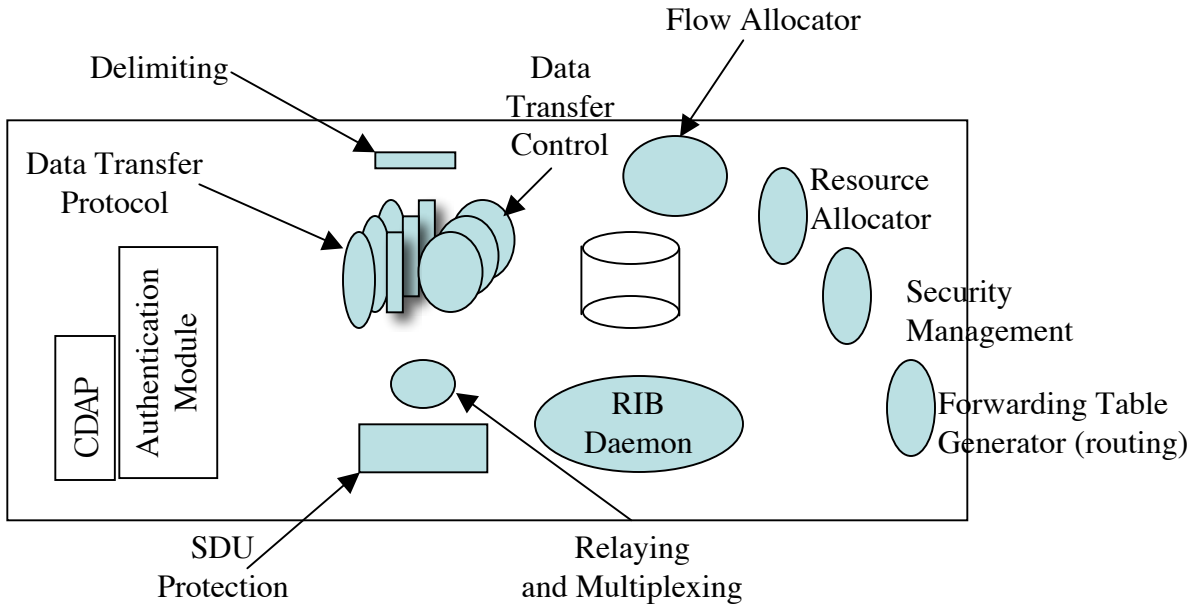
The Nuts and Bolts of RINA

FutureNet Tutorial Part III

John Day

May 2010

The Elements of an IPC Process

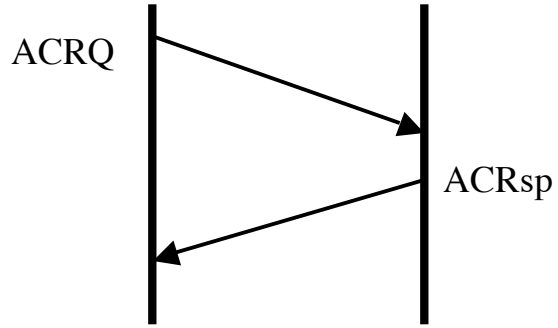


- Lets Look at Each Group of Elements in Turn
  - The Common Distributed Application Protocol
  - IPC Modules
  - IPC Management

Common Distributed Application Protocol

- The only application protocol that is required
  - DAFs may use others for backward compatibility
  - Used in DIFs for all non-data transfer communication
- Three components:
  - Application Connection Establishment
  - Authentication (policy)
  - CMIP-like operations
- Distinct flows allow operations on specific sets of objects

CDAP:
Application Connection Establishment



- Request/Response that carries:
  - Source and Destination Application-Process names and Application-Entity-identifiers (including instances) as necessary.
  - Application-context definition (object set available)
  - Syntax negotiation

CDAP

- Why CMIP? It's a management protocol.
  - Not really, it is a distributed object-oriented intermediate language
  - That does create/delete, read/write (get/put), action (start/stop)
  - With OO-support in Scope and Filter:
    - Scope selects the base object alone, the nth level of the base object, the base object and all its subordinates to and including the nth level, or the base object and all its subordinates.
      - This can be quite powerful in minimizing traffic.
    - Filter is an expression of assertions grouped using AND, OR, and NOT to determine equality, ordering, presence, or set comparison.
      - Extensions to scope and filter should be considered.
- CMIP does everything required and nothing more.

Why Not SNMP?

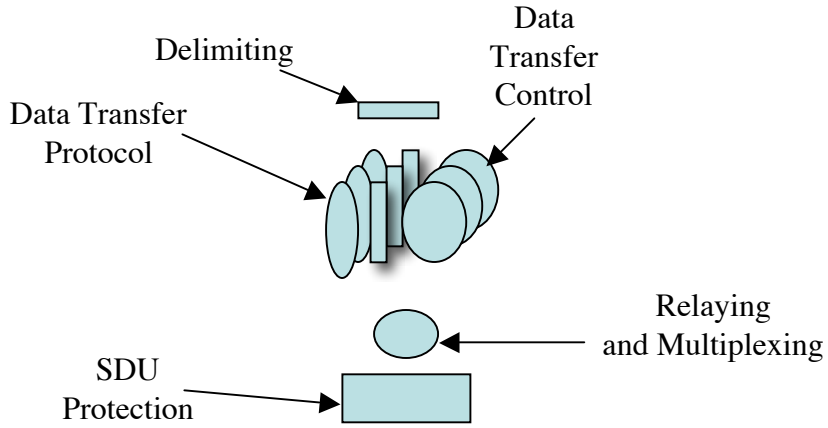
- Too complex.
  - Implementation is larger than CMIP.
- Not object-oriented, no leverage.
- Too many restrictions that contribute to complexity.
  - Can't retrieve entire tables
  - Can't modify multiple attributes in a single operation.
  - Can't, Can't, Can't.
  - It was a good protocol for the mid-80s, but was obsolete by the time it was proposed.

The Use of CDAP

- There are three primary uses of CDAP in a DIF:
  - Joining a DIF
    - Establishing a management connection between an new IPC Process and the members of the DIF, authenticating it, initializing it, and assigning it a forwarding-id.
  - Maintaining the Resource Information Base
    - The RIB Daemon (see below)
  - Flow Allocation (see below)

  - Everything but IPC – the actual data flow – itself

The IPC Modules



- There are Five Modules:
  - Delimiting
  - The Error and Flow Control Protocol consisting of:
    - Data Transfer (Sequencing/Fragmentation)
    - Data Transfer Control (Flow control and rexmsn control)
  - Relaying and Multiplexing
  - SDU Protection

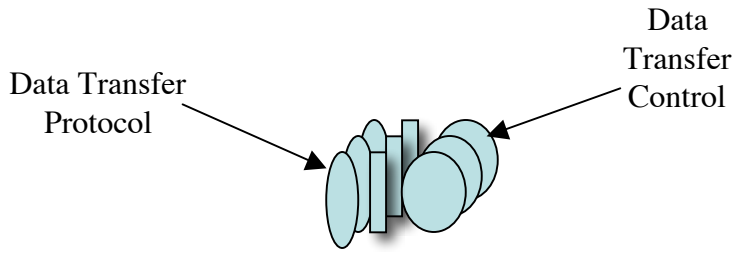
Delimiting

- This Module delimits SDUs to ensure that the service maintains the identity on delivery.
  - IPC may fragment or combine SDUs but will deliver same SDUs to the destination.
- External and Internal Delimiting are possible.
- This module is entirely policy.
  - For a flow that appears to be streaming, the entire flow is a single SDU and early delivery is allowed.

