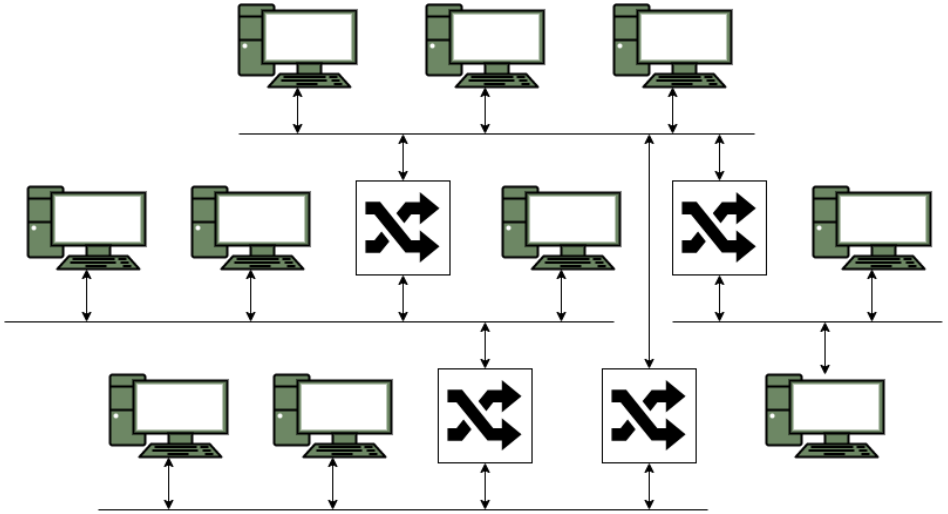
CS4650 Topic 22: Internet Basics

Internet Basics

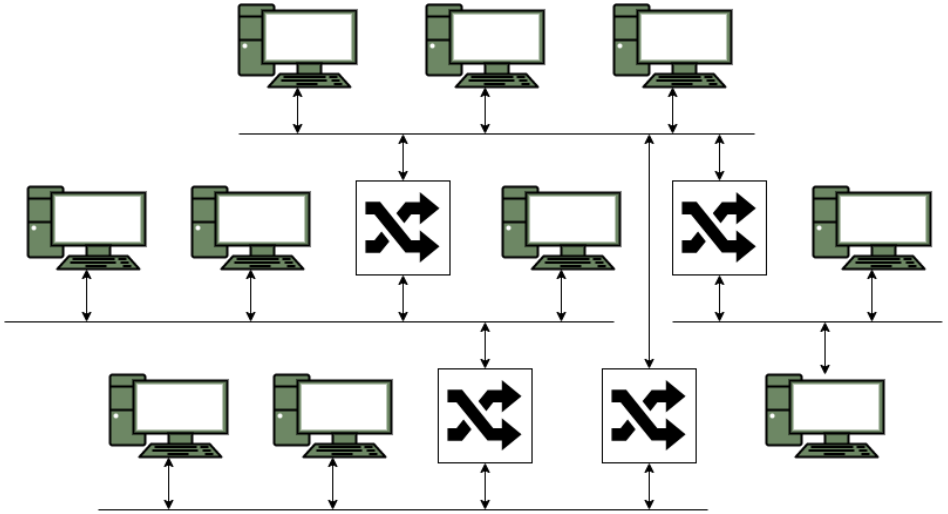
- To understand the operation of some cloud computing mechanisms, a basic understanding of the architecture of the Internet is useful.
- A deeper dive into the Internet is provided in CS3800!

Interconnected Networks



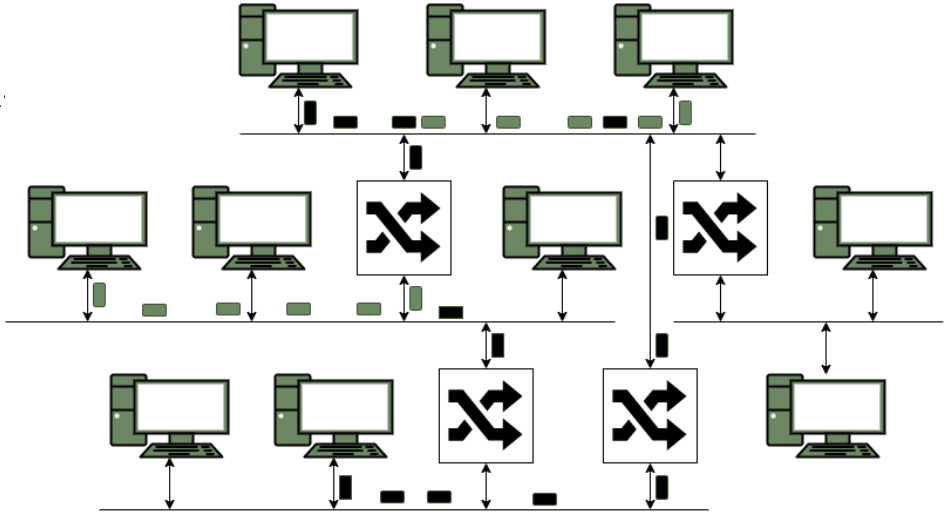
- The Internet consists of many computers, or *nodes*, interconnected by *networks*.
- There are many types of computers on the Internet.
- There are many types of networks, such as ethernet and wifi, using various mediums, such as copper wires, fiber optics, or radio waves.
- Each network has a limited size and capacity (number of nodes). To make a complete Internet, the networks are interconnected by *switches*.

Switches



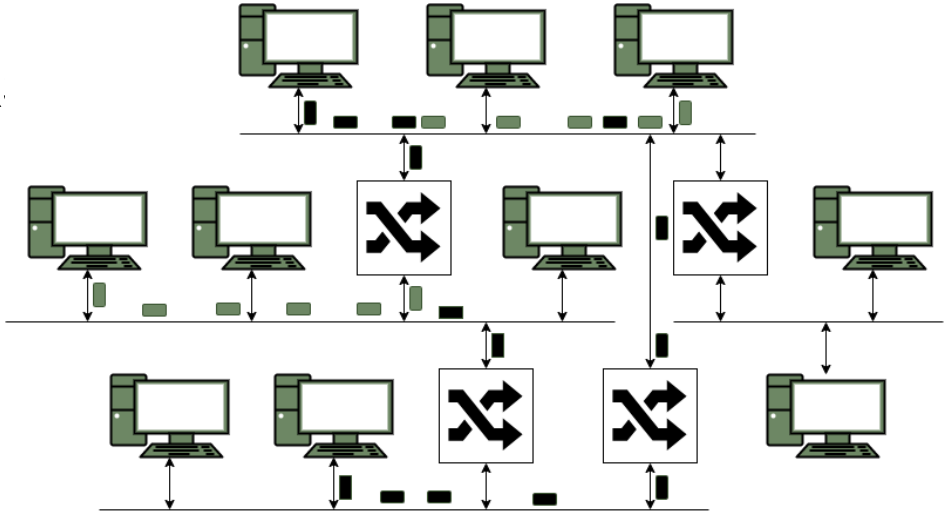
- Larger networks can be formed by interconnecting networks, or *links*, with *switches*.
- A switch is a node (computer) that connects to two different links (or maybe more).
- A switch listens for messages on both of its links. When it finds a message on one link that should be passed to the second link, it reads that message in, then writes it on the other link.

Packet Switched Network



- In a packet switched network, long messages are split into tiny packets.
- The packets are sent, one at a time, through the network.
- When a sender is going to send a packet, it first listens to the link to see if it is busy.
- If the link is busy, the sender waits.
- If the link is not busy, the sender sends the packet to the link.

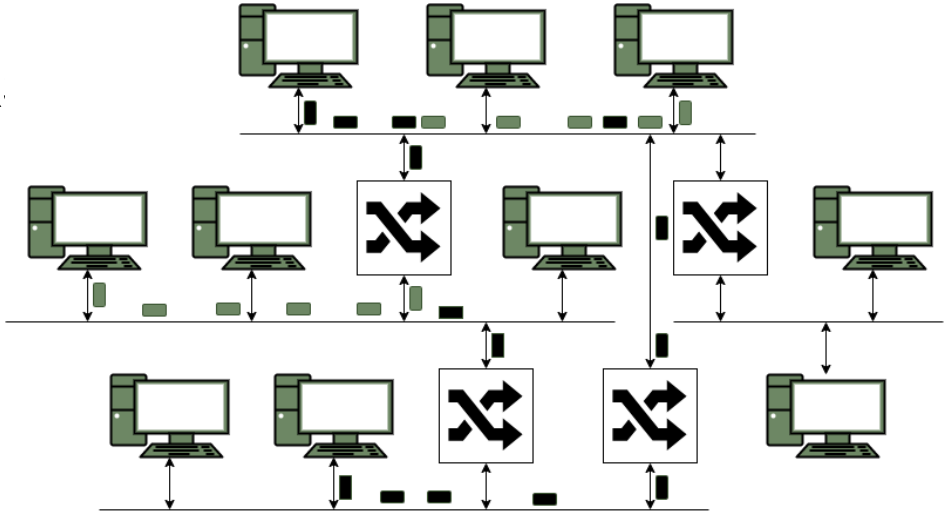
Packet Switched Network



- As the packet is being sent over the link, all nodes on the link (including switches) listen to see if the packet is for them.
- A switch may see that the packet should be sent to another link, so the switch will load the packet, storing it in a buffer. It will then watch the 'output' link, and when this link is free, it will pass the packet to that link.
- In this way, the packet will migrate through the network.

Packet Switched Network

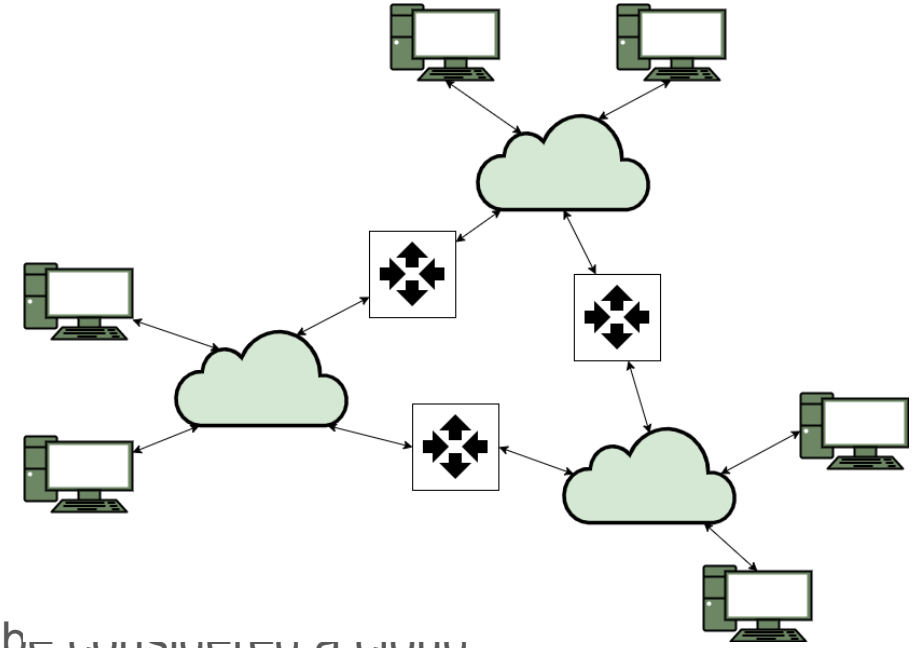
- Note that the various packets of a message may take different paths through the network.
- When the packets arrive at the destination, the packets are sorted and reassembled into the complete message.



Clouds

- A network like we saw above is an example of a *cloud*.
- The entire Internet could be formed as a single cloud.
- However, additional efficiencies can be achieved by limiting the size of clouds, but providing interconnections between the clouds.
- The interconnections between clouds are called *routers*.

Clouds



- In this diagram we see computers connected to clouds, and clouds interconnected by routers.
- A router is very similar to a switch. For our present discussion we will consider them identical.
- This resulting network is itself could be considered a cloud.
- Consequently, this cloud could itself be one cloud in a larger network.
- This recursive structure is what allows the Internet to span the whole globe.

Reliability

- The network as we have described it seems powerful and capable.
- What could possibly go wrong?

Reliability

- Electrical signals can be corrupted by noise, so some of the bits might be incorrect.
- As packets are waiting in buffers of the routers, the net congestion may be so great that a router runs out of buffers, so some packets are dropped.
- A router may die, or a link may break, so a connection is lost. The packets in transit are lost, and new packets can't go that way.
- For mobile devices, the person may move from one cell tower to another, so that packets enroute may not reach their destination.
- Other types of errors may also occur.
- *How do we handle these?*

Best Effort

- The goal of the Internet is to provide *best effort* service. It does its best to correctly transfer information, but does not guarantee the results.
- But things are not quite so grim:
- The Internet provides *error detection and correction*, that checks for corruption of data in the packets. If a received packet has errors, a request is sent to resend the packet.
- Software layers at the terminals (client and server computer) verify that all the packets got there. If some packets were lost, the packets will be resent.

Internet Application Details

- There are a lot of technical details required to implement a network like the Internet:
  - Dealing with the electronics, optics, and the raw bits of data
  - Collecting the bits into frames, or extracting bits from frames, buffering the data
  - Routing data from one node to another
  - Breaking large messages into packets, reassembling the message from packets. Dealing with broken packets, missing packets, duplicate packets, and out-of-order packets.
  - Dealing with how data is represented across the network: image formats, byte order, sizes of integers.
  - Encryption and security
- The above list represents a *lot*^{2} of code

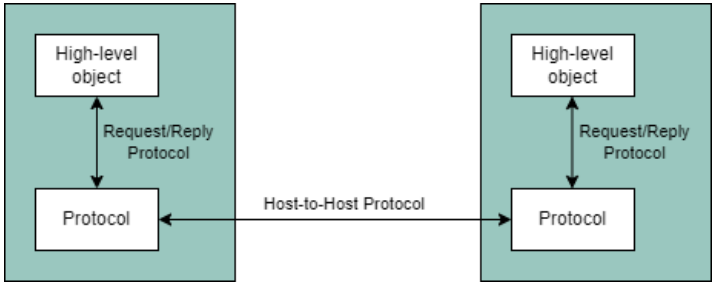
Internet Application Details

- Putting all of that network code into every Internet Application is problematic:
  - Did I mention this is a *lot* of code.
  - Error prone.
  - Difficult to fix bugs (if a bug is found in the network software, then every program that uses the network has to be fixed).
  - As the network and technology evolves, all of the programs have to be updated.
- Does this sound like Object Oriented Design? (hint: no)

Object Oriented Design Key Principles

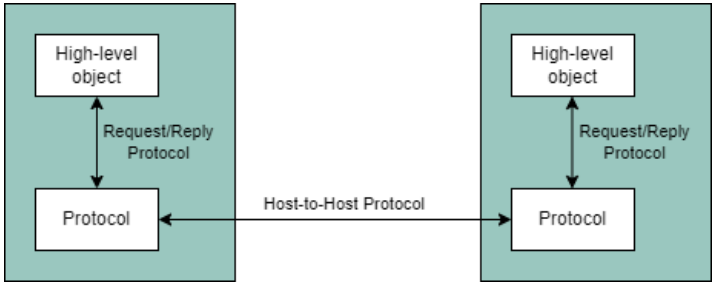
- There are a dozen key principles of Object Oriented Design, but two of these stand above the rest:
- Abstraction: Provide a simple, clean interface between the external code and the internal code.
- Encapsulation: Hide all of the internal details.
- With these, the system can be used simply by working with the interface.
- The external code cannot interfere with or introduce bugs to the internal code.
- The internal code (and internal algorithms, technology, etc) can be changed without any changes to the external code.
- This also promotes code reuse, reducing the amount of code to be written and supported.

