

Virtualization: Indirection, Heterogeneity and Scale

Shivkumar Kalyanaraman

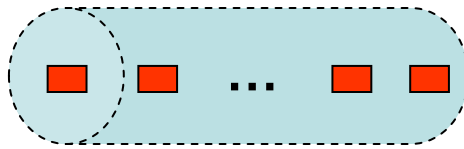
Principles/Take-away Messages:

- Virtualization is a form of abstraction, particularly useful in computer systems. A virtual entity resembles lower-level resources, though its semantics and performance attributes differs dramatically.
 - Virtualization is implemented over a multiplexed underlying resource using a technique called indirection.
- An indirection is a dynamic mapping from one software entity to another. The entities can be distributed in space, and the mapping can be achieved across time, with state information used for coordination.
 - Bind and unbind operations using state epitomize the indirection approach.
 - IDs (identifiers) are the simplest kind of *state* used in distributed indirections (eg: sending a packet from source to destination). A large part of protocol headers are just identifiers in disguise used to facilitate indirections.
 - The indirection *viewpoint* is useful in understanding and designing systems. For example, we can view bridges, switches and routers as pure indirection entities, and all state information (eg: headers, forwarding tables) used to facilitate indirections.
 - Common indirection patterns in protocols include: “decouple-and-indirect”, “late-binding” and “indirect-with-filtering.”
 - Indirection is everywhere in protocols and used to achieve a variety of objectives. Review a number of examples: from SONET to p2p.
- Scalability:
 - Indirection, combined with filtering, embodied in devices such as bridges, switches and routers allow systems to scale. The quality of filtering determines the efficiency and scaling gains derived.
 - The Internet can be viewed as a virtual switch, which in turn can be viewed as an indirection device providing virtual link abstractions.
 - The higher quality of filtering needed for scalability can be achieved through a variety of means: hierarchies & structured IDs, explicit coordination (eg: routing vs bridging), DHTs.
 - Scalable indirection in dynamic networks: overlay p2p, MANETs. Key: limiting scope of flooding (a.k.a. filtering).
- Heterogeneity:
 - Translation and overlays are two indirection models for handling heterogeneity.
 - Translation has its pitfalls, but is commonly used because of its simplicity and expediency (eg: bridging, NATs).
 - Overlays can be largely successful if the semantics/syntax to be mapped is simple.
 - IP-over-ATM demonstrates how even simple models can be hard to map onto incompatible lower layers.
- Indirection used to tackle protocol complexity, architectural inflexibility
 - Protocol complexity: decouple-and-indirect, flexible cross-layer designs etc
 - Problems with the internet architecture today can be traced to overloaded and inflexible indirections
 - Layered naming with multiple levels of indirection can provide an evolutionary path
- Summary

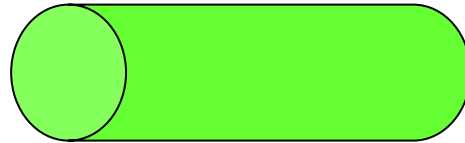
1 Virtualization

The idea of a "virtual resource" is central in computer systems. A virtual resource is a software entity that built out of a (shared or multiplexed) physical resource. In this sense, virtualization is just a useful higher-level abstraction that is appealing because humans tend to relate easily with the concept of the underlying physical resource. They can hence reason about the virtual resource and compose even higher level virtual resources using these new building blocks. Though the virtual resource shares some essential defining aspects of the underlying physical resource, the former's *semantic* and *performance* attributes tend to be significantly different from the underlying physical resource, and those differences are often where its value lies.

For example, to a programmer, virtual memory "feels" like physical memory, but offers a lot more flexibility and new capabilities (eg: larger size, protection across processes etc). A UDP/IP flow of packets (i.e. a best-effort packet switched virtual link) offers no performance guarantees and has packet-based semantics compared to a physical link that accepts bits and offers fixed capacity, fixed delay and tight control over delay jitter (**Figure 1**). A TCP/IP connection is a virtual link that can be imagined as a local file-system interface or as a reliable physical byte-stream link, but with a twist: one that offers variable performance and is not secure. A SSL-session adds security features to this abstraction, but performance is still variable. With end-to-end per-flow QoS support from the network, performance assurances can be added, though there still will remain minor differences (eg: packetized service vs bit-level service). Observe that the SSL and TCP virtualizations add *new semantics*: security and reliability capabilities. What is particularly remarkable is that these "virtual" capabilities of security and reliability are built over un-secure and unreliable end-to-end paths respectively!



Best-Effort Packet Switched Virtual Link



Link: Fixed Capacity, Delay, No-Jitter

Figure 1: Virtual Link vs Physical Link. Performance and Semantics Differ.

Early virtualization designs such as virtual memory in computer architecture or multi-tasking in operating systems were built on top of *local* resources such as physical memory and the processor. In contrast, the Internet offers a "virtual link" abstraction built over *large, distributed set of heterogeneous, autonomously managed* physical resources: links, routers, switches and networking software layers. Besides hiding complexity and accommodating heterogeneity, virtualization in a distributed context also provides some degree of location transparency. For example, with virtual storage and file systems (eg: SANs, NFS or network-attached storage), the user does not need to know where the underlying physical storage resources are. Similarly the mobile IP virtualization hides the location of the mobile node. The distributed nature of these kinds of virtualizations, combined with their accommodation of heterogeneity and scale poses new design challenges. In particular, to cope with complexity, we need to view this problem as a higher-level virtualization problem, and decompose it into multiple lower-level indirection operations (described next) that are re-composed in a distributed manner.

2 Indirection

The dynamic association between the virtual resource and the underlying set of shared physical resources is referred to as "*indirection*", alluding to the fact that the user achieves the virtualization through *indirect*

access to physical resource¹. In its simplest form, an indirection is just a *functional mapping* from one software entity (like an abstraction, virtual entity, or any protocol component) to other entities (eg: instances, physical entities, or lower-layer protocol entities). The reason indirection is attractive is because it allows the definition of very general functional mappings to be realized in a dynamic, distributed way, and coordinated through minimal state information (eg: embedded in headers or placed at nodes). Several design issues in computer systems may be boiled down to the question of the attributes the virtual resource should have, and the design of the underlying indirection infrastructure to achieve the virtualization. The internet protocol infrastructure in particular may be thought of as an elaborate indirection infrastructure. We will see that this *indirection viewpoint* of protocols helps us understand the tradeoffs in particular protocol designs and the topic of peer-to-peer networks in particular.

Lets us develop and exercise this “indirection” viewpoint of protocol design with a simple example. **Figure 2** shows an indirection operation for unicast transmission of a packet over a shared local area network. The source node desires to send a piece of information to the destination node. It views this unicast problem as a functional mapping of a payload (set of bits) to the destination, and not to any other host node. Indirection refers to the process of implementing this desired mapping. In particular, the source node adds a piece of state or meta-data called the multiplexing identifier (ID) to the information payload to create a “packet.” The association of an ID to the packet is sometimes referred to as a “*binding*” operation, similar to the way a pointer (in a programming language like C) binds to a memory location. Potential destinations on the path of the packet examine this ID and *only* the destination who’s configured ID matches the ID on the packet, will “*unbind*” the meta-data header and claim the packet. Observe this “unbind” operation is similar to the de-referencing operation on a C pointer.



Figure 2: Indirection Prototype: Decoupled Entities (Source, Destination), State (ID in header) and Bind/Unbind Operations

The indirection has achieved its unicast objective: a functional mapping of the packet’s payload from the source to the destination only (and not to any other node), achieved in a decentralized way, through a simple packet header field (an identifier). The indirection has also created a virtualization: an abstraction of a virtual, packetized link built over the physically shared LAN. Note that a separate management process is required to handle the assignment of unique IDs to nodes and facilitate this indirection process. The indirection operates in the context where entities are *decoupled* (source and destination node entities) and offers *flexible, distributed re-coupling* using state information (eg: IDs). The coupling or mapping between the packet payload and the ID was done just-in-time, sometimes called a “late” binding operation. We will see this pattern of *decouple-and-indirect* and *late-binding* in several protocol designs.