SDU Protection

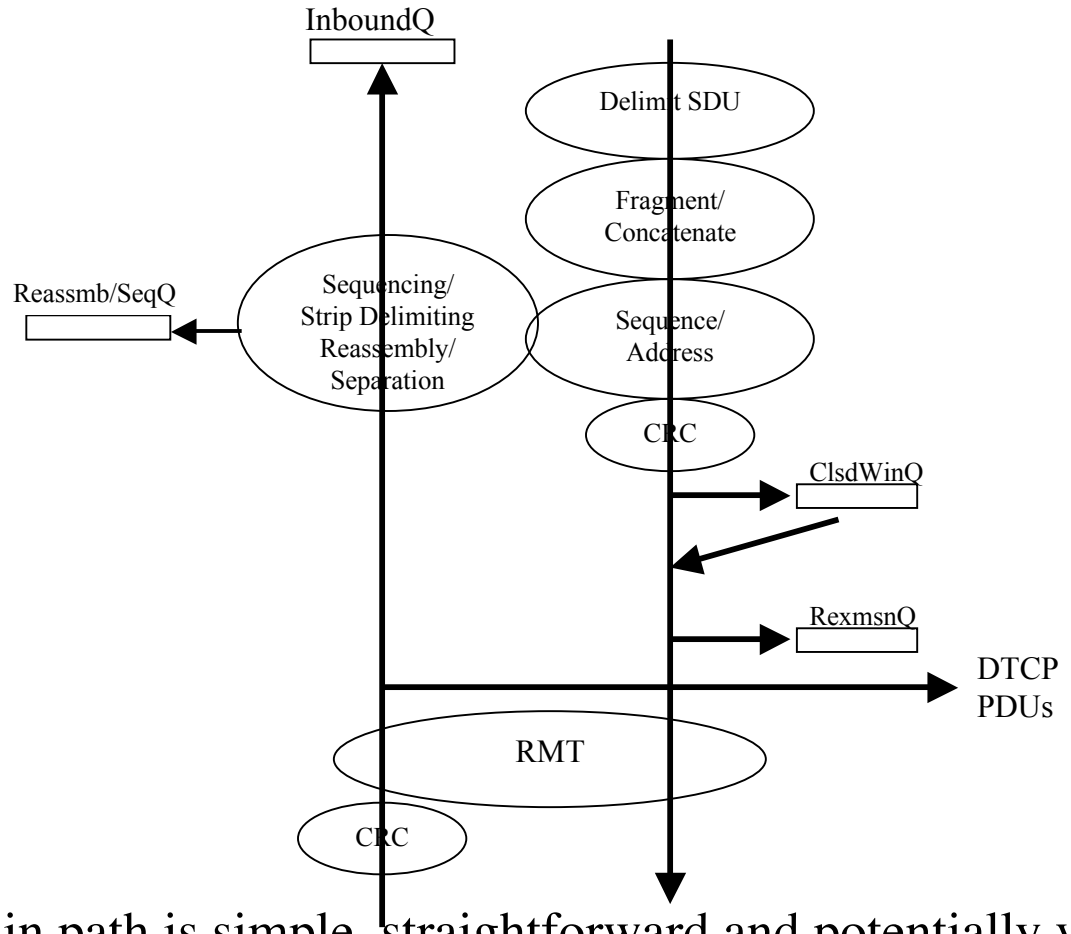
- SDU Protection is also entirely policy and may include:
  - Data Corruption Protection (CRCs or FECs)
  - Time to Live
  - Integrity and Confidentiality (encryption)
    - Below we will explore why the heavy duty mechanisms of IPsec like solutions are not required.
  - Compression - not really protection but this is the right place for it

The Error and Flow Control Protocol



- Based on delta-t with mechanism and policy separated.
  - Naturally cleaves into Data Transfer and Data Transfer Control
    - Data Transfer consists of tightly bound mechanisms
      - Roughly similar to IP+UDP
    - Data Transfer Control, if present, consists of loosely bound mechanisms.
      - Flow control and retransmission (ack) control
- One instance per flow; policies driven by the QoS parameters.
- Comes in several syntactic flavors based on the length of (address, connection-endpoint-id and sequence number)
    - Addresses: 8, 16, 32, 64, 128, variable.
    - CEP-id: 8, 16, 32, 64
    - Sequence: 4, 8, 16, 32, 64

Data Transfer



- The main path is simple, straightforward and potentially very fast (see specifications for details).

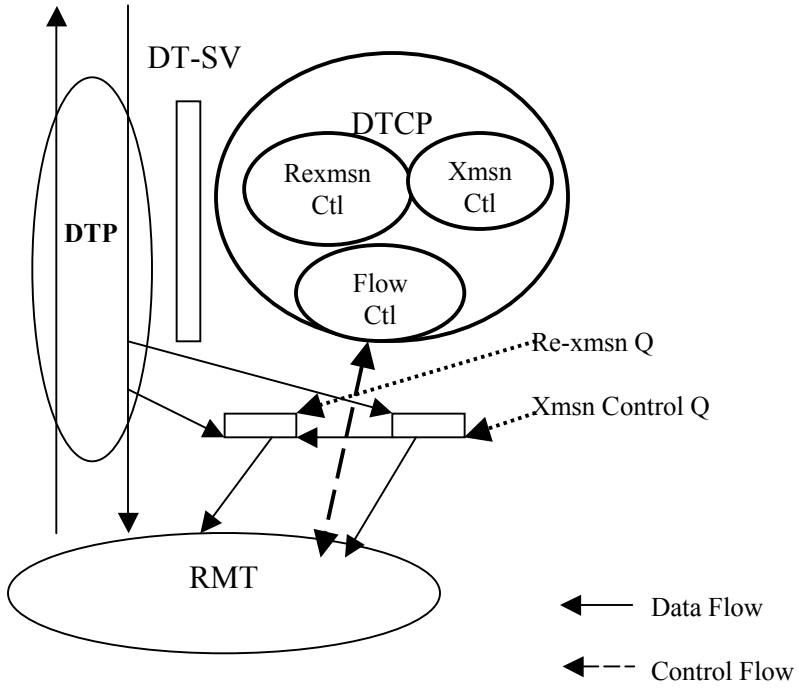
Data Transfer PDU

- Version: 8 Bit
- Destination-Address: Addr-Length
- Source-Address: Addr-Length
- Flow-id: Struct
  - QoS-id: 8 Bit
  - Destination-CEP-id: Port-id-Length
  - Source-CEP-id: Port-id-length
- PDUType: 8 bits
- Flags: 8 bits
- PDU-Length: LengthLength
- SequenceNumber: SequenceNumberlength
- Sequence User-Data{DelimitedSDU* | SDUFrag}

