LECTURE 2
Connecting Networks of Networks: The Internetworking Problem

This lecture looks at the issues—primarily those of heterogeneity and scale—that arise in internetworking different networks together. Using the Internet and its network protocol, IP, as an example, we will discuss some important design principles and the techniques that have enabled the Internet to scale to many millions of connected networks. Many of these principles were articulated in Clark's retrospective paper on the design of the Internet protocols. A section at the end gives a quick summary of Cerf and Kahn's seminal paper on the protocol that eventually led to the TCP/IP network stack.

■ 2.1 The Internetworking Problem: Many Different Networks

In the previous lecture, we discussed the principles underlying packet switching, and developed a scheme for connecting local-area networks (LANs) together. The key idea was a device called a switch, which forwarded packets between different LANs.

While such networks work and are common, they are not enough to build a global data networking infrastructure. They have two main failings:

1. *They don't accommodate heterogeneity.*

2. *They don't scale well to large networks.*

Each point bears some discussion. Although common, Ethernets are by no means the only network link technology in the world. Indeed, even before Ethernets, packets were being sent over other kinds of links including wireless packet radio, satellites, and telephone wires. Ethernets are also usually ill-suited for long-haul links that have to traverse hundreds of miles. Advances in link technologies of various kinds has always been occurring, and will continue to occur, and our goal is to ensure that *any* new network link technology can readily be accommodated into the global data networking infrastructure.

The LAN switching solution we studied in Lecture 1 requires switches to store per-host (actually, per-interface) state in the network; in the absence of such state for a destination,

switches revert to flooding the packet to all the links in the network. This does not scale to more than a moderate number of hosts and links.

The internetworking problem may be stated as follows: Design a scalable network infrastructure that interconnects different smaller networks, to enable packets to be sent between hosts across networks of very different performance characteristics.

Some examples of the differing characteristics include:

- **Addressing.** Each network might have its own way of addressing nodes; for example, Ethernet's use a 6-byte identifier while telephones use a 10-digit number.
- **Bandwidth & latency.** Heterogeneity in bandwidths range from a small number of bits/s (e.g., power lines) to many Gigabits/s, spanning many orders of magnitudes. Similarly, latencies can range from microseconds to several seconds.
- **Packet size.** The maximum packet sizes in general will vary between networks.
- **Loss rates.** Networks differ widely in the loss rates and loss patterns of their links.
- **Packet routing.** Packet routing could be handled differently by each constituent network, e.g., packet radio networks implement different routing protocols from wireline networks.

Our scalability goal is to reduce the state maintained by the switches and the bandwidth consumed by control traffic (e.g., routing messages, periodic flooding of packets/messages, etc.) as much as we can.

2.2 Handling Heterogeneity: A Universal Network Layer

Communication between different networks in an internetwork is facilitated by entities called *gateways.* Gateways interface between two networks with potentially differing characteristics and move packets between them. There are at least two ways of interfacing between different networks: *translation* and a *unified network layer*.

2.2.1 Translating Gateways Considered Harmful

Translation-based gateways work by translating packet headers and protocols over one network to the other, trying to be as seamless as possible. Examples of this include the OSI X.25/X.75 protocol family, where X.75 is a translation-based "exterior" protocol between gateways.

There are some major problems with translation.

- **Feature deficiency.** As network technologies change and mature and new technologies arise, there are features (or bugs!) in them that aren't preserved when translated to a different network. As a result, some assumption made by a user or an application breaks, because a certain expected feature is rendered unusable since it wasn't supported by the translation.
- **Poor scalability.** Translation often works in relatively small networks and in internetworks where the number of different network types is small. However, it scales

poorly because the amount of information needed by gateways to perform correct translations scales proportional to the square of the number of networks being interconnected. This approach is believed to be untenable for large internetworks.[1]

■ 2.2.2 Universality and Robustness Design Principles

The designers of the ARPANET (the precursor to today's global Internet) protocols recognized the drawbacks of translation early on and decided that the right approach was to standardize some key properties across all networks, and define a small number of features that all hosts and networks connected to the Internet must implement. In their seminal 1974 paper on IP and TCP, Cerf and Kahn describe the rationale behind this decision, pointing out the drawbacks of translation gateways mentioned above.

