Networks

Content taken from IPP 2.3.3 and the following:

"Distributed Systems: Principles and Paradigms" by Andrew S. Tanenbaum and Maarten Van Steen (Chapter 4)
Various online sources (including wikipedia.org and openclipart.org)
Overview

- **Topologies** – how a network is arranged (hardware)
- **Routing** – how traffic navigates a network (hardware and software)
- **Protocols** – how machines communicate (software, low-level)
- **IPC paradigms** – how processes communicate (software, high-level)
Part 1

- **Topologies** – how a network is arranged (hardware)
A network topology is an arrangement of components or nodes in a system and their connections (e.g., a graph).
A network topology is an arrangement of components or nodes in a system and their connections (e.g., a graph)

- Ring, star, line, and tree allow simultaneous connections but disallow some pairs of point-to-point communication
- Fully connected and bus allow any-to-any communication but do not scale well
Evaluating topologies

- **Bandwidth**: maximum rate at which a link can transmit data
  - **Throughput**: measured rate of actual data transmission (usually less than bandwidth)
- **Latency**: time between start of send and reception of first data
- **Diameter**: maximum number of hops between nodes on a network
- **Bisection**: divide the network into two sections
  - **Bisection width**: how many communications could happen simultaneously between the two sections?
  - **Bisection bandwidth**: what is the bandwidth between the sections?
- **Important**: how do these metrics scale as you add nodes?
Crossbar switches

- **Switched** interconnects allow multiple simultaneous paths between components
  - *(Graphically, use squares for nodes and circles for switches)*
- A **crossbar switch** uses a matrix of potential connections to create ad-hoc paths between nodes
Omega networks

- Omega network: crossbar of crossbars
  - Each individual switch is a 2-by-2 crossbar
Butterfly networks

- **Multi-stage** network w/ dedicated switching nodes
  - Easy routing based on binary host numbers (0=left, 1=right)

from https://en.wikipedia.org/wiki/Butterfly_network
HPC interconnects

- In an HPC system, the network is called an interconnect
  - Common patterns: switched bus, mesh/torus, hypercube
  - Connected via switches vs. connected directly

Our cluster (switched bus)

Toroidal Mesh
Meshes and tori

- Nodes are connected to several neighbors
  - Non-uniform memory access to non-immediate neighbors

2D Regular Mesh

2D Torus
(or “toroidal mesh”)

3D Torus

https://en.wikipedia.org/wiki/Torus_interconnect
**Hypercubes**

- **Inductive definition:**
  - **0-D hypercube**: a single node
  - **n-D hypercube**: two (n-1)-D hypercubes with connections between corresponding nodes
  - E.g., a 3-D hypercube contains two 2-D hypercubes

https://en.wikipedia.org/wiki/Hypercube
Fat trees

- Hierarchical tree-based topology
  - Links near the root have a higher bandwidth
HPC Interconnect Technologies

- **Ethernet**: 10/100 Mbps – 100 Gbps
  - Early versions used shared-medium coaxial cable
  - Newer versions use twisted pair or fiber optic with hubs or switches
- **InfiniBand (IB)**: 24-300 Gbps w/ 0.5μs latency
  - Packet-based switched fabric (bus, fat tree, or mesh/torus)
  - Very loose API; more formal spec provided by OpenFabrics Alliance
  - Used on many current high-performance clusters
  - Vendors: Mellanox, Intel, and Oracle
- **OmniPath (Intel)** or **Aries / Slingshot (Cray)**
  - Proprietary interconnects for HPC machines
Part 2

- **Routing** – how traffic navigates a network (hardware and software)
Routing

- **Circuit switching**
  - Paths are pre-allocated for an entire session
  - All data is routed along the same path
  - Higher setup costs and fewer simultaneous communications
  - Constant latency and throughput

- **Packet switching**
  - Break data into independent, addressed packets
  - Packets are routed independently
  - No setup costs and no restriction on simultaneous communications
  - Resiliency to network failures and changing conditions
  - Variable (and often unpredictable) latency and throughput
Routing

Unicast (one-to-one)

Anycast (one-to-nearest)

Geocast (one-to-proximate)

Multicast (one-to-many)

Broadcast (one-to-all)

from https://en.wikipedia.org/wiki/Routing#Delivery_schemes
Overlays

- **Overlay**: a network built on top of another network
  - **IP multicast**: technique for sending data to multiple recipients over an IP network using UDP
    - Group addressing (IGMP)
    - Tree-based distribution
  - Distributed hash tables (e.g., Chord)
  - **XMPP** – Jabber/Gtalk chat protocol
  - **Tor** network
Tor network

- Overlay network for anonymity
- **Onion routing**: multiple layers of obfuscation
  - At each layer, data is encrypted and sent to a random Tor relay
  - Sequence of relays form a virtual circuit that is difficult to trace
  - No single relay connects the source and destination directly
Part 3