Data Transfer Policies and Parameters

- **UnknownFlowPolicy** – When a PDU arrives for a Data Transfer Flow terminating in this IPC-Process and there is no active DTSV, this policy consults the ResourceAllocator to determine what to do.
- **SDUReassemblyTimer Policy** – this policy is used when fragments of an SDU are being reassembled and all of the fragments to complete the SDU have not arrived. Typical behavior would be to discard all PDUs associated with the SDU being reassembled.
- **SDUGapTimer Policy** – this policy is used when the SDUGapTimer expires and PDUs have not been received to a sequence of SDUs with no gaps greater than MaxGapAllowed. Typically, the action would be to signal an error or abort the flow.
- **ClsdWindPolicy** - This policy determines what to do if the PDU should not be passed to the RMT.
- **MaxPDUSize** – The maximum size in bytes of a PDU in this DIF.
- **MaxFlowPDUSize** – The maximum size in bytes of a PDU on this Flow.
- **SeqRollOverThres** – The value at which a new flow is created and assigned to this Port-id to support data integrity.
- **MaxGapAllowed** – The maximum gap in SDUs that can be delivered to the (N)-DIF-port without compromising the requested QoS.

Data Transfer Control



- Control stays out of the main data flow.
- This module will not exist for flows that don't need it.

Data Transfer Common Control Syntax

Common Control PDU
Version: 8 Bits
Destination-Address: Addr-Length
Source-Address: Addr-Length
Flow-id: Struct
    QoS-id: 8 bits
    Destination-CEP-id: CEP-id-Length
    Source-CEP-id: CEP-id-length
PDUType: 8 bits
Flags: 8 bits
PDU-Length: LengthLength
SequenceNumber: SequenceNumberLength

Ack/Flow ControlPDU
Common Control PDU
PDU TYPE = X'800C' Ack only
PDU TYPE = X'800D' Ack and Flow Control
PDU TYPE = X'8009' Flow Control only
Ack: Integer(SeqNbrLength)
RightWindowEdge:SequenceNbrLength
NewRate: RateLen
TimeUnit: TimeLen

Selective Ack PDU
Common Control PDU
PDU TYPE = X'8004'
Ack/Nack: Integer(SeqNbrLength)
Ack/Nack List Length: Integer(8)
Ack/Nack List: Sequence(StartingNbr Integer (SeqNbrLength), Ending Integer(SeqNbrLength))

Forced NackPDU
Common Control PDU
PDU TYPE = X'8006'
Ack/Nack: Integer(SeqNbrLength)
Ack/Nack List Length: Integer(8)
Ack/Nack List: Sequence(StartingNbr Integer(SeqNbrLength), Ending Integer(SeqNbrLength))

Data Transfer Control
General Parameters and Policies

- **TA –** Maximum time an ack is delayed before sending
- **TG –** Maximum time to exhaust retries.
- **TimeUnit –** for rate based flow control, i.e. # of PDUs sent per TimeUnit
- **FlowInitPolicy –** Data Transfer Control initialization policy
- **SVUpdatePolicy –** Updates the State Vector on arrival of a TransferPDU
- **LostControlPDUPolicy –** What to do if a Control PDU is lost?

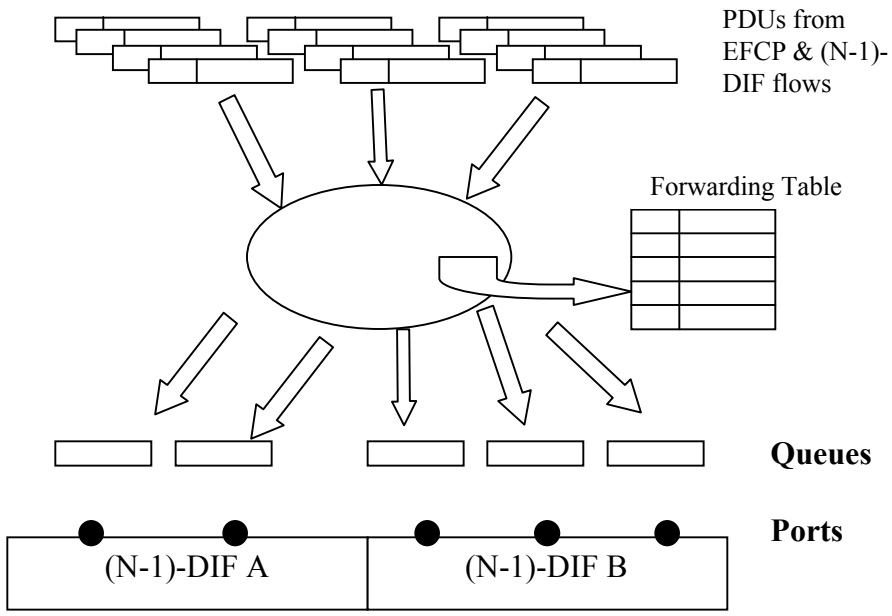
Data Transfer Control Retransmision Policy

- **RTTEstimator Policy –** the algorithm for estimating RTT
- **RetransmissionTimerExpiryPolicy** - what to do when a Retransmission Timer Expires, if the action is not retransmit all PDUs with sequence numbers less than this.
- **ReceiverRetransmission Policy** - This policy is executed by the receiver to determine when to positively or negatively ack PDUs.
- **SenderAck Policy** - provides some discretion on when PDUs may be deleted from the ReTransmissionQ. This is useful for multicast and similar situations where one might want to delay discarding PDUs from the retransmission queue.
- **SenderAckList Policy** - similar to the previous one for selective ack

Data Transfer Control Flow Control Policies

- **InitialCredit Policy** - sets the initial amount of credit on the flow.
- **InitialRate Policy** - sets the intial sending rate to be allowed on the flow.
- **ReceivingFlowControlPolicy** - on receipt of a Transfer PDU can update the flow control allocations.
- **UpdateCredit Policy** – determines how to update the Credit field, i.e. whether the value is absolute or relative to the sequence number.
- **FlowControlOverrun Policy** - what action to take if the credit or rate has been exceeded.
- **ReconcileFlowConflict Policy** - when both Credit and Rate based flow control are in use and they disagree on whether the PM can send or receive data.

Relaying and Multiplexing Task

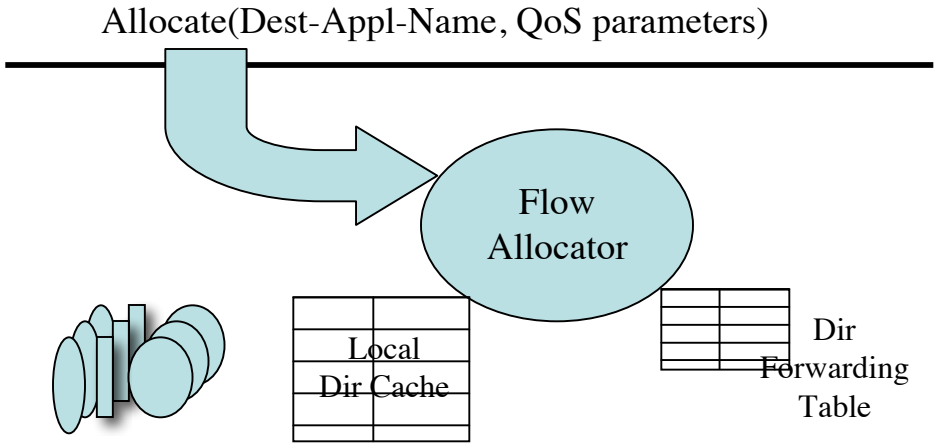


- Queues at the top are at least one per QoS Class.
- Queues at the bottom are created by the Resource Allocator and may be related to the number of QoS Classes provided by the lower DIF.

