Review of Networking and Design Concepts (I)

http://www.pde.rpi.edu/

Or

http://www.ecse.rpi.edu/Homepages/shivkuma/

GOOGLE: “Shiv RPI”

Shivkumar Kalyanaraman
Rensselaer Polytechnic Institute
shivkuma@ecse.rpi.edu

Based in part upon slides of Prof. Raj Jain (OSU), S. Keshav (Cornell), L. Peterson (Princeton), J. Kurose (U Mass)
The goal of networking: “Connectivity”
- direct (pt-pt, N-users),
- indirect (switched, inter-networked)

Concepts: Topologies, Framing, Multiplexing, Flow/Error Control, Reliability, Multiple-access, Circuit/Packet-switching, Addressing/routing, Congestion control

Data link/MAC layer:
- SLIP, PPP, LAN technologies …

Interconnection Devices

Chapter 1-7, Keshav’s book

Reading: Saltzer, Reed, Clark: "End-to-End arguments in System Design"
Reading: Clark: "The Design Philosophy of the DARPA Internet Protocols"
Reading: RFC 2775: Internet Transparency: In HTML
What is networking?

- The goal of networking is to provide connectivity or a “virtual link” between end-points.
  - Different networking architectures implement specialized virtual link abstractions.

- Virtual link vs Physical link:
  - Different performance and semantic characteristics.
    - **Virtual link:** "un-secure, unreliable, best-effort packet-switched service" vs
    - **Physical link:** "secure, reliable, bit-stream, guaranteed QoS service" provided by a physical point-to-point link.

- Networking today is getting integrated with distributed systems.
  - From virtual links and virtual resources to virtual services …
  - … abstraction realized over physically distributed components…
What’s a network: “nuts and bolts” view (h/w building blocks)

- **network edge:** millions of end-system devices:
  - pc’s workstations, servers
  - PDA’s, phones, toasters running network apps
- **network core:** routers, switches forwarding data
  - packets: packet switching
  - calls: circuit switching
- **communication links**
  - fiber, copper, radio, …
A closer look at network structure:

- **network edge:** applications and hosts
- **network core:**
  - routers
  - network of networks
- **access networks,**
  **physical media:** communication links
The network edge:

- **end systems (hosts):**
  - run application programs
  - e.g., WWW, email

- **client/server model**
  - client host requests, receives service from server
  - e.g., WWW client (browser)/server; email client/server

- **peer-peer model:**
  - host interaction symmetric
  - e.g.: Gnutella, KaZaA

- **service models:**
  - connectionless (UDP) or connection-oriented (TCP) service
The Network Core

- mesh of interconnected routers
- **the fundamental question**: how is data transferred through net?
  - circuit switching: dedicated circuit per call: telephone net
  - packet-switching: data sent thru net in discrete “chunks”

- [Ckts: network resources are chopped up in units of “circuits”]
- Pkts: data is chopped up in units of “packets”]
Q: *How to connect end systems to edge router?*

- residential access nets
- institutional access networks (school, company)
- mobile access networks

**Keep in mind:**

- bandwidth (bits per second) of access network?
- shared or dedicated?
- Symmetric or asymmetric? (inbound b/w > outbound b/w)
Example access net: home network

Typical home network components:
- ADSL or cable modem
- router/firewall
- Ethernet
- Wireless access point

Diagram:
- Cable modem
- Router/firewall
- Wireless access point
- Ethernet (switched)
- Wireless laptops

Diagram: Connections from/to the cable headend.
Beyond hardware components: **Protocols** (s/w building blocks)

Why protocols or interactions? [for a distributed function] Because the information & resources are not in one place…

Goal of protocol design: **minimize** interactions!

Hi

Got the time?

2:00

TCP connection req.

TCP connection reply.

Get http://gaia.cs.umass.edu/index.htm

<file>
Internet protocol stack

Why layering? Modularity (for software productivity!) & support for evolution (in line with Moore’s law!)

- **application**: supporting network application
  - ftp, smtp, http
- **transport**: host-host data transfer
  - tcp, udp
- **network**: routing of datagrams from source to destination
  - ip, routing protocols
- **link**: data transfer between neighboring network elements
  - ppp, ethernet
- **physical**: bits “on the wire”
Layering: logical ("virtual") communication

E.g.: transport
- take data from app
- add addressing, reliability check info to form "datagram"
- send datagram to peer
- wait for peer to ack receipt
- analogy: post office
- Virtual link abstraction (aka peer-to-peer interface)
Layering: *physical* communication
Internet structure: network of networks

- roughly hierarchical
- at center: “tier-1” ISPs (e.g., UUNet, BBN/Genuity, Sprint, AT&T), national/international coverage
- treat each other as equals

Diagram:
- Tier-1 providers interconnect (peer) privately
- Tier-1 providers also interconnect at public network access points (NAPs)
Internet structure: network of networks

- “Tier-2” ISPs: smaller (often regional) ISPs
  - Connect to one or more tier-1 ISPs, possibly other tier-2 ISPs

**Diagram:**
- Tier-2 ISP pays tier-1 ISP for connectivity to rest of Internet
- Tier-2 ISP is customer of tier-1 provider
- Tier-2 ISPs also peer privately with each other, interconnect at NAP

**Legend:**
- Tier 1 ISP
- Tier 2 ISP
- NAP
Internet structure: network of networks

- “Tier-3” ISPs and local ISPs
- Last hop (“access”) network (closest to end systems)

Local and tier-3 ISPs are customers of higher tier ISPs connecting them to rest of Internet.
Internet structure: network of networks

- A packet passes through many networks!
- Creates an end-to-end “Virtual Link” abstraction

Try a traceroute!
Pause! … List of Ideas …

- Networking **goal**: realize connectivity or virtual link abstractions
- **High-level network structure**: end/access/edge/core …
  - **Tiered** hierarchical structure of Internet
- Hardware and software **building blocks**: hosts, routers, protocols, layering
- **E2E Service Models**: connectionless (UDP) vs connection-oriented (TCP)
- **Network Transport Models**: circuit-switched vs packet-switched.
Fundamental Building Blocks...
How to provide (or implement) the connectivity abstraction?