Encryption and Security



- Not every application will need encryption and security, so we can separate out this part.
- In addition, there might be several different kinds of encryption and security, so we might have multiple channels that provide these features, with different characteristics for each.
- We call this 'layer' a *protocol*.

Encryption and Security



- Actually, our encryption and security 'protocol' has two sub-protocols. (I know that is confusing…not my fault!)
  - One protocol deals with how the application program and server program connects to and uses this protocol. This is the RRP.
  - The second protocol deals with how the 'protocol' in the client and the 'protocol' in the server communicate. What does the data look like, what are the controls. This is the HHP.

Interfaces

- The RRP (Request/Reply Protocol) defines the *service interface*, which is how the application on that computer uses this service. What types of data are expected, what values are needed, what options are available. How to set up and use the service.
- The HHP (Host-to-Host Protocol) defines the *peer interface*, which is how the code on this computer interacts with its counterpart on the other computer. What data is sent over the link, in what format, and so on.
- The RRP is the external interface, the HHP is the internal interface.

A Little Better

- Applications that don't need encryption and security are a little better, they have fewer things to implement:
  - Dealing with the electronics, optics, and the raw bits of data
  - Collecting the bits into frames, or extracting bits from frames, buffering the data
  - Routing data from one node to another
  - Breaking large messages into packets, reassembling the message from packets. Dealing with broken packets, missing packets, duplicate packets, and out-of-order packets.
  - Dealing with how data is represented across the network: image formats, byte order, sizes of integers.
- Unfortunately, the encryption and securing package is still pretty complex, because it has to deal with *all* of the details.
- So *some* applications are simpler, but not all.

Doing This Again

- Feeling a little bit better after splitting off encryption and security, we now do the same thing at the next level, peeling off data representation and formats.

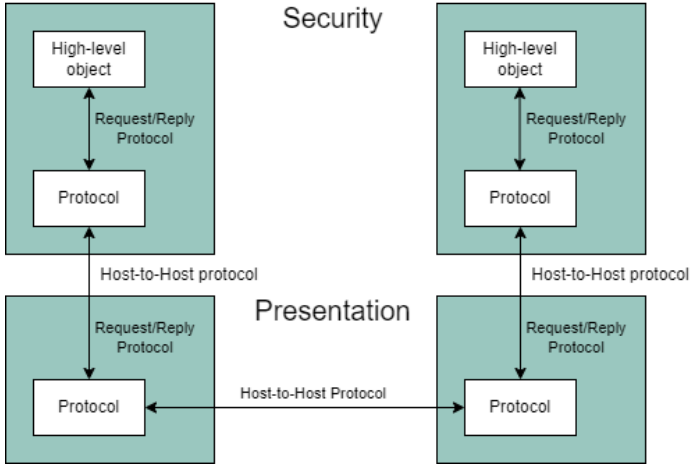
- We define interfaces for another protocol, what we might call *Presentation*.

Revisiting Encryption and Security

- Now that we have a nice protocol for presentation, the people writing and supporting the protocol for encryption and security are quite put out!
- They had to do a lot of work to handle all of the details of using the internet (electronics, frames, routing, …, presentation). All of the work that the presentation protocol provided for the applications.
- Then the insight: why couldn't the security protocol use the presentation protocol, be built on top of the presentation protocol?

Revisiting Encryption and Security

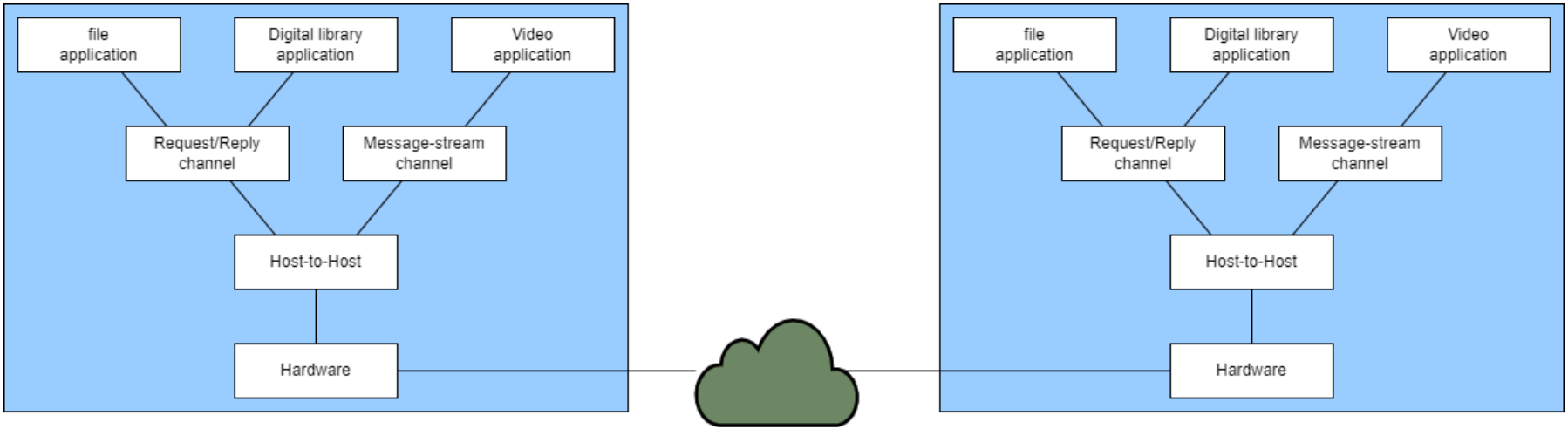
- The applications that use the Security protocol saw no change, but now the Security protocol was a lot simpler.
- The presentation protocol gave a nice simple interface for Security.
- The host-to-host connection of Security now uses the Presentation protocol, and through that all of the code for electrical, frame, routing, and so on.



Layering

- This same technique is used to divide the tasks into manageable, useful *layers*.
- The bottom layer is the only layer that communicates through the links to other computers.
- All of the other layers talk to layers beneath it in the *protocol graph*.
- Any particular layer does not necessarily connect to one previous layer, it may connect to two or three other layers, at various points in the graph.
- Each layer abstracts all of the layers below it, providing a clean, simple interface to applications and layers above it.

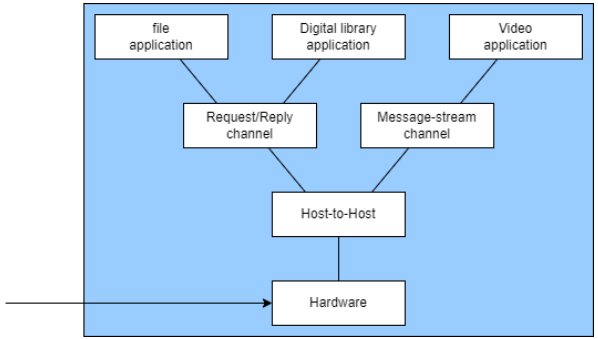
Protocol Graph



- Here is an example of a protocol graph. Not every endpoint computer necessarily has all of these protocols. But for any application that uses this computer, it has all of the protocols needed for those applications.

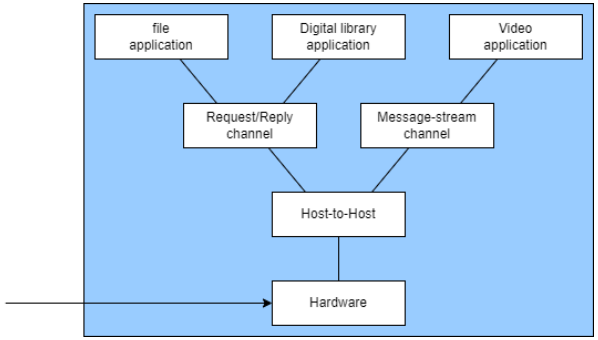
Demultiplexing

- Consider what happens when a server receives a message.
- The message goes to the lowest protocol in the stack.
- After the lowest level handles its part of the processing, it passes the result to the next level in the stack (graph).
- If there are multiple protocols above this level, which path does it use? Does *Host-to-host* pass this to *Request/Reply channel* or *Message-Stream channel*?



Demultiplexing

- When the *Host-to-host* layer of the source machine received its data from either the *Request/Reply channel* or *Message-Stream channel*, it added a little information to the head of the message, indicating where it came from.
- When the *Host-to-host* layer of the destination machine receives its data from the hardware layer, it pulls this header off the message, examines that information, and so can then see where to send the message.
- This process is known as *multiplexing/demultiplexing.*



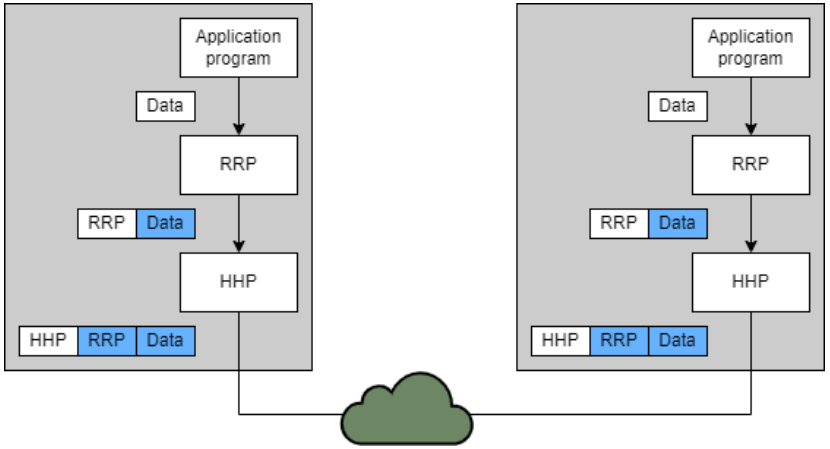
Encapsulation

- Consider a File Application that uses the RRP and HHP in its network stack.
- The File Application sends a message through its RRP.
- The RRP does not understand the contents of the message, it just passes that message along. However, it attaches a *header* to the start of the message containing instructions for the receiving RRP, so that *it* knows what to do with the message.
- *Note: The RRP may modify the message contents, such as encrypting or compressing the data, but the RRP on the other end reverses this operation, so the final data received by the destination application exactly matches the message that was first sent.*

Encapsulation

- The message that RRP sends is therefore the concatenation of the RRP header and the original message.
- The format and contents of the RRP are part of the peer interface specification of the RRP. No other software need know or care about this header.
- The HHP does the same thing, attaching its own header to the start of the message, to be removed at the other end of the pipe.
- As the packets are being passed through the Internet, the switches and routers may peek inside the HHP header, since this contains the routing information. But the rest of the message is opaque.

Encapsulation



- Here we see each protocol treats the data it receives as opaque, but adds a header, to be used by its counterpart in the destination machine.
- The network itself is part of the lowest level, so the network can see that header. That header includes the destination address, so the network knows where to send the packet.

Layer Standards

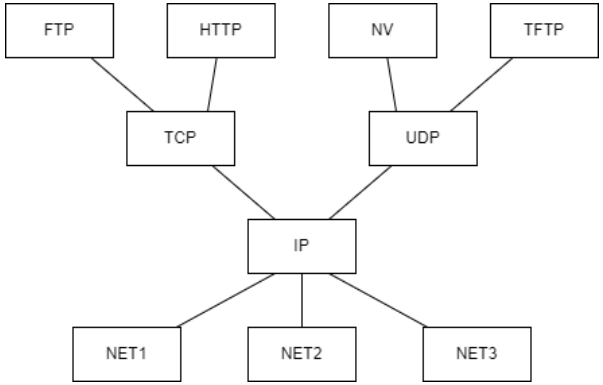
- There are two primary layer 'stacks' for networks:
  - The 7-Layer Model: The ISO architecture.
  - The Internet Architecture (a 4-layer model).
- These are similar, but we will be using the Internet Architecture.

The 7-Layer Model

From the top (application) to the bottom (hardware), here are the layers in the 7-Layer Model:

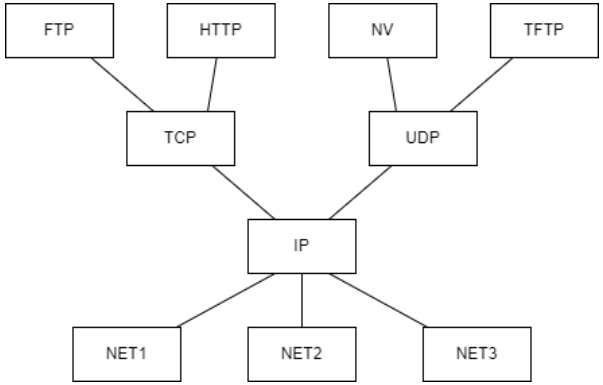
- Application (HTTP, etc)
- Presentation (size of ints, byte order, etc)
- Session (used to tie multiple channels to one program)
- Transport (process-to-process channel)
- Network (routing of packets among nodes)
- Data Link (collecting bits into frames)
- Physical (transmission of raw bits)

Internet Architecture



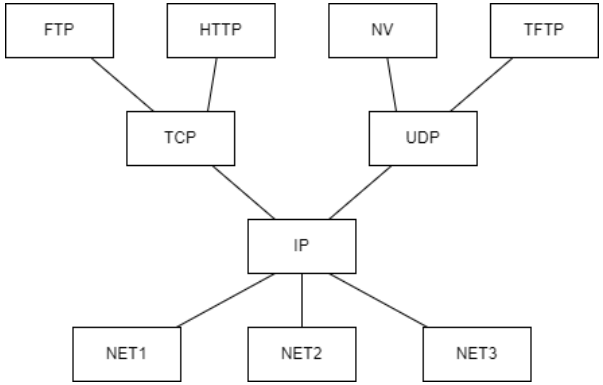
- The lowest layer in the Internet Architecture is the network interface layer.
- Examples would be ethernet or wireless protocols.
- Each of these might be fairly complex, and can have layers of their own.
- The Internet layer is not concerned with those details, they are abstracted and encapsulated.

