

Things They Never Taught You About Naming and Addressing (FutureNet Tutorial Part II)

John Day

May 2010

**“Did I ever tell you about Mrs. McCave
Who had 23 sons and she named them all Dave?
Well she did and that was not a very smart thing to do
Because now when she calls, “Yoo-hoo, come into the house, Dave!”
All 23 of her sons come on the run
“And now she wishes that she had named them . . .”**

**<there follows a wonderful list of Dr. Seuss names she wishes she'd
named them and then concludes with this excellent advice.>**

“But she didn't do it and now it is too late.”

**- Dr. Seuss
Too Many Daves**

Going Back to Fundamentals

- Develop the concepts moving from more fundamental to more specific.
 - Fundamentals of Naming
 - Naming in Computing Systems
 - Naming for Communicating Processes
 - Naming for IPC

Names and Name Spaces

- A **name space**, \mathbf{NS} , is a set $\{\mathbf{N}\}$ of names from which all names for a given collection of objects are taken. A name from a given name space may be bound to one and only one object at a time.
- A **name** is a unique string, \mathbf{N} , in some alphabet, \mathbf{A} , that unambiguously denotes some object or denotes a statement in some language, \mathbf{L} . The statements in \mathbf{L} are constructed using the alphabet, \mathbf{A} .
- A function, $\mathbf{M}_{\mathbf{NS}}$, which defines the class of objects, \mathbf{M} , that may be named with elements of \mathbf{NS} . This is referred to as the **scope** of the name space (see below). This may refer to actual objects or the potential for objects to be created.
- A function, $\mathbf{F}_{\mathbf{MNS}}$, that defines the mapping of elements of \mathbf{NS} to elements of \mathbf{M} . This function is one-to-one and onto and is called a **binding**.

Operations on Names

- *Assignment*, allocates a name in a name space, essentially marks it in use. *Deassignment*, removes it from use. Assignment makes names available to be bound. This allows certain portions of a name space to be “reserved,” not available for binding.
- *Binding*, maps a name to an object. Once bound, any reference to the name accesses the object. *Unbinding* breaks the binding. Once unbound, any reference will not access any object.
 - An object ceases to exist when the last name referring to it is unbound.
- Saltzer [1977] defines “resolve” as in “resolving a name” as “to locate an object in a particular context, given its name.”
 - An object cannot be identified without locating it nor located without identifying it.

Types of Names

- Objects may be assigned more than one name. These are called *synonyms* or *aliases*.
 - Objects that may have more than one name, the names are unambiguous.
 - Objects that must have only one name, the names are unique.
- Names may also denote sets of names. Associated with the set is a rule that determines which names are returned when the name of the set is resolved. In networking,
 - A rule that returned all members has been called *multicast*.
 - A rule that returned one member has been called *anycast*.
 - We will refer to all forms as whatevercast!
- Synonyms or names of sets may be taken from the same or different name spaces.
 - The name of an object is the name of a set of one element.

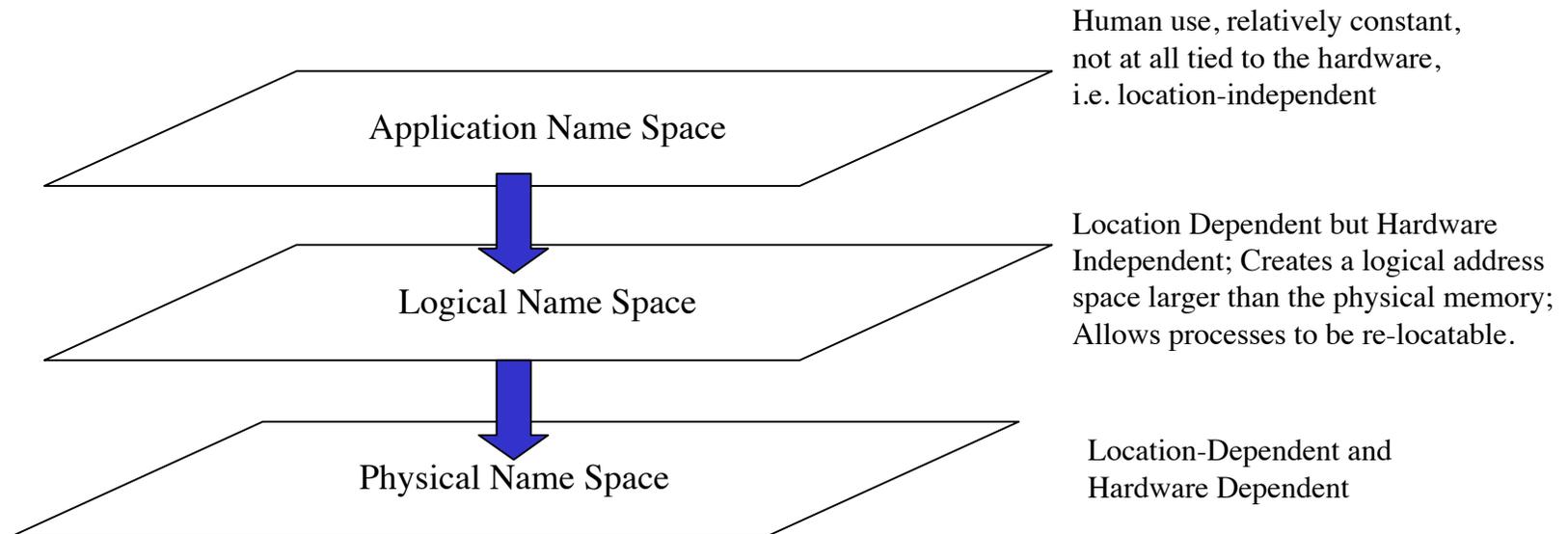
Resolving Names

- There are 2.5 means to resolve names:
 - Exhaustive Search
 - The name provides hints to narrow the search.
- The “half” is indirection:
 - The name or part of the name points to an object that points to a name.
- Hierarchy is the most common form of embedding hints in a name.
 - Hierarchy imposes a topological structure on the name space, which constrains the names that can be assigned to an object.
 - There may be *search rules* for how to utilize the “hints.”

- Every thing up to this point is applicable to all naming in computing systems.