The example we used above is a stylized version of a general pattern of protocol interactions. If we examine the content of IPv4 or IPv6 headers (see **Figure 3**), we find that it is dominated by identifier (ID) fields facilitating indirections! The following fields are just IDs in disguise: source/destination addresses, version number, type-of-service or class field, protocol type and flow label. Even fields like sequence

¹ The term “indirection” is a generalization of several similar terms or phrases such as “resolution”, early/late “binding”, “delegation”, “translation”, “mapping”, “(de)-referencing”, tight/loose/flexible “coupling”, “interfacing”, “dynamic/flexible composition”, “relocation” etc.

numbers and fragment offset can be viewed as IDs referring to a particular byte in a byte stream. Packet headers can thus be viewed *largely* as state information, typically IDs of some sort, with which binding/unbinding operations are performed. Similarly higher layer protocols such as TCP, UDP and RTP use a number of ID fields (eg: port numbers, sequence numbers etc).

IPv4 & IPv6 Header Comparison

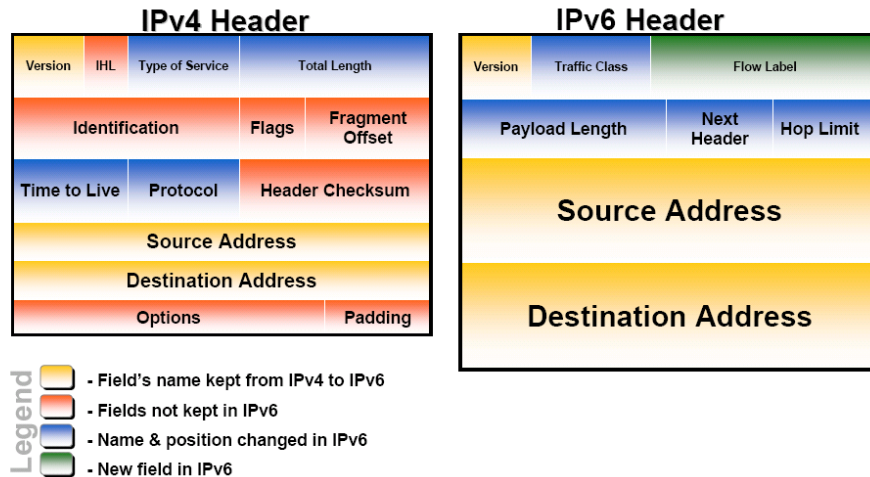


Figure 3: Many Protocol Header Fields are Identifiers (IDs) in Disguise, Facilitating Indirections

Of course, identifiers are not the only type of per-packet state through which an indirection operation is done. Broadly speaking, any information put in headers or trailers of packets can be used as the basis for indirect, remote operations. Some quick examples include count fields (eg: TTL, length), error control fields (eg: CRC, header FEC). More complex forms of indirection involve the use of complex state such as bloom filters, XML-based self-describing control information, active programs and network coding. Signaling protocols can be viewed as mechanisms for mapping global IDs (eg: addresses, path specifications) to local IDs (eg: labels, VCIDs) and local reservations of resources.

2.1 Indirection Everywhere

Indirection mechanisms are literally everywhere in Internet protocols. Indeed the entire protocol infrastructure can be viewed as an indirection infrastructure. Let us start with **Figure 4** which gives a few examples of the indirection concept in the Internet.

A common problem in resource discovery or routing involves mapping a persistent (or human-friendly) name/ID to a locator, i.e. another ID that encodes routing hints to reach the resource. **Figure 4(a)** illustrates how DNS uses indirection to map a human-friendly name to an IP address (a “locator”). The IP address is then associated with data for communication. As we shall discuss later, the DNS name itself is hierarchically structured, and the name servers are organized hierarchically to support autonomy of namespace management and scalability of name resolution. **Figure 4(b)** shows how Mobile IP provides location transparency by mapping a persistent ID (a home IP address) to the mobile’s locator, called the care-of-address. When a mobile joins a new network, it needs to register the new mapping with the home agent. Since the mobile IP bindings are updated dynamically once the mobile moves to a new network, this is an example of late-binding between the ID and locator. Observe the similarity in concept between DNS and mobile IP, despite their being at different layers in the protocol hierarchy. Similarly ARP (address resolution protocol) maps a layer-3 ID (eg: next-hop IP address) to a layer-2 locator (eg: next-hop Ethernet address).

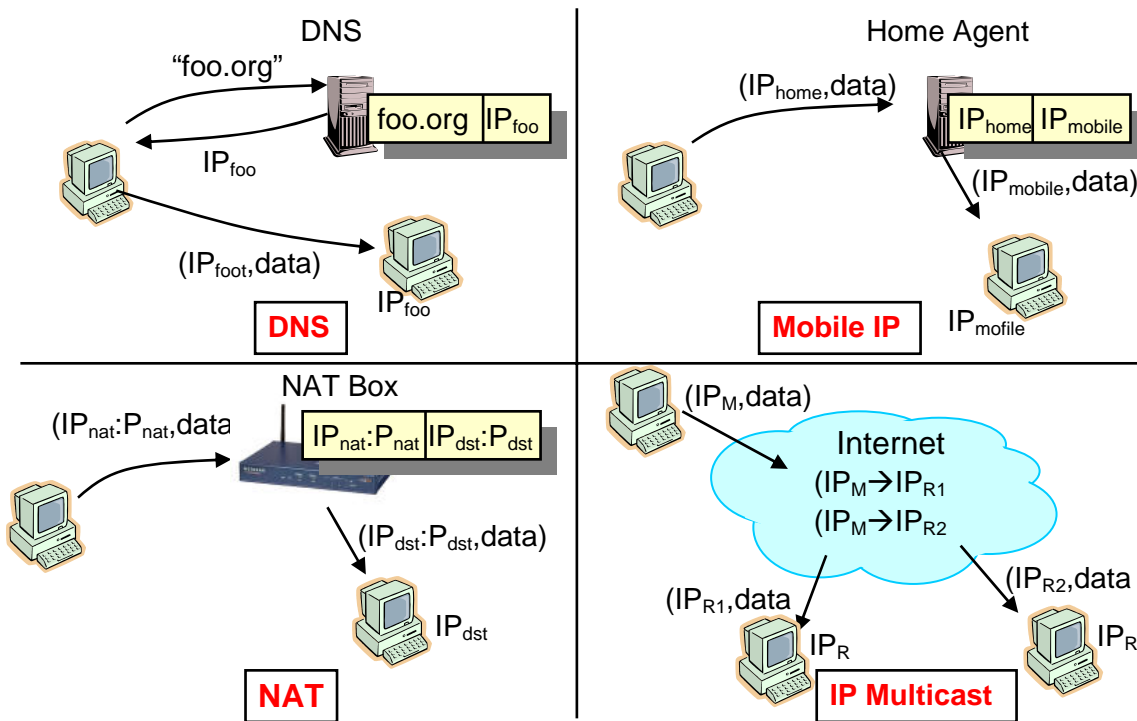


Figure 4: Indirection Examples: DNS (name resolution), Mobile IP (location transparency), NAT (address translation), Multicast (1-to-many mapping)

Figure 4(c) shows how NAT (network address translation) maps a public IP address (and a public port number) to a private IP address (and private port number) and vice versa. Its objective is to multiplex a scarce pool of public addresses, amongst a larger pool of nodes. DHCP also does IP address space multiplexing amongst a set of nodes, but NAT in addition allows internal IP communications within the private network using private IP addresses. Public IP addresses are used only when external IP connectivity is needed. Also NAT is transparent, whereas DHCP is a request-response protocol. NAT also is an example of the translation approach (similar to how bridging was used between heterogeneous LANs). The problem with translation is that when heterogeneity increases, the number of dialects the translator needs to know also increases. NAT also has to deal with public IP address dependencies in older versions of higher layer protocols (eg: FTP encodes IP addresses in its messages). NAT also breaks the IP model of having layer-3 IDs that are meaningful end-to-end, and available for referencing by applications. Peer-to-peer applications like Skype and Bittorrent which require peer-to-peer IP connectivity therefore have to incorporate NAT negotiation methods and use of a centralized login service to discover the public IP addresses currently associated with peers.

Figure 4(d) shows how a new virtual pipe abstraction - multicast - is realized using indirection within the network. An internal network node has to map the multicast IP address to one or more next-hops. The network needs to transparently build such mappings and dynamically update them when destinations join or leave a group. Multicast also is an example of how complex the indirection infrastructure can become in a large-scale, heterogeneous internet. Recently, there have been proposals to make the multicast abstraction source-specific (SSM) which simplifies the management issues of address allocation and control-plane problems such as multicast routing. The I3 project suggests implementing multicast as an overlay on the internet, aided by a general purpose indirection mechanism, called the Distributed Hash Table (DHT). DHTs map IDs (or keys) to values, providing a simple put() and get() interface. The mapping (or hash) table is broken up and distributed over the network to form specific graph structures. ID resolution then becomes a matter of efficient routing over this overlay graph.