- **Starting point:** Physical Building Blocks
  - links: coax cable, optical fiber...
  - nodes: general-purpose workstations...

- **Direct** connectivity:
  - point-to-point
  - multiple access
Connectivity… (Continued)

- **Indirect** Connectivity
  - switched networks
  - \( \Rightarrow \text{switches} \)

- inter-networks
  - \( \Rightarrow \text{routers} \)
Summary: How to Implement “Connectivity”? 

- **Direct or indirect access** to every other node in the network:
  - Using nodes, (shared/dedicated) links, switches, routers + protocols
  - End result: Virtual link abstraction

- **Tradeoff:** Performance & semantic characteristics different vs physical link!
Virtual link vs Physical Link

- Internet:
  - *Best-effort*  
    *(no performance guarantees)*  
  - *Packet-by-packet*

- A pt-pt link:
  - *Always-connected*
  - *Fixed bandwidth*
  - *Fixed delay*
  - *Zero-jitter*
Indirect Connectivity: Misc...

- The architectural split between data, control and management planes become explicit as we build scalable "indirect" connectivity abstractions over heterogeneous components or networks.

- Topics like security, multicast and wireless/mobility can be viewed as advanced virtual link abstractions.
Direct Connectivity: Details...

- A. Connecting 2 users with a virtual link:
  - Point-to-point connectivity
  - Higher level abstraction than the “raw” physical link

- B. Connecting N users and creating virtual link abstractions between any pair
  - Topologies, MAC protocols
Point-to-Point Connectivity Issues

- **Physical** layer: coding, modulation etc
- **Link** layer needed if the link is shared bet’n apps; is unreliable; and is used sporadically
  - **New Ideas**: frame, framing, multi-protocol encapsulation, error control, flow control
- **No need (yet) for protocol concepts** like addressing, names, routers, hubs, forwarding, filtering …
- **Tradeoffs**: connects only 2 users; not scalable
Concept of “Frame” or “Packet”

- What is a Frame?
  - Limited number of bits + meta-data or timing clues for delimiting the frame (PHY-level hints)

- Why frame?
  - Can share link between multiple protocols (“multi-protocol encapsulation”)
  - Frame = unit for error detection and correction.
    - Bit stream is highly likely to have errors => split into “blocks” for error control. (‘CRC”, “FEC”, “ARQ”)
  - Frame = unit or sub-unit for flow control. Larger unit = “window”

- Why meta-data (header/trailer) vs low-level timing clues?
  - More flexibility: avoid need for synchronization, convey more protocol control information in header fields, allow statistical sharing, fit well with “layering” (onion-like header fields for each layer)
Example Link Layer: Serial IP (SLIP)

- Simple: only framing = Flags + byte-stuffing
- Compressed headers (CSLIP) for efficiency on low speed links for interactive traffic.
- Problems:
  - Need other end’s IP address a priori (can’t dynamically assign IP addresses)
  - No “type” field => no multi-protocol encapsulation
  - No checksum => all errors detected/corrected by higher layer.
- RFCs: 1055, 1144
Flag Fields

- Delimit frame at both ends
- Flag code = 01111110
- May close one frame and open another
- Receiver hunts for flag sequence to synchronize frame
- Bit stuffing used to avoid confusion with data containing 01111110
  - 0 inserted after every sequence of five 1s
  - If receiver detects five 1s it checks next bit
  - If 0, it is deleted
  - If 1 and seventh bit is 0, accept as flag
  - If sixth and seventh bits 1, sender is indicating abort
Bit Stuffing

- Example with possible errors

Original Pattern:

11111111111011111101111110

After bit-stuffing

11111011111011011111101011111010

(a) Example

(b) An inverted bit splits a frame in two

(c) An inverted bit merges two frames
Link Layer: PPP

- **Point-to-point protocol**
- Frame format similar to HDLC
- Multi-protocol encapsulation, CRC, dynamic address allocation possible
  - key fields: flags, protocol, CRC
  - Note: protocol field is an “identifier” or “address” to aid multiplexing/demuxing
- Asynchronous and synchronous communications possible
Link Layer: PPP (Continued)

- Link and Network Control Protocols (LCP, NCP) for flexible control & peer-peer negotiation
- Can be mapped onto low speed (9.6Kbps) and high speed channels (SONET)
- RFCs: 1548, 1332
SONET (STS-1) Frame Format

90 Bytes
Or “Columns”

Small Rectangle = 1 Byte

Two-dimensional frame representation (90 bytes x 9 bytes)…

Frame Transmission: Top Row First, Sent Left To Right

- **Time-frame:** 125 μs/Frame
- **Frame Size & Rate:**
  810 Bytes/Frame * 8000 Frames/s * 8 b/byte = 51.84 Mbps
- For STS-3, only the number of columns changes (90x3 = 270)

STS = Synchronous Transport Signal

Shivkumar Kalyanaraman
Reliability & Error Control

- **Goal**: recovery from failure (e.g., bit/packet errors)
- Reliability => requires **redundancy** to recover from uncertain loss or other failure modes.

- Two types of redundancy:
  - **Spatial redundancy**: independent backup copies
    - Forward error correction (FEC) codes (intra-pkt or per-window)
    - Problem: requires **overhead and computation**. Also, since the FEC is also part of the packet(s) it cannot recover from erasure of all packets
  - **Temporal redundancy**: retransmit if packets lost/error
    - Lazy: trades off **response time** for reliability
    - Design of status reports and retransmission optimization important
Bit level error detection

EDC = Error Detection and Correction bits (redundancy)
D = Data protected by error checking, may include header fields

- Error detection not 100% reliable!
  - Protocol may miss some errors, but rarely
  - Larger EDC field yields better detection and correction
Error Checks: Parity Checking & CRC

**Single Bit Parity:**
Detect single bit errors

**Two Dimensional Bit Parity:**
Detect and correct single bit errors

Much more powerful error detection/correction schemes: Cyclic Redundancy Check (CRC)

Simple form of forward error correction (FEC)
Temporal Redundancy Model (ARQ)

- Packets
  - Sequence Numbers
  - CRC or Checksum
  - Proactive FEC (optional)

- Status Reports
  - ACKs
  - NAKs,
  - SACKs (complex)
  - Bitmaps (complex)

- Retransmissions (ARQ)
  - Packets
  - Reactive FEC (optional)

Timeout
Forward Error Correction (FEC):
Eg: Reed-Solomon RS(N,K)

- RS(N,K)
- FEC (N-K)
- Block Size (N)
- Data = K
- >= K of N received
- Recover K data packets!