Address Spaces in Operating Systems

(From my OS Course)



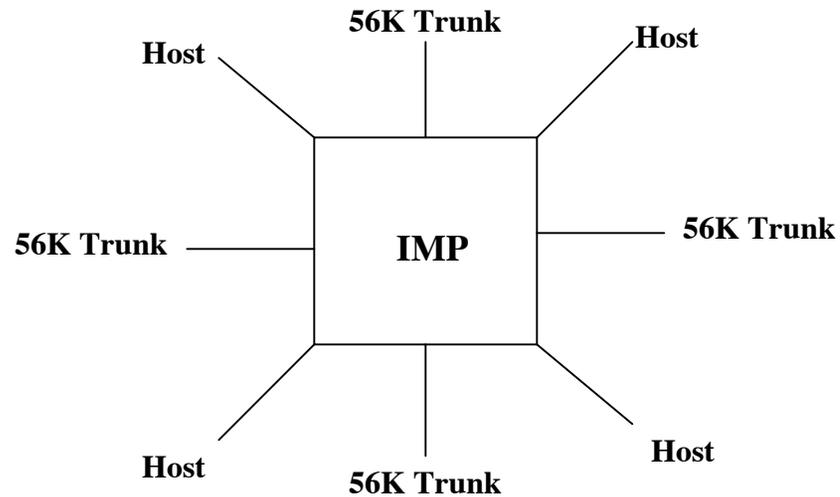
An name space is defined as a set of identifiers with a given scope.

An address space is a location-dependent name space.

In Operating Systems, we have found a need for 3 such *independent* spaces.

Virtually all uses of names in computing are for *locating*.

The Root of Internet Addressing Problems



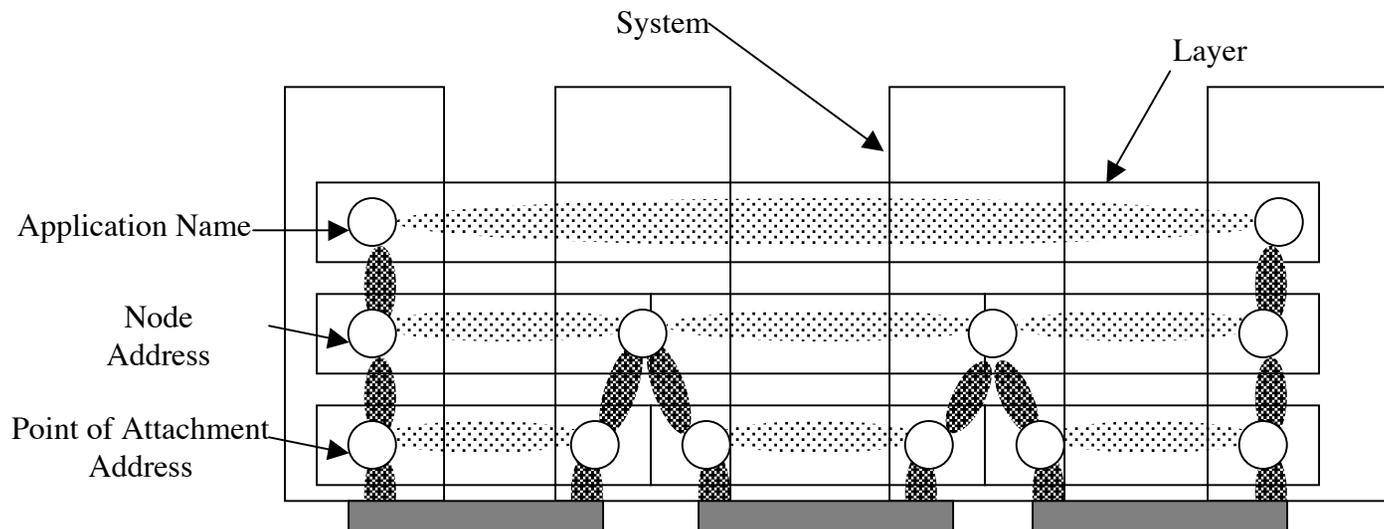
A host's address was its IMP Port Number. It was the common approach at the time. The ARPANet was first. Everyone else afterwards fixed the problem, except the Internet. So What is the Fix?

OS's Were the Guide

- Shoch [1978] put it eloquently:
 - Application names indicate *what* is to accessed
 - Node addresses indicate *where* it is
 - Routes tell you *how* to get there.
- In 1982, Jerry Saltzer published the “other” major paper on addressing.
 - Wrote down the OS view of network addressing:
 - Saltzer, Jerry. “On the Naming and Binding of Network Destinations” in Local Computer Networks, edited by P. Ravasio et al. P North-Holland Publishing Company, pp 311-317, 1982, republished as RFC 1498.
 - Application names that were location independent
 - Node addresses that were location dependent (the logical address)
 - Point of attachment addresses that were route-dependent (the physical address)
 - Routes which were actually a sequence of point of attachment addresses
 - And the three mappings between them.
- Saltzer works through the problem in somewhat more detail.
 - Colored by the hardware of the time and the OS perspective.

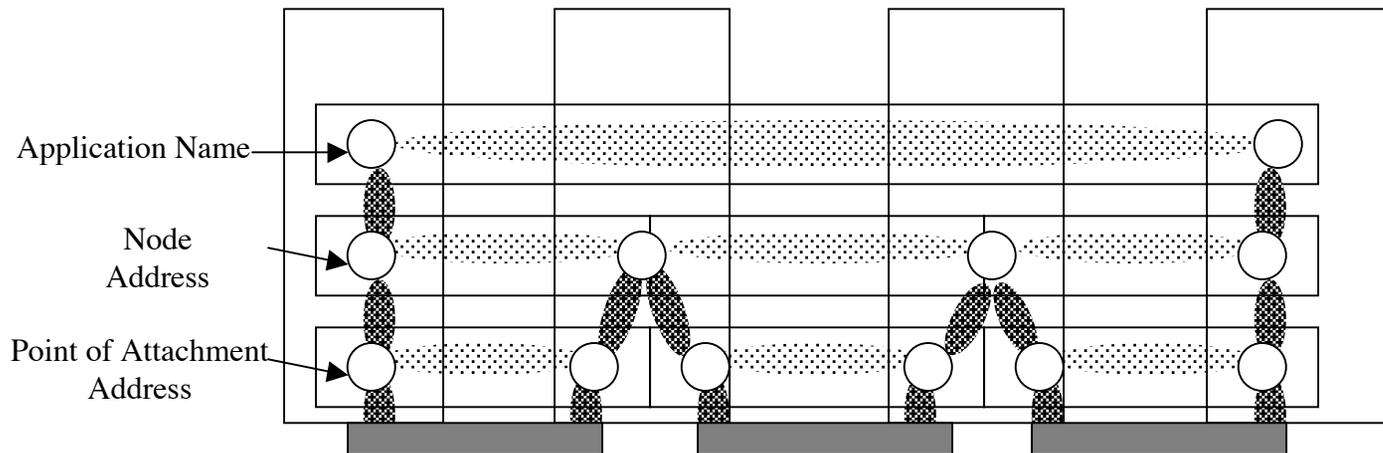
Modeling Saltzer

- A Logical Model of a Network System
 - Naming the binding between a (N)- and (N-1)-PM is equivalent to naming the (N-1)-PM. Need to do one or the other.
- Saltzer seems to favor naming PMs.
 - Since ultimately we must name the application and we know it isn't a binding, naming PMs makes sense.