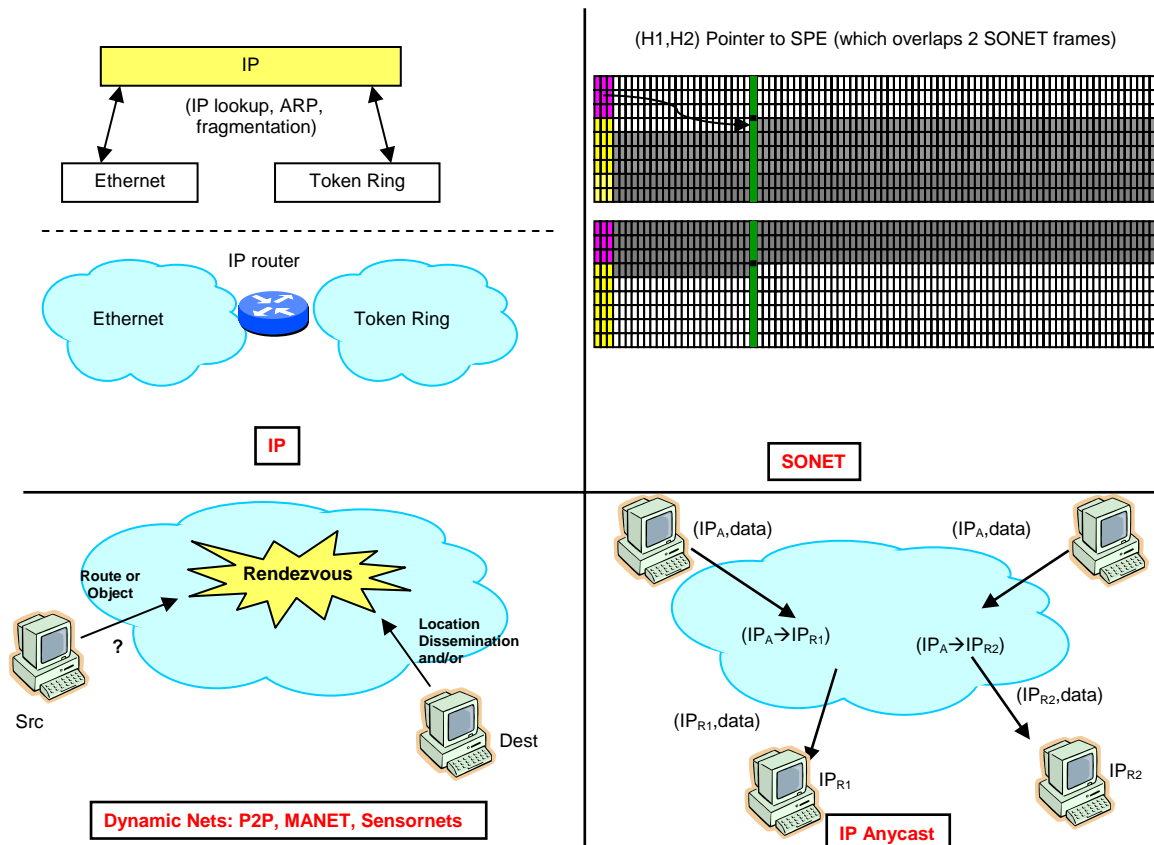


Figure 5: Indirection Examples (Continued): IP internetworking (overlay vs translation), SONET, Dynamic Networks (data-centric routing), Anycast

Figure 5 provides further examples of indirection. **Figure 5(a)** depicts the idea of indirection using overlays. IP’s *overlay* approach to internetworking contrasts with the translation approach used in layer-2 bridges and application-level gateways (ALGs). Instead of requiring translation between many layer-2 protocols, the IP overlay approach requires each link layer to be mapped to just one overlay protocol (IP). The “thin waist” or “hour-glass” concept of IP further implies that the IP-layer is lightweight and easy to map to. While this model has been a great success, there have been considerable difficulties in mapping IP and internet routing protocols (eg: OSPF, BGP) to non-broadcast medium access (NBMA) networks such as ATM networks. Interestingly, such difficulties led to the emergence of MPLS (multi-protocol label switching) which was easier to map IP to, and to the demise of ATM networks as a competitive end-to-end internetworking technology. The term “overlay” also refers to protocols operating on top of the Internet, and using the core internet protocols for point-to-point connectivity. Peer-to-peer data-centric systems are a large emerging class of such overlay systems. Planetlab is a worldwide non-profit organization led by Princeton university that has nodes in hundreds of locations worldwide, and users have the ability to deploy long-running prototype overlay applications over this infrastructure. To facilitate the concurrent operation of multiple overlays, Planetlab divides individual machines into virtual “slices” and allocates slices to users.

Figure 5(b) shows the use of pointer-based indirection in the physical layer (SONET). SONET uses a two-dimensional time-slotted frame. There is a time-synchronized frame of length $125 \mu\text{s}$ into which aggregates of telephone calls (eg: T1 or T3 trunks) can be placed. Specifically, each call can occupy an 8 bit slot in this frame, and the number of slots within the $125 \mu\text{s}$ frame determines the level of aggregation (eg: STS-3, STS-12 etc). The frames can also be used in a concatenated mode for data. An interesting aspect in SONET is the difference between the frame and the payload (called the “synchronous payload envelope” (SPE)). The SPE can “float” in the time-frame, i.e. it can start at an arbitrary location. A pointer is placed in header fields H1, H2 to indicate the beginning of the frame. This is done to allow flexibility in managing

synchronization between SONET equipment, despite operating at ever higher speeds (eg: today OC-192 equipment operates at 10 Gbps!). Techniques like scrambling and byte stuffing are used in the plesiochronous T-system hierarchy at lower bit rates; these techniques are not scalable for high bit rates (155 Mbps+). Considering the fact that most of telephony infrastructure has to operate in a tightly synchronized fashion, this introduction of flexibility through indirection was a revolutionary idea. This indirection feature along with SONET's operations & management (OAM) features led it to be a huge success as the basis for both telephony and data networking at the physical layer.

Figure 5(c) is a generalization of several protocols in emerging dynamic networks such as mobile ad-hoc wireless networks (MANETs), peer-to-peer networks (P2P), disruption-tolerant networks (DTNs) and wireless sensor networks. A common indirection feature involves a *query* originating from a source that traverses the network and *rendezvous* with *state information* or object *replicas* injected on behalf of the destination object. For example, in MANETs, the source sends a “reactive” route query and may optionally be replicated (using controlled flooding or multiple random walks) to reach either the destination directly or rendezvous with “proactive” state information injected by the destination. In sensor networks, where the network is viewed as a data-centric system, an application query is disseminated into the network (using techniques like directed diffusion) to sensors which may have the answers to the query. The sensors cooperatively respond to the query since each may have only a partial answer. Data or responses need to be aggregated and routed back to the source. In peer-to-peer applications (eg: Gnutella, Kazaa or eDonkey), the goal is to discover the location of a resource given an object ID. The query is optionally replicated (through flooding or k-random walks); and the object itself may be replicated based upon popularity. Object routing may use properties of the underlying p2p graph (eg: super-nodes or nodes with high degrees). In contrast to flooding or unstructured methods, Distributed Hash Tables (DHTs) offer a structured indirection method where the mappings are based upon special name-space and graph structures (eg: ring, torus, deBruijn graphs). The objective is again to achieve rendezvous between one or more query trajectories and the destination-injected state or replica locations.

Figure 5(d) depicts the idea of “anycast” routing. In this model, the destination associated with the same ID might be different depending upon where the query originates. The source may care about access to a nearby service, and be indifferent to who provides it. Anycast is provided as a service in IPv6 and in overlay protocols, and is considered an important primitive facilitating incremental deployment of new architectures and protocol designs [1]. It is interesting to note that services like anycast, multicast and support for mobility can be provided over a DHT infrastructure as an overlay on the Internet. The Internet Indirection Infrastructure (I3) [2] project in UC Berkeley aims to do this, and has been deployed as a publicly available overlay system service on Planetlab. This can be viewed as a composite of several of the indirection examples provided in this section. OpenDHT [3] is another publicly deployed DHT service available through Planetlab. A wide variety of data-centric services and systems have been developed over these core overlay building blocks.