Over time, we have been able to identify and articulate several important design principles that underlie the design of Internet protocols. We divide these into *universality* principles and *robustness* principles, and focus on only the most important ones (this is a subjective list).

Universality principles

U1. IP-over-everything. All networks and hosts must implement a standard network protocol called **IP**, the **Internet Protocol**. In particular, all hosts and networks, in order to be reachable from anywhere, must implement a standard well-defined addressing convention. This removes a scaling barrier faced by translation-based approaches and makes the gateways simple to implement.

The ability to support IP is necessary (and largely sufficient) for a network to be part of the Internet. The "sufficient" part also requires that the network's IP addresses be globally reachable, in which case the network is said to be part of the "global Internet".

U2. Best-effort service model. A consequence of the simple internal architecture of the gateways is that the service model is best-effort. All packets are treated (roughly) alike and no special effort is made inside the network to recover from lost data in most cases. This well-defined and simple service model is a major reason for the impressive scalability of the architecture. A key consequence of the best-effort design decision is that it is almost trivial for a link technology to support IP.

U3. The end-to-end arguments. The end-to-end arguments, articulated after the original design of TCP/IP, permeates many aspects of the Internet design philosophy. Its layered architecture keeps the interior of the network simple and moves all the complicated machinery for reliability, congestion control, etc. to the end-hosts where they can be implemented more completely. Historically, the original design of the Internet protocols had TCP and IP combined both at the end-points and in the gateways. The separation between the two, with TCP functionality being removed from the gateways, was an important step in the evolution of the Internet.

[1] As an aside, it's worth noting that the widespread deployment of NATs is making portions of the Internet look a lot like translation gateways! We will revisit this point later in this lecture when we discuss scaling issues, but the important difference is that NATs are used to bridge different *IP* networks and don't suffer from the translation-scalability problem, but prevent applications that embed IP addresses in them from working well (among other things).

Robustness principles

R1. Soft-state inside the network. Most of the state maintained in the gateways is *soft-state*, which is easily refreshed when it is lots or corrupted. Routing table state is a great example of this; gateways use these tables to decide how to route each packet, but the periodic refreshes in the form of routing updates obviate the need for complex error handling software in the routers. In this way, gateways and routing protocols are designed to overcome failures as part of their *normal* operation, rather than requiring special machinery when failures occur.

R2. Fate sharing. When a host communicates with another, state corresponding to that communication is maintained in the system. In the Internet, the critical state for this is shared between the two (or more) communicating entities and not by any other entities in the network. If one of the end hosts fails, so do the others; in this way they share fate. If a gateway fails en route, the end-to-end communication doesn't—the state in the gateway is soft and the gateway doesn't share fate with the end hosts. This is in stark contrast with the OSI X.25 approach where gateways maintained hard connection state and end hosts shared fate with them.

R3. Conservative-transmission/liberal reception. "Be conservative in what you send; be liberal in what you accept."[2] This guideline for network protocol design significantly increases the robustness of the system. This principle is especially useful when different people, who may have never spoken to each other, code different parts of a large system. This is important even when writing code from a specification, since languages like English can be ambiguous, and specifications can change with time. A simple example of this principle is illustrated in what a TCP sender would do if it saw an acknowledgment that acknowledged a packet it had never sent. The worst behavior is to crash because the receiver isn't prepared to receive something it didn't expect. This is bad—what's done is instead is to silently drop this acknowledgment and see what else comes from the peer. Likewise, if a sender transmits a packet with a non-standard option, it should do so only if it has explicitly been negotiated.

■ 2.2.3 Weaknesses

The Internet architecture does have some weaknesses, many of which are a consequence of its original design goals (see Clark's retrospestive paper).

- The architecture fundamentally relies on the trustworthiness of end-systems. It does not consider the possibility of malicious end systems and how to build a trusted network despite this.

- Greedy sources aren't handled well (non-cooperating users, buggy or malicious implementations). TCP does a good job of sharing bandwidth, but there's no enforcement mechanism in place both for non-TCP protocols and for TCP-based bandwidth hogs.

[2]This statement has been attributed to Jon Postel.

- Security issues, for a long time, weren't considered as paramount as they are these days. We will discuss security problems at length in later lectures, including hijacking, denial-of-service, and virus attacks.