Note: Since Error Detection + ARQ is more reliable & simpler than FEC, it is more common in older protocols.
Types of errors and effects

- **Forward** channel bit-errors (garbled packets)
- **Forward** channel packet-errors (lost packets)
- **Reverse** channel bit-errors (garbled status reports)
- **Reverse** channel bit-errors (lost status reports)

- **Protocol-induced** effects:
  - Duplicate packets
  - Duplicate status reports
  - Out-of-order packets
  - Out-of-order status reports
  - Out-of-range packets/status reports (in window-based transmissions)
Link-level Reliability Mechanisms

- **Mechanisms:**
  - **Checksum**: detects corruption in pkts & acks
  - **ACK**: “packet correctly received”
  - **Duplicate ACK**: “packet incorrectly received”
  - **Sequence number**: identifies packet or ack
    - 1-bit sequence number used *both in forward & reverse channel*
  - **Timeout only at sender**

- **Reliability capabilities achieved:**
  - An *error-free* channel
  - A *forward & reverse* channel with bit-errors
  - Detects *duplicates* of packets/acks
  - *NAKs eliminated*
  - A *forward & reverse* channel with packet-errors (loss)
Link-level Flow Control: (Stop and Wait)

Stop-and-wait is quite efficient for medium-speed LANs (low \( \alpha \)):
Indeed, Ethernet CSMA/CD uses stop-and-wait!
(collision = NAK, no collision = implied ACK)

It is a terrible idea for larger B-D product channels (high \( \alpha \))!

\[
\alpha = \frac{t_{\text{prop}}}{t_{\text{frame}}}
\]

\[
= \frac{\text{Distance/Speed of Signal}}{\text{Frame size /Bit rate}}
\]

\[
= \frac{\text{Distance} \times \text{Bit rate}}{\text{Frame size} \times \text{Speed of Signal}}
\]

\[
U = \frac{t_{\text{frame}}}{2t_{\text{prop}} + t_{\text{frame}}}
\]

\[
= \frac{1}{2\alpha + 1}
\]

Light in fiber:
- Speed = 200 m/\( \mu \)s
Electricity:
- Speed = 250 m/\( \mu \)s
Concepts like sliding windows, sequence numbers, feedback, timeouts are common between “reliability” and “flow/congestion control” functions

⇒ These functions are often “coupled” in protocol design…
Coupled functions are double-edged swords!
Stop and Wait: For Flow Control + Reliability
Window: Flow Control + Reliability

“Go Back N” = Sliding Window + Retransmit Entire Window
Window: Flow Control + Reliability

“Selective Reject” = Sliding Window + Selectively Retransmit
Pause! … List of Ideas …

- Realizing connectivity (direct vs indirect)
- Pt-Pt (direct) connectivity:
  - **Framing**: SLIP vs PPP
  - **Error control/Reliability**:
    - CRC/checksum: check errors
    - FEC, ARQ, seq #s, timeouts, ack/nak: recover from errors
  - **Flow control**: stop-n-wait, sliding window,
- Synergies between flow control & reliability: go-back-N, selective retransmit ARQ
**Connecting N users: Directly...**

- Pt-pt: connects only **two** users **directly**...
- How to connect **N** users **directly**? Ans: **share** links!

- **Bus** (shared link)

- **What are the costs** of each option?
- **Does this method of connectivity** **scale**?
Building Block: Multiplexing
Multiplexing: Outline

- Single link:
  - Channel partitioning (TDM, FDM, WDM) vs Packets/Queuing/Scheduling

- Series of links:
  - Circuit switching vs packet switching
  - Statistical Multiplexing (leverage randomness)
    - Stability, multiplexing gains, Amdahl’s law
  - Distributed multiplexing (MAC protocols)
    - Channel partitioning: TDMA, FDMA, CDMA
    - Randomized protocols: Aloha, Ethernet (CSMA/CD)
    - Taking turns: distributed round-robin: polling, tokens
Building Block: Multiplexing

- Multiplexing = sharing
  - Allows system to achieve “economies of scale”
  - **Cost:** waiting time (delay), buffer space & loss
  - **Gain:** Money ($$) => Overall system costs less

Eg: Full Mesh  
Eg: Bus (shared link)

Note: share (or optimize) only **expensive** resources, **NOT** cheap ones!
Multiplexing at a Single Link

- Chop up the link (channel partitioning):
  - into time-slots: Time-division multiplexing (TDM), SONET
  - into frequency (or wavelength bands): FDM or WDM

- Chop up the input traffic into packets:
  - Packet switching => queuing (store-and-forward)
  - Buffer management & Scheduling
    - Scheduling: FIFO, priority or round-robin based
    - BM: early/random drop, drop-tail etc

- Hybrids: FDD or TDD to separate uplink from downlink and then other methods within each band
Coordinating a Series of Multiplexed Links: Circuit-Switching

- Divide link bandwidth into “pieces”
- Reserve pieces on successive links and tie them together to form a “circuit”
- Map traffic into the reserved circuits
- Resources wasted if unused: expensive.