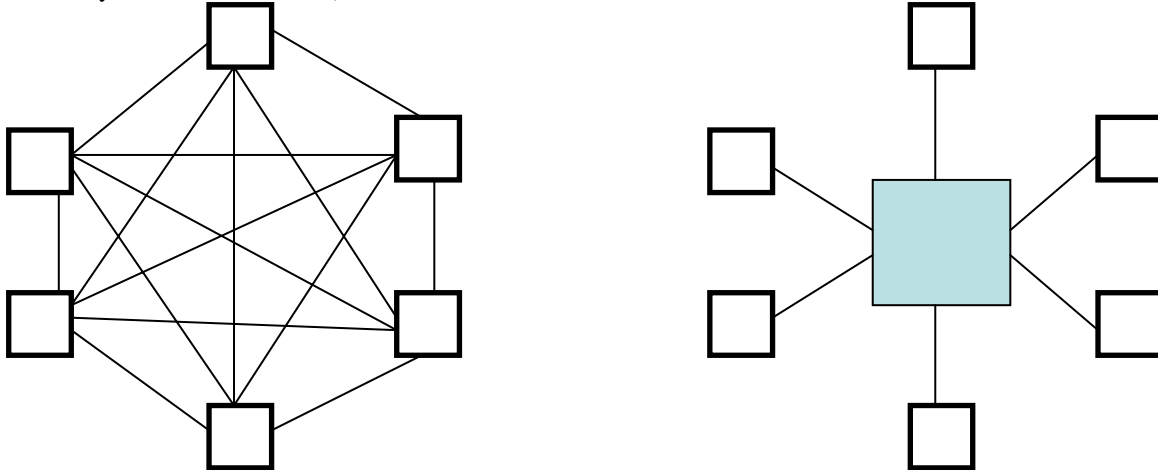
2.2 System Scalability Using Indirection

Since indirection is a general functional mapping mechanism, it can be used to achieve a variety of system design goals. We illustrate this concept using the topics of scalability, heterogeneity and protocol complexity. Let us first consider the problem of system scalability.

2.2.1 Scalability: The Role of Switches, Bridges and Routers

Consider a network composed of nodes and links. Scalability is a property which allows the network to efficiently increase its performance as more resources are added (eg: adding more nodes). This implies two things. *First*, we should be able to increase the system size as a function of the resource without hitting any hard limits. For example, a bus-based Ethernet has hard scaling limits on how many nodes can be added on a single segment. *Second*, by “efficiently” we mean that system costs should not grow super-linearly with a linear increase of one resource. For example, a full mesh network will require $O(N^2)$ links to be added to support $O(N)$ nodes – if links are costly, or per-node link interfaces are limited, this design will not scale.

To support scalability, we typically introduce new components into the system that facilitate *indirection* and *filtering*. For example, in Ethernet, we introduce hubs, bridges, switches and routers. What do these network elements have to do with indirection? Why exactly does indirection help solve the scalability problem? To answer this, we revisit the simple indirection example from the previous section: sending a packet from source to destination using an ID field to facilitate the indirection operation. In that example, nodes whose IDs did *not* match the packet ID did not waste further time and resources to process the packet. Only the destination node whose configured ID matched the packet ID picked up the packet. We refer to this as a *filtering* operation: the mapping is done between a source node and a *subset* of all nodes (in this case, only the destination node).



**Figure 6: Full Mesh Network vs Hub-and-Spoke “Switched” Network.
The switch design requires an indirection and provides a virtual link abstraction.**

Consider the full mesh network design which requires the addition of $N-1$ links for the addition of the N th node. Compare it to a switched network with a star or hub-and-spoke topology (see **Figure 6**). In the latter design, the switch looks at the destination ID on the packet and maps the packet *only* to the appropriate output port to which the destination is connected. The switch is where the “late binding” or indirection from the source to the destination address is made. The packet was *not* flooded on ports to which the destination was not connected. Thus the *indirection-and-filtering* operation and a dedicated indirection system component (the switch) have reduced the number of links required. The number of auxiliary resources needed (in this case, links) to support the addition of a new node has reduced from $N-1$ to just one (a link from the new node to the switch). The switch has created a *virtual link* abstraction to replace the missing links. The efficiency obtained in terms of number of links required directly translates to system scalability. Efficiency and scalability are two sides of the same coin. As an aside, note that we now need free ports on the switch to attach the new node. If we hit a limit, we will have to use a network of switches and multiple indirection-and-forwarding hops to create the same scalable indirection abstraction as a single switch (see **Figure 7**).

Let us now come back to the Ethernet scaling example. There is a limit on the number of nodes that can be on a single collision domain (i.e. a set of segments connected by repeaters) due to the nature of the MAC algorithm, CSMA/CD. Therefore, to allow extended LANs, the concept of Ethernet bridges was proposed. An ethernet bridge looks at the destination ID in the packet (the Ethernet address), compares it to a local forwarding table and if a match is found, maps the packet to a single port. This indirection which maps a packet to a port is also a filtering operation because the bridge has restricted the set of nodes this packet will encounter. Through a series of such filtering indirection operations at multiple bridges, the packet reaches the destination. When a match is not found in bridges, the packet is flooded on all ports (i.e. it is *not* a filtered indirection and is not scalable in general)!

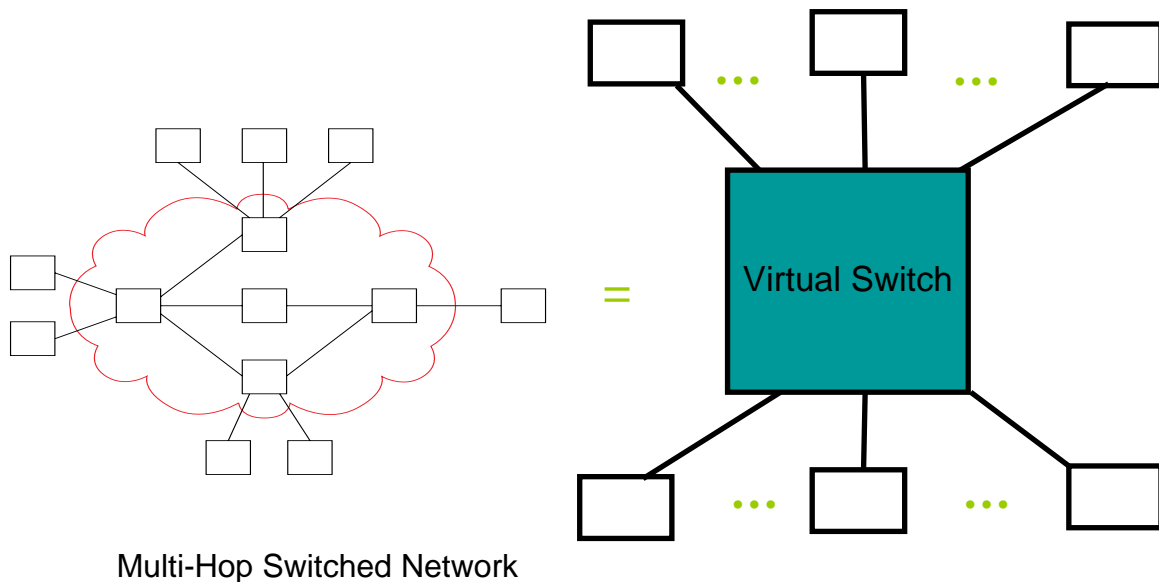


Figure 7: Multi-Hop Switched Network: provides a virtual switch abstraction.

IP routers are similar to bridges in the indirection function they perform. The key difference is the *quality* of the forwarding table in routers vs bridges. Bridges build a forwarding table by just looking at *local* information (snooping at packets passing by), whereas routers use an *explicit coordination protocol* called the routing protocol to exchange global topology information and condense it into local forwarding tables. Specifically, the router is better informed where the destination is in the topology, and chooses a next hop for its forwarding table appropriately. As a result, there will *always* be a next-hop match in any router forwarding table (or a default route), whereas bridges can frequently not find matches in large bridged networks. Indeed, a distinctive property of routers is that they *never* flood data packets on all ports, and instead always find forwarding table matches, i.e. they always filter. We will encounter the issue of flooding vs filtered indirections in other contexts (eg: peer-to-peer lookups and mobile ad-hoc network routing).