- Weak accounting and pricing tools. Although one of the original goals, the ability to perform fine-grained accounting and provide fine-grained usage-based pricing hasn't gained much traction. The accounting area has improved significantly in recent years, but it is not clear that usage-based pricing is even desirable (in terms of what Internet customers seem comfortable with).

- Administration and management tools are not particularly mature. Again, this has been changing especially over the past few years, but it's extremely hard to diagnose problems when they occur.

- The need for incremental deployment. Not really a fundamental "weakness," but this property has to be recognized by every protocol designer. Deploying a new idea or a protocol modification has to be done incrementally, because "flag days" when one can bring the entire Internet to a halt and upgrade the infrastructure are impossible!

2.2.4 The standards process

An interesting aspect of the continuing development of the Internet infrastructure is the process by which standards are set and developments are made. By and large, this has been a very successful exercise in social engineering, in which people from many different organizations make progress toward consensus. The Internet Engineering Task Force (IETF), a voluntary organization, is the standards-setting body. It meets every 4 months, structured around short-lived working groups (usually less than 2 years from inception to termination). Internet standards are documented in RFCs ("Request for Comments" documents) published by the IETF (several different kinds of RFCs exist), and documents in draft form before they appear as RFCs are called "Internet Drafts." Check out http://www.ietf.org/ for more information.

Working groups conduct a large amount of work by email, and are open to all interested people. In general, voting on proposals are shunned, and the process favors "rough consensus." Running, working code is usually a plus, and can help decide direction. It used to be the case that at least two independent, interoperating implementations of a protocol were required for a standard to be in place, but this seems to be generally ignored these days.

2.3 Achieving scalability

The fundamental reason for the scalability of the Internet's network layer is its use of **topological addressing.** Unlike an Ethernet address that is location-independent and always identifies a host network interface independent of where it is connected in the network, an IP address of a connected network interface depends on its *location in the network topology*. This allows routes to IP addresses to be *aggregated* in the forwarding tables of the routers,

and allows routes to be *summarized* and exchanged by the routers participating in the Internet's routing protocols. In the absence of aggressive aggregation, there would be no hope for scaling the system to huge numbers; indeed, this is challenging enough even with the techniques we use today.

Because an IP address signifies location in the network topology, the Internet's network layer can use a variant of classical *area routing* to scalably implement the IP forwarding path.

■ 2.3.1 The area routing idea

The simplest form of classical area routing divides a network layer address into two parts: a fixed-length "area" portion (in the most significant bits, say) and an "intra-area" address. Concatenating these two parts gives the actual network address. For example, one might design a 32-bit area routing scheme with 8 bits for the area and 24 bits for the intra-area address. In this model, forwarding is simple: If a router sees a packet not in its own area, it does a lookup on the "area" portion and forwards the packet on, and conversely. The state in the forwarding tables for routes not in the same area is at most equal to the number of areas in the network, typically much smaller than the total number of possible addresses.

One can extend this idea into a deeper hierarchy by recursively allocating areas at each level, and performing the appropriate recursion on the forwarding path. With this extension, in general, one can define "level 0" of the area routing hierarchy to be each individual router, "level 1" to be a group of routers that share a portion of the address prefix, "level 2" to be a group of level 1 routers, and so on. These levels are defined such that for any level i, there is a path between any two routers in the level $i-1$ routers within level i that does not leave level i.

There are two reasons why this classical notion of area routing does not directly work in practice:

1. The natural determinant of an area is administrative, but independently administered networks vary widely in size. As a result, it's usually hard to determine the right size of the "area" field at any level of the hierarchy. A fixed length for this simply does not work.

2. Managing and maintaining a carefully engineered explicit hierarchy tends to be hard in practice, and does not scale well from an administrative standpoint.

■ 2.3.2 Applying area routing to the Internet: Address classes

The second point above suggests that we should avoid a deep hierarchy that requires manual assignment and management. However, we can attempt to overcome the first problem above by allocating network addresses to areas according to the expected size of an area.