- Mapping can be done without “headers”.
- Everything inferred from relative timing.
Coordinating a Series of Multiplexed Links: *Packet-Switching*

- Chop up data (not links!) into “packets”
  - Packets: data + meta-data (header)
- “Switch” packets at intermediate nodes
  - *Store-and-forward* if bandwidth is not immediately available.
  - I.e. build up “packet queues”

Bandwidth division into “pieces”
Dedicated allocation
Resource reservation
Another Viewpoint: Spatial vs Temporal Multiplexing

- **Spatial multiplexing**: Chop up resource into chunks. Eg: bandwidth, cake, circuits…

- **Temporal multiplexing**: resource is shared over time, i.e. queue up jobs and provide access to resource over time. Eg: FIFO queuing, packet switching

Packet switching is designed to exploit *both* spatial & temporal multiplexing gains, provided performance tradeoffs are acceptable to applications.

- *Packet switching is potentially more efficient => potentially more scalable than circuit switching!*
Statistical Multiplexing

- Smartly reduce resource requirements (e.g., bus capacity) by exploiting statistical knowledge of the load on the resource.
  - (yet offering acceptable service)

- Key (first order) requirement:
  - average rate $\leq$ service rate $\leq$ peak rate

- If service rate $<$ average rate, then system becomes unstable!!
  - We will later see that there are other forms of instability (in a control-theoretic sense) caused in feedback-control systems

- Lesson 1: Design to ensure system stability!!

Shivkumar Kalyanaraman

Rensselaer Polytechnic Institute
Stability of a Multiplexed System

Average Input Rate > Average Output Rate => system is unstable!

How to ensure stability?

1. Reserve enough capacity so that demand is less than reserved capacity
2. Dynamically detect overload and adapt either the demand or capacity to resolve overload

- **Cost**: self-descriptive header per-packet, buffering and delays due to statistical multiplexing at switches.
- **Need to either reserve resources or dynamically detect and adapt to overload for stability**
What is relevant “statistical knowledge”?

PDF

- In general: (use measurement/modeling to get this!)
- 1 R.V.: mean/median/mode, variance/SIQR/CoV, skew/kurtosis/heavy-tail measures etc
- Multiple RVs: joint pdf, marginal pdfs, conditional pdfs, covariance/correlation
- Random process/time series/finite states: IID or autocorrelation function, Markovian/chains, covariance matrix, aggregate limit processes (e.g. gaussian, self-similar/Hurst parameter)

CCDF: for heavy tailed distributions

FIGURE 2.4. Probability distribution for a conceptual population of yield values.
Time-Series “Statistical” Models (of burstiness) …

model packets, flows, connections, and meta-connections

meta-connection

connections

flow

packets
Statistical Multiplexing (Continued)

- Once you have a \textbf{stable} multiplexed system, then try to tune the \textbf{tradeoffs} using statistical knowledge
  - i.e. tradeoff whatever is “cheap” and optimize on whatever is “expensive” or unavailable. \textit{Egs (TCP)}:
    - Tradeoff delay to maximize goodput
    - Tradeoff feedback complexity to ensure stability
- At links: Tradeoff muxing gain to reduce queuing delays, buffering requirements & packet losses
  - \textit{Gain} = peak rate/service rate.
  - \textit{Cost}: buffering, queuing delays, losses.

\textbf{Recall:} You \textbf{MUST} have tradeoffs! Identify them…
There is no free lunch in system design.
What’s a performance tradeoff?

- A situation where you cannot get something for nothing!
- Also known as a zero-sum game.

- \( R = \) link bandwidth (bps)
- \( L = \) packet length (bits)
- \( a = \) average packet arrival rate (pkts/s)

Traffic intensity = \( \frac{La}{R} \)
What’s a performance *tradeoff*?

- $La/R \approx 0$: average queuing delay small
- $La/R \rightarrow 1$: delays become large
- $La/R > 1$: average delay infinite (*service degrades unboundedly => instability*)!

Summary: Multiplexing using bus topologies has both *direct* resource costs and *intangible* costs like potential instability, buffer/queuing delay.

Shivkumar Kalyanaraman
Design Process: Amdahl’s Law

- If design implies a set of tradeoffs, the question is how to redesign components so that the system cost-performance tradeoff is improved?

- Amdahl’s law talks about the maximum expected improvement to an overall system when only a part of the system is improved.
  - Statement of “diminishing returns”
  - System Speedup = \[
  \frac{1}{(1 - P) + \frac{P}{S}}
  \]

- Guides the iterative design process.
Lessons from Amdahl’s Law

- If a part of a system accounts for 12% of performance ($P = 0.12$) and
- You speed it up 100-fold ($S = 100$)
- The actual system speedup is only: 13.6% !!!!

$$\frac{1}{1 - 0.12} = 1.136$$

- **Lesson #1:** Find and optimize the common cases *(that account for a large fraction of system performance)*
- **Lesson #2:** Bottlenecks *shift*! Once you optimize one component, another will become the new bottleneck!
Statistical multiplexing useful only if peak rate differs significantly from average rate.

Eg: if traffic is smooth, fixed rate, no need to play games with capacity sizing based upon complicated statistics that are hard to forecast/estimate...

(Traditional) Circuit-switched telephony does not exploit statistical multiplexing within a single circuit

TDM: 64kbps is reserved (8 bits per 125 usecs), and wasted if no load (voice sample).