A separate control-plane protocol infrastructure (eg: routing protocols or local snooping in bridges) has to handle the creation and management of the ID-to-next-hop mappings in forwarding tables. The indirection approach usually leads to these auxiliary control-plane and management-plane problems which may themselves be complex to solve and require further levels of indirection!

The filtering capability offered by indirection components such as bridges and routers is essential to scaling. Why? By restricting resource consumption using filtering, those resources have been freed for concurrent use by other nodes. The system can support more concurrent packet transfers now, i.e. its performance has scaled to support a larger number of nodes. The better the quality of the filtering (eg: use of routers vs bridges), the lesser the instances of flooding (i.e. failure of filtering), and greater the scalability of the system. Internal to bridges and routers, the development of switch fabrics allowed high-performance parallel forwarding paths between ports rather than going through a central bottleneck. This development, combined with per-port lookup and buffering, led to high-performance, low-cost bridges and routers. These components are often called layer-2 switches and layer-3 switches respectively. We will view both routed and bridged networks as examples of “multi-hop switched” networks and both provide a virtual switch abstraction, which in turn provides a virtual link abstraction to the connected nodes (**Figure 7**). Note that our discussion did not make reference to layers until now, because the real story is indirection and filtering. Indeed as we go up the layers, filtering and indirection tends to be more sophisticated and based upon higher-layer IDs (eg: IP address or DNS names instead of Ethernet addresses).

In summary, the indirection viewpoint has therefore given us a new realization: *bridges, switches and routers are pure indirection and filtering entities that enhance network scalability.*

2.2.2 Scalable Internetworking: Direct vs Indirect Connectivity

We described routers and bridges above as examples of “switches” and mentioned that routers offer a higher quality of filtering. IP routers are distinctive in another way: they are “switches” that interconnect bridged networks to form a *network of networks*. This network of heterogeneous, autonomous networks is called the Internet². The Internet therefore provides a virtual switch abstraction, which in turn provides virtual link abstractions between pairs of nodes (see **Figure 8**).

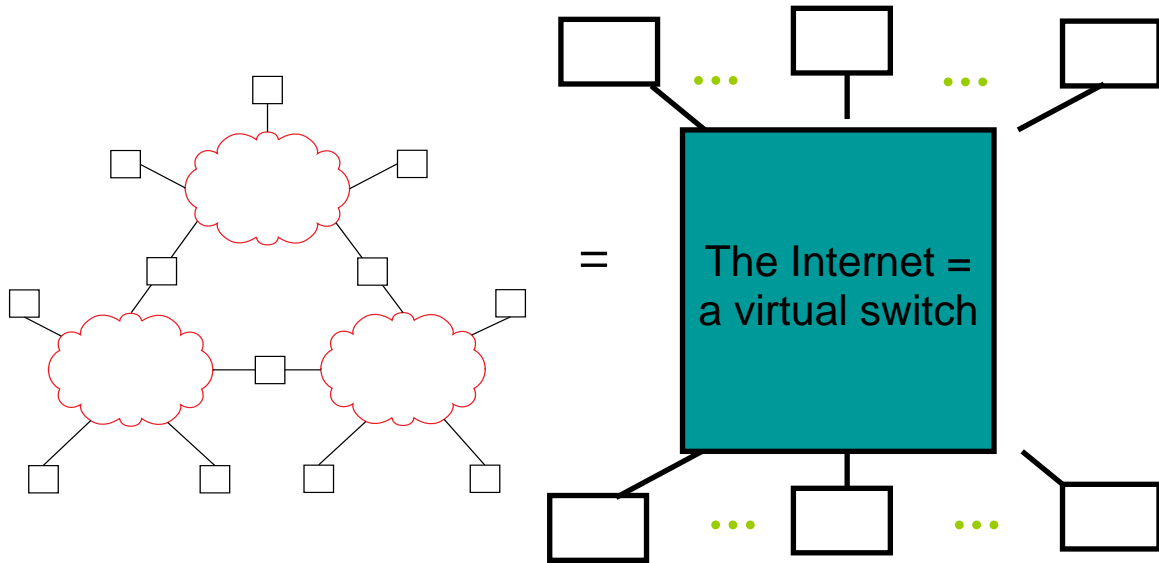


Figure 8: Internet is a network of networks, that also provides a virtual switch abstraction

Why is the quality of filtering offered by a bridge different from that offered by a router? While a bridge allows a source to indirectly access a remote link, an IP router allows an IP source to *indirectly* access a remote *network*. To facilitate this, the IP address is structured *hierarchically*. The IP address is not just *flat* destination interface ID, but it also encodes the network ID on which the destination interface is located (**Figure 9**). The boundary between the network ID and host ID is specified separately by a “mask” at individual routers, often called the subnet mask or supernet mask. Indeed, the IP address encodes several such network *locators* or network IDs at several bit boundaries.

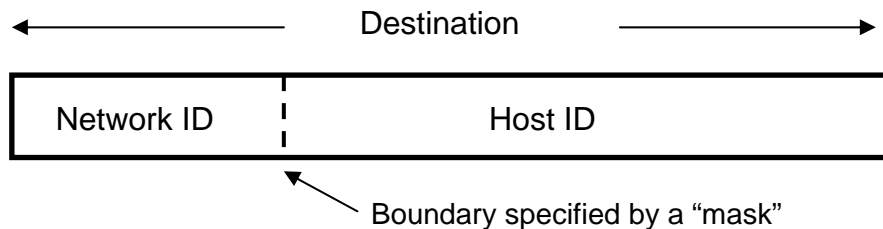


Figure 9: Hierarchical IDs Facilitate Scalable Indirection

IP routers use this embedded structure in the following way. If the network ID matches a local network ID, then the destination is declared to be *directly* connected (via a bridged network) to the router. If the network

² We will deal with the issues surrounding heterogeneity of networks later in Section ...

ID does not match a local network ID, it is *indirectly* connected, and the forwarding table specifies another router (a next-hop) which can get the packet closer to the destination network. The packet hence jumps from one network cloud to another network cloud (**Figure 8**) as it goes through a series of routers and indirectly accesses the final network on which the destination ID is directly connected. Indirect connectivity or indirection through routers - that in turn depend upon hierarchical addresses and forward packets on well-defined paths instead of flooding - is a critical contributor to the scalability of the internet.

The same idea of a hierarchically organized ID-space for scalability is used by the domain name system (DNS). Internet domain names are variable-length human-friendly names, but have a hierarchical structure that facilitates autonomous, decentralized management of name spaces within each domain. In the name resolution process the DNS resolver contacts a remote authoritative name server either through a recursive or iterative process that leverages this hierarchy. A key difference between IP forwarding and name resolution is that the latter is an overlay operation assuming point-to-point connectivity is already available. Similarly IP forwarding is an overlay operation that assumes the layer 2 connectivity (through a network of bridges, and ARP for address resolution) is already available.

2.2.3 Scalable Overlay P2P Systems: Flat IDs & Distributed Hash Tables (DHTs)

Though hierarchical ID structures enable scalable indirections, it does not offer a feature called *data-independence*. Application-level use of data is not decoupled from data organization, layout and access. For example, DNS uses host-centric names that are embedded in a domain context. DNS thinks of www.cnn.com as a *host* and resolves it to an IP address. However, users think of www.cnn.com as an *object*, i.e. a piece of data. In other words, internet use at the application-level is increasingly data-centric, though the DNS model is “host-centric.” One manifestation of this feature is our preference for flat IDs (keywords) and the use of search engines like Google to resolve such flat keywords to hierarchically-structured URLs. Another issue with host-centric viewpoint is the limits it places on location transparency. For example, the object www.ecse.rpi.edu/~shivkuma/index.html is bound to the domain www.ecse.rpi.edu. Moving the object to a different domain means that the name has to change (i.e. DNS offers location transparency only within a domain, and not across domains).

Flat, location-independent names for data are preferred to build database-style data-centric systems over the Internet. In such systems, a large number of objects are stored in a distributed manner, and we require an object/resource discovery mechanism, i.e. the resolution of a flat object ID to an IP address (

Figure 10). A scalable indirection mechanism is again needed to map such flat names to the location of data objects. How can this scalability be achieved *without* the hierarchical embedded structure of names? To contrast this problem, recall that bridged Ethernet used flat IDs, but had limited scalability.