When version 4 of IP ("IPv4") was standardized with RFC 791 in 1981, addresses were standardized to be 32-bits long. At this time, addresses were divided into classes, and organizations could obtain a set of addresses belonging to a class. Depending on the class, the first several bits correspond to the "network" identifier and the others to the "host" identifier. *Class A* addresses start with a "0", use the next 7 bits of network id, and the last 24 bits of host id (e.g., MIT has a Class A network, with addresses in dotted-decimal

notation of the form 18.*). *Class B* addresses start with "10" and use the next 14 bits for network id and the last 16 bits for host id. *Class C* addresses start with "110" and use the next 21 bits for network id, and the last 8 bits for host id.[3]

Thus, in the original design of IPv4, areas were allocated in three sizes: *Class A* networks had a large number of addresses, 2^{24}, *Class B* networks had 2^{16} addresses each, and *Class C* networks had 2^8 addresses each.

The router forwarding path in IPv4 is a little more involved than for the exact-match required by LAN switches, but is still straightforward: a router determines for an address not in its area which class it belongs to, and performs a fixed-length lookup depending on the class.

■ 2.3.3 The problem with class-based addressing

The IETF realized that this two-level network-host hierarchy would soon eventually prove insufficient and in 1984 added a third hierarchical level corresponding to "subnets". Subnets can have any length and are specified by a 32-bit network "mask"; to check if an address belongs to a subnet, take the address and zero out all the bits corresponding to zeroes in the mask; the result should be equal to the subnet id for a valid address in that subnet. The only constraint on a valid mask is that the 1s must be contiguous from most significant bit, and all the 0s must be in the least significant bits.

With subnets, the class-based addressing approach served the Internet well for several years, but ran into scaling problems in the early 1990s as the number of connected networks continued growing dramatically. The problem was *address depletion*—available addresses started running out.

It's important to understand that the problem isn't that the entire space of 2^{32} addresses (about 4 billion) started running out, but that the class-based network address assignment started running out. This is the result of a fundamental inefficiency in the coarse-grained allocation of addresses in the Class A and (often) Class B portions of the address space.[4] The main problem was that Class B addresses were getting exhausted; these were the most sought after because Class A addresses required great explanation to the IANA (Internet Assigned Numbers Authority) because of the large (2^{24}) host addresses, which Class C addresses with just 256 hosts per network were grossly inadequate. Because only $2^{14} = 16384$ Class B networks are possible, they were running out quickly.

Solution 1: CIDR

In a great piece of "just-in-time" engineering, the IETF stewarded the deployment of **CIDR** (pronounced "Cider" with a short "e"), or **Classless Inter-Domain Routing** recognizing that the division of addresses into classes was inefficient and that routing tables were exploding in size because more and more organizations were using non-contiguous Class C addresses.

CIDR optimizes the common case. The common case is that while 256 addresses are insufficient, most organizations require at most a few thousand. Instead of an entire Class

[3]Class D addresses are used for IP multicast; they start with "1110" and use the next 28 bits for the group address. Addresses that start with "1111" are reserved for experiments.

[4]For instance, MIT and Stanford each had entire Class A's to themselves. MIT still does!

B, a few Class C's will suffice. Furthermore, making these *contiguous* will reduce routing table sizes because routers aggregate routes based on IP prefixes in a classless manner.

With CIDR, each network gets a portion of the address space defined by two fields, A and m. A is a 32-bit number (often written in dotted decimal notation) signifying the address space and m is a number between 1 and 32. If a network is assigned an address region denoted A/m, it means that it gets the 2^{32-m} addresses all sharing the first m bits of A. For example, the network "18.31/16" corresponds to the 2^{16} addresses in the range [18.31.0.0, 18.31.255.255].

"Solution 2": NAT

NATs are an expedient solution to the address depletion problem. They allow networks to use *private IP addresses*, which different networks can reuse, and deploy globally reachable gateways that *translate* between external address/transport-port pairs and internal ones.

We will discuss NATs in more detail in Recitation #3 (T3). RFC 1631 gives a quick and simple overview of NATs that's easy to understand.

Initiating connections from inside a private network via a NAT is easier (and has been supported from the first deployed NATs). The NAT maintains mappings between internal and external address/port pairs, and modifies the IP *and TCP* header fields on the connection in both directions. Because the TCP checksum covers a portion of the IP header (including the IP addresses), the TCP header needs to be modified.

NATs break several applications, notably ones that use IP addresses in their own payload or fields. Usually, these applications require an application-aware NAT (e.g., FTP).

