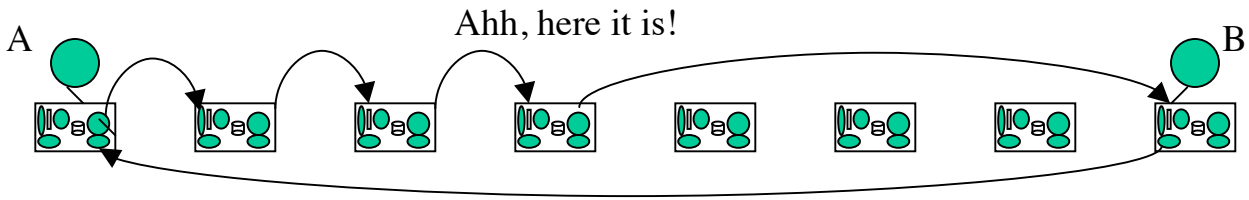
Joining a Layer



- Nothing more than Applications establishing communication (for management)
  - Authenticating that A is a valid member of the (N)-DIF
  - Initializing it with the current information on the DIF
  - Assigning it a synonym to facilitate finding IPC Processes in the DIF, i.e. an address

How Does It Work?

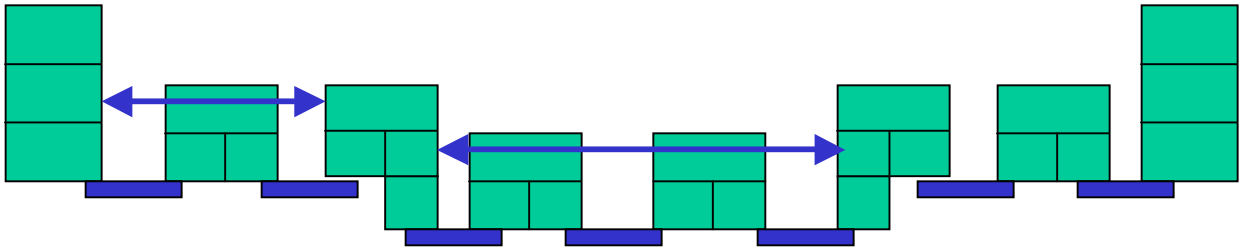
Establishing Communication



- Simple: do what IPC tells us to do.
  - A asks IPC to allocate comm resources to B
  - Determine that B is not local to A use search rules to find B
  - Keep looking until we find an entry for it.
  - Then go see if it is really there and whether we have access.
  - Then tell A the result.
- This has multiple advantages.
  - We know it is really there.
  - We can enforce access control
  - We can return B's policy and port-id choices
  - If B's has moved, we find out and keep searching

How Does It Work?

"Congestion Control"

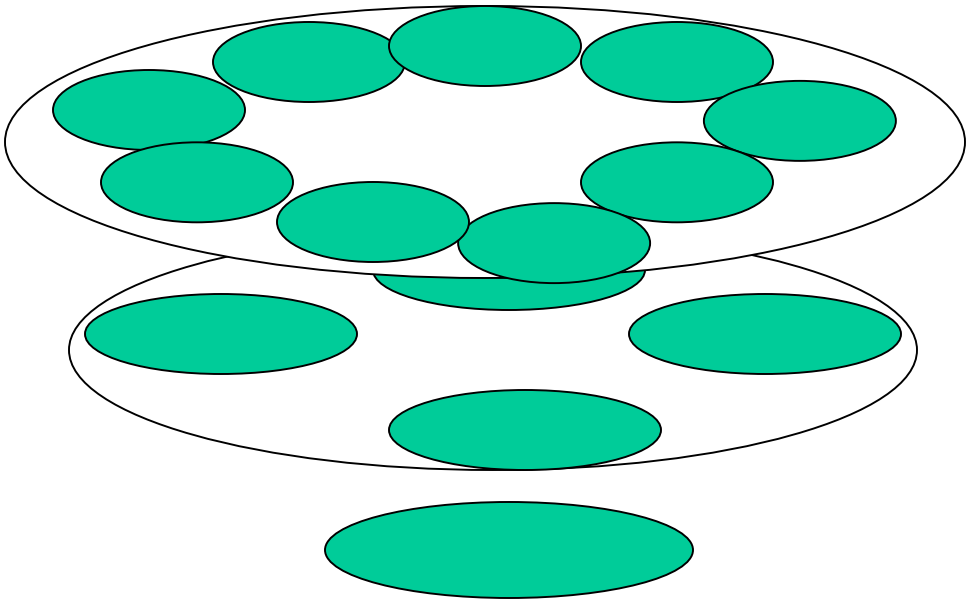


- Congestion Control in TCP was always known to be a stop-gap.
- A DIF always has the potential for the full capability of functions.
- Do flow control (without retransmissions) between intermediate points.
  - Better congestion control, really flow control
  - Allocate different resources to different e-malls.
  - Allows provider much more effective management of resources.
  - Provides means to throttle flows being used for denial of service attacks
  - All of these places? Doubtful. Research topic..