Implications also for network topology, routing etc

Statistical muxing IS exploited at higher levels (eg: poisson, Erlang models used) to size network capacity
Multi-Access LANs: Distributed Multiplexing

- **Medium Access Control (MAC) Protocols:**
  - Arbitrates the distributed multiplexing process
  - ALOHA, CSMA/CD (Ethernet), Token Ring ...
  - **Key:** Use a single protocol in network

- **New Concepts:** address, forwarding (and forwarding table), bridge, switch, hub, token, medium access control (MAC) protocols
MAC Protocols: a taxonomy

Three broad classes:

- **Channel Partitioning:** TDMA, FDMA
  - divide channel into “pieces” (time slots, frequency)
  - allocate piece to node for exclusive use

- **Random Access:** Aloha, Ethernet CSMA/CD, WiFi CSMA/CA
  - allow collisions
  - “recover” from collisions

- **“Taking turns”:** Token ring = distributed round-robin
  - Coordinate shared access using turns to avoid collisions.
  - Achieve statistical multiplexing gain, but at greater complexity
  - CDMA can be loosely classified here (orthogonal code = token)

**Goal:** efficient, fair, simple, decentralized
Channel Partitioning
MAC protocols. Eg: TDMA

TDMA: time division multiple access

- Access to channel in "rounds"
- Each station gets fixed length slot (length = pkt trans time) in each round
- Unused slots go idle
- Example: 6-station LAN, 1,3,4 have pkt, slots 2,5,6 idle
- Does not leverage statistical multiplexing gains here
Partitioning (FDMA, TDMA) vs CDMA
“Taking Turns” MAC protocols

Channel partitioning MAC protocols:
- share channel efficiently at **high load**
- inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

Random access MAC protocols [discussed a little later]
- efficient at **low load**: single node can fully utilize channel
- **high load**: collision overhead

“Taking turns” protocols
look for best of both worlds!
Similar to Round-Robin!

- Round Robin: scan class queues serving one from each class that has a non-empty queue
Distributed Round-Robin

Polling:
- Master node "invites" slave nodes to transmit in turn
- Request to Send, Clear to Send messages
- Concerns:
  - polling overhead
  - latency
  - single point of failure (master)

Token passing:
- Control token passed from one node to next sequentially.
- Token message
- Concerns:
  - token overhead
  - latency
  - single point of failure (token)
More complex “turns” methods

Reservation-based a.k.a Distributed Polling:

- Time divided into slots
- Begins with N short reservation slots
  - reservation slot time equal to channel end-end propagation delay
  - station with message to send posts reservation
  - reservation seen by all stations
- After reservation slots, message transmissions ordered by known priority
Building Block: Randomization
Building Block: Randomization

- **Insight**: randomness can be used to **break ties** (without sharing **state** in a distributed system!).
  - Tradeoff: performance degradation at high load
  - Stateless => no need for protocol messages!

- If multiple nodes transmit at once, it leads to a “collision” or a “tie”.
  - Randomly choose to transmit: i.e. when you arrive [Aloha]
  - Still leads to ties when the load increases (multiple users): “birthday problem.”
  - Slotting helps focus ties, but not enough [slotted Aloha]
  - p-persistence adds to delays [p-persistent Aloha]

- **Refinement**: randomization + **local state**:
  - Carrier sense (CS) before transmitting: avoid obvious “ties”
  - Collision detect (CD) => reduce “tie” penalty
  - Exponential backoff: Retransmit after a random interval whose length increases exponentially (reduce average load)
Randomized MAC Protocols

- **Aloha** at University of Hawaii:
  Transmit whenever you like
  Worst case utilization = $1/(2e) = 18\%$

- **CSMA**: Carrier Sense Multiple Access
  Listen before you transmit

- **CSMA/CD**: CSMA with Collision Detection
  Listen while transmitting.
  Stop if you hear someone else.

- **Ethernet** uses CSMA/CD.
  Standardized by IEEE 802.3 committee.
Ethernet

- single shared broadcast channel
- 2+ simultaneous transmissions by nodes: interference
  - only one node can send successfully at a time
- *multiple access protocol*: distributed algorithm that determines how nodes share channel, i.e., determine when a node can transmit
CSMA/CD Operation

TIME $t_0$
A's transmission
C's transmission
Signal on bus

TIME $t_1$
A's transmission
C's transmission
Signal on bus

TIME $t_2$
A's transmission
C's transmission
Signal on bus

TIME $t_3$
A's transmission
C's transmission
Signal on bus
Digression: More on Randomization

- **Stateless, distributed tie-breaking** property used often!
  - **Auto-configuration** in Appletalk, IPv6: choose a random L3 address (and optionally check to avoid a “tie”, i.e. duplicate address).
  - **Reliable multicast**: Avoid feedback implosion (i.e. a “tie” by having a random node ack, and others suppress their acks.
  - **Active Queue Management (AQM)**: RED uses randomization to break synchronization (i.e. “ties”) in participating TCP window dynamics
  - **Random exponential backoff** is also used in TCP timer backoff

- **Failure of naive randomization**:
  - Aloha variants under-perform CSMA/CD (**local state helps**)!
    - Slotting, p-persistence are cute tricks that help a little…
  - **Random walks** are a poor routing strategy…
    - They do not reach destination often and increase load!
    - Better to have a “**stateful**” routing strategy (with carefully constructed forwarding tables)
Digression: More on Randomization

- More: Randomization and Ties (goal: lower complexity):
  - **FEC:** Randomized LDPC erasure codes: check symbols cover a random subset of data symbols
    - Simple XOR operations. (↓ computational complexity)
    - Avoid overlaps by picking numbers from a carefully chosen distribution.
    - Tradeoff: K+ε check packets instead of K in reed-solomon.
  - **Statistical multiplexing:** peak rate transmissions (“ties”) can be absorbed in buffers, and the capacity => muxing gain…
  - Randomization in arrivals (aka burstiness or “ties”) causes all queuing! A stable D/D/1 system needs a minimal buffer size.

- Tradeoff between performance and correctness:
  - Randomized approximation algorithms, sampling techniques in measurement
  - **Data structures:** Bloom filters for set-membership checking, small false positive probability
    - (see “Probability and Computing” by Mitzenmacher, Upfal)
Building Block: Identifiers
Building Block: Identifiers & Addresses

- **New concept:** (after sharing links)
  - **Address** to identify nodes.
  - Needed if we want the receiver **alone** to consume the packet! (i.e. “filter” the packet)
  - Else resources consumed at all nodes unnecessarily.
What’s the big deal about an identifier?