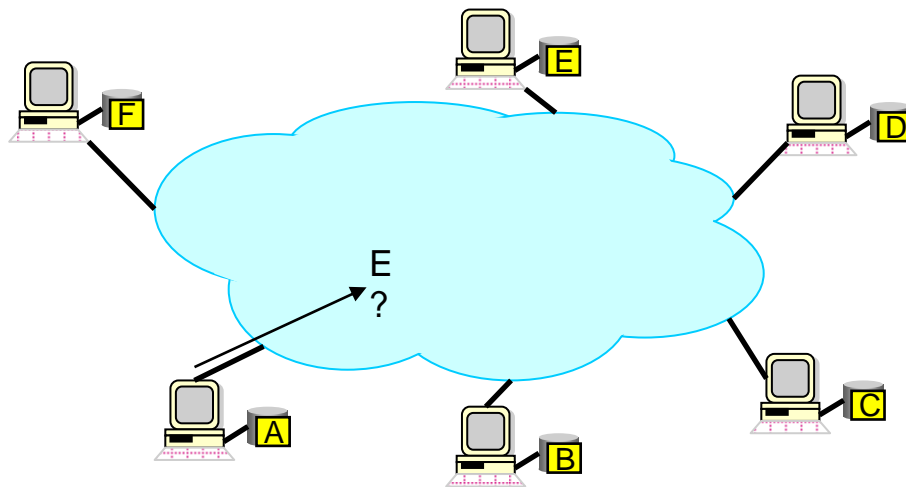


Figure 10: Resource Discovery in Large-Scale P2P Systems: Name-Resolution Problem with Flat IDs

Distributed Hash Tables (DHTs) solve this problem by partitioning the flat name space using a particular *structure* (eg: rings, toruses, DeBruijn graphs) and storing the mappings for the subsets in individual nodes. Since DHTs are implemented as an *overlay* over the Internet, like DNS, they assume that point-to-point connectivity is already available³. Object ID query resolution is performed by a series of indirections (i.e. routing) over the DHT structure (**Figure 11**), and each “hop” in this overlay route corresponds to an IP end-to-end path. The similarity of DHT put-get operations with routing is not coincidental: both are indirection operations! Indeed, the I3 project [2] builds an overlay indirection infrastructure that can be used to provide a wide variety of flexible network- and content-based services at the overlay-level.

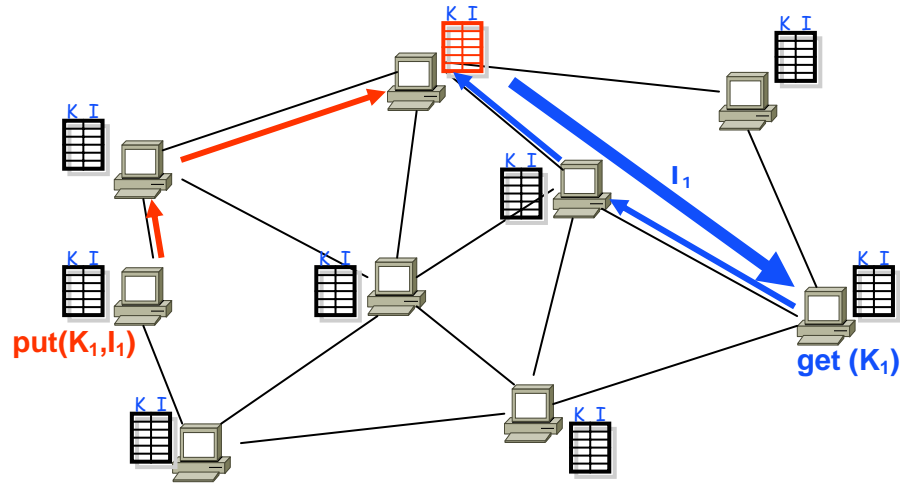


Figure 11: Distributed Hash Table (DHT): Structured decomposition of Hash Table. Supports efficient Put(key,value) and Get(key) operations with guaranteed recall.

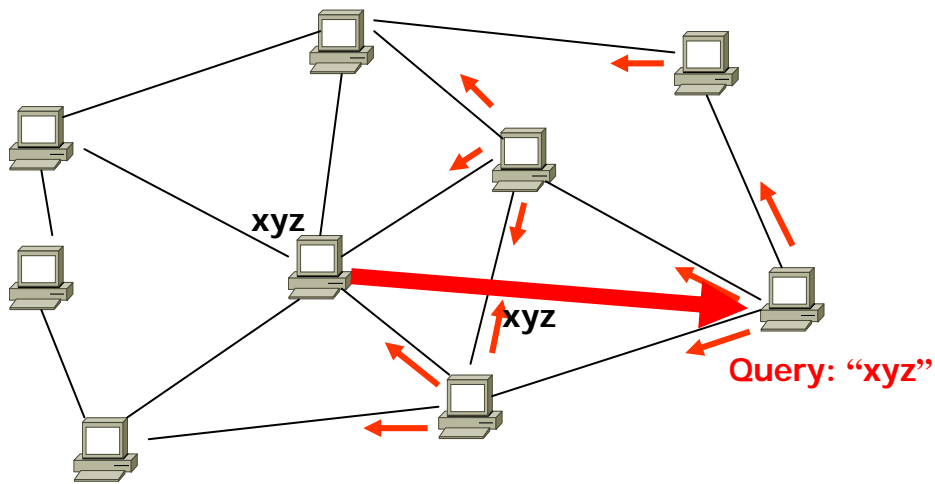


Figure 12: Flooding in Unstructured P2P Networks (Eg: Gnutella, KaZaa)

DHTs offer worst case performance bounds (eg: $O(\log N)$ overlay hops), guaranteed recall (i.e. any object inserted can be queried), and robustness to node failures. If guaranteed recall is not critical for rare objects, and the system only needs to facilitate rapid indirections for popular objects, then *unstructured* methods are sufficient. The simplest unstructured indirection mechanisms are flooding (eg: Gnutella) and

³ This is a key difference from bridged Ethernet which is not an overlay, and could not assume that point-to-point connectivity was available.

hierarchical/controlled flooding at super-peers (eg: KaZaa). As shown in **Figure 12**, a query is simply flooded over the overlay graph with a TTL bound, and a reply is sent from a node that finds a match. Other unstructured indirection methods include gossiping, epidemic routing, and k-random walks. An open problem today is to construct general-purpose scalable unstructured indirection mechanisms that also provide guaranteed recall like DHTs. Structured indirection (DHTs) and unstructured indirection methods are the foundation of modern peer-to-peer (p2p) systems.

2.2.4 Scalable Indirections in MANETs

Peer-to-peer systems belong to the larger class of *dynamic* networks. Other dynamic network systems include mobile ad-hoc wireless networks (MANETs), sensor networks and disruption-tolerant networks (DTNs). A common problem in routing in such networks involves the *indirection* from a *persistent name* (or *ID*) to a *locator*. In a dynamic network, the locator and ID-to-locator mappings are both dynamic. The dynamism increases if the locator is expressed in terms of graph topology because the link state entropy is usually higher than node geo-location entropy. Dealing with dynamic indirections in a scalable manner usually involves *delaying* the creation of the ID-to-location mapping (a.k.a “*late-binding*”), and reducing the *number and dynamism* of bindings. We illustrate these ideas in the context of MANET routing.

MANET routing classifies routing into two classes: *proactive* routing (using early binding) and *reactive or on-demand* routing (involving late-binding). In the latter class, a route is not computed ahead of time, but instead, a route query is generated on-demand. We can view this route query as similar to a name resolution process. The resolution of this route query is either a full route specification (eg: DSR), or node location (in geographical or coordinate-space routing). It is interesting to note that unstructured methods like flooding and controlled flooding are used in the MANET “route query resolution” problem. However, since “guaranteed recall” (i.e. route query to any node) is required, there is a limit on scalability of such MANET routing schemes. Scalability is recovered by placing some restrictions on relative mobility (eg: a node does not move very far in a very short time), and by using hierarchical indirection schemes that exploit this restriction. Again, like peer-to-peer networks and routed inter-networks, the key to scalability is the quality of filtering performed and the extent to which flooding is limited.