How Does It Work?

The Internet and ISPs

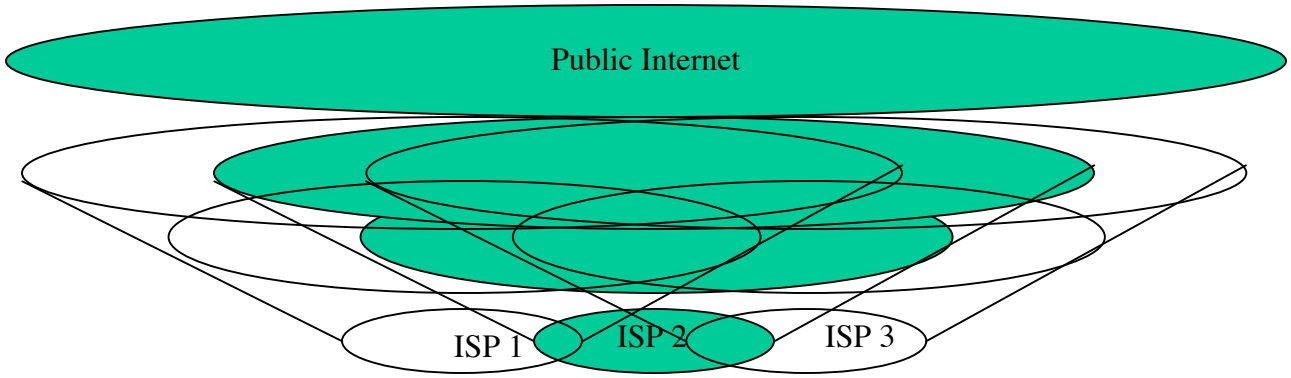
- ISPs have as many layers as they need to best manage their resources.



How Does It Work?

The Internet and ISPs

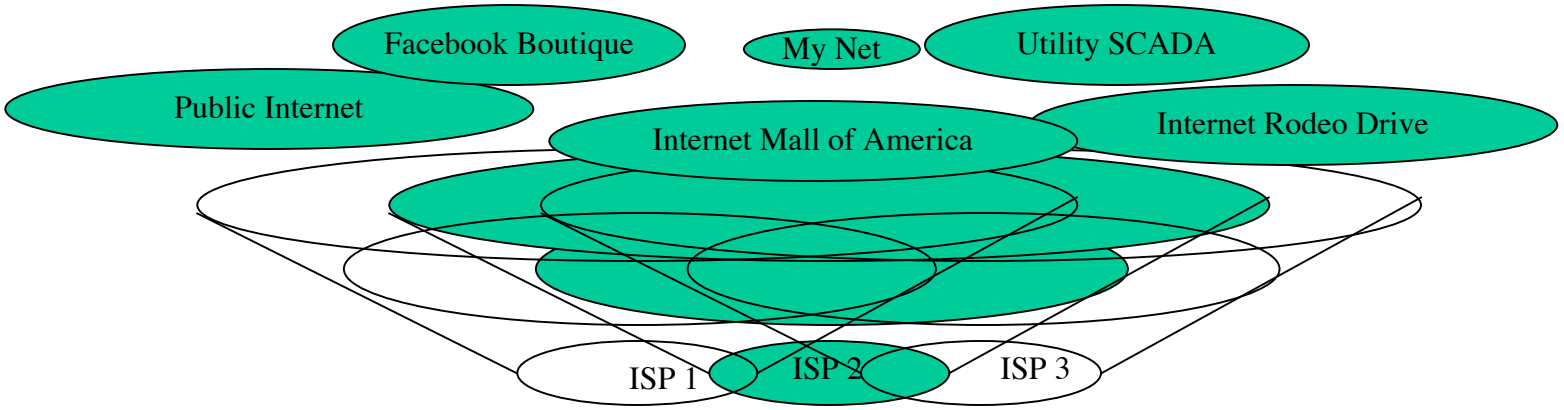
- The Internet floats on top of ISPs, a "e-mall."
  - One in the seedy part of town, but an "e-mall"
  - Not the only emall and not one you always have to be connected to.



