Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.892 Fall 1999 **Lecture 2: Internetworking** September 13, 1999

Overview. In this lecture, we'll look at the issues that arise in internetworking different networks together. We will study the Internet design principles, IP, the protocol that binds different networks together, the Internet's best-effort service model, its design philosophy, protocol deployment issues, the standards process in the IETF, IPv4, and IPv6.

Readings: CK74, Cla88, KP91

1 Internet Design Principles

1.1 The Problem: Many Different Networks

The fundamental problem of internetworking is how to send packets between different networks with very different characteristics. 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 will in general be different in different networks.
- **Loss rates.** Networks differ widely in the loss rates and loss patterns of their links.
- **Packet routing.** Packet routing could be different in different networks, e.g., packet radio networks implement different routing protocols from wireline networks.

The primary challenge of internetworking is to handle these differences and efficiently transfer data between receivers connected over paths that span many different networks.

1.2 Gateways

Communication between different networks in an internetwork is facilitated by entities called *gateways.* Gateways are entities that 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 *standardization.*

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 users and applications breaks, because a certain expected feature is rendered unusable because 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 inter-connected. This approach is clearly untenable for large internetworks.

1.3 Design Principles

The designers of the Internet ("big I") protocols recognized the drawbacks of translation early on and (correctly) 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 (then ARPANET) must implement. This brings us to the first design principle underlying the Internet protocols: *IP-over-everything.*

P1. 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 fundamental scaling barrier faced by translation-based approaches and makes the gateways simple to implement.

P2. End-to-end argument. This principle, articulated much after the original, 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.

P3. 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.

P4. Soft-state. 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 sue these tables to decide how to route each packet, but the periodic refreshes in the form of routing updates obviates 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.

P5. 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.

P6. Conservative-transmission/liberal reception. "Be conservative in what you send; be liberal in what you accept." This is an unwritten rule in (Internet) protocol design and 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 it isn't prepared to receive something it didn't expect. This is bad—what's done is instead is to silently drop this ACK 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.

P7. Layering. But with a twist—bypasses are possible across layers. Furthermore, the Internet uses a simpler layering model than the complex 7-layer OSI stack (no one has actually seen some of the OSI layers in existence!): physical, link, network, transport, and application. Because of IP-over-everything, the protocol layer structure has an "hourglass" feel to it, with IP being the thin part of the hourglass, running numerous applications over numerous network technologies.

P8. Decentralization. No single entity administers the Internet today.

1.4 Weaknesses

Some weaknesses of the Internet architecture include:

- Greedy sources aren't handled well (non-cooperating users, buggy or malicious implementations). TCP does a good job of sharing bandwidth, but more and more traffic isn't TCP.
- Weak accounting and pricing tools. One of the original goals, but hasn't developed much.
- Administration and management tools are not particularly mature.
- Incremental deployment. Not really a fundamental "weakness," but has to be recognized by every protocol designer.

1.5 Standards process

- The IETF: voluntary organization, meetings every 4 months, short-lived working groups (usually less than 2 years). Email discussions.
- The dogma: "We reject kings, presidents, and voting; we believe in rough consensus and running code." (Dave Clark, 1992).
- Usually need at least 2 independent, interoperating implementations for a standard.

2 The Original TCP/IP

Cerf & Kahn describe their solutions to the internetworking problems in this 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: A great paper that details an internetworking protocol of great significance. Enough details to actually implement it, but not a dry document to read!

3 IPv4

3.1 Addressing

When version 4 of IP was standardized in 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 net, 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. 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.

This two-level network-host hierarchy quickly proved insufficient and in 1984 a third hierarchical level corresponding to "subnets" was added. 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.

In the early 1990s, IPv4 addresses started running out. In a great piece of "just-in-time" engineering, the IETF deployed 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. 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.

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 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. We will discuss routing table aggregation in more detail when we discuss routing protocols in a few weeks.

3.2 Fragmentation and Reassembly

Different networks do not always have the same MTU, or Maximum Transmission Unit. When an IP gateway receives a packet[1] 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] Or *datagram*.

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.

3.3 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.

3.4 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.

3.5 Protocol field

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

3.6 Header checksum

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

3.7 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.

4 IPv6

IPv6 (or IPng, for "next generation") is the next version of IP. Its development is motivated by several shortcomings and problems with IPv4. 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 autoconfiguration 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., secutiry 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.

4.1 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.

4.2 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.