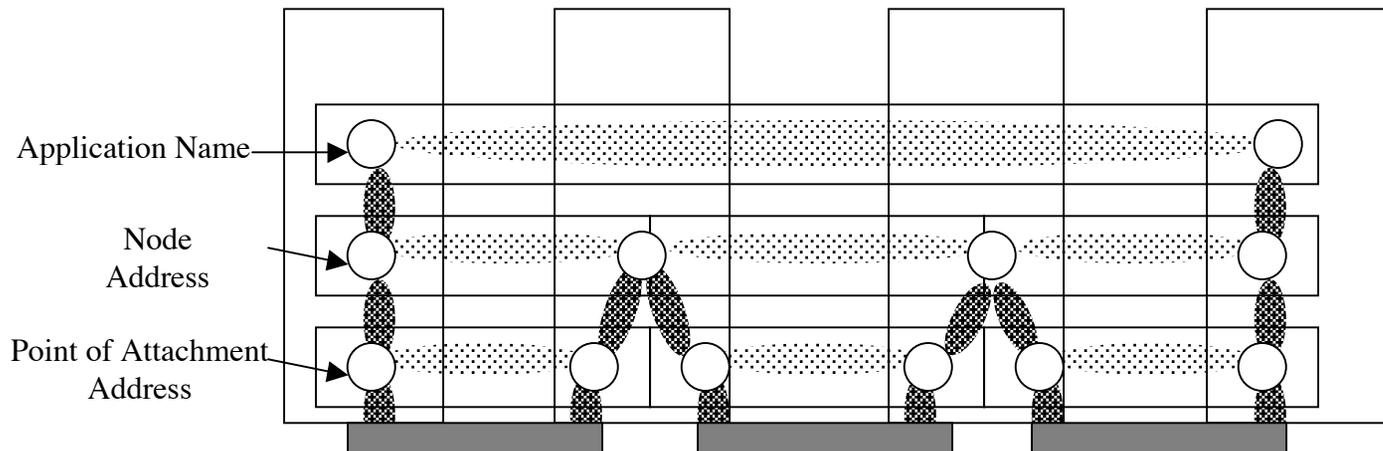
Saltzer's *View of Networks*

- Application names map to node addresses.
- Node addresses map to points of attachment addresses.
- Routes are sequences of points of attachments.
 - Just as in an operating system.
 - But networks are more general than operating systems.



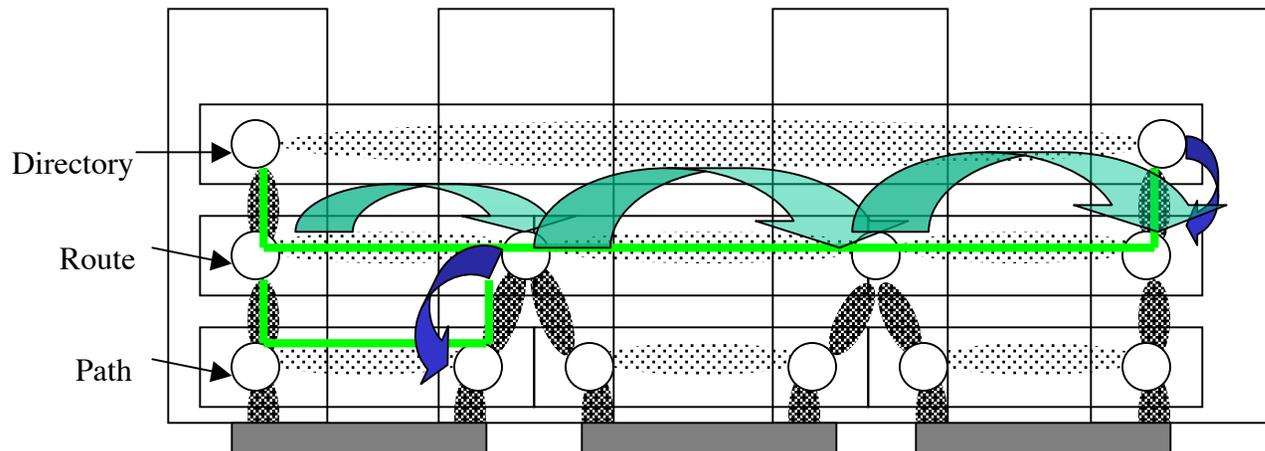
But Saltzer Missed a Case

- There can be More than One path to the Next Hop.
 - This case does not occur in computing systems. There is only one path from the CPU to memory.
- Must route on the Node addresses, not the point of attachments



Generalizing Saltzer to Networks of Networks

- Directory maintains the mapping between Application-Names and the node addresses of all Applications reachable without an application relay.
- Routes are sequences of node addresses used to compute the next hop.
- Node to point of attachment mapping for all nearest neighbors to choose path to next hop. (Saltzer missed this because they hadn't occurred yet.)
- This last mapping and the Directory are the same:
 - Mapping of a name in the layer above to a name in the layer below of all nearest neighbors.



The Real World is More Diverse

(for better or worse, OSI understood that)

Subnet Independent
Subnet Dependent
Subnet-Access

- Two distinct cases: there may be a wire or another network.
 - If the former, then it is a simple Data Link Layer; otherwise
 - Subnetwork Access, Point of Attachment (interface) addresses (X.25, IP).
 - X.25 not data link, because it used HDLC (LAPB) which was.
 - Subnet dependent convergence - if the network needed more error control
 - Subnet Independent (node) addresses at the top of the Network Layer (CLNP)
 - » (Another indicator of a repeating pattern)
- If there is routing in the underlying network, it is that network's issue.
 - Traditional concept of layering caused problems: don't repeat functions.

To Recap

Mappings between Name Spaces

- Mapping between application names and node addresses: Directory.
 - Allows for applications moving to different systems or systems moving.
- Sequence of node addresses: Routes.
 - Node address space is a logical abstraction of point of attachment address spaces.
 - Node address space must be de-coupled from the point of attachment address space(s).
 - Any form that concatenates a local (N)-layer identifier to an (N-1)-address to create an (N)-address will make the node address space route dependent.
 - From this we determine the next hop.
- Mapping of node addresses to point of attachment addresses: Path.
 - A node needs this mapping for itself and all nearest neighbors.
 - Point of attachment addresses distinguish multiple paths to the next node (hop).
 - Note that this is the same as the Directory.
- And it repeats: Node and Point of Attachment are relative, not absolutes.