- **Protocols** – how machines communicate (software, low-level)
Networking principles

• Distributed system components are often unreliable

• How do we build a **reliable** network using **unreliable** hardware and software?
  - **Abstraction** helps by hiding details where possible
  - **Protocols** define well-structured communication patterns
  - **Layered / stacked** protocols build on each other
  - Each layer adds **metadata** to help solve a specific problem

• Another guiding principle: the **end-to-end principle**
  - **Application-specific functions ought to reside in the end hosts of a network rather than in intermediary nodes whenever possible.**

For more info:
QoS concerns

- Quality of Service (QoS) guarantees
  - Possible reasons to violate end-to-end principle
  - Minimum required bit rate (bandwidth)
  - Maximum delay to set up a session
  - Maximum end-to-end delay (latency)
  - Maximum delay variance (jitter)
  - Maximum round-trip delay
  - Possibility of expedited forwarding
  - Synchronization mechanisms
  - Examples: MPEG-2, HLS
Networking protocols

- **Routing**: choosing a path through a network
- **Datagram**: self-contained, encapsulated package of data and metadata capable of being routed
  - Also called a **frame**: (layer 2), a **packet**: (layer 3), or a **segment**: (layer 4)
- **Protocol**: rules for exchanging data (often using datagrams)
- **Checksums**: data integrity verification mechanism

---

IPv4 header

(from https://www.tutorialspoint.com/ipv4/ipv4_packet_structure.htm)
Protocol design issues

- **Connectionless vs. connection-oriented**
  - Is there a setup/teardown procedure required for communication?
  - No setup costs vs. faster speed after connection
- **Synchronous vs. asynchronous**
  - Does the sender block after sending?
    - E.g., MPI_Ssend vs. MPI_Isend
  - Easier to debug and verify vs. faster communication
- **Persistent vs. transient communication**
  - Are messages stored by the middleware?
  - Guaranteed delivery vs. simplicity of middleware
1) **Physical**: Transmission of raw bits over a physical medium (Ethernet, 802.11)
2) **Data link**: Reliable transmission of frames between two nodes (FC, 802.11)
3) **Network**: Structured transmission on a multi-node network (IP, ICMP)
4) **Transport**: Reliable transmission on a multi-node network (TCP, UDP)
5) **Session**: Managed communication sessions (RPC, NFS)
6) **Presentation**: Encoding and conversion of data (HTML, XML, JSON)
7) **Application**: Application-level abstractions (FTP, HTTP, SSH, MPI)
Part 4

• **IPC paradigms** – how processes communicate (software, high-level)
IPC paradigms

- Inter-process communication (IPC)
  - Message-passing (explicit)
    - Symmetric (SPMD) vs. asymmetric (differentiated hosts)
  - Remote procedure calls (implicit)
    - Synchronous vs. asynchronous
Berkeley / POSIX Sockets

- API for inter-process communication
  - Originally designed for BSD
  - Later evolved into a POSIX standard
  - Often used for low-level TCP and UDP communication
  - Hosts identified by address (usually IP) and port number
  - Passes "messages" (packets) between hosts
  - Can use Unix pipes if both endpoints are on a single host
Socket primitives

- **Server**
  - **Socket**: Create a new endpoint
  - **Bind**: Attach a local address to a socket
  - **Listen**: Announce readiness for connections
  - **Accept**: Block until a request arrives

- **Client**
  - **Connect**: Attempt to establish a connection

- **Server & client**
  - **Write**: Send data over a connection
  - **Read**: Receive data over a connection
  - **Close**: Destroy a connection

*Figure 4-15. Connection-oriented communication pattern using sockets.*
MPI (Message Passing Interface)

- MPI_Send
- MPI_Recv
- MPI_Bcast
- MPI_Scatter
- MPI_Gather
- MPI_Allgather
- MPI_Reduce
- MPI_Allreduce
- MPI_Alltoall

from https://computing.llnl.gov/tutorials/parallel_comp/
Remote Procedure Call (RPC)

- **Key idea:** transparency
  - It should look like the procedure call is happening locally
  - Similar in spirit to PGAS remote memory accesses
  - Implement server / client stubs to handle the call

- **Parameter** marshalling
  - Preparing parameters for transmission over a network

---

**Figure 4-6.** Principle of RPC between a client and server program.

**Figure 4-7.** The steps involved in doing a remote computation through RPC.
Asynchronous RPC

Figure 4-10. (a) The interaction between client and server in a traditional RPC. (b) The interaction using asynchronous RPC.

Figure 4-11. A client and server interacting through two asynchronous RPCs.
Summary

- **Topologies** – how a network is arranged (hardware)
- **Routing** – how traffic navigates a network (hardware and software)
- **Protocols** – how machines communicate (software, low-level)
- **IPC paradigms** – how processes communicate (software, high-level)

Next time: how do we **identify** hosts on a network? (e.g., what is a host’s name)