More recent work has shown that it is possible to use NATs to obtain a form of "global" reachability, by using DNS as a way to name hosts uniquely inside private networks. The result of a DNS lookup in this case would be an IP address/transport-port pair, corresponding to the globally reachable NAT. The NAT would maintain an internal table translating this address/port pair to the appropriate internal address/port pair.

Solution 3: IPv6

The long-term solution to the address depletion problem is a new version of IP, IPv6. We will discuss this in Section 2.4.2 after discussing the IPv4 packet forwarding path in a switch (i.e., what a switch must do to forward IP packets).

■ 2.3.4 CIDR lookups: Longest prefix match

The forwarding step with CIDR can no longer be based on determining the class of an address and doing a fixed-length match. Instead, a router needs to implement a *prefix match* to check if the address being looked-up falls in the range A/m for each entry in its forwarding table.

A simple prefix match works when the Internet topology is a tree and there's only one shortest path between any two networks in the Internet. However, the toplogy of the Internet is not a tree; many networks *multi-home* with multiple other networks for redundancy and traffic load balancing (redundancy is the most common reason today).

The consequence of having multiple possible paths is that a router needs to decide on its forwarding path which of potentially several matching prefixes to use for an address

being looked-up. Be definition, IP (CIDR) defines the correct address as the *longest prefix* that matches the sought address. As a result, each router must implement a *longest prefix match (LPM)* algorithm on its forwarding path.

2.4 Other IP layer details

We now look at a few other details of the IP layer, dividing our discussion into IPv4 and IPv6 (the next version of IP from 4[5]).

2.4.1 IPv4

Fragmentation and Reassembly

Different networks do not always have the same MTU, or Maximum Transmission Unit. When an IP gateway receives a packet[6] that needs to be forwarded to a network with a smaller MTU than the size of the packet, it can take one of several actions.

1. Discard the packet. In fact, IP does this if the sender set a flag on the header telling gateways not to fragment the packet. After discarding it, the gateway sends an error message using the ICMP protocol to the sender. [What are some reasons for providing this feature in the Internet?]

2. Fragment the packet. The default behavior in IPv4 is to fragment the packet into MTU-sized fragments and forward each fragment to the destination. There's information in the IP header about the packet ID and offset that allows the receiver to reassemble fragments.

Time-to-live (TTL)

To prevent packets from looping forever, the IP header has a TTL field. It's usually decremented by 1 at each router and the packet is discarded when the TTL is zero.

Type-of-service (TOS)

The IP header includes an 8-bit type-of-service field that allows routers to treat packets differently. It's largely unused today, although we'll later see how differentiated services intends to use this field.

Protocol field

To demultiplex an incoming packet to the appropriate higher (usually transport) layer, the IP header contains an 8-bit protocol field.

Header checksum

To detect header corruption, IP uses a (weak) 16-bit header checksum.

[5] IPv5 didn't last long!
[6] Or *datagram.*

IP options

IP allows nodes to introduce options in its header. A variety of options are defined but are hardly used, even when they're useful. This is because it's expensive for high-speed routers (based on the way they're designed today) to process options on the fast path. Furthermore, most IP options are intended for end-points to process, but IPv4 mandates that all routers process all options, even when all they have to do is ignore them. This makes things slow—and is a severe disincentive against options. Engineering practice today is to avoid options in favor of IP-in-IP encapsulation or, in many cases, having the routers peek into transport headers (e.g., the TCP or UDP port numbers) where the fields are in well-known locations and require little parsing.

■ 2.4.2 IPv6

IPv6 is the next version of IP. Its development is motivated by several shortcomings and problems with IPv4. The main problem that IPv6 solves is the address space shortage problem.

Its design challenges, and how it achieves these goals include:

1. Incremental deployment. No more flag days are possible on an Internet with over tens of millions of nodes. Painstakingly engineered to coexist and seamlessly transition from IPv4.

2. Scale. It has been designed for a scale much larger than today's Internet. The main feature that enables this is a *huge* 128-bit address space.

3. Easy configuration. Attempts to reduce manual network configuration by an auto-configuration mechanism. Here, end node interfaces can pick an address automatically using a combination of provider prefix and local hardware MAC address.