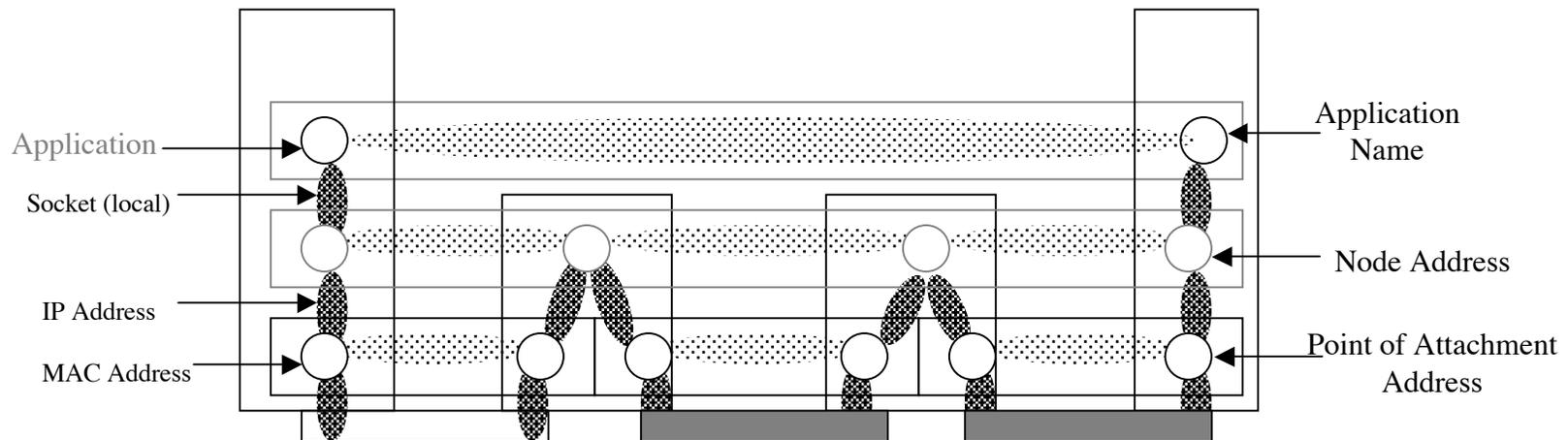
The Fundamental Flaw in the IP Architecture

- Given what we have seen already we can see the mistake,
 - By following Saltzer and routing on the interface, the Internet architecture assumes there is a point-to-point line under IP.
- An *Internet* Protocol should have *networks* under it.
- A Network is a “wire with multiple ends”, i.e. requires addresses.
 - Points of attachment
 - Where are these in the architecture?
- By routing on the interface,
 - IP is a Network Protocol not an *Internet* Protocol
 - After IPng’s refusal to name the node, there has been a search for a workaround. There is none. Loc/id split is simply the last attempt. It is in a very real sense post-IPng trauma.

Applying Results to Real Architectures: The Internet

(This is a *Network* Architecture)

- The most striking feature is that half of the addressing architecture is missing.
 - No wonder there are addressing problems.
 - The only identifier we have for anything is the IP address.
- There are no node addresses and no application names.
 - And the point of attachment is named twice!
 - If this is an *Internet* Protocol, where are the *Network* Addresses?
 - Domain Names are synonyms for IP addresses. URLs are pathnames through the stack and location dependent.



As if your computer worked only with absolute memory addresses.

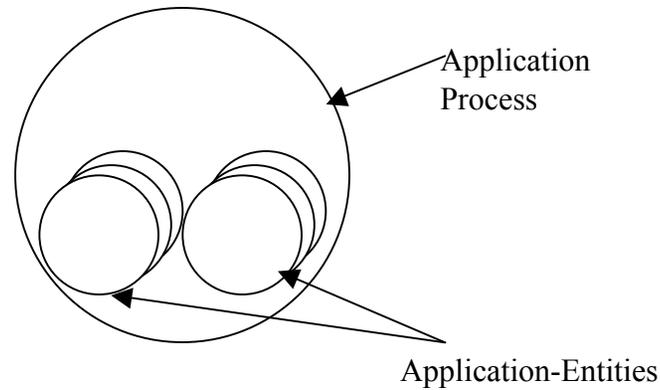
What Does RINA Tell Us?

- If networking is a distributed application that does IPC, we should start there.

Communicating Processes

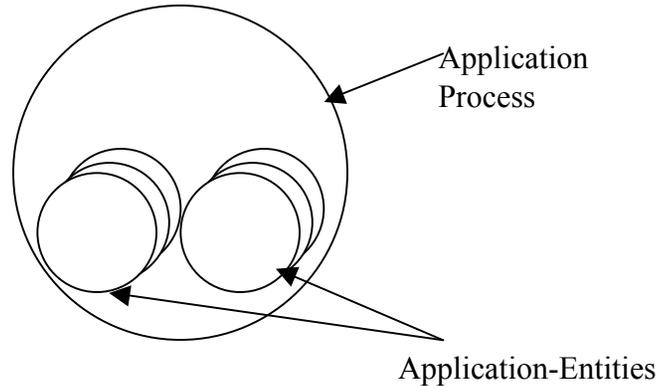
- Application Processes exist to do work. They communicate to do that work. Communicating is not (usually) their primary *raison d'être*.
- So how do applications relate to communication?
 - Communicating applications share state on some things.
 - They can't be unaware of communicating.
 - So what is the nature of the relation between the communication mechanism and the application process?
- Now we need the Application Process/Application Entity distinction

Applications and Communication



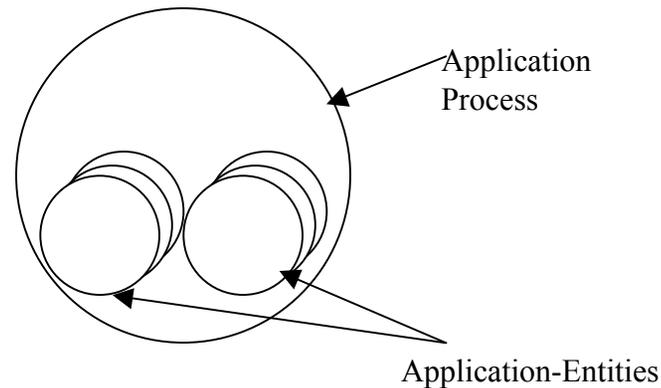
- The Application-Entity (AE) is that part of the application concerned with communication, i.e. shared state with its peer.
- The rest of the Application Process is concerned with the reason for the application in the first place.
- An Application Process may have multiple AEs, they assumed, for different application protocols.
 - An HTTP library linked into a web browser is an AE; FTP is another.