Internet Architecture



- The second layer is the IP layer, the Internet Protocol.
- One of the key principles of the Internet Architecture is that there is exactly one protocol at this layer. Thus everything that is built at higher levels goes through this one interface, and every type of network below here is fed from this one interface.
- This isolates all of the application code and higher network code from the lower-level details.

Internet Architecture



- The third layer has two primary protocols, TCP and UDP.
- TCP (Transmission Control Protocol) deals with reliable delivery of streams of data, dividing the streams into packets, and assuring that the packets are correctly received and reassembled.
- UDP (User Datagram Protocol) provides unreliable delivery of small datagrams.

Reliable Transmission

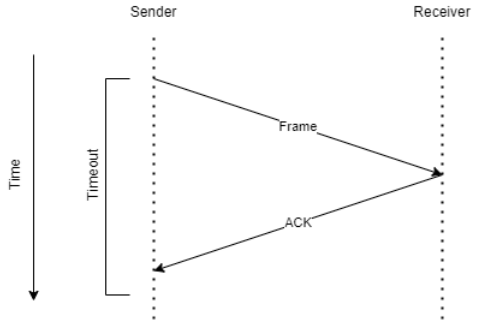
- As we have seen, some packets may be 'broken' as they are transmitted across the medium.
- Corrupt frames must be discarded.
- To provide reliable transmission, the link-level protocol must be able to recover from these discarded frames.
- There are a few layers in the stack where reliability is provided.
    - Once a layer provides reliability, any layers higher on the stack do not also have to include the logic to make the link reliable.
    - Sometimes the lower-level layers omit the logic for reliability, relying on the higher-level protocols to provide this feature.

Reliable Transmission

- Reliable transmission usually uses a combination of two mechanisms: *acknowledgement* and *timeout*.
- Acknowledgement (ACK) is a small reply message that the receiver sends upon successfully receiving a packet. When the sender sees this message, it knows that the receiver successfully received the original packet.
- The sender expects to see the ACK message within a given amount of time. If it does not see this reply, a timeout has occurred, at which point the sender retransmits the packet.
- This strategy is called *automatic repeat request* (ARQ).
- We will explore three ARQ algorithms.

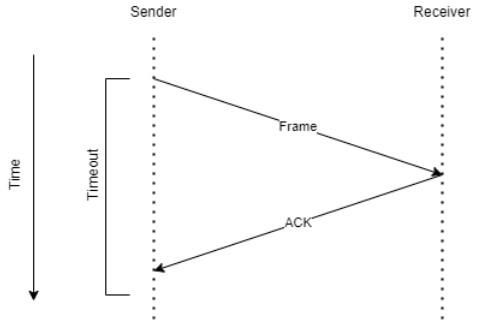
Stop-And-Wait

- The first ARQ algorithm is *stop-and-wait*:
  - The sender sends one packet, then waits for the ACK.
  - If the sender receives the ACK before the timeout, it then sends the next packet.
  - If the sender has a timeout before receiving the ACK, it retransmits the packet.
  - The packet may be transmitted any number of times, until the ACK is finally received.
    - The protocol may limit the number of times a packet may be sent -- perhaps the link or destination is simply down.
  - The packets are always sent, and received, in the correct order.



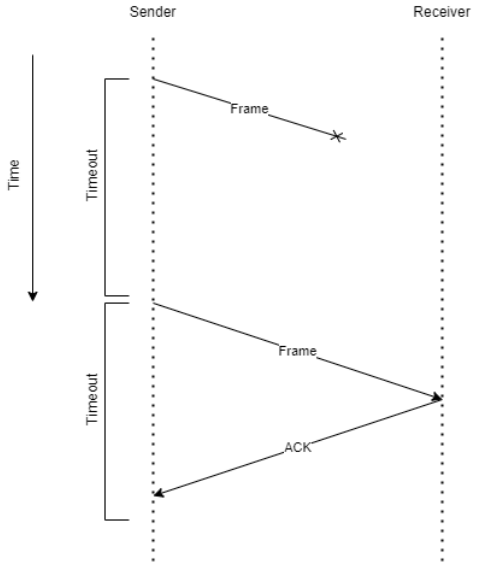
Stop-And-Wait

- This diagram shows the successful transmission of one packet.
- The sender sends the packet, and starts a timer.
- Some while later the receiver receives the packet and checks to see if there was an error in the transmission.
- If all is well, the receiver sends the ACK.
- The sender receives the ACK before the timeout, so it can then move on to the next packet.
- This repeats until the whole message has been delivered.



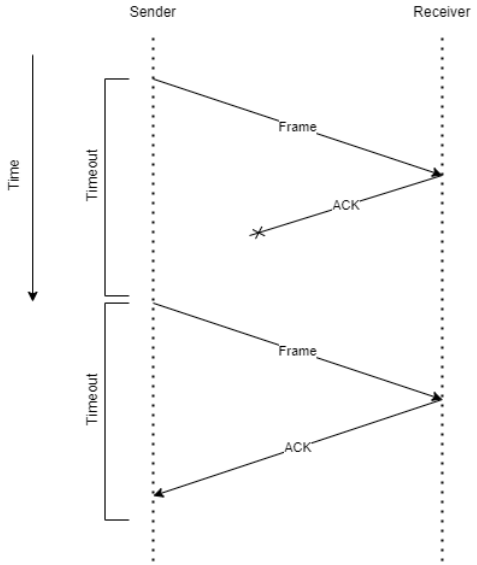
Stop-And-Wait

- Suppose the packet is lost in transit, so the receiver never sees the packet.
- In this case, the receiver never sends the ACK.
- The sender will timeout, so then the packet will be resent.
- Perhaps this time the receiver will see the packet, and so acknowledge it.



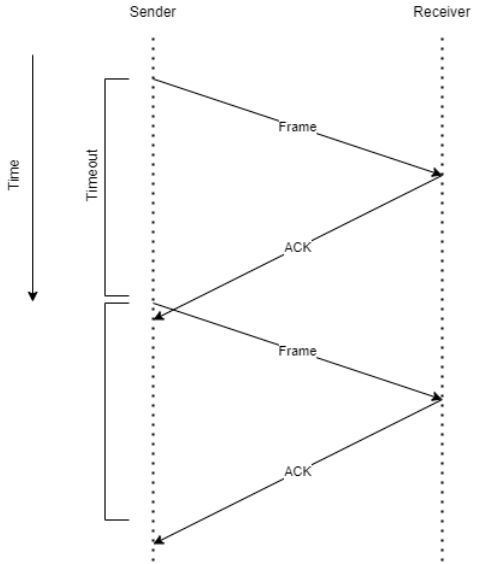
Stop-And-Wait

- Perhaps the receiver does receive the message, and sends the ACK.
- What if the ACK gets lost?
- At this point, the receiver is happy, but the sender is still curious, "did it get there?"
- Eventually the sender times out. The sender doesn't know if the receiver never got the message, or if the ACK was lost. Doesn't matter, the sender resends the packet.
- The receiver receives a copy of the packet, which is discarded, but acknowledged.



Stop-And-Wait

- In this last scenario, the sender sent and the receiver received and acknowledged.
- However, the process was too slow, so the sender timed-out before receiving the ACK.
- Consequently, the sender resent the packet, and the receiver ignored the new copy, but sent an ACK.
- The sender sees the first ACK, so it knows the packet got there. It doesn't know if this ACK is for the first or second copy. But it stops sending this packet.
- Eventually, the sender sees (and ignores) the second ACK.

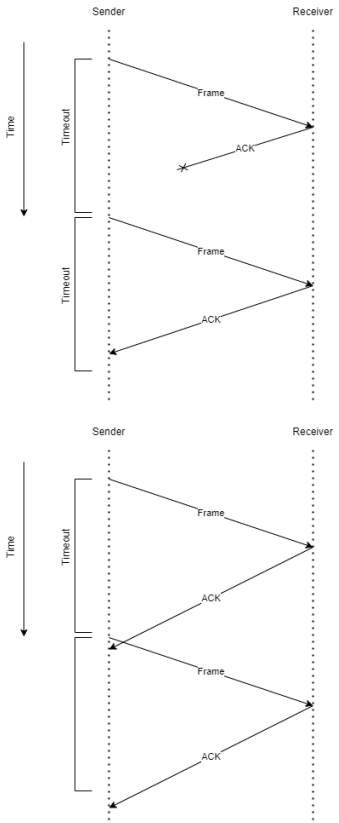


Stop-And-Wait

- When we say that a packet (or ACK) was *lost*, we mean that either the packet never got to the desired recipient, or that the packet was damaged in transit, so that the recipient saw the error detection flag.
- In either case, the packet is deemed as lost, the sender will time-out, and the packet will be retransmitted.

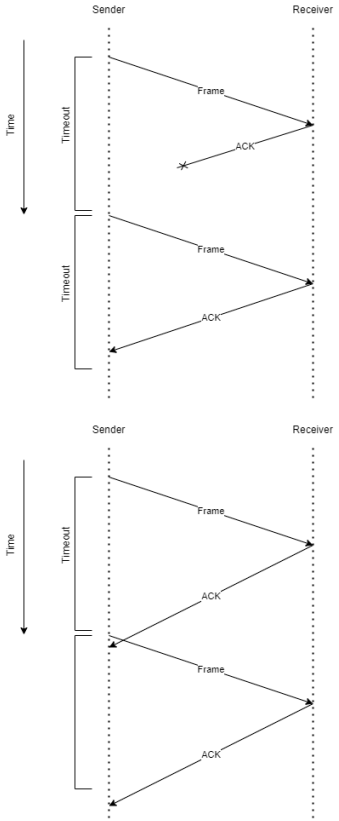
Stop-And-Wait

- There is one subtle problem that can occur with stop-and-wait, in the cases where the receiver got the packet and responded with the ACK, but the ACK was either lost or delayed.
- The sender will have resent the packet.
- But the receiver will expect that the previous packet was acknowledged, so it will assume that this new packet is in fact the *next packet in the message*:
- The received message will contain a duplicate of this packet.



Stop-And-Wait

- By the same token, when the sender receives the ACK messages, it might misinterpret which packet was being acknowledged. Consider the second of these diagrams:
  - The sender sends the first packet, but the response is slow.
  - The sender times out, so resends the first packet.
  - The sender then receives the ACK. Was this the ACK for the first packet, or was the first packet lost and this is the ACK for the resend?
  - So the sender goes ahead and sends the next packet.
  - The sender then receives the second ACK for the first packet, but thinks this is the ACK for the second, so it goes ahead and sends the third.



Stop-And-Wait

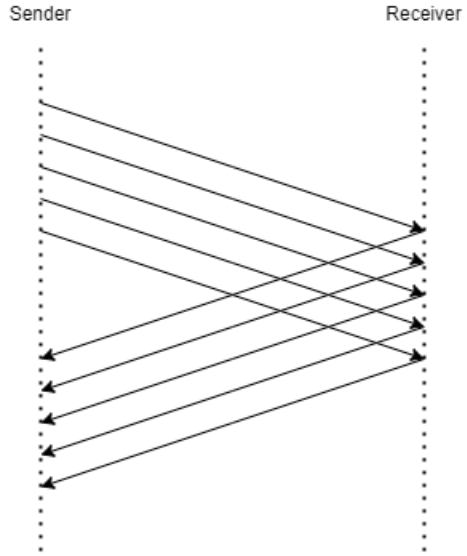
- To fix this problem, the stop-and-wait protocol includes a 1-bit sequence number. For all the even-numbered packets, this bit is set to 0, and for all odd-numbered packets, this bit is set to 1.
- This bit is present for both the original packets and for the ACK responses.
- Only one bit is required, because the protocol can only be "off by 1":
  - For the receiver getting the packets, it only has to know is this packet a resend of the previous packet or is it the next packet.
  - For the sender getting the acknowledgements, it only has to know if this ACK is a resend for the previous packet or it is the ACK for the current packet.

Stop-And-Wait

- The main shortcoming of stop-and-wait is that there is only one packet being sent at a time.
- For each packet, the sender has to wait at least the RTT to get the acknowledgement before moving to the next packet.
- Consider a 1.5-Mbps link with a 45-ms RTT. The delay x bandwidth product is 67.5 Kb, or about 8 KB. If the frame size is 1 KB. So if the sender has to wait for the ACK, then only about ⅛ of the link's capacity is being used.
- To use the link fully, 8 packets could be sent before the sender would receive any of the ACK messages.

Sliding Window

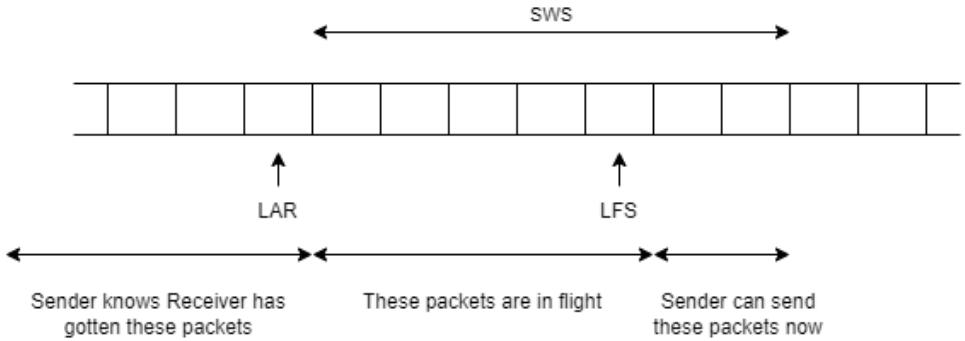
- The second ARQ algorithm we will discuss is *Sliding Window*.
- Consider again the case where the link has delay x bandwidth product of 8 KB but the frames are 1 KB. We would really like to keep the pipe full by sending 8 packets before expecting to see the ACK for the first packet.
- At about the same time as we receive ACK 1, we are sending packet 9, and so on.