RMT Policies

- RMTQMonitorPolicy – Policy for monitoring the status of the RMT and the QoS being provided by the (N-1)-DIF from data available to the RMT.
- RMTSchedulingPolicy – Policy determines what flows are mapped to what RMT queues and how the queues are serviced.
- MaxQPolicy – Policy invoked if a queue reaches its limit.

Flow Allocator



- When Application Process generates an Allocate request, the Flow Allocator creates a flow allocator instance to manage each new flow.
- The Instance is responsible for managing the flow and deallocating the ports
  - DTP/DTCP instances are deleted automatically after 2MPL with no traffic,
- When it is given an Allocate Request it does the following:

Flow Allocator

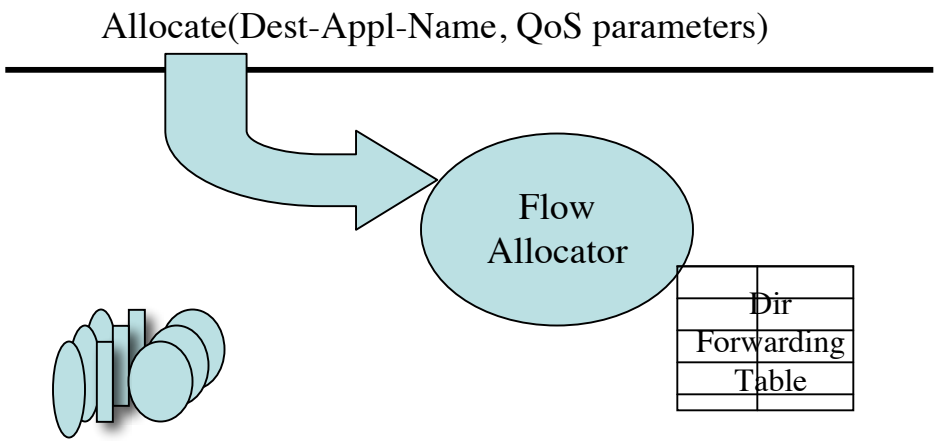
Input: Allocate Request

1) It inspects the Allocate request and maps the parameters to the appropriate QoS Class and the associated policy set.
2) It instantiates a DTP (and DTCP if necessary) for this flow.
3) Checks its local directory cache for the destination application name. If found, it sends a Create Flow request to destination address; otherwise consults the Dir Forwarding Table and sends the Create Flow request to the address noted there.
4) When it receives an Create Flow Response it executes an Allocate Response API call and modifies the state of the DTP as necessary.

Flow Allocator
Input: Create Flow Request

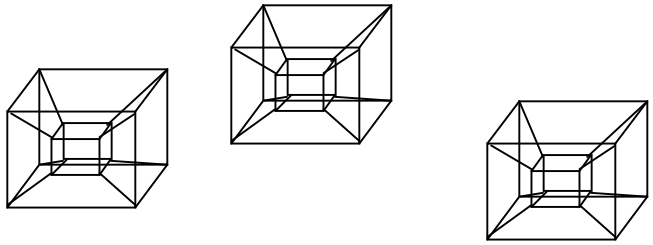
- Inspect local Directory Cache for an entry that indicates where to forward the request.
  - Each look up hopefully gets it closer to its destination.
  - The directory forwarding table can create a hierarchy of databases.
- When the lookup finds that the application is here, create a new flow allocator instance to manage this end of the flow, do access control, initiate the application if necessary, and give it the Allocate indication.
- When the Application responds, if positive, create the local DTP/DTCP instances, allocate local ports: if negative discard the flow allocator instance; Regardless send the create flow response to the source address with the appropriate outcome.

Flow Allocator
Subtleties



- Separating port allocation from synchronization eliminates well-known ports, and several SYN hijacking attacks.
  - Using Delta-t is inherently more secure the TCP
- The Create Flow allows for a list of connection-endpoint-ids that can be used either in parallel or serially.
  - In parallel, might be used for things like p2p [sic] do.
  - Used serially, avoids the need for a separate security connection as in IPsec.

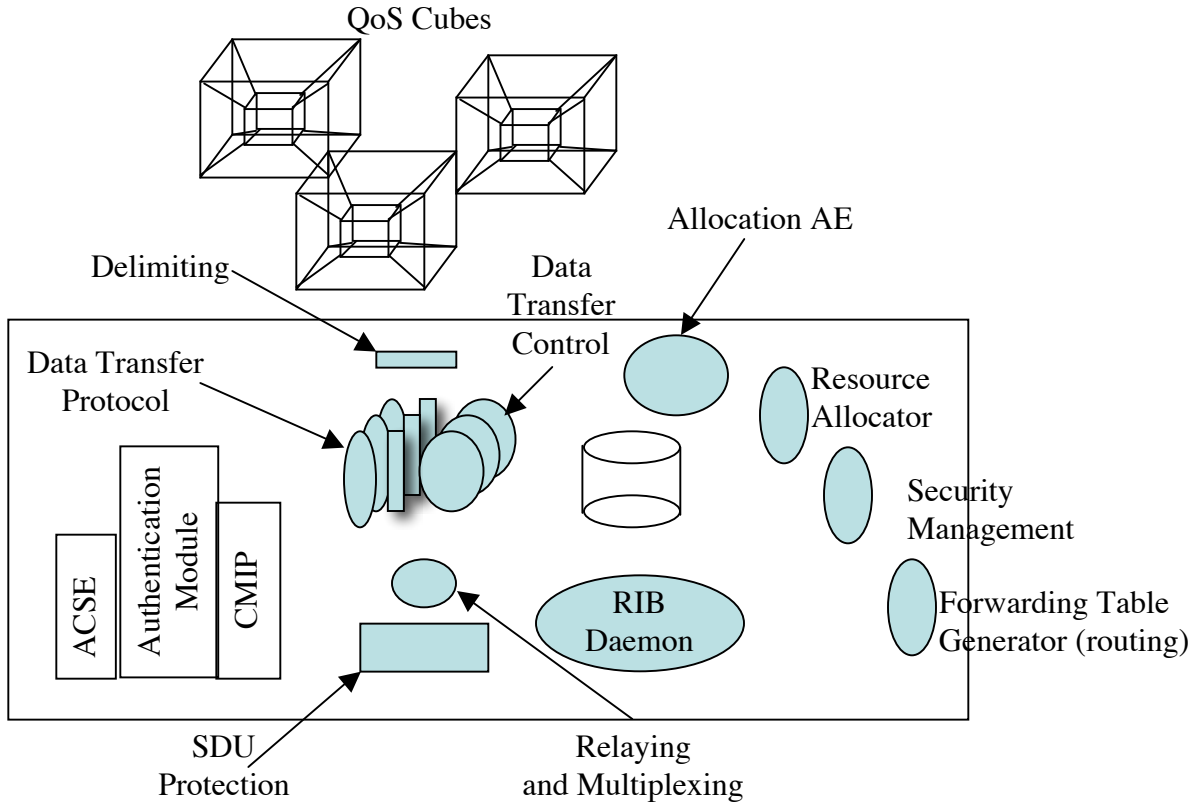
Requests are mapped to QoS Cubes



- Systems are not sufficiently sensitive to be able to deliver precise QoS values. The best that can be done is a range. Parameters identified are:

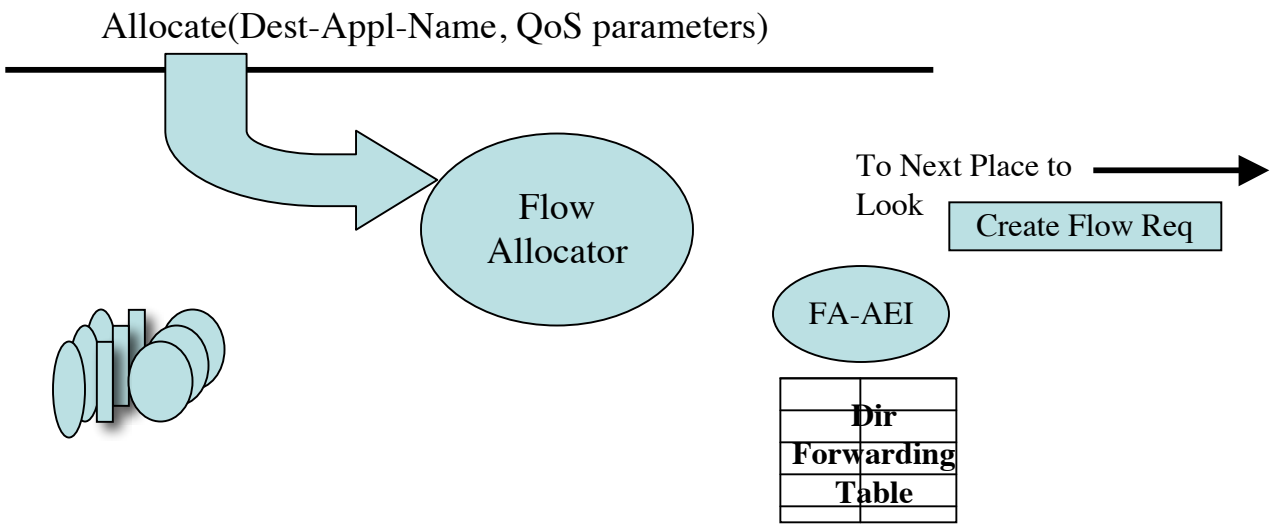
Average Bandwidth	Undetected bit error rate
Average SDU bandwidth	Partial Delivery
Peak bandwidth-duration	Order
Peak SDU bandwidth-duration	Max allowable gap in SDUs
Burst period	Delay
Burst duration	Jitter

Protocols are Irrelevant



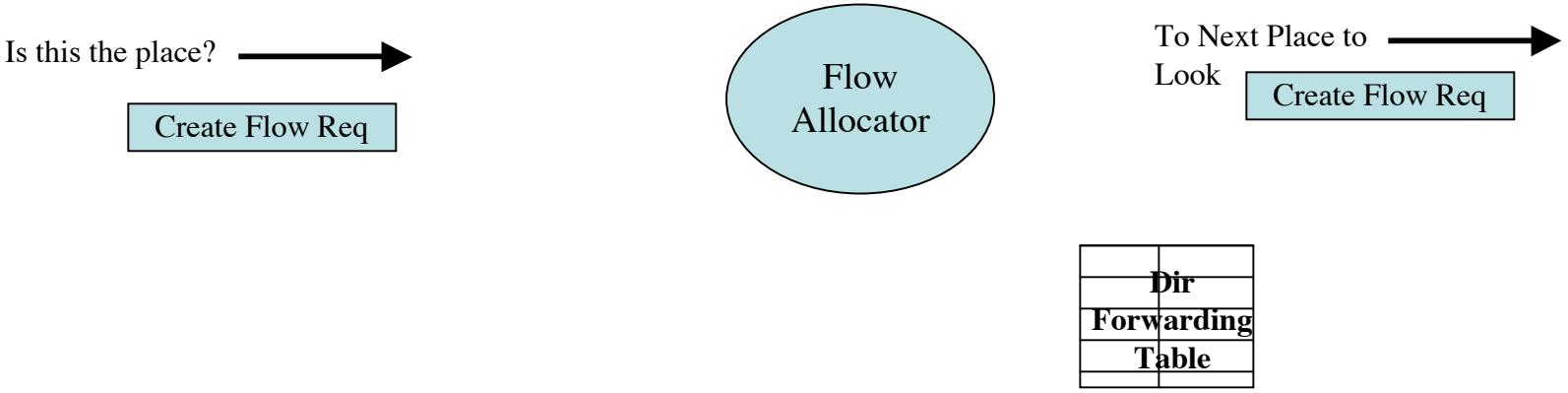
- Notice that we are defining policy sets driven by QoS classes and objects for management that configure a DIF.
- Protocol becomes non-issue.

So How Does a Flow Get Allocated?



- An Application issues an Allocate Request to the Flow Allocator.
- If well-formed, spawns an instance to manage the request and the flow.
- The Flow-Allocator instance looks up the destination-application-name in its local cache and finds an address to look for the requested application.
- It then instantiates an EFCP instance (whose id is the CEP-id).
- Forms a Create flow request and sends it.

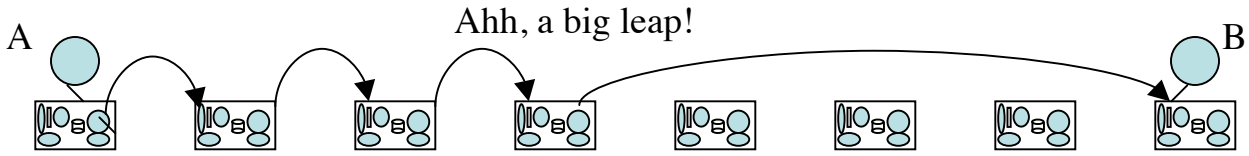
At The Next Place To Look



- Create Flow Request arrives at the next place to look.
- Flow Allocator looks up the destination application name in its “local cache”
- Address returned isn’t ours, so not here.
- Send it there