Application Naming



- Application-process names (APN) are globally unambiguous and location-independent, but system-dependent.
 - They may have synonyms of less scope from the same or different name space.
 - There may be multiple instances of the process in the same system.
 - APN-instance-identifiers are unambiguous within the scope of the Application Process.
- Application-entity-identifiers are unambiguous within the application process.
 - There may be more than one Application-entity (AE) in a process.
 - Unambiguous within the scope of the Application Process.
 - There may be more than one instance of each type of Application-Entity.
 - AE-instance-identifiers are unambiguous within the scope of the AE.
- Distributed Application Name is the name of a set of application processes and system-independent.
- Few applications need all of these but a complete theory requires all of them.

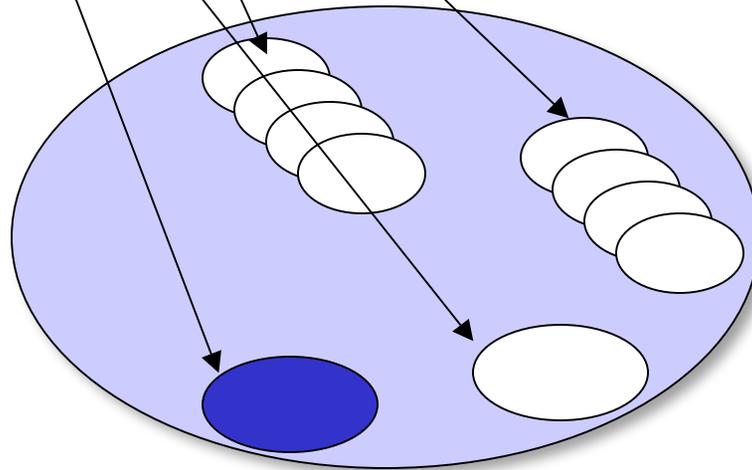
What Good is All This?



- Many capabilities not possible today or that require specific protocols are a consequence of naming and enabled by a complete architecture.
 - Handing off a connection from one system to another;
 - The need to pass IP addresses among applications is avoided;
 - Opening multiple connections with different “protocols” to the same instance of an application process.
 - Connecting to an existing “conference” call, etc.

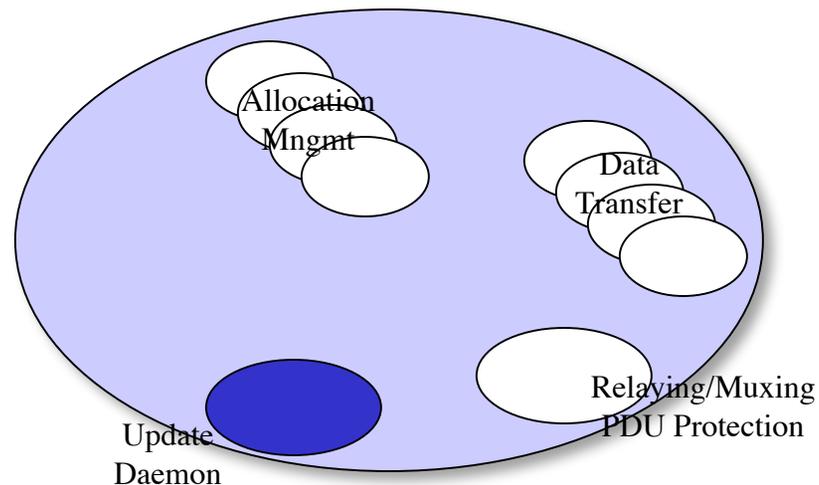
What about IPC?

- The IPC Model says that an IPC Process is simply an application process dedicated to doing IPC. What does this tell us about that?
 - There are three kinds of Application Entities:
 - A Flow Allocator AE to manage the creation of flows,
 - A Data Transfer AE to instantiate a flow, and
 - RIB Daemon AE to maintain shared state among the IPC Processes
 - Managing IPC with the (N-1)-DIF/layer



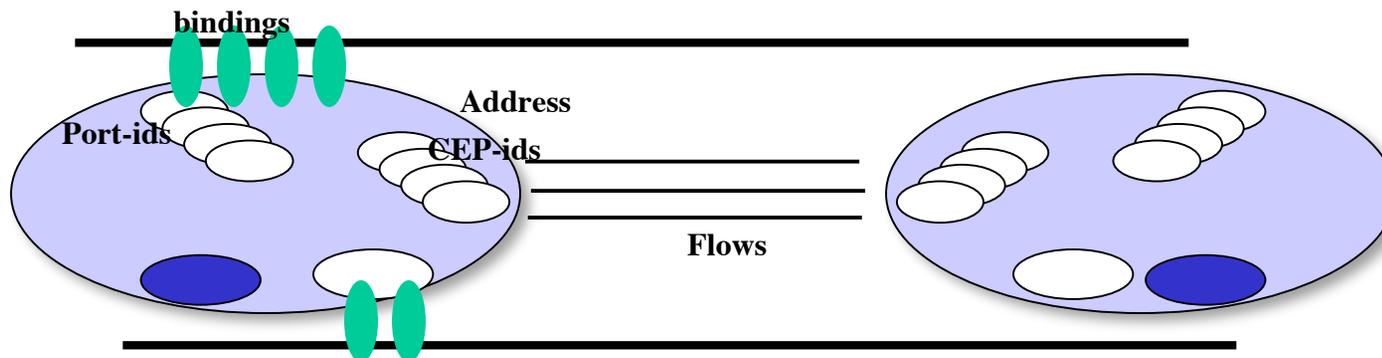
What is the “Application” of an IPC Process?

- What is the main work of an IPC Process? Management!
 - Not, Data transfer.
- Managing the IPC within the Layer.
 - Resource Allocation
 - Maintaining a Resource Information Base
 - Routing
 - Security Management

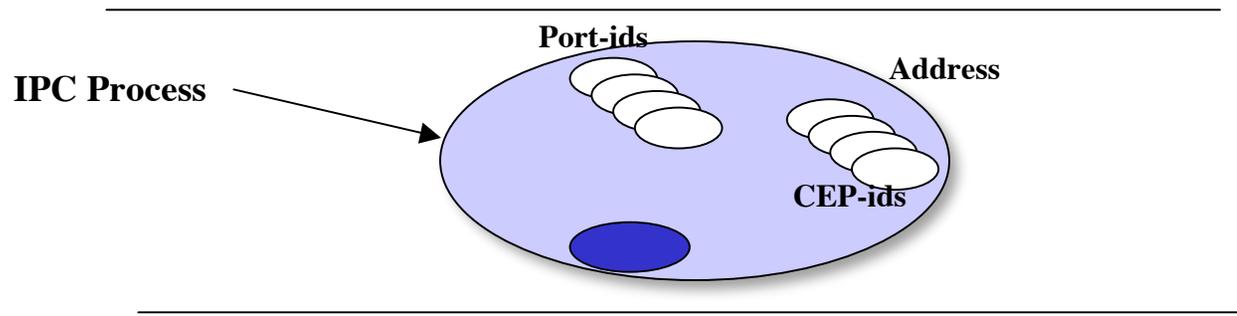


What about IPC Naming?