Sliding Window

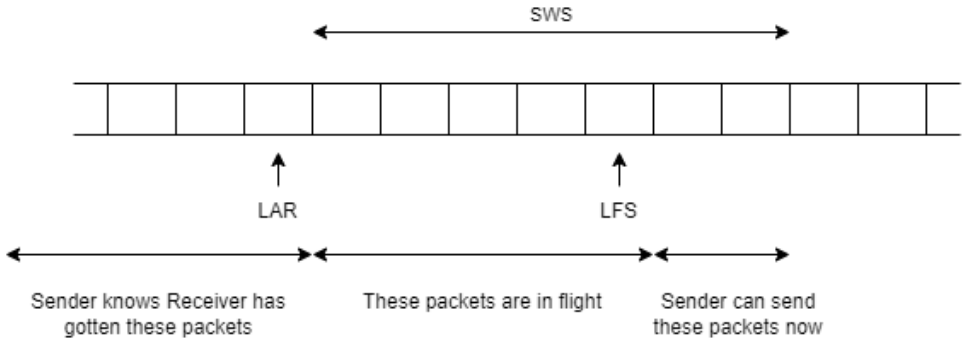
- For the sliding window algorithm, we will assign a sequence number to the packets.
- Recall that for the stop-and-wait algorithm, we needed a 1-bit sequence number to disambiguate the packet numbers and acknowledgement numbers.
- For sliding window, if we can have *n* packets in flight, then we need sequence numbers up to *2n*. We would normally round up to the next power of 2.

Sliding Window: Sender



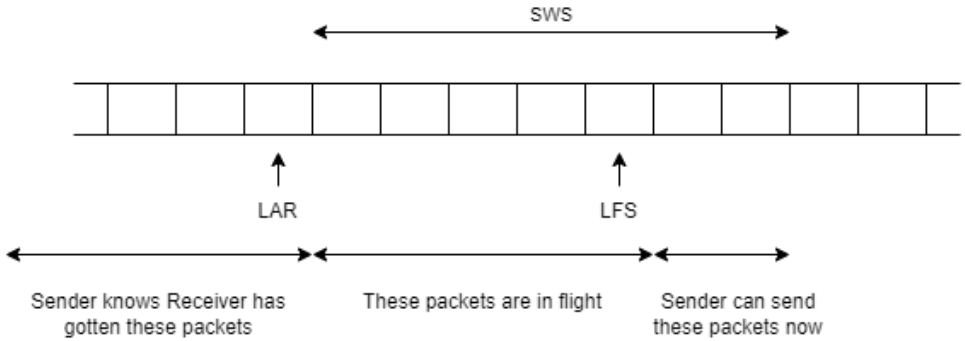
- The sender has three numbers, and a series of timers:
  - The first number, the *send window size*, or SWS, is the maximum number of packets we want in flight at a time. For the example presented above, SWS would be 8.
  - The *last acknowledgement received*, or LAR, is the largest sequence number we have received from any ACK. The sender knows that all packets numbered from 1 to LAR have been successfully received.
  - The *last frame sent*, or LFS, is the sequence number of the last packet that was sent.

Sliding Window: Sender



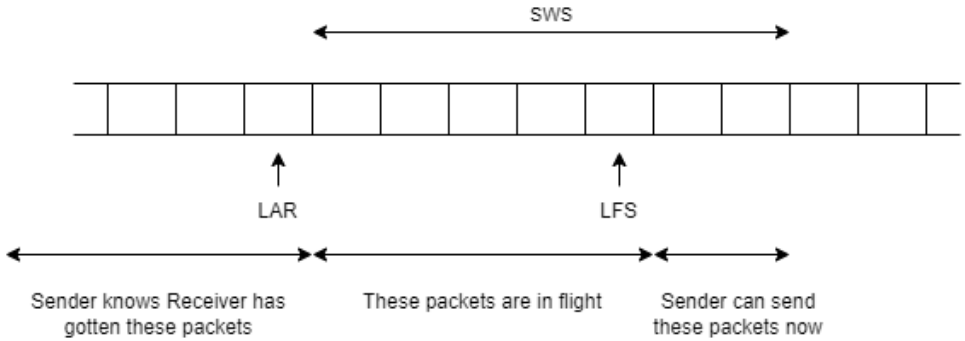
- The sender has three numbers, and a series of timers:
  - The sender also maintains a timer for all packets that have been sent, that haven't been acknowledged yet. If any of these timers times out, then the sender resends that packet.
  - The number of timers needed will be SWS. The sender must also have this many buffers to hold the packets that are in flight, in case any of these need to be resent.

Sliding Window: Sender



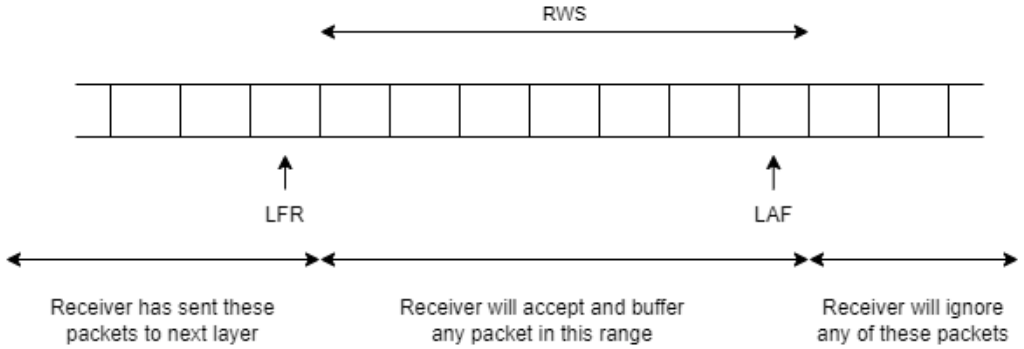
- Initially LAR is set to 0, since we haven't received any ACKs yet.
- The sender sends packets from 1 up to SWS (if there are that many packets).

Sliding Window: Sender



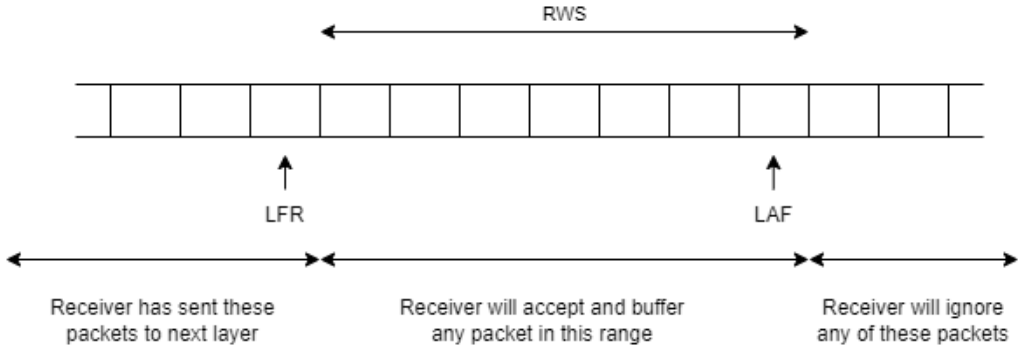
- Whenever the sender receives an ACK, it sets the LAR to that number.
  - Actually, in a large network, the ACKs might be received in a different order than they are sent, so the sender would only ever *increase* the LAR. Suppose LAR was 15, then an ACK for 13 was received. In this case, LAR can remain at 15.
- Whenever the LAR increases, new packets can be sent, up to packet number LAR + SWS.

Sliding Window: Receiver



- The receiver for the sliding window algorithm is just a little more complex.
- The receiver also has three numbers (but no timers):
  - The *receiver window size*, RWS, gives the upper bound on the number of out-of-order packets that the receiver is willing to accept. For example, the receiver might say that it can take up to 4 out-of-order packets. It has buffer space to store these packets.

Sliding Window: Receiver



- The receiver also has three numbers (but no timers):
  - The *last frame received*, LFR, gives the sequence number of the last packet *that the receiver received and passed on to the higher level protocol*. For example, the receiver may have received and passed on packets 1 - 5, but also received packet 7, which it stored in a buffer. The receiver has to get packet 6 before it can pass both 6 and then 7 to the higher protocol. The LFR would be 5.
  - The *largest acceptable frame*, LAF, which is <= RWS + LFR. Since the receiver can only buffer RWS packets, then it is looking for packets numbered LFR + 1 through LFR + RWS.

Sliding Window: Receiver

- When the receiver receives a packet, it looks at the sequence number, SeqNum.
- If SeqNum <= LAF, the packet is ignored. The receiver has already processed this packet (the packet was resent, either the ACK was corrupted or came too late).
- If SeqNum > LAF, the packet is ignored. The receiver only has SWS number of buffers, and this would be beyond that limit.
- If the SeqNum is between those limits, the receiver can process this packet. (see next slide)

Sliding Window: Receiver

- When the receiver accepts a packet, it stores the data into one of the buffers.
- Recall that the receiver has already received, and passed on, all of the packets from 1 to LFR.
- It now looks to see if LFR + 1 is in the buffer.
- If so, it passes this buffer to the next higher protocol, and increments LFR.
- It repeats these steps until it finds an empty slot, a packet that it has not yet received.
- At this point, the receiver sends an ACK containing the LFR number.
- The receiver also updates LAF to be LFR + RWS.

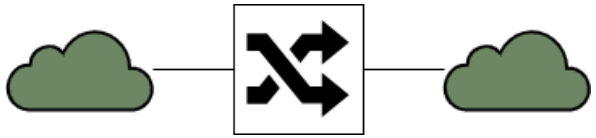
Sliding Window: Receiver

- For example, suppose the RWS is 4, and the receiver has already received and processed packets 1 - 5, so LFR is 5. Consequently, LAF is 9, and the receiver is listening for packets 6 through 9.
- Suppose packets 7 and 8 come in. These are both in the active range, so they are accepted. They are stored in two of the buffers of the receiver.
- Because packet 6 is still missing, nothing can be sent to the higher-level protocols, and the receiver does not send an ACK (or it sends ACK 5).
- When packet 6 arrives, it is placed in the buffer.
- The receiver then sends packets 6, 7, and 8 to the higher-level protocol.
- It updates LFR to 8, and sends an ACK 8. It updates LAF to 12.

Internetworking

- Interconnecting nodes on one specific network is not that complex:
  - Each node has a unique network address.
  - Each message has the address of its source and its destination.
  - All nodes listen to all messages on the network, to see if any are addressed to it.
- Those techniques cannot be scaled up to build a global network.
- To interconnect these networks, we need to solve:
  - Switches or bridges to interconnect networks, forwarding messages *when appropriate* from one network to the other.
  - Routers to connect between different *types* of networks. These require significantly more logic and computing power than simple repeaters.
  - Finding a suitable path for a message through the network, as there are many possible paths.
  - The design of devices capable of performing these tasks.

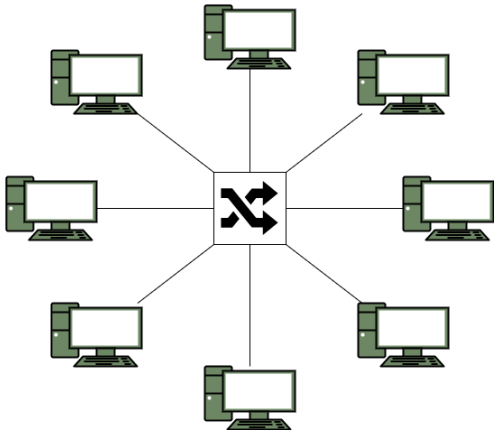
Switches



- A *switch* is a device that connects two (or more) links from different sub-networks, to form a larger network.
- Like a hub or repeater, a switch can take a message from one link and copy it to the other link.
- As the other devices, the switch runs in both directions.
- The difference is that switches are *selective*: Not all messages are repeated, only ones that 'belong' (or 'pass through') the other network.
- If a switch has more that two ports, the message is not sent to *all* of the other ports, just the one leading to the destination.

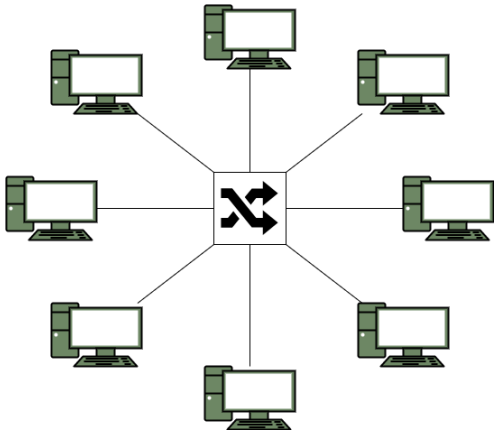
Switch Ports

- Each connection between a switch and a link is called a *port*.
- A switch has at least two ports, but may have more. In our examples we routinely use switches with four or eight ports.
- The ports of a switch are typically identified by number, so the switch here would have ports numbered from 0 to 7.



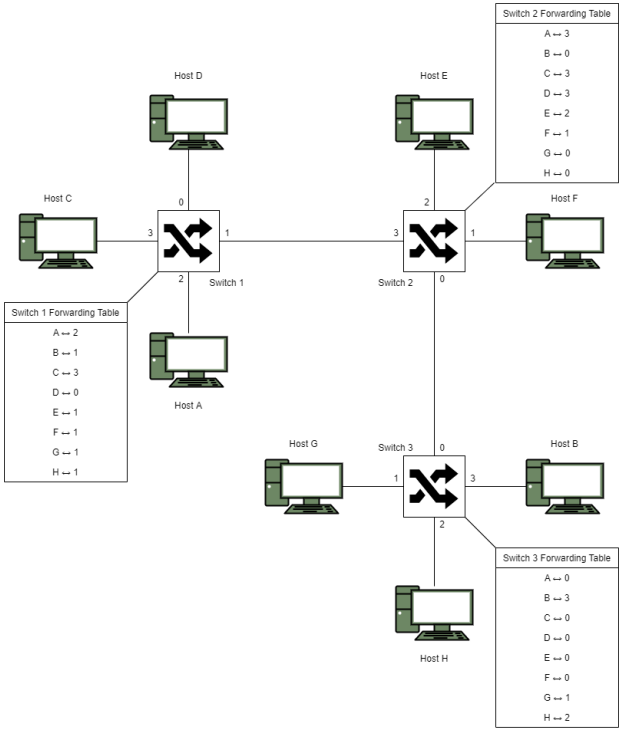
Switch Ports

- When a switch receives a packet on one port, how does it know to which port the packet should be sent?
- An identifier in the header of the packet is used to determine the port to use.
- There are three general approaches:
  a. Datagram or connectionless
  b. Virtual circuit or connection-oriented
  c. Less commonly, source routing.
- We will briefly visit the Datagram technique.