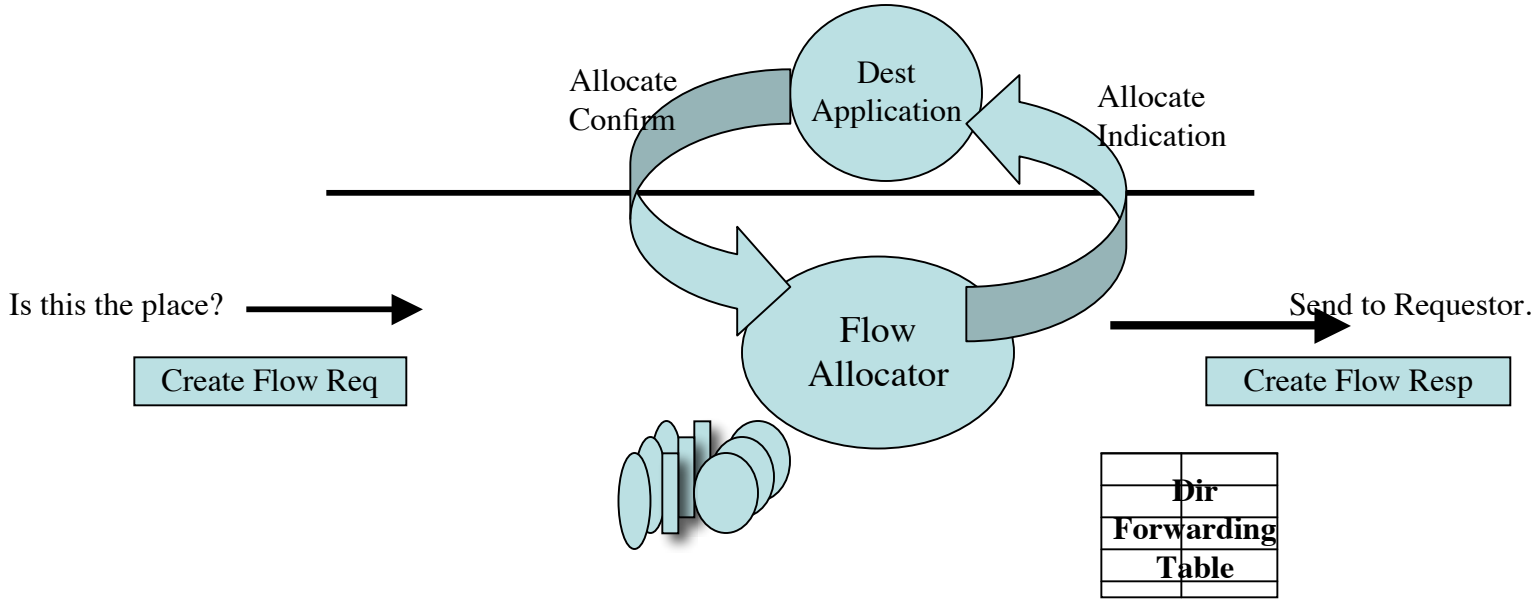
Stepping Back

Establishing Communication



- Remember this?
- We go along looking for places to look for the requested application.

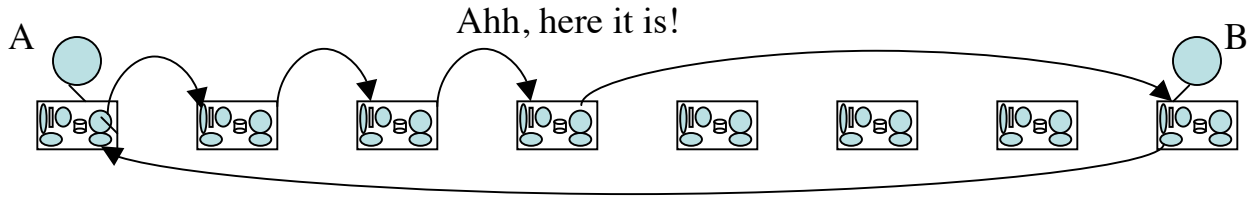
Yet Another Place to Look



- Create Flow Request arrives at yet another place to look.
- Flow Allocator looks up the destination application name in its "local cache"
- Address returned is ours. Its here! Check requestors credentials, if okay.
- Deliver Allocate indication to the destination application
- Which accepts or rejects, if accepts sends a Create Flow Response and
- Instantiates a EFCP-instance
- The connection is established.

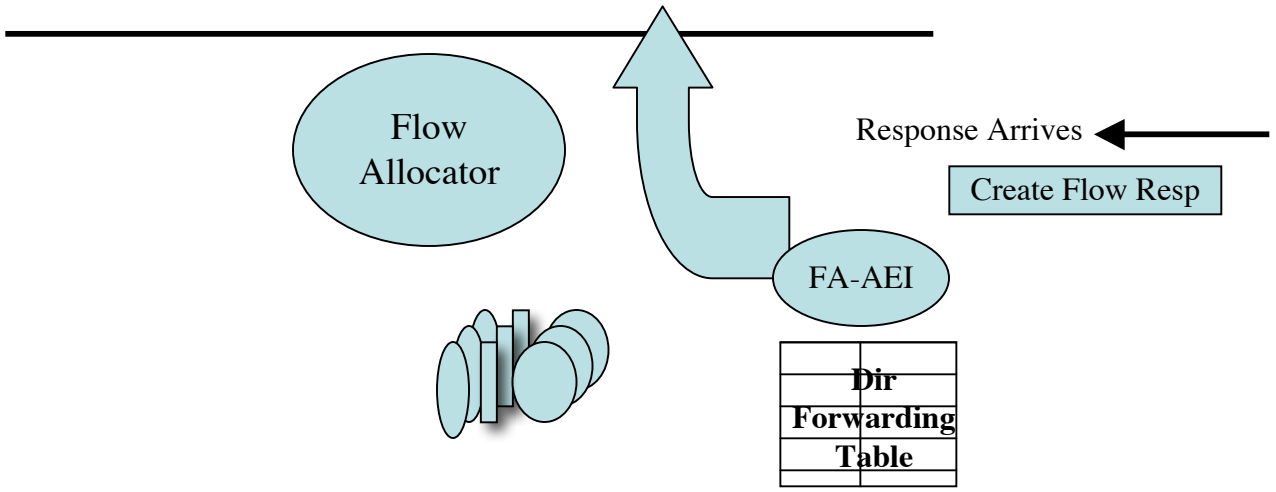
Finishing Up

Establishing Communication



- The Create Flow Response is sent back to the originator.
- Data can now flow.
  - Remember we instantiated the EFCP at the source before we started.
  - Just in case data gets back before the Create Response.
- Either end may start the Application Protocol exchange.

The Flow is Now Allocated



- The Create Flow Response arrives from the Destination
- The Application is notified that the Allocate was successful and is given a port-id (in Unix this might be a file descriptor), i.e. the FA-AEI-identifier.
- The application can now start exchanging information.

The Management Side
RIB Daemon

- The OIB/RIB Daemon is a generalization and combination of three facilities: routing update, event management and the management agent and performs the following:
  - periodically request or notify all or a subset of members of the current value of selected information (routing update);
  - upon certain events, notify all or a subset of members of the current value of selected information;
  - the latter implies that all event notifications occurring within the DIF should be delivered to the RIB Daemon because there may be a subscription that is triggered by the event, (event management);
  - given that elements of the DIF members should be able to request immediate notification of an event's arrival, have it recorded in the RIB, or both (event management);
  - if a log of events received is to be kept (the black box function), then the RIB Daemon is the natural place or it (event management);
  - given that the RIB Daemon responds to requests for information from other members of the DIF, then it is the natural place to respond to external requests for information from the DIF Management System (DMS), (management agent).