How Does It Work?

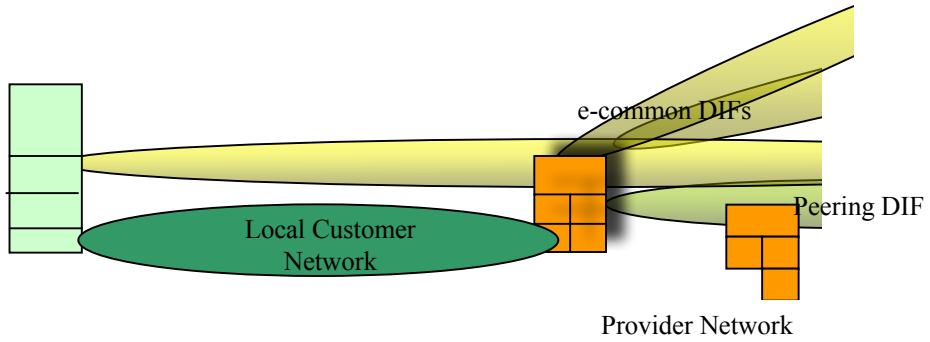
The Internet and ISPs

- But there does not need to be ONE e-mall.
  - You mean!
    - Yes, it is really an INTERnet!



How Does It Work?

The User's Perspective

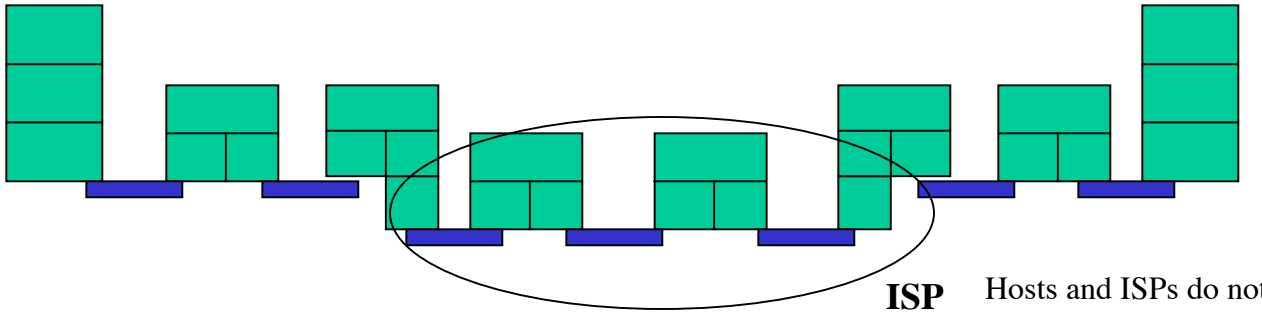


A Customer Network has a border router that makes several e-malls available. A choice can be made whether the entire local network joins, a single host or a single application.

In this case, one host on the local network chooses to join one of the available e-malls.

How Does It Work?