4. Simplification. Gets rid of little used stuff in the IP header, such as fragment offsets, etc. Even though addresses are 4X bigger, the base header size is only doubled from IPv4.

5. Improved option processing. Changes the way options are encoded to enable efficient forwarding. Options (e.g., security options) are placed in separate extension headers that routers can easily ignore.

6. Authentication & privacy. Network-level security is built-in for applications that desire it.

Addresses & Flows

IPv6 addresses are 128 bits long. The paper describes the details of how they are organized. An important point to note is that most wide-area communication is expected to use *provider-based* addresses, where contiguous address blocks are allocated to Internet service providers (ISPs) rather than directly to organizations. [What does this mean when an organization changes its ISP?]

IPv6 also has *local-use* addresses that don't have global scope, with link-local (e.g., same LAN) and site-local (within the organization) scope. These allow intra-organization communication even if disconnected from the global Internet.

To enable incremental deployment, IPv6 addresses can contain embedded IPv4 addresses. IPv6 also supports *anycast*, where routers will route a packet to the closest (defined in terms of the metric used by routers, such as hop count) of many interfaces that answers to that address. Anycast addresses aren't special in IPv6; they come from the space of unicast addresses.

Flows are an important useful addition to IPv6. You can think of a flow as a TCP connection or as a stream of UDP traffic between the same host-port pairs. IPv6 explicitly formalizes this notion and makes flow labels visible to routers and network-layer entities in the IP header. Flow labels are chosen randomly by end-hosts to enable good classification by network entities.

Fragmentation

IPv6 does away with network-layer fragmentation and reassembly. End-points are expected to perform path-MTU discovery. Locally, links are free to perform fragmentation and reassembly if they so desire. All IPv6 networks must handle an MTU of at least 1280 bytes.

We will study more details of IPv6 in Recitation #3 (T3).

■ 2.5 Summary

The Internet architecture is based on design principles that provide universality and improve robustness to failures. It achieves scaling by using an addressing structure that reflects network topology, enabling topological address information to be aggregated in routers.

■ Appendix: Notes on Cerf & Kahn's original TCP/IP [CK74]

Cerf & Kahn describe their solutions to the internetworking problems in their seminal 1974 paper. This early work has clearly had enormous impact, and this is one of the main reasons we study it despite the actual details of today's IP being very different. Some of the design principles have been preserved to date. Furthermore, this is a nice paper in that it presents a design with enough detail that it feels like one can sit in front of a computer and code up an implementation from the specifications described!

The following are the key highlights of their solution:

- Key idea: **gateways.** Reject translation in favor of gateways.
- IP over everything—beginnings of protocol hourglass model evident in first section of paper.
- As a corollary, addressing is common to entire internetwork. Internal characteristics of networks are of course different.

- Standard packet header format. But notice mixing of transport information in routing header. (And some fields like ttl are missing.)
- Gateways perform *fragmentation* if needed, *reassembly* is done at the end hosts.
- **TCP:** Process-level communication via the TCP, a reliable, in-order, byte-stream protocol. And in fact, they had the TCP do reassembly of segments (rather than IP that does it today at the receiver). TCP and IP were tightly inter-twined together; it took several years for the TCP/IP split to become visible.
- The spent a lot of time on figuring out how multiple processes between the same end hosts communicate. They considered two options:

  1. Use same IP packet for multiple TCP segments.
  2. Use different packets for different segments.

  They didn't consider the option of many TCP connections between the same hosts!
- 2 is simpler than 1 because it's easier to demultiplex and out-of-order packets are harder to handle in 1. [Under what conditions might 1 be better?]
- Demux key is port number (typically one process per port).
- Byte-stream TCP, so use ES and EM flags. EM, ES, SEQ and CT adjusted by gateways. I.e., gateways look through and modify transport information. End-to-end checksums for error detection.
- Sliding window, cumulative ACK-based ARQ protocol (with timeouts) for reliability and flow control. Similar to CYCLADES and ARPANET protocols.
- Window-based flow control. Also achieve process-level flow control.
- Implementation details/recommendations for I/O handling, TCBs.
- First-cut connection setup and release scheme. (Heavily modified subsequently to what it is today.)
- Way off the mark on accounting issues! (The problem is a lot harder than they anticipated.)
- One-line summary of impact: A seminal paper that led to today's dominant data network architecture.