- An identifier is a piece of state (i.e. information stored across time): most header fields are just IDs!!!
  - Eg: Ethernet/IP address, port numbers, protocol ID, OSPF Area ID, BGP autonomous system ID, DNS names (URLs, email IDs).
- Allows sharing (i.e. multiplexing) of the link
- Enables filtering (avoid needless resource consumption)
  - Eg: put IDs in packets or in forwarding

- Requirements:
  - Uniqueness to enable filtering
  - Configuration requirement: someone needs to assign unique IDs

- Could be overloaded to encode other semantics:
  - Eg: administrative structure (ethernet), location (IP address), network ID information (hierarchical L3 IDs).
  - Overloading of IDs is a double-edged sword!
### Ethernet & 802.3 Frame Format

**Ethernet**
- Dest. Address: 6 bytes
- Source Address: 6 bytes
- Type: 2 bytes
- Info: 2 bytes
- CRC: 4 bytes
- Size in bytes

**IEEE 802.3**
- Dest. Address: 6 bytes
- Source Address: 6 bytes
- Length: 2 bytes
- LLC: 2 bytes
- Info: 2 bytes
- Pad: 2 bytes
- CRC: 4 bytes

- Maximum Transmission Unit (MTU) = 1518 bytes
- Minimum = 64 bytes (due to CSMA/CD issues)
- Except for length, CRC, other fields are simply IDs!
**Ethernet (IEEE 802) Address Format**

- 48-bit flat address => no hierarchy to help forwarding
  - Hierarchy only for administrative/allocation purposes
  - Assumes that all destinations are (logically) directly connected.

- Address structure does not explicitly acknowledge or encode indirect connectivity
  - => Sophisticated filtering cannot be done!
**Ethernet (IEEE 802) Address Format**

(Organizationally Unique ID)

```
10111101
```

- **G/L bit**: *administrative*
  - Global: unique worldwide; assigned by IEEE
  - Local: Software assigned

- **G/I bit**: *multicast*
  - I: unicast address
  - G: multicast address. Eg: “To all bridges on this LAN”
Direct connectivity for N users => multiplexing, indirection, virtualization (latter two discussed later)

Multiplexing = sharing:
- Statistical multiplexing, relevant statistics, multiplexing gain
- Stability, performance tradeoffs
- Amdahl’s Law: guides iterative design

Identifiers: are unique pieces of state that allow efficient multiplexing through filtering.
- Most header fields in protocols are IDs

Multiplexing, IDs, indirection, virtualization are everywhere in computer systems!
Bridging, Switching, Routing: Origins;

Building Block: Filtering for Scalability
Limits of Direct Connectivity

- **Limited Scalability:**
  - Uses directly connected topologies (e.g., bus), or
  - MAC protocols that share a link don’t scale with increased load

- Interim solution:
  - Break up LANs into bridged domains
  - Indirectly connected with simple filtering components (switches, hubs).
  - Do not use global knowledge to set up tables that help filter packets and avoid flooding of packets
  - Default to flooding when information is insufficient
  - **Limited scalability** due to **limited filtering**
How to build *Scalable* Networks?

- **Scaling**: system allows the increase of a key parameter. Eg: let N increase…
  - **Key Observation**: *Inefficiency limits scaling* …

- Direct connectivity is *inefficient* & hence does not scale
  - Mesh: *inefficient* in terms of # of links
  - Bus architecture: 1 expensive link, N cheap links. *Inefficient* in bandwidth use
Filtering, forwarding ...

- **Filtering**: choose a subset of elements from a set
  - Don’t let information go where it’s not supposed to…
  - *Filtering => More efficient => more scalable*

  **Filtering is the key to efficiency & scaling**

- **Forwarding**: actually sending packets to a filtered subset of link/node(s): a form of indirection…
  - Packet sent exactly to one link/node => efficient

- **Solution**: Build nodes which focus on filtering/forwarding and achieve indirect connectivity

  **“switches” & “routers”**
Methods of “filtering”

- Flat (unstructured) address: destination filters the packet.
- Multicast address: only nodes configured with address filter the packet
- Reduction of flooding/broadcasting:
  - L2 Bridge forwarding table: (filtering through indirection)
    - lookup ID in table and send packet only to the port returned.
    - If lookup fails, then flood, guided by a spanning tree.
  - L3 Router forwarding table:
    - Broadcast is disallowed.
    - Lookup in L3 table ALWAYS succeeds (default route)
- Geographic routing: use GPS or location information to guide forwarding.
- Aggregation of addresses & Hierarchies:
  - Filter both routing announcements and packets
  - Restrict the flows to go through the hierarchy
  - Address Encoding and configuration complexities
- Complex non-hierarchical structures (distributed hash tables): rings, torus, de-bruijn graph (filtering through multiple indirections)
Why indirect connectivity?

- #1. allows **Scalability**
  - Using stronger filtering techniques like table-based unicast (vs flooding), address aggregation (vs flat addresses)
  - Using multi-hop forwarding: switches, routers

- #2. can handle **Heterogeneity**
  - Using an indirection infrastructure (overlay)
Connecting N users: *Indirectly*

- **Star**: One-hop path to any node, reliability, forwarding function
- “**Switch**” S can filter and forward!
  - Switch may forward multiple pkts in parallel for additional efficiency!
Connecting N users: *Indirectly* ...

- Ring: *Reliability* to link failure, *near-minimal* links
- All nodes need “forwarding” and “filtering”
- Sophistication of forward/filter lesser than switch
Topologies: Indirect Connectivity

Note: in these topologies (unlike full-mesh), some links and nodes are multiplexed (shared).

More complex topology => need for intelligent forwarding using addresses/tables.
**Inter-connection Devices**

- **Repeater:** Layer 1 (PHY) device that restores data and collision signals: a digital amplifier

- **Hub:** Multi-port repeater + fault detection
  - Note: broadcast at layer 1

- **Bridge:** Layer 2 (Data link) device connecting two or more collision domains.
  - Key: a bridge attempts to filter packets and forward them from one collision domain to the other.
  - It snoops on passing packets and learns the interface where different hosts are situated, and builds a L2 forwarding table
  - MAC multicasts propagated throughout “extended LAN.”
  - Note: Limited filtering intelligence and forwarding capabilities at layer 2
Interconnection Devices (Continued)

- **Router:** Network layer device. IP, IPX, AppleTalk. Interconnects *broadcast domains*.
  - Does not propagate MAC multicasts.