2.3 Tackling Heterogeneity Using Indirection

One of the remarkable achievements of the internet design is its *inter-networking* ability, i.e. its accommodation of heterogeneity. Heterogeneity manifests both in terms of the transmission media and applications. Each application needs to be mapped to each link in order to enable communication (See **Figure 13**). This is an $O(N^2)$ mapping problem, where N is the number of entities that need to be mapped to each other. There are two approaches to dealing with this problem: translation and overlays. The *translation* approach attempts to do this $O(N^2)$ mapping *directly* and is therefore complex. Imagine the complexity if you as a tourist had to translate all the world’s different languages! The *overlay* approach suggests an *indirect* or *indirection-based* approach, i.e. a new intermediate layer (IP) to which the applications and links are mapped (**Figure 13**). The number of mappings is now reduced to $O(N)$. Moreover, the incremental number of mappings needed to introduce a new link or a new application into the system is just one, i.e. map to IP. Continuing the language analogy, if everyone had to learn one world-standard language, English, in addition to their local languages, the communication problem between locals and tourists is solved.

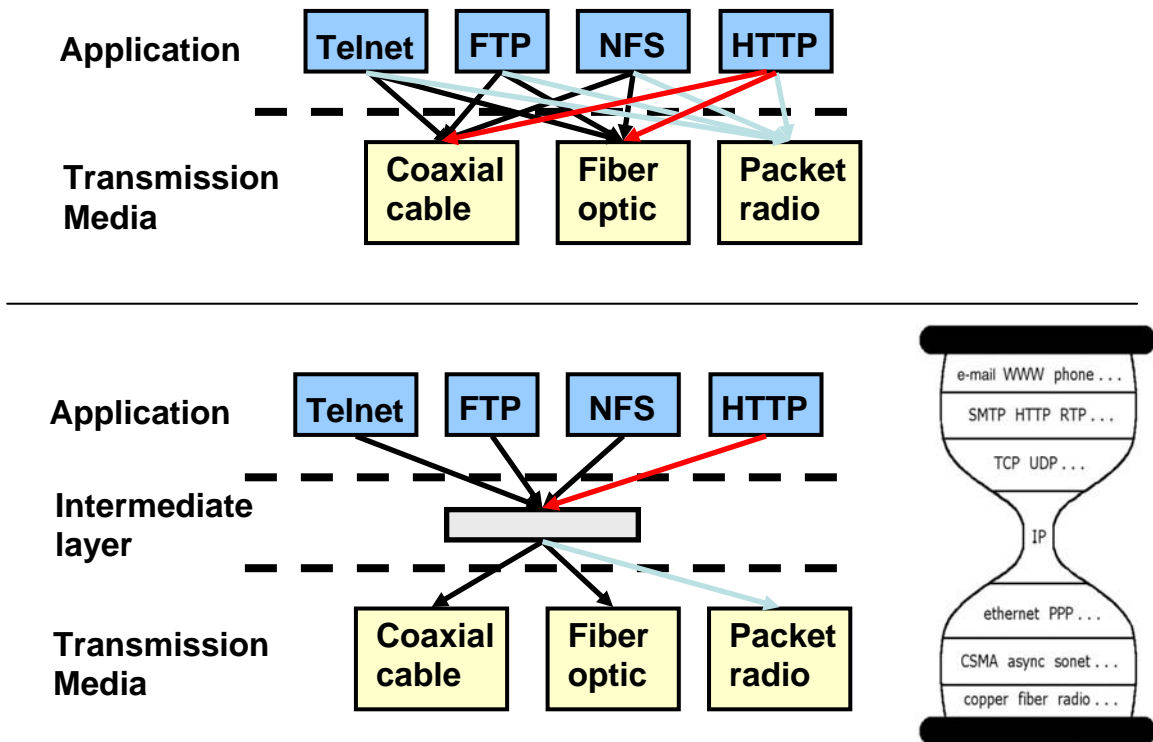


Figure 13: Internetworking Models: Translation vs Overlay. The IP HourGlass.

Observe the similarity between **Figure 13** (internetworking models) and **Figure 6** (full mesh vs hub-and-spoke networks). The switch or hub in the latter case was introduced as a pure indirection entity that enables scalability, i.e. the switch reduces the number of links needed. The role of IP in the internetworking context is similar; IP reduces the number of protocol mappings needed. Some of the other issues with translations have been discussed using the specific example of NAT earlier. It is useful to keep in mind that though the translation approach has pitfalls, it is commonly used because of simplicity, lower short-term costs and expediency.

Another lesson from IP internetworking is that overlays can be successful if the semantics and syntax to be mapped between layers is simple. Since IP offers a best-effort datagram model, with flexible data-gram sizes, it is easy to map to. However, mapping IP to lower layers involves two functions: **(a)** mapping an IP datagram to a link-level frame, and **(b)** mapping a next-hop IP address to a next-hop link-layer address. The first function is provided within IP by the *fragmentation and reassembly* procedures. The second function is provided by a separate protocol, *Address Resolution Protocol (ARP)*. In addition to IP, the control-plane infrastructure, especially routing protocols like OSPF and BGP must be easy to map to lower layers as well.

These mapping problems are relatively simple over link layers such as Ethernet that support broadcast operation or point-to-point links, having only one next-hop node. ARP can broadcast a request for a L3-to-L2 address mapping and get a unicast response. Routing protocols work well on point-to-point links. Over broadcast media, OSPF has to make certain adjustments to ensure that Hello messages, Link-State encoding, link-state packet (LSP) flooding and link-state database synchronization are done efficiently. The core issue is that an extended Ethernet LAN is logically a shared channel between nodes and routers and can provide broadcast/multicast support for adjacency maintenance/flooding etc; but Dijkstra's algorithm views it as a full-mesh of point-to-point links between the routers. A designated router (DR) and backup designated router (BDR) are used in the LSP generation, database synchronization and LSP flooding processes.

In the case of Asynchronous Transfer Mode (ATM) networks and other non-broadcast medium access (NBMA) networks, this mapping problem is non-trivial. These networks need to set up expensive signaled virtual circuits (VCs) prior to communication. A protocol like ARP cannot broadcast a mapping request (this would require a full-mesh of VCs). ARP therefore has to be done through a server, called an ATMARP server in ATM networks. In larger networks, multiple servers will be needed and the servers need to have a full mesh between them to ensure consistency of mappings. The NBMA mapping of protocols like OSPF and BGP tend to be even more complex. OSPF uses concepts like designated routers (DRs) and backup designated routers (BDRs) similar to broadcast LANs. However, communication between regular routers and DR/BDRs will be established through explicitly signaled virtual circuits. IP makes a simple assumption that nodes on a single subnet can communicate directly; however this feature will require a full mesh of VCs between all nodes and all routers. The OSPF point-to-multipoint weakens this IP-model assumption and allows partial meshes between routers. To avoid full meshes between every node, a large ATM network may need to be organized into multiple subnets, which implies the need to do explicit routing between these subnets since the IP model does not allow node on different subnets to connect directly. Proposals like RFC 1577 (Classical IP-over-ATM), RFC 2332 (NHRP: NBMA Next-Hop Resolution Protocol) and MPOA (ATM Forum's Multi-protocol over ATM standard) were developed to deal with these thorny internetworking problems.

The result was a very complex IP-to-ATM mapping. Multi-protocol Label Switching (MPLS) grew out of this debate, pioneered initially by the IP switching proposal of Ipsilon Inc. (now part of Nokia Corp.) and later by several companies including Cisco's tag switching. The rough idea was a *hybrid mapping*: use the data-plane of ATM networks (label-switching, which was at that time much faster than IP), and the control-plane of IP. This way the complex mapping of control planes can be avoided. Later, MPLS has become the basis for *traffic engineering*, i.e. providing a general set of mechanisms to trunk traffic and map them to independently set up paths (including failover paths) at both the optical- and IP levels. MPLS today substitutes a separate, sophisticated control plane in place of the standard IP control plane.

The purpose of this detour into IP-over-ATM is to emphasize how complex the overlay mappings can be when the underlying systems are diametrically opposite in architectural design (eg: VC-based vs datagram-based). The story of MPLS also teaches us that pure overlay mapping complexities can be avoided by using a new unified control-plane.