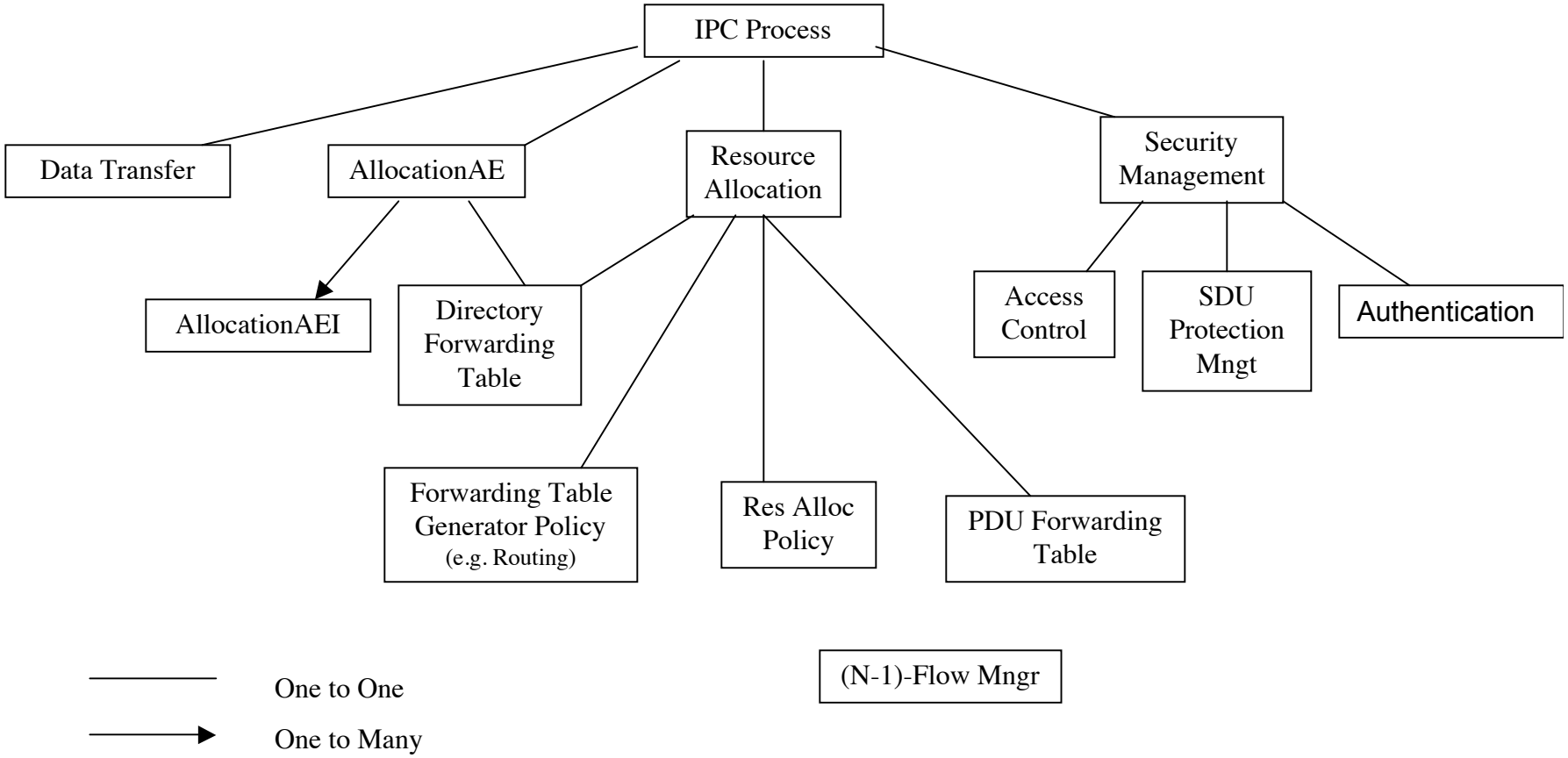
The Management Side
Resource Allocator

- The resource allocator is the core of management in the IPC Process. The degree of decentralization depends on the policies and how it is used.
- The RA has a set of meters and dials that it can manipulate
- The meter fall in 3 categories:
  - Traffic characteristics from the user of the DIF
  - Traffic characteristics of incoming and outgoing flows
  - Information from other members of the DIF

