CSE 588: Network Systems

Discussion Notes:

Review: Priorities

Interoperability with existing networks (e.g., Ethernet, 802.11, …)

Robustness (availability/correctness): how much is enough? (99%; 99.9999%)
Side note: robustness for application vs. robustness for network. Web documents are often cached, so higher availability than uptime for telecon.

Scalability (to a trillion hosts)

Flexibility to deal with any kind of application (e.g., real-time, secure, …)

Extensibility
Perhaps this should be first: allows the others to be built over time

Cost-performance
Or just cost? Cheap, mass-produced, easy to use technologies get adopted, so want to be on that train.

Roles:

What should a router do? Anything that can't be done by the endpoint? Why? (Promotes rapid change, compared to embedding functionality in the network.)

Sidebar: how long did the following technologies take to become widely adopted? Object oriented programming, RISC computing, RAID, relational databases, bitmap displays, threads, … Why do some take longer than others?

Design principles:

We didn't get to this, but it is pretty important. How do you achieve robustness? Can you do it by design?

Example: end to end principle was motivate by robustness, turned out its main benefit was flexibility/extensibility.

Example: soft state. Make limited assumptions about what other nodes in the network know or don't know; add redundant updates for self-correction. Do you buy this?

Third paper is about: make limited assumptions about correct behavior of other nodes. Perhaps you shouldn't trust what others are telling you.

We'll revisit in context.

Today's discussion: TCP/IP

IP: the interoperability layer for communication across multiple networks

What needs to go in the IP header? Destination address

Addressing: need some way of talking about remote machines. What should this look like?

Options:

Global addresses. How long? 32 bit? 128 bit? (a trillion is 40 bits)

In this case, how do you go from user names to addresses? DNS – global naming service. How does this work?

How do you route? How large do the interior router tables need to be, and how do you need to do address assignment so that works? Need hierarchical address assignment. (Analogy: area codes in phone numbers). Sidebar: how does CIDR work? How does the internals of router lookup work? How fast does it need to go?

What about multihoming? Breaks aggregation.

What about hosts with multiple interfaces (on different networks?)

What about firewalls?

What about mega-services? Load balancers? Physically distributed services/load balancers?

What about mobility?

What about multicast? Are group addresses global? Local?

What happens if you need to change an address? (or phone #)

Local address realms (NATs): current practice, in part because of running out of 32 bit IP addresses, in part because of security concerns.

Sidebar: how does a NAT work?

Use global addresses for talking between realms, local addresses within a realm.

Names?

How do 1-800 numbers work? Allow arbitrary reforwarding, based on time of day, source of call, …

Would eliminate a layer of confusion, allow for arbitrary extensibility, mobility, …

How do users register the location of names? How do they control how names get routed?

Example: communication over challenged networks; what if no end to end connectivity? Painful to send message to do name translation, then wait a week, send real message, wait a week, get back announcement that address has changed, …

Any downsides to using names as addresses?

Any implications of this choice on robustness?

Some mechanics:

How does a host get an IP address?
- Static assignment
- DHCP
- Embedded MAC address

How does a host know its IP router's MAC address?

How does a router know the destination's MAC address? (ARP)

What about security?

How do you enforce that source addresses aren't spoofed?

How do you know someone hasn't spoofed your router?

How do you know someone hasn't spoofed the ARP reply? (In fact, ability to spoof ARP is needed by NAT failover mechanisms!)

Lack of robust security infrastructure makes many of these low level details very hard to solve.

Circuits and/or packets?

IP service model is best-effort packet switching. Packets can be dropped (how?), reordered (how?), routes can change during a connection without any visibility to higher layers.

How do we do real-time guarantees in this context?

One approach: soft-state, refreshable guarantees (if routing is infrequent, and ok to break guarantees)

Circuits are another option. How do these work? Circuit tables. Faster lookup, easier to integrate per-connection guarantees, accounting.

Example: MPLS: point to point tunnels. Send connection setup request, get back ack; then send data into tunnel.

Should circuits be end to end, or interior to the network? If end to end, what does that mean for scalability? If interior, can the host specify tunnel?