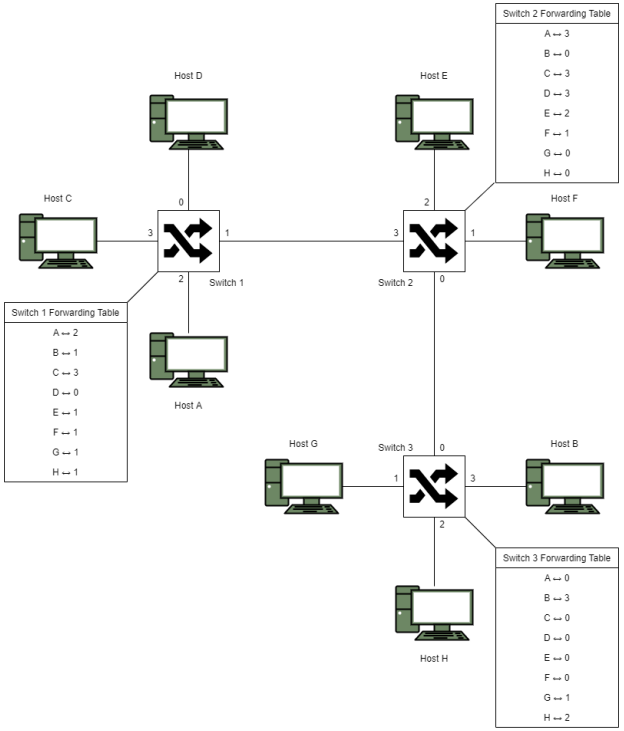
Datagrams

- The idea behind datagrams is incredibly simple:
  - Each packet has the destination address.
  - Each switch in the network has a *forwarding table* that indicates, for this switch and for every host (possible destination in the network), which port points to that host.
- For example, host A sends a packet to host B.
- Switch 1 gets the packet, and sees that it should send the packet out port 1.
- Switch 2 sends the packet out port 0.
- Switch 3 then sends it out port 3.



Datagrams

- For a small, static network, building the forwarding tables is pretty straightforward.
- It is a lot harder to build these tables with a large, complex, dynamically changing network with multiple paths.
- This harder problem is known as the *routing* problem.
- In many cases, the forwarding tables are populated in the background so that when packets arrive, the tables are already built.



Datagrams

- Datagram networks have the following characteristics:
  - A host can send a packet anywhere at any time, since all switches know where to send any possible packet.
  - When a host sends a packet, it has no way of knowing if the network is capable of delivering it or if the destination is up and running.
  - Each packet is sent independent of previous packets to the same destination. They might go on different routes if the forwarding tables change.
  - A switch or link failure can block transmissions until an alternate route can be encoded in the forwarding tables.

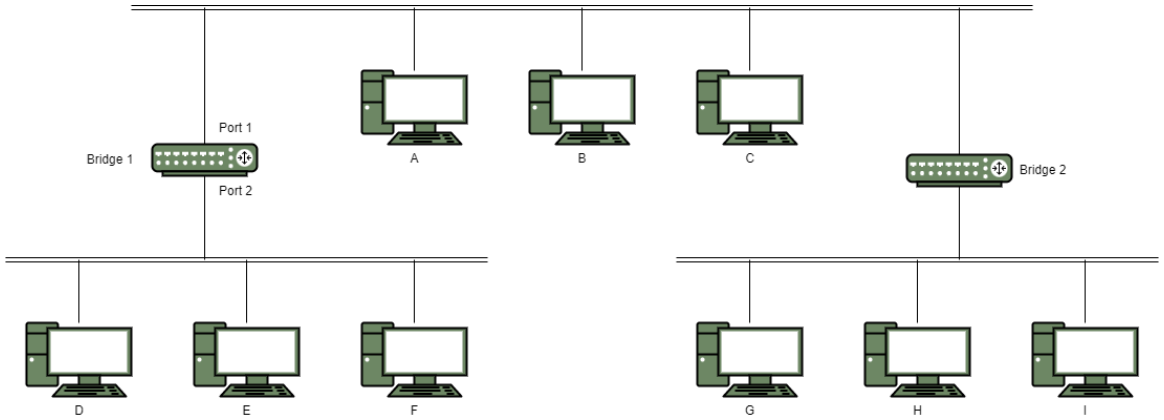
Bridges and LAN Switches

- Consider the specific application of forwarding packets between LAN networks, such as ethernets.
- Historically, these switches have been called *bridges*.

Bridges and LAN Switches

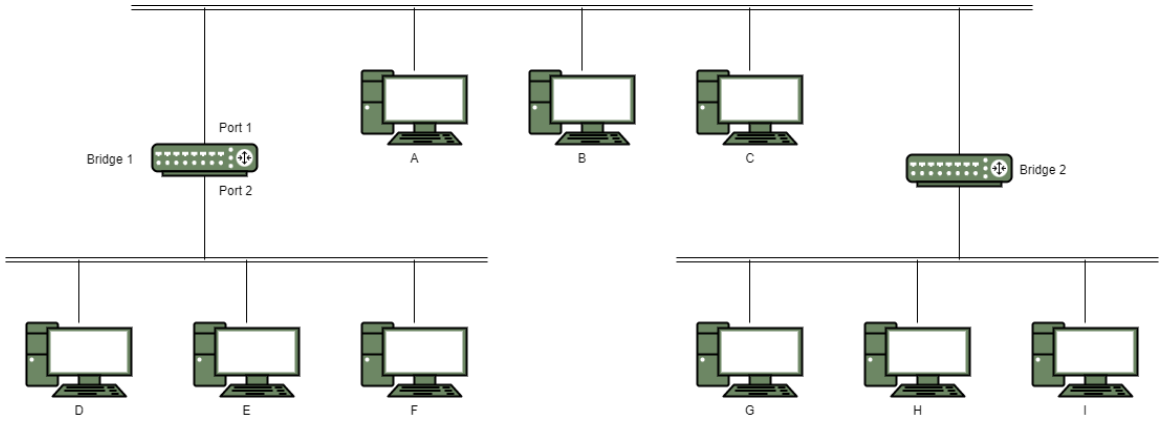
- The bridge is basically a computer with two ethernet adapters.
- The bridge works on packets, not on bits: The adapter on the bridge loads the entire packet from one link, then *may* pass this to the other link.
- When forwarding the packet, the bridge waits until the second link is idle.
- If there is a collision on the second link, the bridge retries again.
- This solves the congestion problem: The congestion on one network is limited to that network, and each of the two networks can be simultaneously sending packets.
- Also, the bridge only forwards the packet if the destination is 'on the other side'.

Bridges



- Consider this network. There are three ethernet spans connected by two bridges.
- If Host A sends a packet to Host B, neither bridge need forward the packets.
- If Host F sends a packet to C, then Bridge 1 forwards the message.
- If Host H sends a packet to E, then both Bridges forward the message.

Bridges

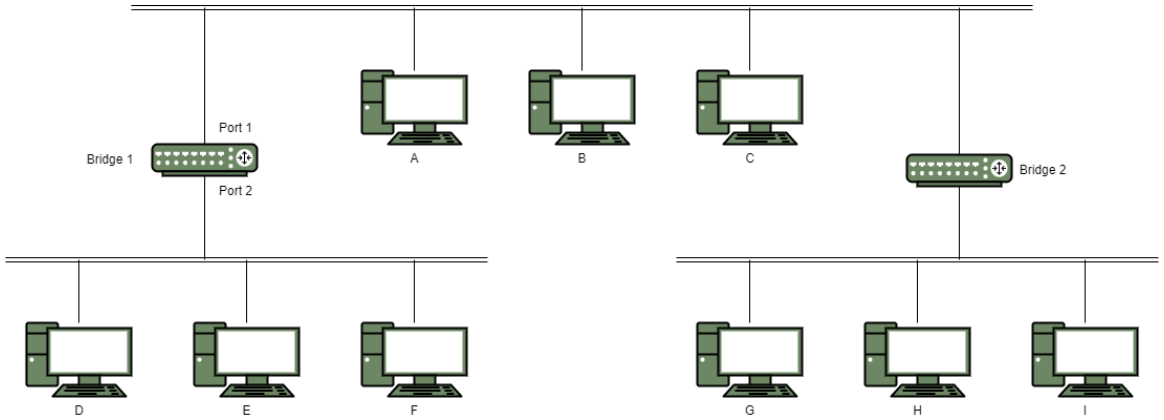


- How does a bridge know when to forward a packet? If it sees a packet with a destination 'on the other side'.
- When F sends a packet to C, the packet enters Port 2 of Bridge 1.
- Bridge 1 looks in its directory and finds that Host C is on the Port 1 side, so it forwards the packet.
- Note that Host H would also be considered on the Port 1 side of Bridge 1.

Bridges

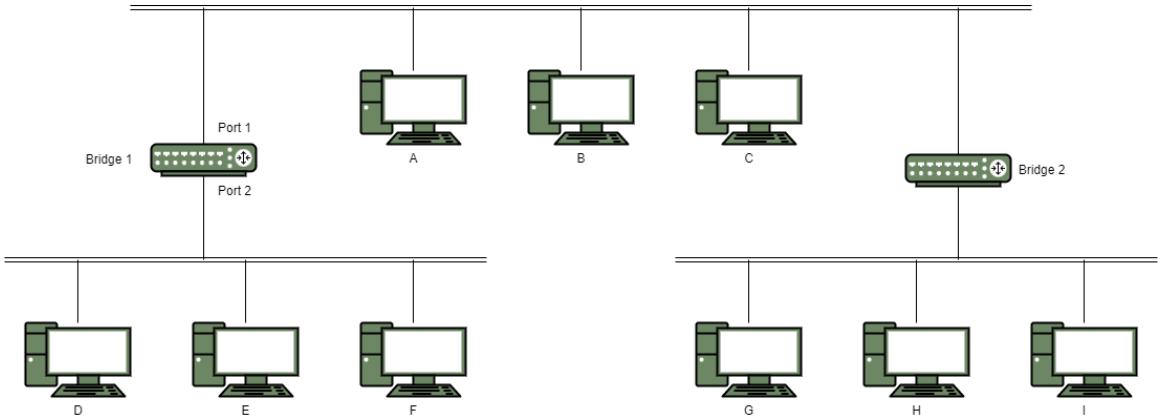
- So how does a bridge get this directory, so that it knows which host is on which port? It *learns* the contents of the directory!
- Initially, the directory is empty.
- When the bridge sees a packet appear on one of its ports, it examines the *source* address.
  - If the bridge does not have a directory entry for this Host, it records that that host is on the receiving port side.
- The bridge then examines the *destination* address:
  - If the bridge has an entry for this Host, and if host is on the other port, the packet is forwarded.
  - If the bridge does *not* have an entry for the Host, it automatically forwards the packet: this is called *broadcasting*.

Bridges



- The packet from F to C appears on the lower left ethernet.
- Bridge 1 records that F is on Port 2 side.
- Bridge 1 doesn't know where C is, so it forwards the message.
- Bridge 2 will see this forwarded message, so it will record that F is on its Port 1 side.
- Bridge 2 doesn't know where C is, so it also forwards the message.
- So in this case, all of the links will see the message, but all bridges now know where F is located.

Bridges



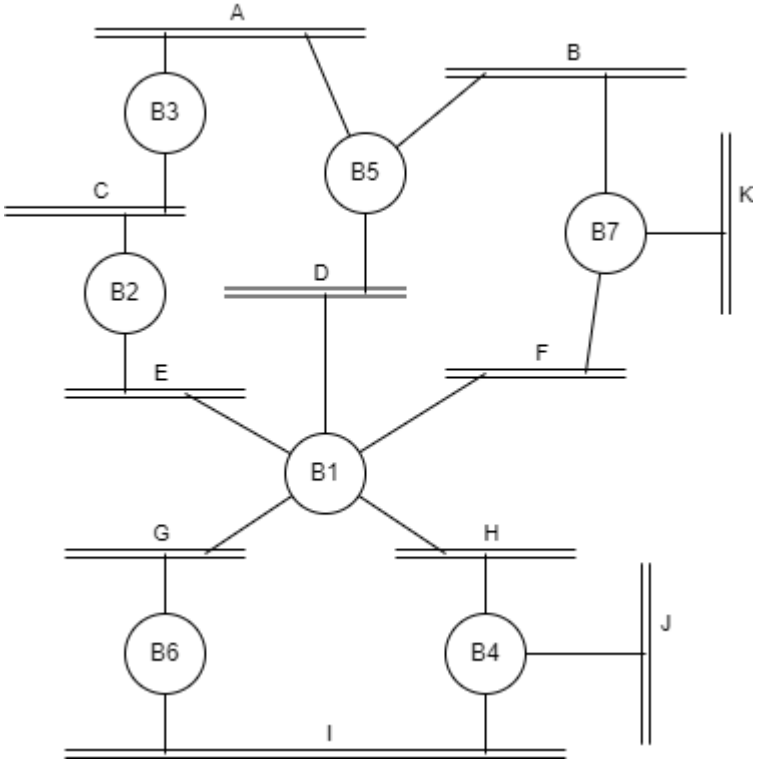
- As long as C never *sends* a packet, the bridges will not know where C is located, so all messages to C will be propagated through the whole network.
- However, when C does send a packet, maybe a response to F, then the network learns where C is located, and from that point one, the packets only go where needed.

Bridges

- From time to time, the network topology changes:
  - A new host may be added
  - A host may move
  - A link may fail
  - A new link may be added
  - Bridges can be added or removed
- When this happens, the directories must be updated.
- The typical way to do this is to simply have directory entries *time out*: they are only kept in the directory for a period of time. The next time this host sends a message, the network re-learns where it is located.