- **Switch:**
  - **Key:** has a switch fabric that allows parallel forwarding paths
  - **Layer 2 switch:** Multi-port bridge w/ fabric
  - **Layer 3 switch:** Router w/ fabric and per-port ASICs

These are functions. Packaging varies.
Interconnection Devices

LAN = Collision Domain

Extended LAN = Broadcast domain

Router

Application
Transport
Network
Datalink
Physical

Gateway

Router

Bridge/Switch

Repeater/Hub

Application
Transport
Network
Datalink
Physical

Shivkumar Kalyanaraman
Rensselaer Polytechnic Institute
Layer 1 & 2: Repeaters, Hubs, Bridges

- **Layer 1:**
  - *Hubs* do not have “forwarding tables” – they simply broadcast signals at Layer 1. No filtering.

- **Layer 2:**
  - Forwarding tables not required for simple topologies (previous slide): *simple forwarding rules suffice*
  - The next-hop could be *functionally* related to destination address (i.e. it can be computed without a table explicitly listing the mapping).
    - This places too many *restrictions* on topology and the assignment of addresses vis-à-vis ports at intermediate nodes.
  - Forwarding tables could be statically (*manually*) configured once or from time-to-time.
  - Does not accommodate *dynamism* in topology
Layer 2: Bridges, L2 Switches

- Even reasonable sized LANs cannot tolerate above restrictions

- **Bridges** therefore have “**L2 forwarding tables**,” and use dynamic learning algorithms to build it locally.
  - Even this allows LANs to scale, by limiting broadcasts and collisions to collision domains, and using bridges to interconnect collision domains.

- The learning algorithm is purely local, opportunistic and expects no addressing structure.
  - Hence, bridges often may not have a forwarding entry for a destination address (i.e. **incomplete**)
  - In this case they resort to **flooding** – which may lead to duplicates of packets seen on the wire.

- Bridges coordinate “globally” to build a **spanning tree** so that flooding doesn’t go out of control.
Layer 3: Routers, L3 Switches

- **Routers** have “L3 forwarding tables,” and use a distributed protocol to *coordinate* with other routers to *learn and condense a global view* of the network in a *consistent and complete* manner.

- Routers **NEVER** broadcast or flood if they don’t have a route – they “pass the buck” to another router.
  - The good filtering in routers (i.e. restricting broadcast and flooding activity to be within broadcast domains) allows them to *interconnect broadcast domains*.

- Routers **communicate with other routers**, typically neighbors to collect an abstracted view of the network.
  - In the form of distance vector or link state.
  - Routers use **algorithms** like Dijkstra, Bellman-Ford to compute paths with such abstracted views.
Additions to List of Issues

- Filtering techniques:
  - Learning, routing
- Interconnection devices:
  - Switching, bridging, routing
- Accommodating diversity, dynamism in topologies
Inter-networking
Inter-Networks: *Networks of Networks*

Our goal is to design this black box on the right.

Shivkumar Kalyanaraman

Rensselaer Polytechnic Institute
Virtualization: Virtual vs Physical Link

- **Internet:**
  - *Best-effort*
  - *(no performance guarantees)*
  - *Packet-by-packet*

- **A pt-pt link:**
  - *Always-connected*
  - *Fixed bandwidth*
  - *Fixed delay*
  - *Zero-jitter*
Inter-Networks: Networks of Networks

- What is it?
  - “Connect many disparate physical networks and make them function as a coordinated unit … ” - Douglas Comer
  - Many => scale
  - Disparate => heterogeneity

- Result: Universal connectivity!
  - The inter-network looks like one large switch,
    - Recall a switch also looks like a virtual link to users
  - User interface is sub-network independent
Internetworking involves two fundamental problems: **heterogeneity and scale**

- **Concepts**: [indirection infrastructure]
  - Translation, overlays, address & name resolution, fragmentation: to handle *heterogeneity*
  - Hierarchical addressing, routing, naming, address allocation, congestion control: to handle *scaling*

- Two broad approaches: circuit-switched and packet-switched
Building Block: Indirection
Building Block: Indirection

in·di·rec·tion  *n.*

1. The quality or state of being indirect.

- **Ingredients:**
  - A piece of **state** (eg: ID, address etc) in packet header,
  - A **pointer-style** reference/dereferencing operation

- Indirection requires operations of **binding & unbinding**…
  - Eg: packets, slots, tokens, (routing) tables, servers, switches etc
  - Internet protocols & mechanisms form an huge indirection infrastructure!
The Power of Indirection

- Just like pointers and “referencing” provides great flexibility in programming… (why?)
- Indirection provides great flexibility in distributed system/protocol design!

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

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

- **Synonyms**: Mapping, Binding, Resolution, Delegation, Translation, Referencing, Coupling, Interfacing, (dynamic or flexible) Composition, Relocation …
Indirection is Everywhere!

DNS Server

"foo.org"

foo.org | IP\textsubscript{foo}

(IP\textsubscript{foo},data)

IP\textsubscript{foo}

DNS

Home Agent

(IP\textsubscript{home},data)

(IP\textsubscript{home},data)

Mobile IP

IP\textsubscript{mobile}

(IP\textsubscript{mobile},data)

NAT Box

(IP\textsubscript{nat}:P\textsubscript{nat},data)

(IP\textsubscript{nat}:P\textsubscript{nat},IP\textsubscript{dst}:P\textsubscript{dst})

(IP\textsubscript{dst}:P\textsubscript{dst},data)

NAT

NAT

Internet

(IP\textsubscript{M} \rightarrow IP\textsubscript{R1})

(IP\textsubscript{M} \rightarrow IP\textsubscript{R2})

(IP\textsubscript{R1},data)

(IP\textsubscript{R2},data)