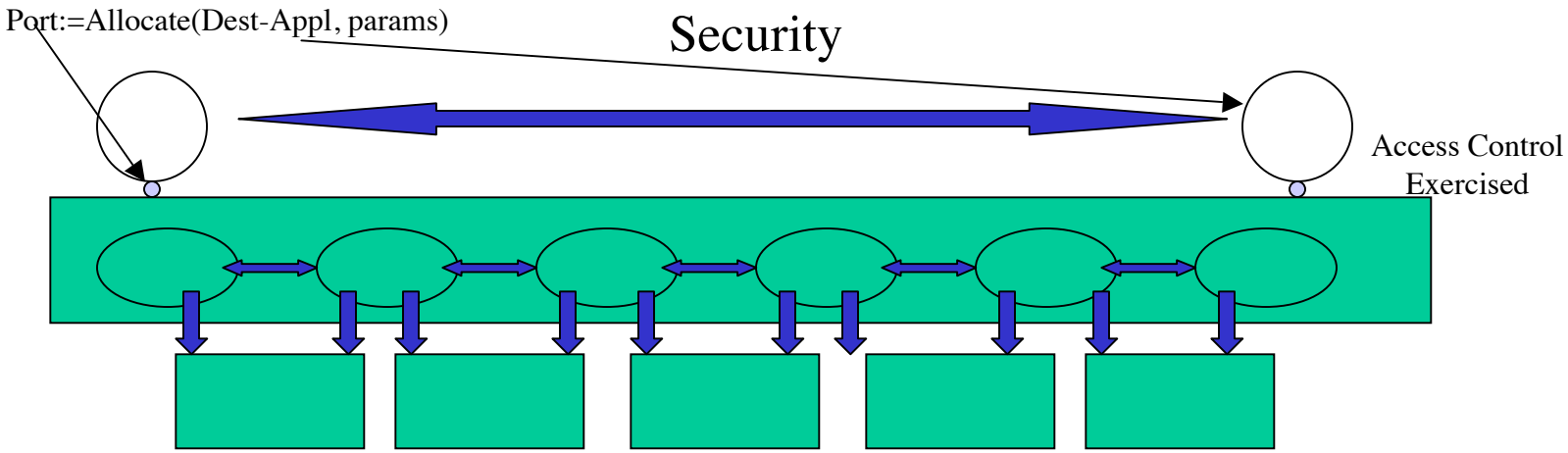
Security



Hosts and ISPs do not share DIFS.
(ISP may have more layers

- Security by isolation, (not obscurity)
- Hosts can not address any element of the ISP.
- No user hacker can compromise ISP assets.
  - Unless ISP is physically compromised.

How Does It Work?

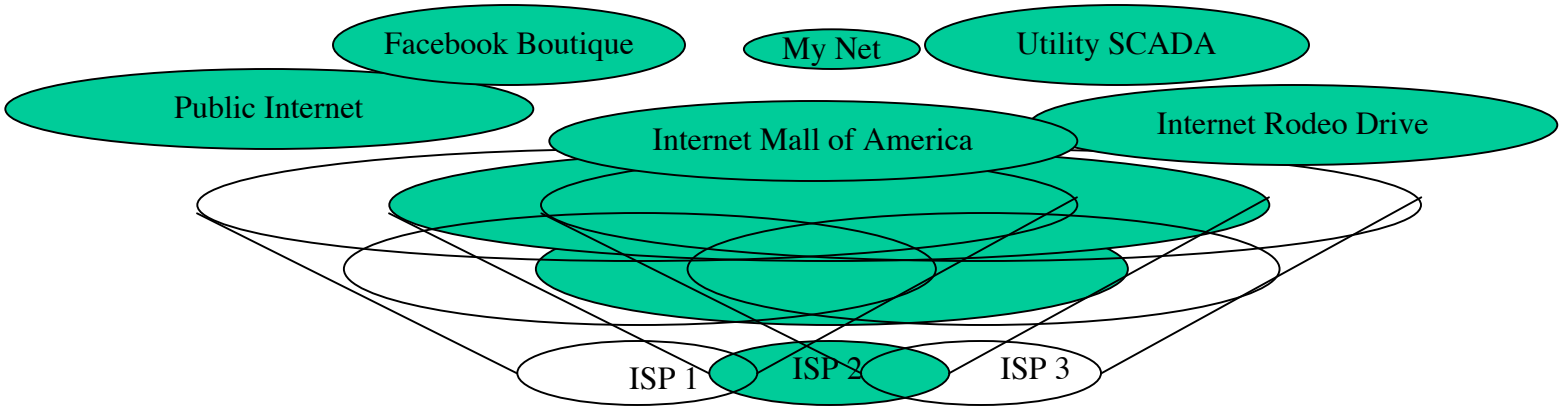


- The DIF is a securable container. DIF is secured not each component separately.
- Application only knows Destination Application name and its local port.
- The layer ensures that Source has access to the Destination
  - Application must ensure Destination is who it purports to be.
- All members of the layer are authenticated within policy.
- Minimal trust: Only that the lower layer will deliver something to someone.
- PDU Protection can provide protection from eavesdropping, etc.
  - Complete architecture does not require a security connection, a la IPsec.

How Does It Work?

Security

- A Hacker in the Public Internet cannot connect to an Application in another DIF without either joining the DIF, or creating a new DIF spanning both. Either requires authentication and access control.
  - Non-IPC applications that can access two DIFs are a potential security problem.



There More to Come

- Next Naming and Addressing
  - It turns out to be quite straightforward and simple.
- Looking at the Internals in a bit more detail
  - What does DTP and DTCP really look like?
  - What does the Flow Allocator and the RIB Daemon do?
  - What goes into defining a DIF?
- A Claim: One will not find a structure that is both as rich and as simple as this that is not equivalent to it. Prove me wrong! ;-)