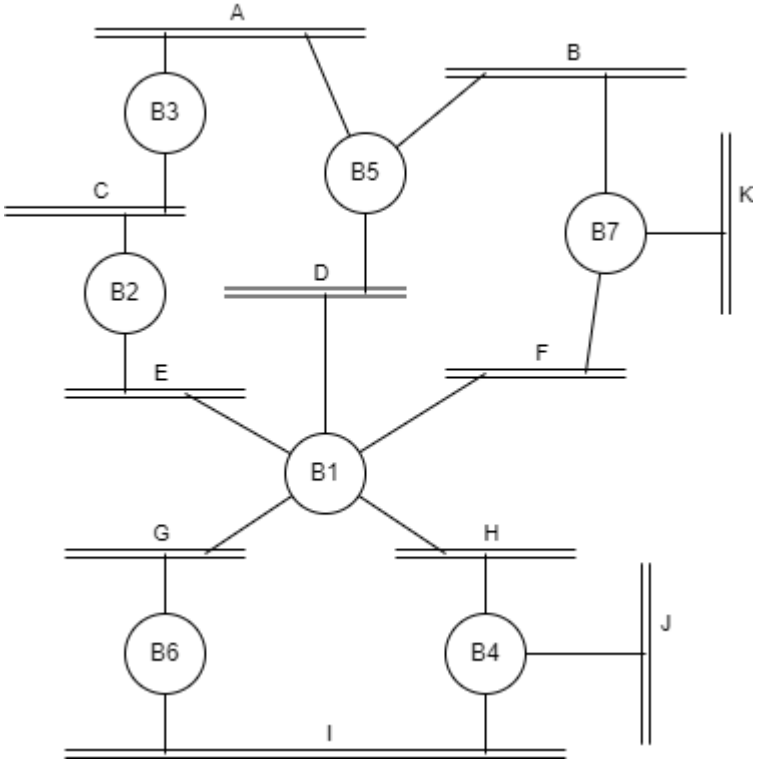
Spanning Trees

- The learning bridge algorithm works well, until the network has a loop. When loops exist, the algorithm fails horribly!
- Consider Host J sends a packet.
- B4 learns where J is, and not knowing the destination's location, sends the packet to H and I.
- B1 learns where J is, but sends the packet to D, E, F, and G.
- Meanwhile, B6 learns where J is, but sends the packet to G.

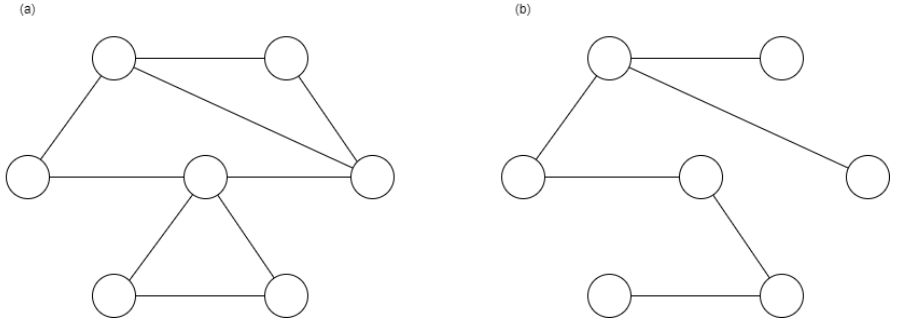


Spanning Trees

- There are now two packets on G.
- B6 will see the one that came from B1.
- It will get confused, now thinking that J is 'on the other side'.
- More than that, it still doesn't know the location of the destination, so it will forward this message back to J.
- B1 does the same with its second message, sending new copies to all of the other ports.
- This process repeats indefinitely!



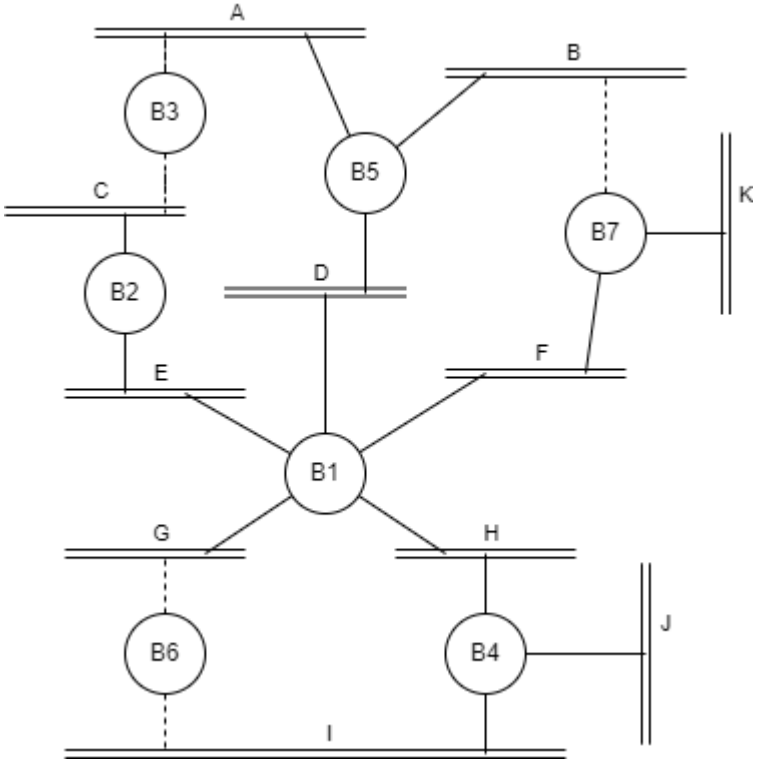
Spanning Trees



- Loops can be removed from a general graph by disabling some of the links (breaking some of the connections.
- The graph on the left has a few loops.
- By breaking three of the links, these loops are removed.
- The network is still fully connected.
- The graph on the right is a *spanning tree*: it covers the whole graph, but it is a tree structure.
- There may be several possible spanning trees for a particular graph.

Spanning Trees

- In this diagram, we see the network, with some of the links being disabled (shown as dotted lines). The remaining active links form the spanning tree.



Basic Internetworking

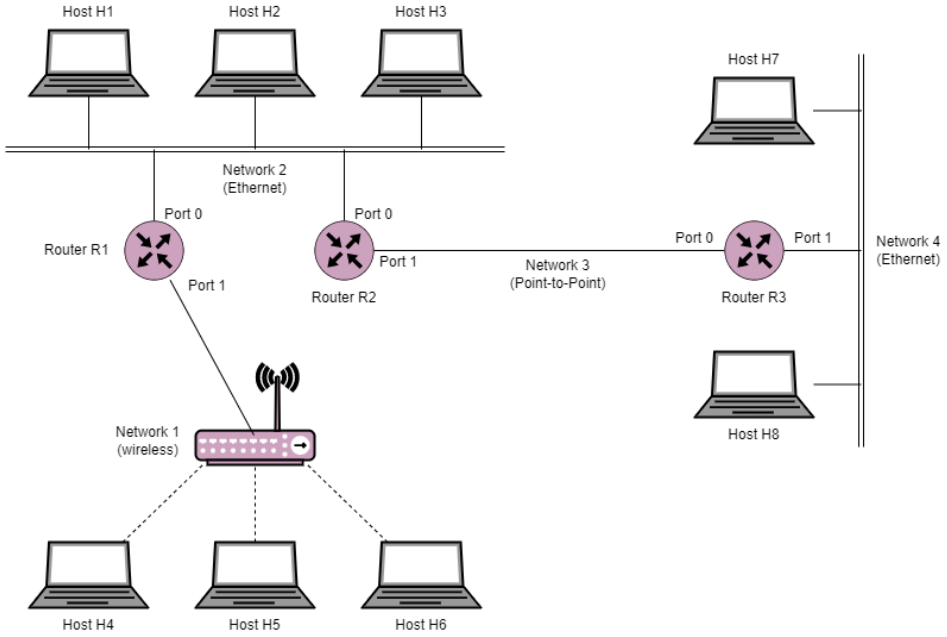
- The networks we've discussed so far use a single technology:
  - Ethernet
  - Wireless, such as 802.11
  - Point-to-Point
- Using routers and bridges, these networks can be extended, to some extent.
- Scaling is limited due to:
  - Signal transmission properties
  - Network congestion
  - Flat addressing
  - Single technology due to specific protocols per technology
- But we would like to scale more than this!

Basic Internetworking

- Let's define *network* to be a direct or switched connection using one technology. We may also call this a *physical network*.
- Let's define *internetwork* to be a *logical network* built by interconnecting diverse physical networks.
- An internetwork is often referred to as a "network of networks".
- The Internet is an example of an internetwork.

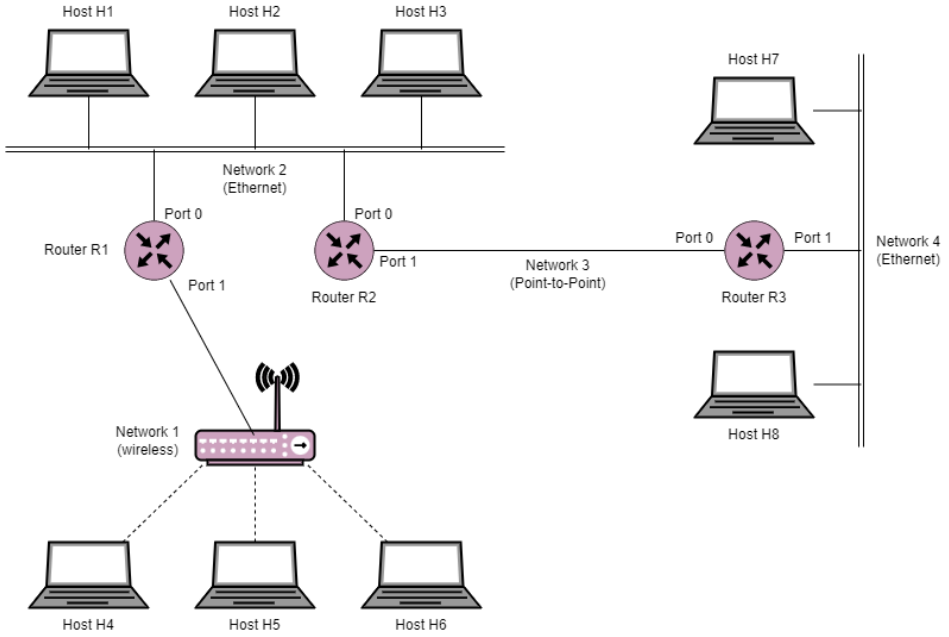
Basic Internetworking

- In this diagram, we have four physical networks linked to form one logical network.
- Two of the physical networks are Ethernets, one is a Point-to-point link, and the final is a wireless network.
- Three routers are used to connect these physical networks.



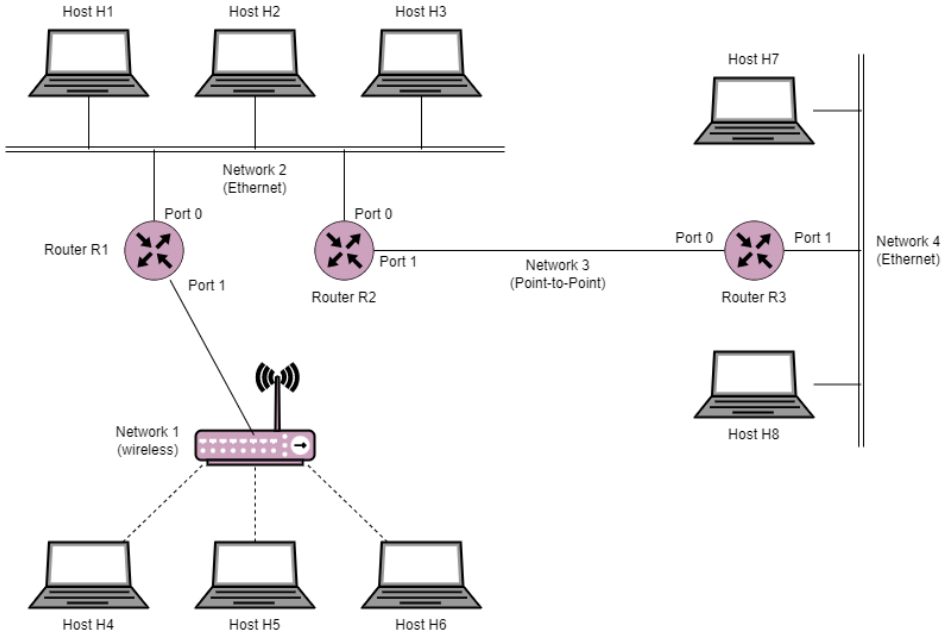
Basic Internetworking

- In the past, routers, switches, and bridges connected networks of the same type.
- Consequently, these devices were customized to match the protocols of that network type.
- Packets could be forwarded with little or no logic, the network protocol extended across the whole network.

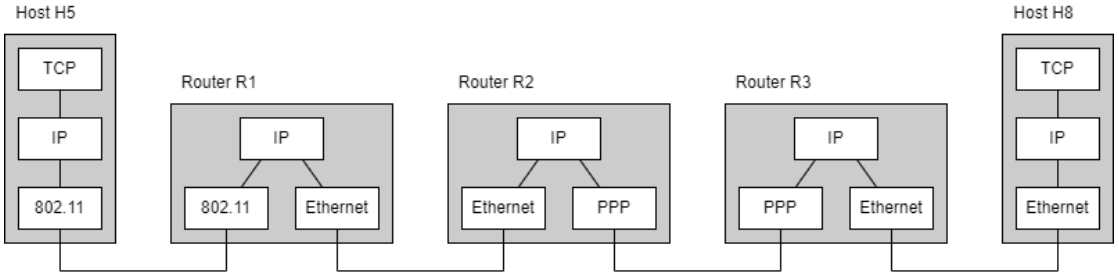


Basic Internetworking

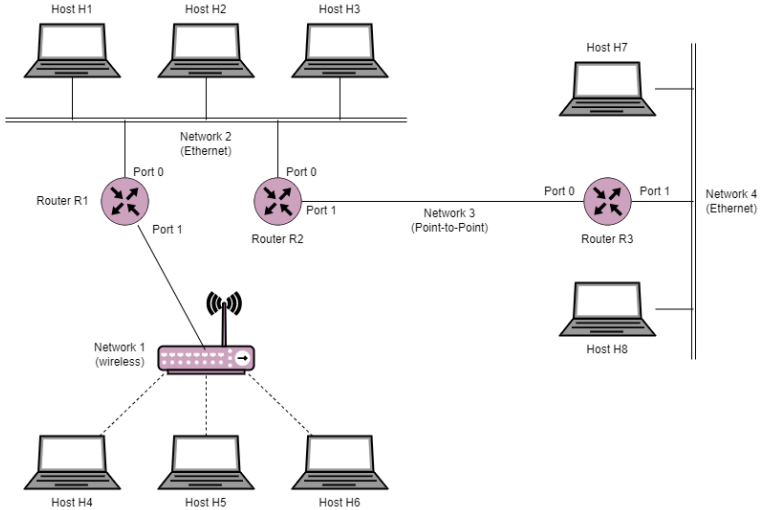
- In an internetwork, a router connects what might be two different types of physical network.
- Consequently, the router has to operate at a higher protocol layer.