IP Multicast

IP\textsubscript{R1}

IP\textsubscript{R2}
In words…

- Data is mapped to a packet that carries a destination address in its header to facilitate forwarding.
  - Application packets are mapped to IP using port numbers.
  - A forwarding table (and the switch fabric) in a router maps and forwards a packet with a destination address to an output port (“next hop”).
  - Layer 3 routers map packets from one L2 network to another, handling heterogeneity through inter-networking.
  - A series of mappings lead to a packet being delivered end-to-end.
- Scarce public address space is shared by dynamically translating private addresses to public addresses (NAT).
  - DHCP leases scarce IP addresses to hosts, i.e. maps hosts to IP addresses.
In words (contd)…

- DNS resolves one kind of ID to another: names to addresses.
  - Names are human friendly and the name-space is organized/managed differently than IP address space.
  - Similarly ARP dynamically resolves (maps) an L3 address to L2 address.
- A persistent identifier (home address) is mapped to an ephemeral location identifier (care-of-address) in mobile IP (aka late binding)
- SONET header has a pointer that can be de-referenced to find the start of the frame and gives it flexibility to handle synchronization despite jitter on physical links.
- A distributed hash table (DHT) is a generic data-structure that maps a key to a value and enables a wide variety of overlay/p2p indirection functions
Building Block: Virtualization
Virtualization

- The multiplexed shared resource PLUS a level of indirection will seem like a *unshared virtual resource*!
- A virtual resource is a *software entity* that is built out of a (shared or multiplexed) physical resource
- I.e. Multiplexing + indirection = virtualization

**Diagram:**

```
   A   ...   B

Physical Bus
```

```
   A   --   B

Virtual Pt-Pt Link
```

- We can “refer” to the virtual resource as if it were the physical resource.
  - Eg: virtual memory, virtual circuits, virtual services...
- **Connectivity:** a virtualization created by the Internet!
Benefits of Virtualization

- **Changes semantics:**
  - Eg: IPSEC tunnels in VPNs provide a secure channel
  - Eg: TCP provides reliable channel over unreliable networks

- **Hides complexity & heterogeneity:**
  - Eg: The internet looks like a simple (best-effort) virtual link.

- **Location transparency:**
  - Eg: Virtual storage: need not know where your files are stored.
  - Eg: Mobile IP hides the location of mobile node.

- **Performance flexibility:**
  - Eg: Virtual memory can appear to be much larger than physical memory and does not have other artificial constraints.
  - Eg: NAT boxes & DHCP allow efficient sharing of scarce IPv4 address space
  - Eg: virtual circuit can have a variety of QoS features compared to a physical link.

- **Bottom Line:** Like magic, you can transform performance and semantic features, and make entirely new features possible!

- **Paradigm shift:** define attributes of desired virtual resource => determines complexity of the indirection infrastructure!
Building Block: Identifiers (more)
Scalable Forwarding: Structured Addresses

- Address has **structure which aids the forwarding process**.

- Address assignment: nodes on the same network **have the same prefix (network ID)**
  
  - Implemented in IP using “subnet” masking
Flat vs Structured Addresses

- **Flat addresses**: no structure in them to facilitate scalable routing
  - Eg: IEEE 802 LAN addresses
- **Hierarchical addresses**:  
  - Network part (prefix) and host part  
  - Helps identify direct or indirectly connected nodes
Structured Addresses: Implications

- Encoding of network ID => encoding the fact of indirect connectivity into the IP address
- A simple comparison of network ID of destination and current network (broadcast domain) identifies whether the destination is “directly” connected
  - I.e. Reachable through L2 forwarding in one hop
  - Else: Needs to go through multiple L3 hops (indirectly) to reach the destination
- Within L3 forwarding, **hierarchical** organization of **routing** domains helps because routing algorithms have other scalability issues.
Overloading of IDs: Issues

- URLs: often name an object AND also its host location (eg: http://www.ecse.rpi.edu/Homepages/shivkuma)
  - Complicates location transparency (eg: object replication, object relocation) etc

- IP address encodes location (a nested set of network IDs)
  - Mobile IP has to create a new indirection (home address, care-of-address)
  - Places configuration & allocation restrictions on IP address: harder to make auto-configurable

- Newer proposals:
  - Separate (“decouple”) host ID from location ID
  - De-coupling is a common theme in network architecture: more flexibility (along w/ indirection to establish flexible/dynamic coupling)
Congestion Control: Origins of the Problem

- **Cost:** self-descriptive header per-packet, buffering and delays due to statistical multiplexing at switches.

- Need to either reserve resources or dynamically detect and adapt to overload for stability.
If information about $\lambda_i$, $\lambda$ and $\mu$ is known in a central location where control of $\lambda_i$ or $\mu$ can be effected with zero time delays,

- the congestion problem is solved!

- The challenge is to solve it with reasonable tradeoffs without all this information!
The Congestion Problem (Continued)

- Problems:
  - Incomplete information (e.g., loss indication, 1-bit feedback)
  - Distributed solution required
  - Congestion and control/measurement locations different
  - Time-varying, heterogeneous time-delay
Additions to Issues List

- Internetworking problems: heterogeneity, scale.
- **Indirection**: state, pointer-style bind/unbind => flexibility!
- **Virtualization**: multiplexing + indirection, virtual software resource created from physical resources
  - Performance and semantics change
- Circuit Switching vs Packet Switching
- Heterogeneity:
  - Overlay model, Translation, Address Resolution, Fragmentation
- Scale:
  - Structured addresses (more IDs!)
  - Hierarchical routing (filtering!)
  - Naming, addressing (more IDs!)
  - Congestion control (statistical muxing origins)
Summary: Laundry List of Problems

- **Basics**: Direct/indirect connectivity, topologies
- **Link layer issues**: Framing, Error control, Flow control
- **Multiple access & Ethernet**: Cabling, Pkt format, Switching, bridging vs routing
- **Internetworking problems**: Naming, addressing, Resolution, fragmentation, congestion control, traffic management, Reliability, Network Management
- **Fundamental building blocks**: multiplexing, identifiers, randomization, indirection, virtualization, filtering for scale