What is the failover procedure when a link goes down and you need to reestablish the connection?

What about client mobility?

What if the host crashes and doesn't tear down the connection?

Another option: source routing. Supported in the Internet, but typically not enabled. Why? Economics of ISPs.

Fragmentation:

Why might I need to fragment? Also need to fragment at TCP level!

How do I know the characteristics of a path (such as MTU)?

Internet today: don't fragment bit. Could we do better? Should we support fragments at all? (requires offset, ident fields)

Other characteristics: Bandwidth? TCP probes bandwidth; causes packet loss even if only one node is using the bottleneck link.

Congestion? Discuss later; one option is to annotate packets with bit that is set to indicate congestion.

Other things in the IP header: (do we need these?)

Length

Type of service (do we need this?)

Checksum (do we need this?)

TTL (why do we need this?)

Protocol (why do we need this?)

ICMP

Error reporting protocol

Example: traceroute, using TTL

Example: unrouteable message in response to receiving packet for which no local host

TCP

How do we make end to end transport work?

API -- What do applications want?

Sockets: two-directional byte stream

One node is the listener (server); one does the connect (client). Applications can write/read arbitrary amounts of data into socket (just like a file or a pipe), asynchronously with arrival of data (block if buffer is full/empty). Either can close the connection.

Socket setup is natural place to specify bandwidth, latency, loss requirements.

Should we optimize for local communication on our local network? Alternative: format every packet with TCP/IP header (example: NFS used to run on UDP, but doesn't anymore)

How do you choose TCP segment size?

What if you have a partial packet to send? Always push? Nagle: push only one partial packet at a time.

Ports

Source/destination port to identify the application

How do I know the port? Well-known ports for services
http://www.iana.org/assignments/port-numbers

Clients dynamically choose their source port; can be any number above 1024

Connection specified by combination of source, destination IP address and port number. So client can make multiple connections to same service port, and server can handle multiple connections at once.

ARQ and sliding window

What if packet is lost? How do we recover? Timeout and retry. How do we tell difference between original, retransmission, and new packet?

Each packet carries sequence # range.

Why do acks carry sequence #?

Why a sliding window? How big does the sliding window need to be?

What if a packet is lost? How does the sender detect this? What does the receiver do with the out of order packets?

Send and receive buffers are not the same as the send and receive windows.

Work through an example: send buffer, receive buffer. Can't send more than will fit in the receive buffer.

Sent and acked
Sent and not acked
Not yet sent

Received and acked
Not yet received

Selective acknowledgements (SACK); precise control over retransmission.

Timeouts

How do you determine timeouts? What if too large? What if too small?

Retransmission ambiguity, and how to deal with it.

RTT estimators, and variance in RTT.

Flow control

How do we keep fast sender from overwhelming a slow receiver? For example, to a printer.

Explicit receive window size, under receiver control.

How does sender get notified when receive window reopens?

What happens if receiver is very slow; will reopen window by a single byte. silly window syndrome avoidance.

Connection setup/teardown

Three way handshake to start (exchange sequence numbers)

Two-way handshake to end

Do we need to optimize this exchange for short connections (e.g., an HTTP object)? If so, how? (or rely on higher level protocols, such as persistent HTTP connections)

How do we distinguish between a new SYN, a retransmitted SYN, and a late SYN?

How do we know that a packet doesn't arrive after a new connection is reestablished with the same port/address?

What if the sender reboots and reconnects?

Can a malicious attacker hijack a TCP connection? How?

TCP SYN flooding: how do we prevent this?

TCP Header

Source port, destination port (16 bits: should be smaller, larger?)

Sequence #, ack # (32 bits: enough? More? Less?)

Advertised window (16 bits: enough?)

Checksum (16 bits: enough?)

UrgPtr (16 bits: needed?)

Flags: syn, ack, fin, reset, push, urg. Do we need them all?

HeaderLength/options (as practical matter, options are almost never supported – and can be used as a backdoor to denial of service attack, since often incur extra processing)

Other measurement tools:

Sting: exploit TCP acks to extract loss rate, latency

Tbit: determine what version of TCP host is running