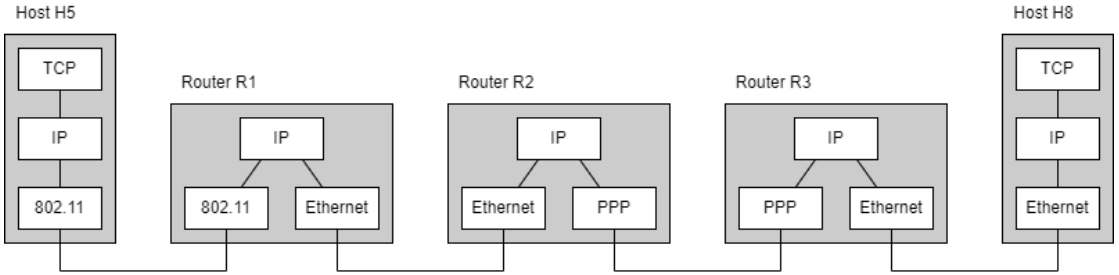
Protocol Layers



- Consider sending a packet from Host H5 to Host H8.
- Router R1 interfaces between a wireless network and an Ethernet network.
- Routers R2 and R3 interfaces between an Ethernet and a Point-to-Point link.



Protocol Layers



- The routers in this configuration operate at the IP (Internet Protocol) layer.
- Each router has two (or more) adapters, or ports. These ports work at the data link and physical layer.
- The ports deal with transporting packets across the physical network.
- The Internet Protocol layer receives packets from the ports, then determines how to forward these packets to other routers or to the final host.

IP Service Model

- To make the Internet as general and extensible as possible, minimal requirements were established.
  - The internet provides minimal constraints upon what a network looks like.
  - The internet provides minimal constraints on how the network is used.
  - The internet provides minimal constraints on the hardware.
- The internet protocol uses datagrams as the underlying packet methodology.
- The internet protocol uses a hierarchical addressing scheme.
- The internet provides a 'best effort' delivery 'promise'.

Service Model: Best Effort

- By 'best effort', we mean:
  - The IP protocol does not guarantee that every packet will get there.
  - The IP protocol does not attempt to recover from failure.
  - Large messages are divided into packets.
  - Packets may be lost, misdelivered, corrupted.
  - Packets may be delivered out-of-order.
  - Packets may be duplicated.
  - Delivery time is not guaranteed.
  - Packets may not take a specific route through the network, and different packets of a message may take different routes.
- The Internet Protocol is an *unreliable* service, but does a pretty good job!
- Higher-level protocols can provide for reliable service.

IP Datagrams

- IP packets are datagrams (connectionless).
  - The packets are pushed into the network.
  - The network determines how to forward the packets toward the destination.
  - Minimal routing and configuration need be done.
- The header that IP adds to the packets provides enough information to determine the destination and to guide the routing of the packets.
- The header has:
  - Source and destination IP addresses.
  - Length values and checksums.
- Consequently, the routers are fairly simple.

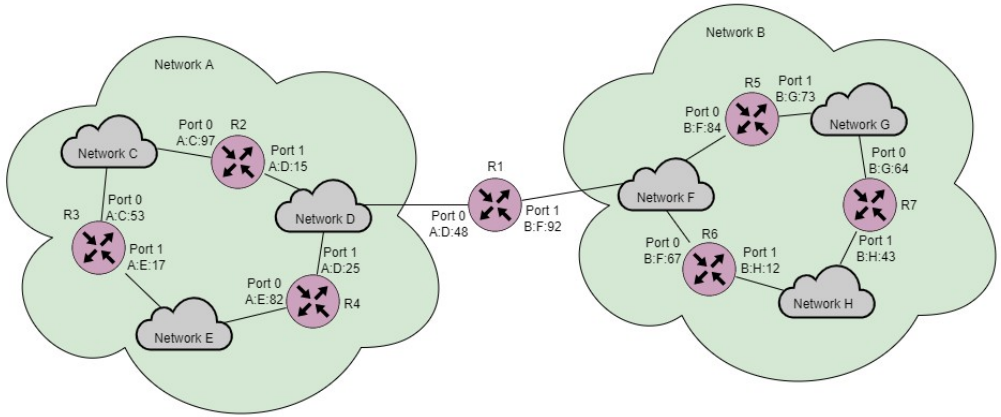
Global Addresses

- Ethernet requires that all adapters have a unique address, so that the network can be sure the packet is going to the correct address.
- Ethernet solved this problem by having the addresses burned into the ROM of the adapters, so these numbers are *globally unique*.
- Internetworking also requires unique addresses.
- Why not use the ethernet address?
- Because the ethernet address is *flat*: Any ethernet address can be anywhere in the world.
- To know where to route a packet, a router would need a directory containing *every ethernet address!*

Global Addresses

- A network of networks has an inherent *hierarchy*.
- Addresses can be split into two parts: The address of a physical network and the address of a host on that network.
- This helps quite a bit: The internetworking routers just need the addresses of each physical network, routing messages to the correct network. Once the correct network is reached, the routers there get the packet to the correct host.
- But we can do better: If we had a network of networks of networks (which we do)...
- *This portion of the lecture is simplified a bit -- IRL things are more complex!*

Global Addresses

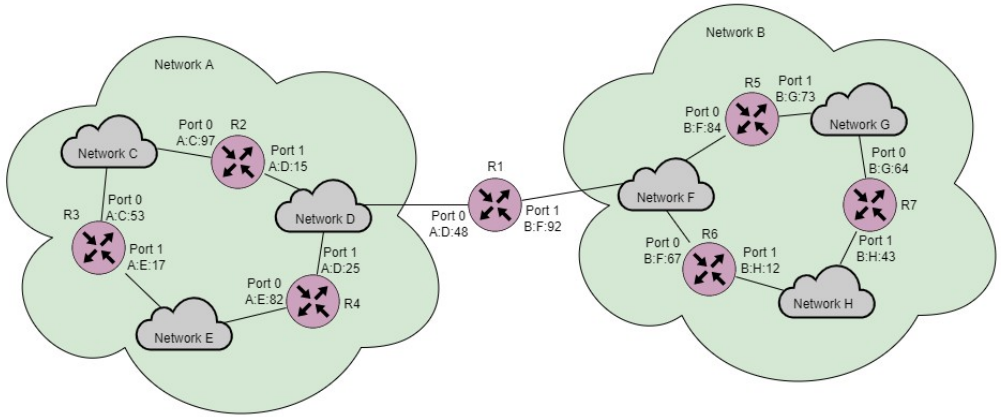


- Consider this small example.
- We have two top-level networks, Network A and Network B, linked by a router, R1.
- Network A is an internetwork of three networks, C, D, and E, with internal routers R2, R3, and R4. Network B is similarly subdivided.

Global Addresses

Host Address

A	C	…	68

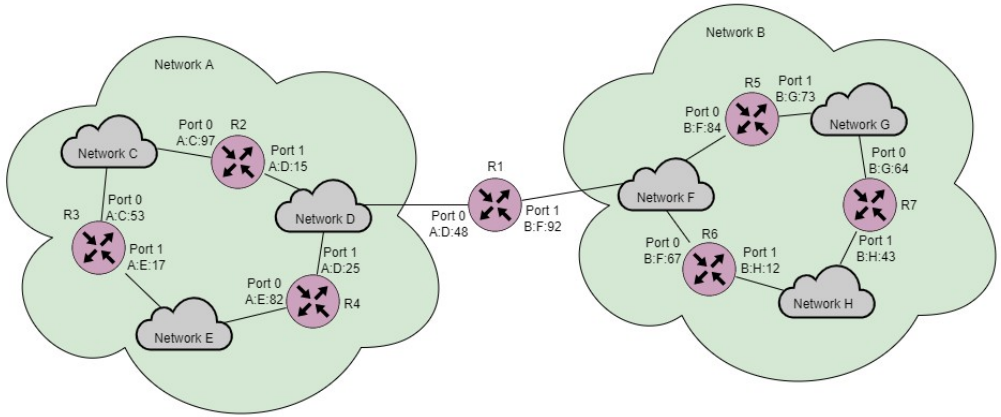


- One of the host computers on Network C might have the address A.C…53.
  - That's not exactly how the address would work, but for a 32 address, for example, the upper bits would indicate either Network A or Network B. The next set of bits would indicate the which network *within* A, and so on. The final bits indicate the host.
- In this way, the internetworking address roughly corresponds with the hierarchy of the global network.

Global Addresses

Host Address

A	C	…	68



- Why is this important?
- Because the overall internetwork has hierarchy: Continent, Country, State, City, and so on.
- Each City has a network, each State has a network of the City networks, and so on.
- The hierarchy is roughly partitioned to match the geography.

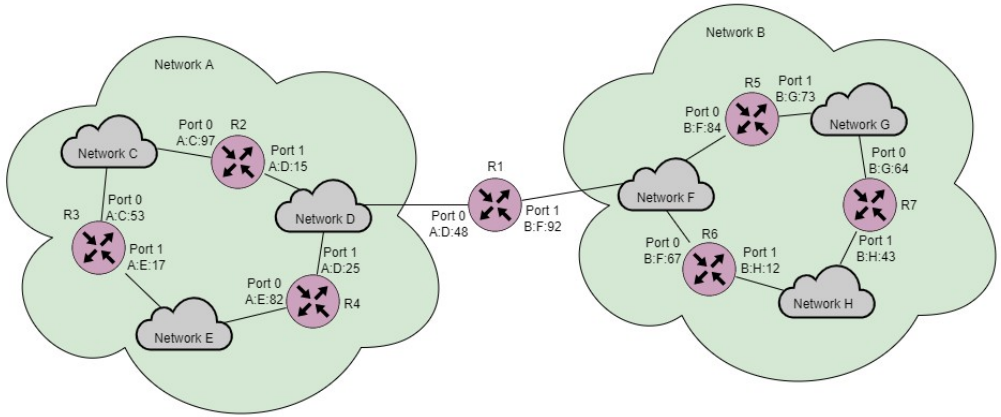
Global Addresses

Source Address

A	C	…	68

Destination Address

A	C	…	41



- Consider a host on Network C that wants to send a packet to another host on Network C.
- In this case, it just sends the message, and the network routes the packet there.

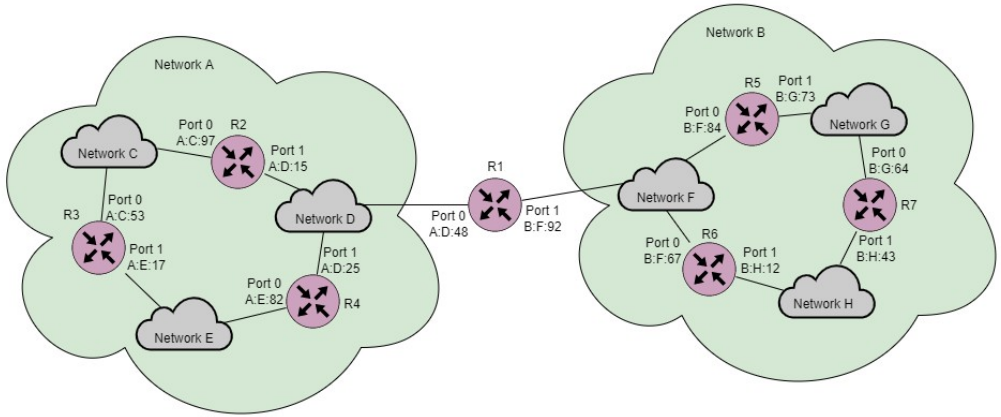
Global Addresses

Source Address

A	C	…	68

Destination Address

A	E	…	36



- What happens when the host on C wants to contact a host on E?
- The IP protocol on the host sees that the destination is on another network.
- It will put another wrapper around the packet, addressing it to A:C:53, the port on router R3.
- Network C then sends the packet to R3.

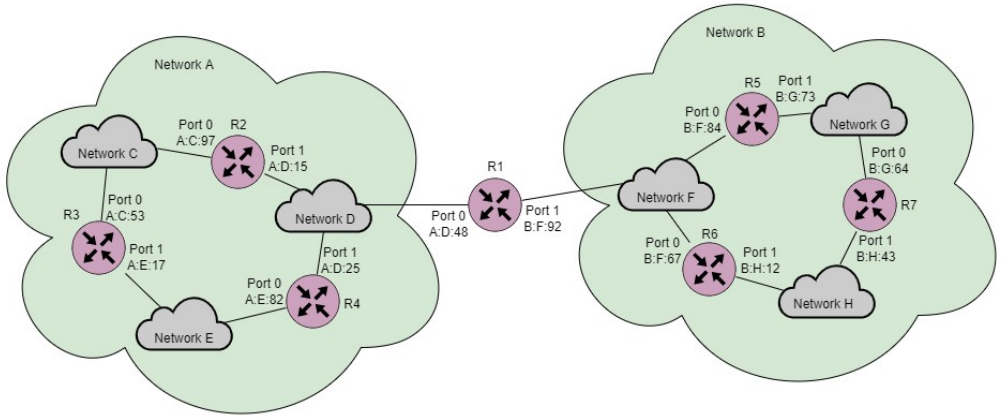
Global Addresses

Source Address

A	C	…	68

Destination Address

A	E	…	36



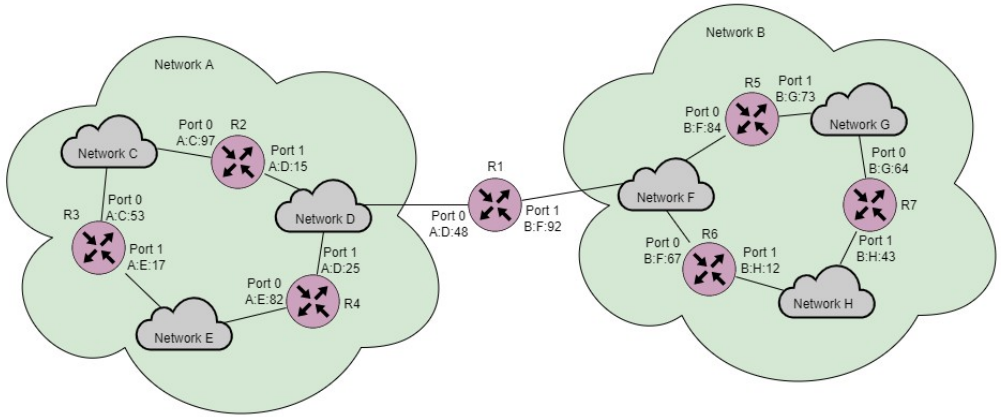
- When R3 sees the packet, in unwraps the packet and sees that the actual destination is in Network E.
- This is the network associated with port 1 of R3, so R3 sends the packet to Network E.
- Network E delivers the message to the proper destination.