- An IPC Process has an Application Process Name, give it a synonym of scope limited to the layer, and if structured to facilitate forwarding.
 - Commonly called an *address*
- There are three local identifiers:
 - A Flow Allocator AE Instance Identifier,
 - Commonly called a *port-id*
 - A Data Transfer AE Instance Identifier, and
 - Commonly called a *connection-endpoint-identifier*
 - RIB Daemon AE to maintain shared state among the IPC Processes
 - Traditionally routing update.
- This coupled with delta-t has major implications (later).



Names for IPC



- An *Address* is a synonym for the IPC Process, with scope restricted to the layer and structured to facilitate forwarding within the layer/DIF.
 - A *port-id* is the handle returned to the calling application to refer to this instance of communication, unique within its AE.
 - A *connection-endpoint-id* (CEP-id) identifies the shared state of one end of a flow/connection, unique within its AE.
- A *connection-id* identifies flows between the same two IPC Processes, formed by concatenating CEP-ids, unique within the pair
- Distributed Application Name is globally unambiguous name for the set of all Application Processes in a Distributed Application, e.g. DIF.

To Summarize

<u>Common Term</u>	<u>Scope</u>	<u>Application Term</u>
Application Process Name	“Global” (unambiguous)	Application Process Name
Address	Layer (unambiguous)	Synonym for IPC Process’ Application Process Name
Port-id	Allocation AE (unique)	Allocation AE-Instance-Identifier
Connection-endpoint-identifier	Data Transfer AE (unique)	Data Transfer AE Instance-Identifier
Connection-id	Src/Dest Data Transfer AE (unique)	Concatentation of data-transfer-AE-instance-identifiers
DIF Management Updates	IPC Process (unambiguous)	AE-identifier

- All identifiers except address and connection-id are local to the IPC Process.
 - Scope of the address is the Layer.
 - Scope of the connection-id is the participants in the connection.
- None has more scope than it needs.

Principles of Naming and Addressing: I

- Application names indicate *what*, not where.
 - Hence, names are organized to put similar “whats” near each other.
 - So what “whats” do we use?
 - There is no single answer. Depends on the purpose and the importance of look up performance generally in a distributed directory.
 - The Scope of an Application Name Space is defined by the chain of databases pointed to by the Directory Forwarding Table.
 - Different Application Name Spaces may name the same objects.
 - It is here that the concepts of query and name merge
 - Some schemes will produce multiple results; some a unique result
 - Given the move to greater granularity more schemes will be useful in keeping them tractable and useful.
 - The only requirement for IPC is that some collection of synonyms be associated with an application process and their mapping to the directory are made known.

Principles of Naming and Addressing: II

- Addresses indicate *where* – Not quite
 - Within a DIF, the “what” of interest is “forwarding.” Synonyms are assigned to facilitate forwarding, call it a forwarding-id or f-id.
 - If the DIF has a sufficiently large number of members (or maybe if it doesn’t) to make F-ids more useful.
 - They are structured to reflect which elements are *near* each other for some concept of *near*. This is an address.
 - This is an “address” in the sense of its normal usage, expressing “where” without indicating “how to get there.”
 - An address is an synonym for an IPC Process whose scope is the DIF and structured to be useful within the layer.
 - “where” is just one kind of “what”
 - Application Names and Addresses are simply two different means for locating an object in different contexts. In this case, the object is an IPC Process.

Principles of Naming and Addressing: III

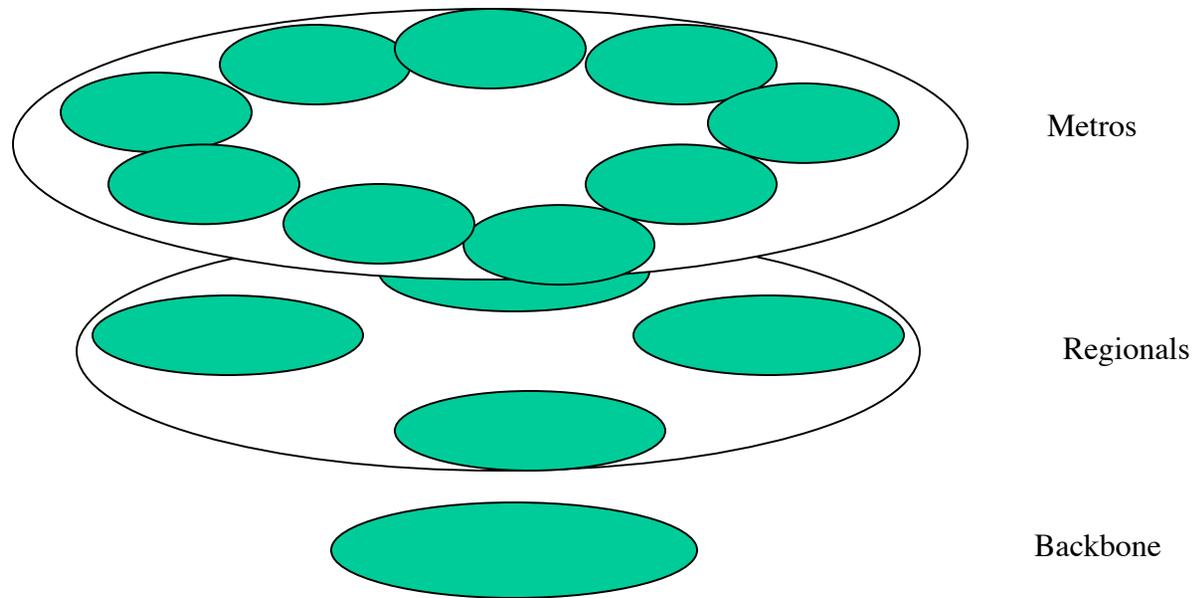
- A F-id (or address) is only unambiguous within the scope of a layer.
 - And no more, otherwise this leads to overloading the semantics of the address and leads to problems
 - MAC addresses are 3 times longer than they need to be.
- Routes are sequences of (node) addresses.
 - Source routing is a “male thing,” not willing to stop and ask directions.
- The relation of node and point of attachment is relative and hence irrelevant.
 - A node address is an (N)-F-id;
 - A point of attachment address is an (N-1)-F-id.
 - A layer routes on its address. A node address routes this layer’s address, the point of attachment address is the address of the node in the layer below and is routed by the layer below, not by this layer.