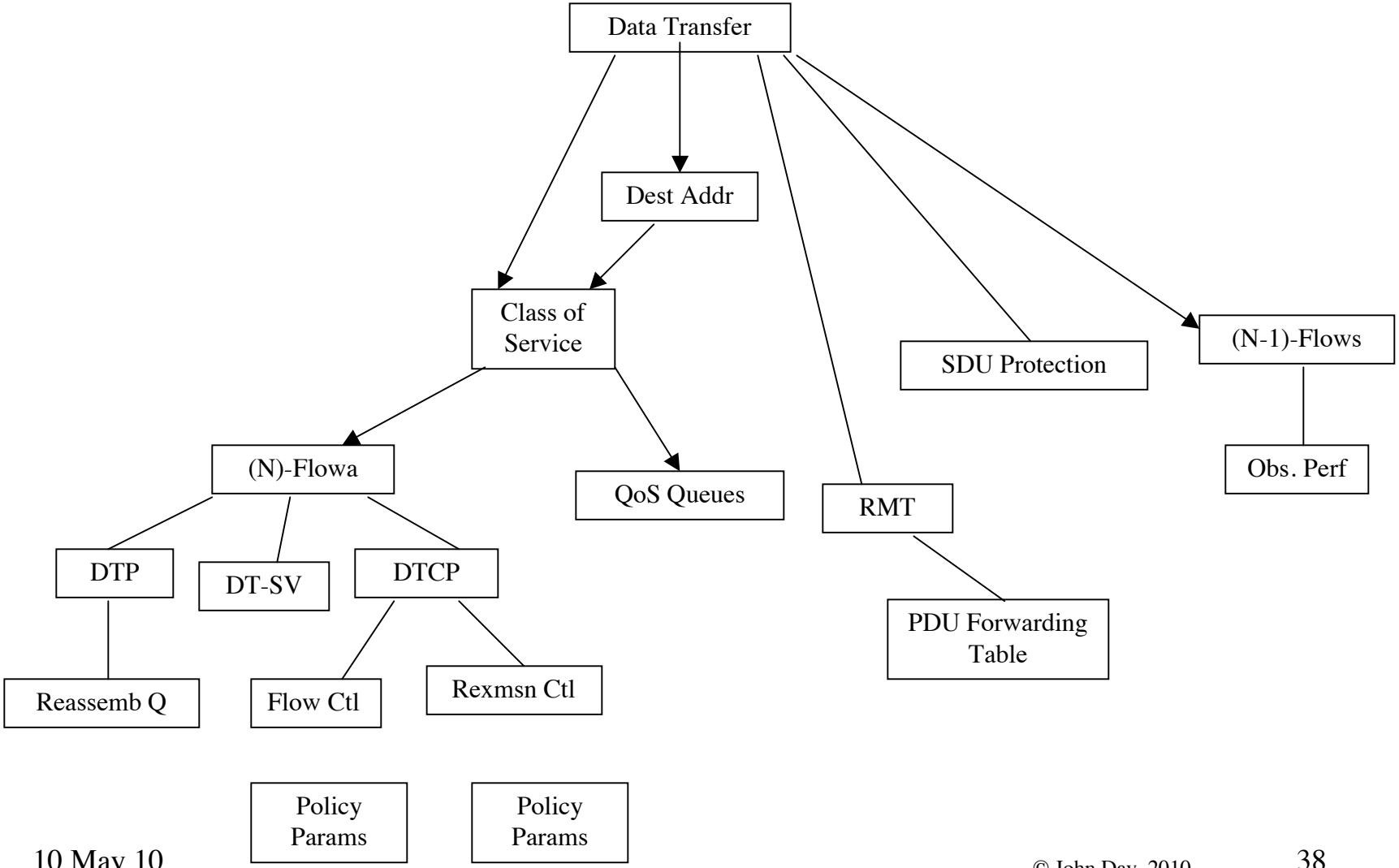
The Management Side
Resource Allocator

- The Dials
  - Creation/Deletion of QoS Classes
  - Data Transfer QoS Sets
  - Modifying Data Transfer Policy Parameters
  - Creation/Deletion of RMT Queues
  - Modify RMT Queue Servicing
  - Creation/Deletion of (N-1)-flows
  - Assignment of RMT Queues to (N-1)-flows
  - Forwarding Table Generator Output

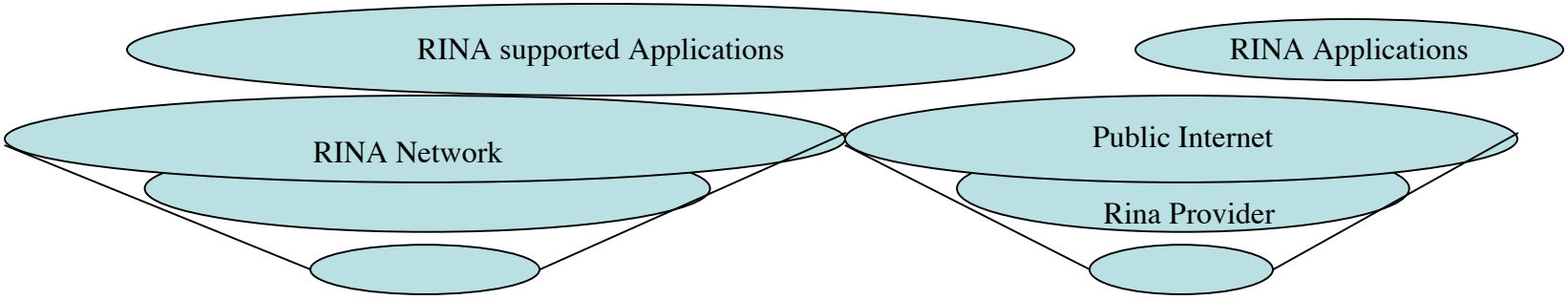
Initial Draft of a MIB: I



Initial Draft of a MIB: II



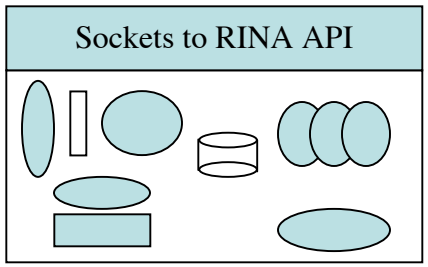
Transition? No, Adoption



- Adopt. Don't transition.
  - If the old stuff is okay in the Internet e-mall, leave it there.
  - Do the new sophisticated stuff in RINA
- Operate RINA over, under, around and through the Internet.
  - The Internet can't be fixed, but it will run better over RINA.
  - New applications and new e-malls will be better without the legacy and run better along side or over the Internet.
    - Microsoft tried to prolong the life of DOS.
      - It still haunts them.
    - A clean break with the past. The legacy is just too costly.
    - We need engineering based on science, not myth and tradition.

Aids to Adoption

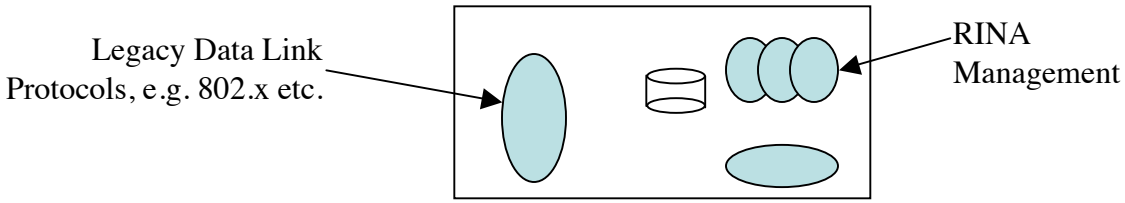
An Internet API



- Allows most Internet applications to work over RINA,
  - Some restrictions on passing addresses since RINA never exposes them.
    - If available on both ends, can be faked.
- But with none of the benefits of RINA.
  - Can only connect to a new instance of an application.
  - There are several ways to do this. None will give you every aspect of RINA.

Aids to Adoption

Data Link Diversity



- To deal with the diversity at the media, use the legacy data transfer protocols, but manage it with RINA.
  - Provides better management.
  - There would be some advantage to moving to RINA data transfer as well.
    - Fewer protocols, less parts count, less cost, easier management, higher performance, etc.
  - But as an Aid to Adoption, this can be investigated.

Next Steps

- Standardization
  - We need multiple implementations to test the concepts, and serve as a sanity check on implementations and interpretation
  - To inter-operate, we need to standardize a number of things
    - Minimum common object sets and operations
    - Common concrete syntax(es) and data representation(s)
    - Interoperable common (sub)sets of policy combinations
  - But not all DIFs have to operate within these constraints, so research and customization is not hampered
  - We need an organization that can help create, promulgate, and support these standards - the Pouzin Society

Next Steps (2)

- Implementation Issues
  - WHY will people do implementations? What can we do to help them so that we raise the acceptance of this technology?
  - Many different execution targets: middleware, microkernel, OS, apps, dedicated/embedded
    - Different concerns may lead to radically different implementations
  - Proprietary concerns
    - If implementations don't interoperate, they don't reinforce one another
    - Basic technology and intellectual property needed for interoperable implementations needs to be availability without burden
      - Product details can generally be abstracted from DIF/protocols

Next Steps (3)

- Many supporting activities will be fruitful
  - Research and analysis of concepts
  - Policy research, separated from the difficulties of making new concepts work within highly-optimized legacy implementations
  - Tools for measurement, testing, configuration, and analysis of implementations
  - Tools to generate and run simulations based on common mechanism, with experimental policies plugged in as appropriate
    - Impractical with hand-coded implementations of protocols
    - Opens new opportunities for research into protocol behavior and design

Next Steps (4)

- We'll be meeting this week to discuss all these issues - please feel free to join us!

Conclusions

- RINA is a new formulation of how to do networking
  - It incorporates the lessons of the last 40 years, where we've learned both good and bad ways to do things. We'd like to step away from many things that have not worked out well.
  - We don't propose that RINA overnight replace the Internet as we know it, but rather that we begin a new age of protocol research and implementation
    - To add new capabilities to networks
    - To make networks more efficient, manageable, and secure
    - To get past problems that have arisen in where we are now
- RINA is in its early stages of development - join us and help us mature it!