2.4 Protocol Complexity & Indirection

Indirection can also be used to divide-and-conquer a complex protocol problem. Here the indirection goes beyond mapping nodes to state variables (such as IDs in packet headers), and involves mapping protocol entities to each other. A larger protocol entity is decoupled into smaller entities and associated together flexibly using indirection mechanisms. This is the same pattern of "decouple-and-indirect" observed earlier. The price paid for decoupling or disaggregating a larger protocol entity (eg: routing and congestion control) is losing the possibility of joint optimization. For example, early internet designers considered the problem of routing and congestion control design jointly [5]. Later, they settled on a *decoupled* design where the routing function was separated from the congestion control problem. Routing is handled at the network layer and congestion control is handled at the transport layer. Congestion control today is a data-plane function assumes a flow or a sequence of packets that follow a single path and operates on small time-scales (milliseconds), whereas routing is viewed as a control-plane function operating at larger time-scales (seconds) whose job is to set up forwarding tables that facilitate the filtering and forwarding of packets belonging to any flow. The decoupled, layered protocol model maps these functions together via common concepts such as packets and forwarding tables. In other words, these *indirection* mechanisms have facilitated a flexible coupling instead of a jointly optimized design. There is also evidence that the price paid due to the loss of joint-optimization can be easily recouped through the gains due to Moore's law that allow rapid evolution, adoption, proliferation and economies of scale [6].

A similar "decouple-and-indirect" trend has been seen in the Internet QoS design space where the evolution of the network level architecture (eg: int-serv, diff-serv, RSVP etc) has been decoupled from the end-to-end QoS adaptation and signaling mechanisms (eg: RTP, SIP, de-jittering buffers, end-to-end FEC, integration

of video with congestion control, multi-paths etc). This decoupled architectural design stands in contrast with the end-to-end QoS designs in ATM networks where the evolution of both end-systems and switching elements is managed jointly. This architectural decoupling of QoS has facilitated the growth of multimedia applications like video streaming, conferencing and VoIP applications over the best-effort internet, without any QoS assurances. This example suggests that we gain several engineering benefits in return for losing the option of joint-optimization (eg: flexibility, evolvability, reuse, avoidance of “spaghetti” code etc).

It is interesting that the question of joint optimization vs architectural flexibility is being re-explored in the context of wireless networks. The challenge is to facilitate such “*cross-layer*” optimizations without losing the benefits of indirection and layering. While joint modulation and coding is performed in the PHY layer and application-layer framing is common at the transport/application interface for multimedia applications, coordination across non-adjacent layers is still not facilitated in the current layered model of network protocol organization. Ideas such as *floating* headers with self-defined semantics could facilitate state-exchanges across non-adjacent layers and distributed nodes (instead of the strict encapsulation of all headers) and play a role in resolving the cross-layer optimization debate. But the very use of such novel headers will raise new concerns of how to coordinate access to, and semantics of this new “floating” state in packet headers. No wonder that David Wheeler prophetically observed:

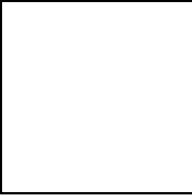
"Any problem in computer science can be solved with another layer of indirection. But that usually will create another problem."

- David Wheeler (1929-2004), chief programmer for the EDSAC project in the early 1950s.

Simple-looking virtualizations can lead to complex indirection infrastructures. For example, IP multicast was proposed as a simple extension of the IP datagram model: replace the destination address with a group address. This “*open group*” model allows arbitrary sources to send messages to a group address and places the responsibility of dynamically discovering geographically dispersed group members and maintaining a tree with the internet control plane. This choice led to complex multicast routing and address allocation protocols. As mentioned earlier, the current consensus is based upon making the IP multicast group abstraction source-specific (called SSM) that simplifies address allocation, group management, access control and multicast routing.

Finally, indirection architectures like the Internet themselves evolve and face stresses and strains. Some of the discontents over the Internet expressed in the recent past include the lack of features (eg: QoS, source-controlled routing, multicast, anycast etc) and lack of accountability and protection (eg: DDoS etc). These architectural brittleness issues can be traced to the fact that some of the indirection mechanisms are overloaded with semantics. Hosts are tied to IP addresses since the latter is both an interface ID and a locator (with several embedded network IDs). This IP address overloading poses problems for multi-homing and mobility. Similarly, services are tied to hosts through domain-specific or host-specific URLs, even though services can be distributed in nature through replication, migration and composition. Packets might require processing at intermediaries such as NATs and firewalls before they reach the destination. These intermediaries are today considered troublesome in the IP architectural design since they break the end-to-end model.

A recent proposal, called *layered naming* [4] suggests resolving this overloading by evolving the name system of the Internet by introducing new levels of indirection, i.e. decouple-and-indirect. In particular, it suggests four levels of IDs: *service identifiers (SIDs)* which are host-independent data names; *endpoint identifiers (EIDs)* which are location-independent host-names, and pure *locators* (IP addresses). Application specific search or lookup using *user-level descriptors* (eg: search keywords) will yield SIDs that is used by the application layer as handles. SIDs are mapped to EIDs, and transport connections opened which then use EIDs as handles. EIDs are mapped IP addresses. SIDs are proposed to be flat, semantic-free IDs, and DHTs proposed to resolve flat names scalably. The proposal generalizes the functions of mobile IP, multi-homing, service location-transparency, intermediary services as indirections or mappings between these basic set of IDs.



To avoid overloading of indirection, layered naming suggests that names should bind protocols only to relevant aspects of underlying structure. To allow flexibility for intermediaries to offer higher-layer processing, layered naming suggests that a network entity be able to direct resolutions of its name to chosen delegates as well as its own location. This delegation principle allows middleboxes like NATs, firewalls, email filters etc to become first-class citizens of the enhanced architecture, and not as layer-violation entities that break end-to-end semantics today.

In summary, virtualization and indirection are evolution-enablers that allow architectural and protocol complexity to be managed as new requirements or constraints are placed on the system. The usual model is to consider overloaded or jointly optimized entities and use the idea of “decouple-and-indirect”, i.e. introduce new levels of carefully designed indirection. Overlay infrastructures like Planetlab, and mechanisms like anycast can then be used to phase in new long-running prototypes of such services onto the Internet.

3 Summary

Virtualization is a useful form of abstraction in distributed computing systems. The virtual entity bears some resemblance to lower-layer entities, but has enhanced or simplified semantic and performance characteristics. Indirection is the means of achieving abstractions such as virtualizations. We have seen how indirection infrastructures are designed to address problems of scalability and heterogeneity. We studied how state information such as identifiers can be used to facilitate distributed indirections. Indirection is a concept that unifies several protocols in the Internet, and we saw several examples of this ranging from SONET at the physical layer and DHTs at the application layer. Filtering combined with indirection allows scalable designs. Scalable indirections may sometimes overload identifiers, by embedded further structure and locator information in them (eg: DNS names, IP addresses). Distributed Hash Tables (DHTs) enable the scalable and guaranteed resolution of flat IDs by moving the structure to the organization of the distributed pieces of the mapping tables in the overlay network. The challenge in modern dynamic networks (eg: p2p, MANETs, sensor networks, DTNs etc) is to provide scalable indirection and rendezvous mechanisms despite strains in maintaining overlay structures and valid state information due to dynamism. The problems of managing protocol complexity and architectural evolution can also be framed as indirection issues. We can then use the method of “decouple-and-indirect” which simplifies indirection entities and introduces new levels of indirection. As networking research merges deeper with middleware research, novel forms of indirection are emerging such as network coding, XML-based self-describing state and active networking (not covered in these notes).

4 References

- [1] S. Ratnasamy, S. Shenker, S. McCanne, "Towards an Evolvable Internet Architecture", In Proc. ACM SIGCOMM Conference 2005, August 2005.
- [2] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In Proc. ACM SIGCOMM Conference 2002, pg. 73-88, August 2002.
- [3] S. Rhea, B. Godfrey, B. Karp, J. Kubiawicz, S. Ratnasamy, S. Shenker, I. Stoica, , and H. Yu, "OpenDHT: A public DHT service and its uses," In Proc. ACM SIGCOMM Conference, August 2005.
- [4] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, Ion Stoica and Michael Walfish, "A layered naming architecture for the internet," In Proc. ACM SIGCOMM 2004, August 2004.
- [5] D. Bertsekas, R. Gallager, "Data Networks," Longman Higher Education, 1986.
- [6] V. Kawadia, P.R. Kumar, "A cautionary perspective on cross-layer design," IEEE Wireless Communications Magazine, Vol. 12, Issue 1, pg 3-11, Feb. 2005