Forwarding Table

- Router R3 builds a *forwarding table*, indicating the ports to which messages should be routed.
- The table has three columns:
  - Network -- The partial address
  - Next Hop -- Explanation coming soon!
  - Port, or Interface -- Which network leads to the partial address
- So far the table looks like this:

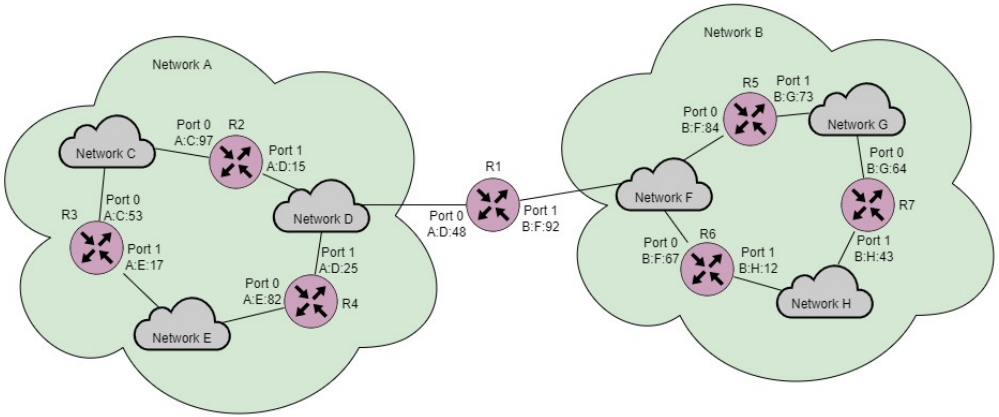
Network	*Next Hop*	*Port*
A.C	--	0
A.E	--	1

Global Addresses



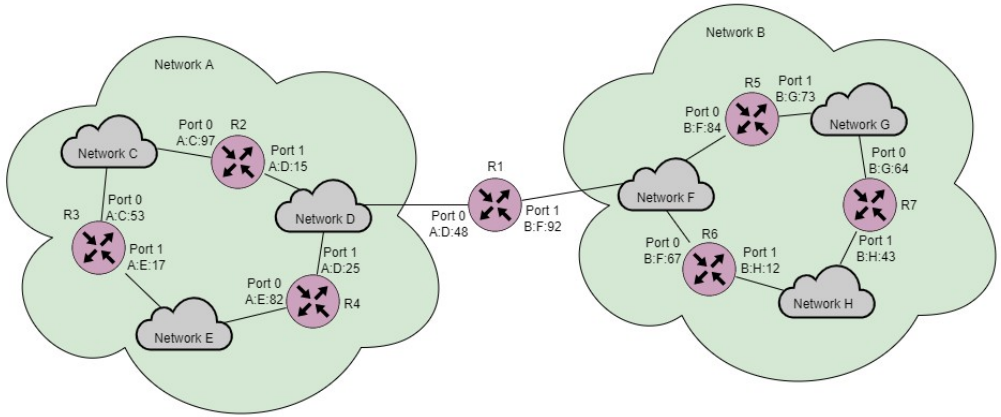
- What happens if R3 receives a packet whose destination is on Network D?
  - In this simple case, that probably wouldn't happen, but in a more complex network, this would be very likely.

Global Addresses



- What happens if R3 receives a packet whose destination is on Network D?
- **R3 would want to pass the message to either R2 or R4.**
- R3 would wrap the message in a *forwarding packet*, addressing it to R2, for example.
- What is the address of R2 (from R3's point of view)?

Global Addresses



- What happens if R3 receives a packet whose destination is on Network D?
- **R3 would want to pass the message to either R2 or R4.**
- R3 would wrap the message in a *forwarding packet*, addressing it to R2, for example.
- What is the address of R2 (from R3's point of view)? **A:C:97**

Forwarding Table

- The forwarding table for R3 now looks like this:

Network	*Next Hop*	*Port*
A.C	--	0
A.E	--	1
A.D	A.C.97	0

- The *next hop* value is the internetwork address for the router that will send the packet toward the target network.

Forwarding Table

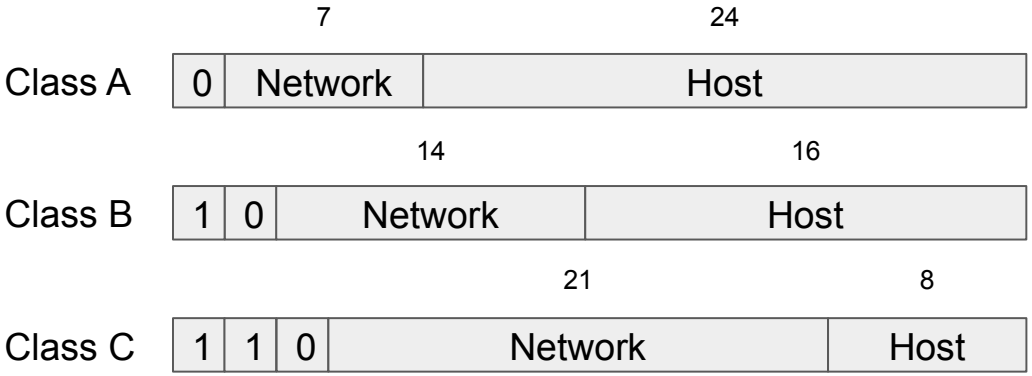
- The forwarding table does not have the complete internetwork address.
- It doesn't have any of the actual host addresses on the final network.
- It just has the address of the network itself.
- In fact, it doesn't have to have the address of the lowest-level network, it can have the address of any network in the hierarchy.
- One additional feature: The forwarding table can have a default router. If no other entry in the table matches, send the packet to the default router.

IP Addresses

- We have been talking conceptually about internetwork addresses.
- We will now discuss the actual details of IP addresses (for the real Internet).
- In this chapter we discuss IPv4 addresses. Much of the Internet still uses IPv4. In a few chapters we will look at a new format, IPv6 addresses.
- IPv4 address are 32 bit numbers, written as decimal versions of each byte. For example, 171.69.210.245

IP Addresses

- IP Addresses are divided into three classes, A, B, and C. Within each class, the number is further split into the Network part and the Host part.



IP Addresses

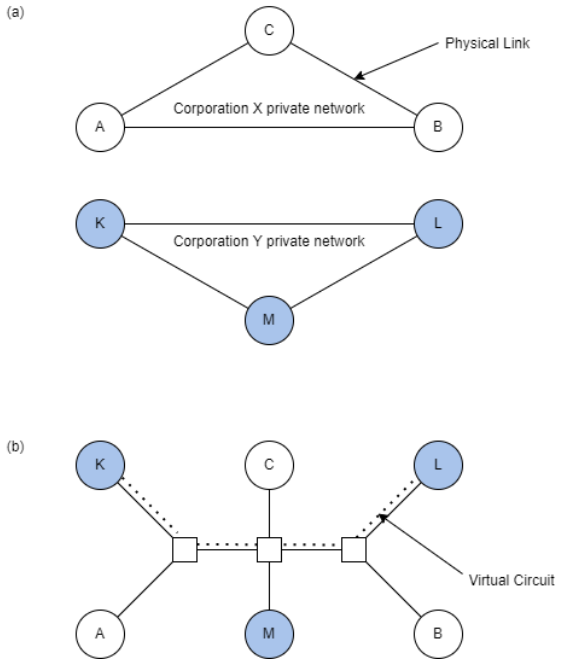
- There can only be 126 Class A networks, but each can hold about 16 million hosts.
- There can be 16,384 Class B networks, but each can hold 65,534 hosts.
- There can be about 2 million Class C networks, but each can hold 254 hosts.
- Yes, these numbers do not quite match the field sizes: two addresses in each set are reserved.
- Due to the hierarchical nature of the Internet, the router tables are actually much smaller that a couple million entries. (Recall that for our example, the routers within Network A did not need to know any of the addresses within Network B.)
- These tables are filled by a routing algorithm (take CS3800)

Virtual Private Networks

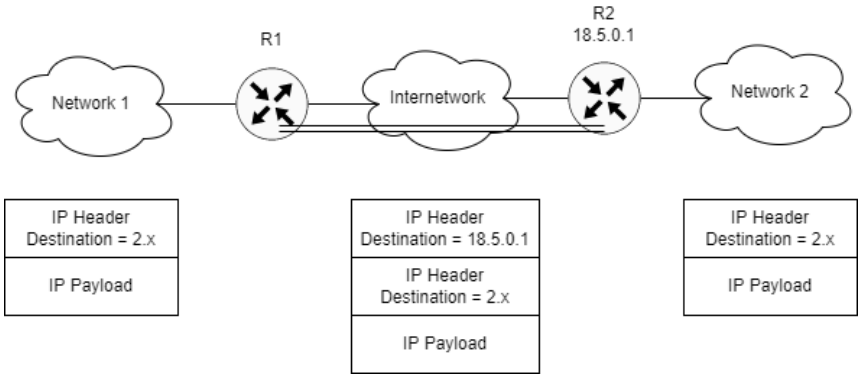
- A corporation may have multiple sites.
- Most likely, the corporation would like these sites on a private network.
  - By private network, we usually mean:
    - The hosts spread throughout the corporation can easily 'see' and communicate with all other hosts.
    - Hosts and routers outside of the corporation cannot see (or if they do see, understand) the packets being sent within the corporation.
    - Many of the hosts inside the corporation may not be visible to hosts outside the corporation (although a host on the inside may see 'out').
- The corporation might implement this by using dedicated physical links between the sites (expensive).

Virtual Private Networks

- Alternatively, a corporation can use public internetworking, but with a *virtual private network*.
- In diagram (a), there are two corporations with multiple sites. Within each corporation, the networks are linked with what looks like private physical links.
- Diagram (b) shows the reality, these sites are linked through the Internet. However, virtual circuits connect the networks (only one virtual circuit is shown).
- This virtual circuit is also called a *tunnel*.

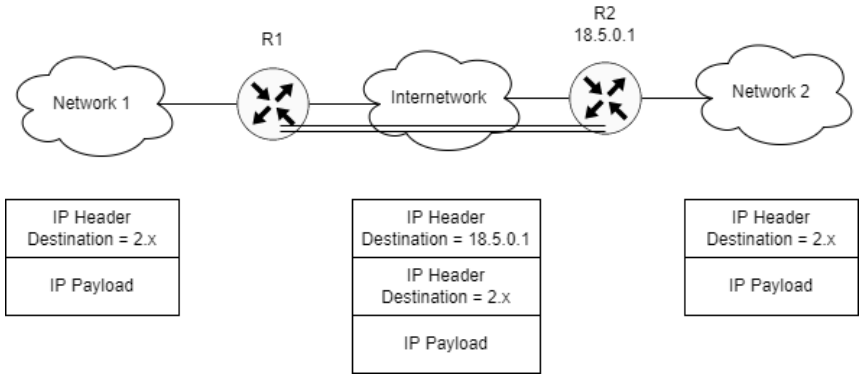


Virtual Private Networks



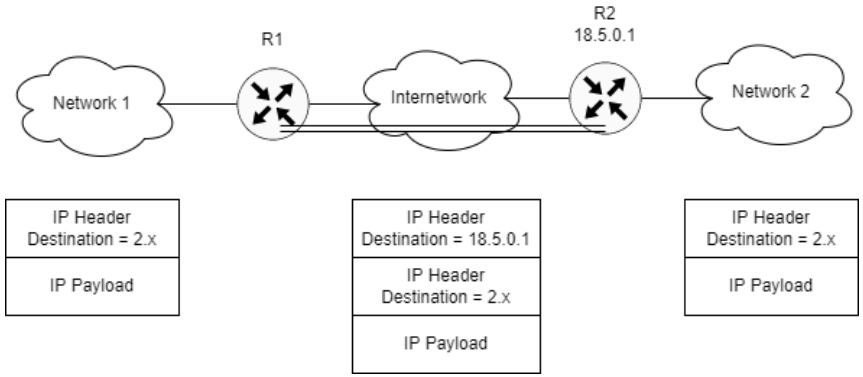
- This diagram shows two networks from one of these companies.
- Network 1 is one of the sites from company Y (for example, site K), while Network 2 is another site for the same company (i.e. site L).
- Suppose some host in network 1 wants to send a packet to a host in network 2. However:
  - The routers within the company know about network 1 and network 2, but routers outside the company do not … that information is private.
  - The only way to actually get there is to go through router R1, to 'into the wild' of the Internet, reach router R2, then go on to the destination.

Virtual Private Networks



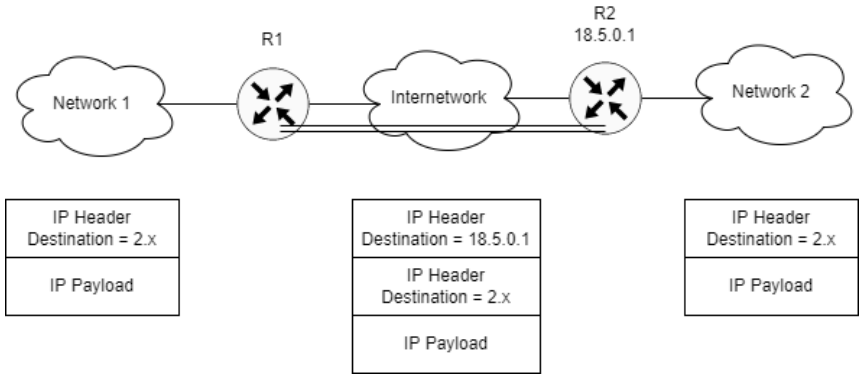
- To achieve this, R1 receives the packet.
- Its forwarding table indicates that this should go through the Internet to R2.
- R1 will slap a new header on the packet, giving a destination of router R2.
- At the same time, R1 will probably encrypt the original packet (including the header).
- This new packet is then sent over the Internet.

Virtual Private Networks



- Any malicious hardware or software that would like to examine the contents of the packet are out of luck, since the packet is encrypted.
- Notice that this encryption is handled in the router, and hence *all* packets, regardless of the type or the application, are protected.

Virtual Private Networks



- When router R2 receives the packet, it removes the outer IP header. The demux key in the header indicates that this is a VPN packet.
- R2 will then decrypt the packet, then see the resulting header. It will then send the message to the correct destination in Network 2.

Wrap Up

- Whew! We covered a lot of stuff in this topic.
- Each of the techniques and technologies we discussed is a significant topic on its own.
- However, having an overview of these pieces is useful.
- We will talk about a few more Cloud Computing techniques, which are based on the information presented here.