Principles of Naming and Addressing: IV

- Since the address is structured to facilitate its use within the layer, it must be assigned by the layer.
 - Only the layer knows *how* to make the synonym useful.
 - Any addresses assigned by anyone else are not addresses.
- Names of systems are useful management functions, but not for forwarding.
 - Names of Hosts are distinct and not related to addressing.
- An address should not be constructed by concatenating an identifier in this layer with one from the lower layer, i.e. don't embed MAC addresses in IP addresses.
 - Why? This construction is called a *pathname*, hence it is route-dependent!
 - Addresses in adjacent layers should be completely independent.
 - Hierarchical address assignment within a layer organizes addresses in this layer, not the layer below.
 - Each layer is a level of indirection.

Implications of the Naming Model: I

- Recursion either reduces the number of routes or shortens them.



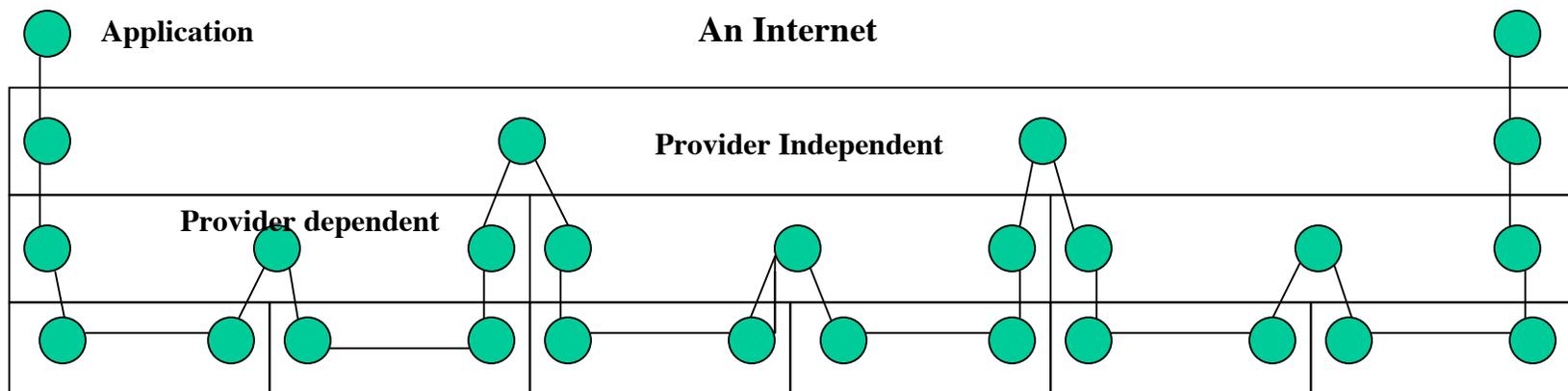
Implications of the Naming Model: III

The Complexity of Routing

	<u>Non-topological</u>	<u>Topological</u>
Metros-DIF 3	$O((2Dn)^2)$	$O((Dm)^2)$ where n is the number of hosts and m is the number of hosts and border routers on a single subnet.
Regionals-DIF 2	$O((2Dn_2)^2)$	$O((Dm_2)^2)$ where n_2 is the number of border routers around the outside and m_2 is number of border routers at this level on a single subnet.
Backbone-DIF 1	$O((2Dn_1)^2)$	$O((2Dn_1)^2)$ where n_1 is the number of border routers on the backbone. Note that $m \ll n$.

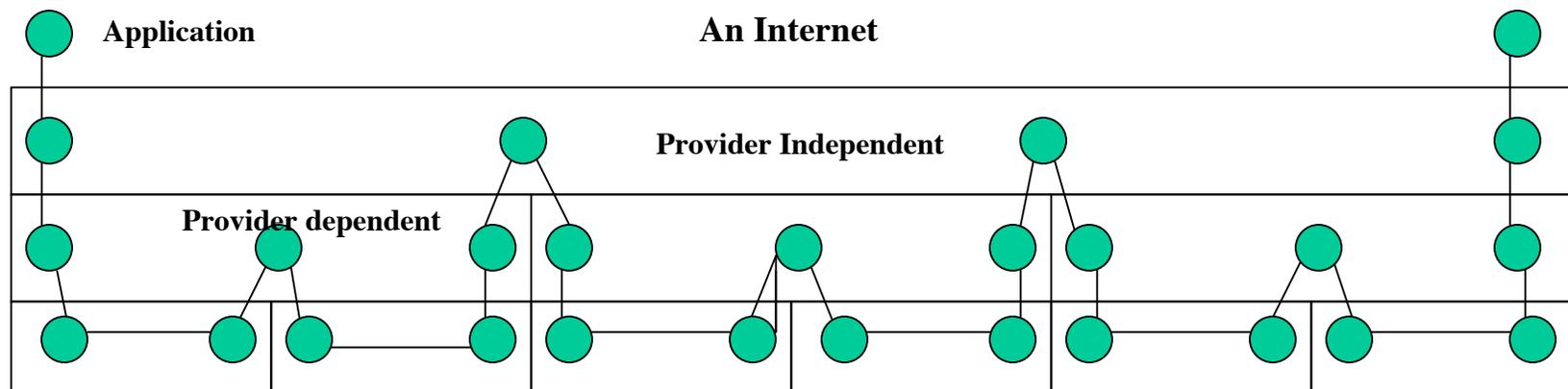
- For the Internet $O((6r)^2)$ where r is the number of routers in the network.

Naming Implications of the Model: II



- Multihoming is a consequence of the structure.
 - Simply a layer having more than one (N-1) binding.
 - Since we are routing in this layer, there is no big deal
 - By routing on the address, the destination address is the destination regardless of which interface the PDU arrived on. It is route-independent.
 - With recursion, no scaling problems, no LISP, no LISP support protocols.

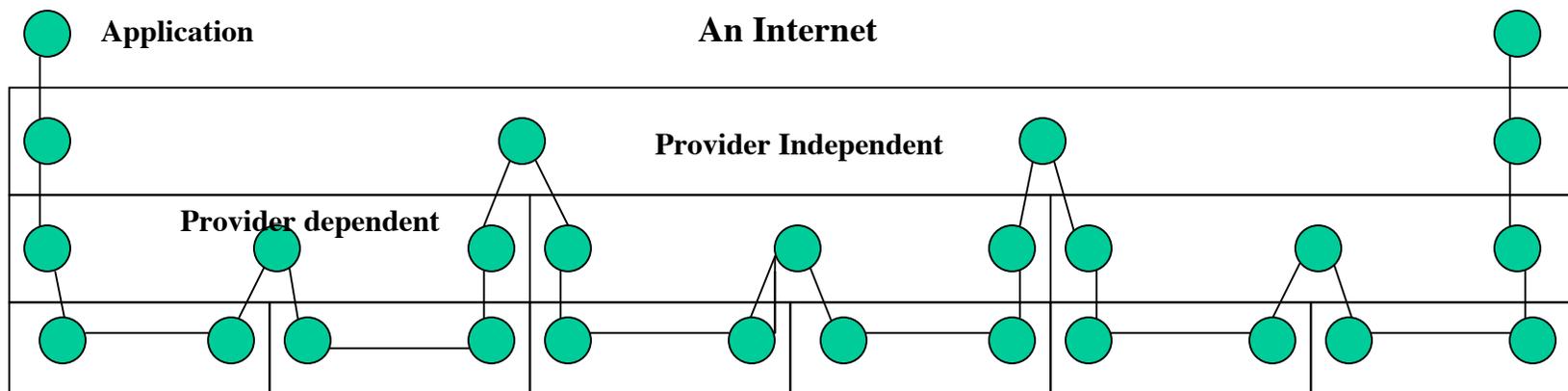
Naming Implications of the Model: III



- Mobility is a consequence of multihoming.
 - Merely acquiring new (N-1)-addresses to use or losing others.
 - No different than so-called “static” case, just more frequent.
 - (N)-address may change either by joining a different DIF
 - Joining another DIF is simply acquiring another point of attachment, another potential path.
 - or if its topological significance changes.
 - Assigning a new address is simply creating adding an additional synonym. Protocol processing is unaffected by the changes.



Naming Implications of the Model: IV

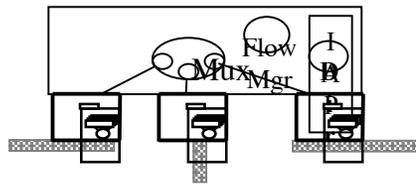


- How to Change the Address of a member of the DIF:
 - Assign the IPC Process a new synonym from the address space;
 - Senders use the new (N)-address as source in all active flows
 - Receivers record the source address of all incoming PDUs as the current address.
 - Advertise routes to the new address, begin to deprecate the old address
 - After a few routing updates, the old address simply disappears.
- Changing addresses is just another name to route to. That two go to the same place is mere coincidence.
- PDU Processing is unaffected by the change.

Implications of the Model: V

Choosing a Layer

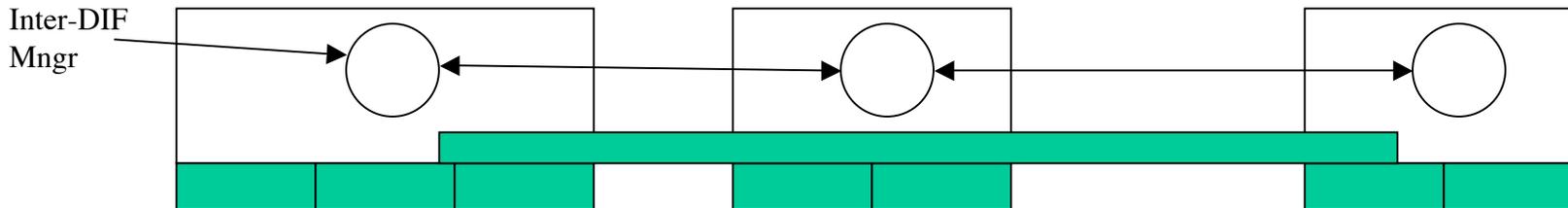
- In building the IPC Model, the first time we had multiple DIFs (data link layers in that case to choose from), we found we needed a task to figure out which DIF to use.



- User didn't have to see all of the wires
 - But the User shouldn't have to see all of the "Nets" either.
- This not only generalizes but has major implications.

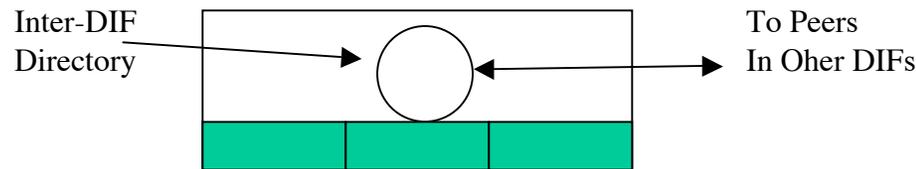
An Inter-Net Directory?

- Inter-DIF Manager determines via what DIFs applications are available.
 - If this system is a not member, it either joins the DIF as before
 - or creates a new one.



- Which Implies that the largest address space has to be only large enough for the largest e-mail.
 - Given the structure, 32 or 48 bits is probably more than enough.
- You mean?
 - Right. IPv6 was a waste of time . . .
 - Twice.

So a Global Address Space is Not Required but Neither is a Global Application Name Space

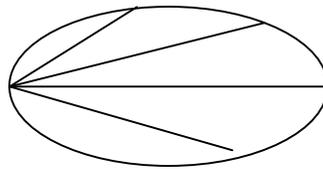


Actually one could still have distinct names spaces within a DIFs (synonyms) with its own directory database.

- Not all names need be in one Global Directory.
- Coexisting application name spaces and directory of distributed databases are not only possible, but useful.
- Needless to say, a global name space can be useful, but not a requirement imposed by the architecture.
- The scope of the name space is defined by the chain of databases that point to each other.

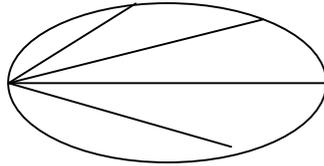
Multicast and Anycast are Simpler Too

- Generalized to Whatevercast:
 - A set and a rule that returns as many members of the set that satisfy the rule.
- Unicast becomes a degenerate case of whatevercast.
 - Forwarding table entry maps Destination Address to a list of next hops. For unicast, the list has one element.
- Primarily handled by hosts or border routers, where all whatevercast traffic is either:
 - On this subnet (only do spanning tree within subnet if there is a lot) or
 - Transit to another subnet, (both cases degenerate to point-to-point).



- So we see Whatevercast devolves into Unicast.

Multicast and Anycast are Simpler Too



- Information in topological addresses imply an approximate spanning tree, which can be used to identify the border routers. Thus, in most cases obviating the need for a separate spanning tree (multicast) routing algorithm.
- And also making it straightforward to multiplex whatevercasts with common sub-trees which will allow even greater efficiency.
 - Note that the common sub-trees do not have to be strict sub-trees but simply have a reasonable degree of commonality to be worthwhile.

Next:
Digging into the Details!
How Does it Really Work?