CS162
Operating Systems and Systems Programming
Lecture 15
Key-Value Storage, Network Protocols

October 28, 2013
Anthony D. Joseph and John Canny
http://inst.eecs.berkeley.edu/~cs162

Goals for Today

- Key-Value Storage
  - Interface and Examples
  - Distributed Hash Tables
  - Challenges and Solutions
- Networking
  - What is a protocol?
  - Layering

Many slides generated from Ion Stoica's lecture notes by Vern Paxson, and Scott Shenker.

Key-Value Storage

- Interface
  - **put**(key, value); // insert/write "value" associated with "key"
  - value = **get**(key); // get/read data associated with "key"
- Abstraction used to implement
  - A simpler and more scalable "database"
  - Content-addressable network storage (CANs)
- Can handle large volumes of data, e.g., PBs
  - Need to distribute data over hundreds, even thousands of machines
  - Designed to be faster with lower overhead (additional storage) than conventional DBMSes.

Database Attributes

Databases require 4 properties:

- **Atomicity:** When an update happens, it is "all or nothing"
- **Consistency:** The state of various tables much be consistent (relations, constraints) at all times.
- **Isolation:** Concurrent execution of transactions produces the same result as if they occurred sequentially.
- **Durability:** Once committed, the results of a transaction persist against various problems like power failure etc.

These properties ensure that data is protected even with complex updates and system failures.

CAP Theorem (Brewer, Gilbert, Lynch)

But we also have the CAP theorem for distributed systems:

Consistency: All nodes have the same view of the data

Availability: Every request receives a response of success or failure.

Partition Tolerance: System continues even with loss of messages or part of the data nodes.

The theorem states that **you cannot achieve all three at once**.

Many systems therefore strive to implement two of the three properties. Key-Value stores often do this.

KV-stores and Relational Tables

KV-stores seem very simple indeed. They can be viewed as two-column (key, value) tables with a single key column.

But they can be used to implement more complicated relational tables:

State	ID	Population	Area	Senator_1
Alabama	1	4,822,023	52,419	Sessions
Alaska	2	731,449	663,267	Begich
Arizona	3	6,553,255	113,998	Boozman
Arkansas	4	2,949,131	53,178	Flake
California	5	38,041,430	163,695	Boxer
Colorado	6	5,187,582	104,094	Bennet
…	…			

Index

KV-stores and Relational Tables

The KV-version of the previous table includes one table indexed by the actual key, and others by an ID.

State	ID
Alabama	1
Alaska	2
Arizona	3
Arkansas	4
California	5
Colorado	6
…	…

ID	Population
1	4,822,023
2	731,449
3	6,553,255
4	2,949,131
5	38,041,430
6	5,187,582
…	…

ID	Area
1	52,419
2	663,267
3	113,998
4	53,178
5	163,695
6	104,094
…	…

ID	Senator_1
1	Sessions
2	Begich
3	Boozman
4	Flake
5	Boxer
6	Bennet
…	…

KV-stores and Relational Tables

You can add indices with new KV-tables:

Thus KV-tables are used for **column-based storage**, as opposed to row-based storage typical in older DBMS.

State	ID
Alabama	1
Alaska	2
Arizona	3
Arkansas	4
California	5
Colorado	6
…	…

Index

ID	Population
1	4,822,023
2	731,449
3	6,553,255
4	2,949,131
5	38,041,430
6	5,187,582
…	…

…

Senator_1	ID
Sessions	1
Begich	2
Boozman	3
Flake	4
Boxer	5
Bennet	6
…	…

Index_2

OR: the value field can contain complex data (next page):

Key-Values: Examples

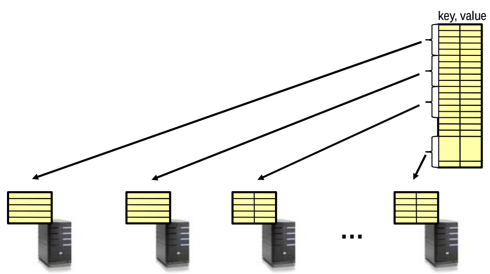
- Amazon:
  - Key: customerID
  - Value: customer profile (e.g., buying history, credit card, ..)
- Facebook, Twitter:
  - Key: UserID
  - Value: user profile (e.g., posting history, photos, friends, …)
- iCloud/iTunes:
  - Key: Movie/song name
  - Value: Movie, Song
- Distributed file systems
  - Key: Block ID
  - Value: Block

System Examples

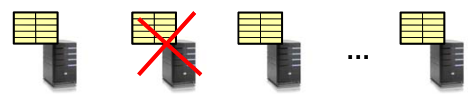
- **Google File System, Hadoop Dist. File Systems (HDFS)**
- **Amazon**
  - Dynamo: internal key value store used to power Amazon.com (shopping cart)
  - Simple Storage System (S3)
- **BigTable/HBase/Hypertable:** distributed, scalable data storage
- **Cassandra**: "distributed data management system" (Facebook)
- **Memcached:** in-memory key-value store for small chunks of arbitrary data (strings, objects)
- **eDonkey/eMule:** peer-to-peer sharing system

Key-Value Store

- Also called a Distributed Hash Table (DHT)
- Main idea: partition set of key-values across many machines



Challenges



- **Fault Tolerance:** handle machine failures without losing data and without degradation in performance
- **Scalability:**
  - Need to scale to thousands of machines
  - Need to allow easy addition of new machines
- **Consistency:** maintain data consistency in face of node failures and message losses
- **Heterogeneity** (if deployed as peer-to-peer systems):
  - Latency: 1ms to 1000ms
  - Bandwidth: 32Kb/s to several GB/s

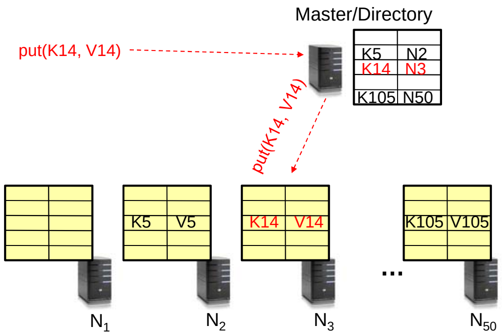
Key Questions

- put(key, value): where do you store a new (key, value) tuple?
- get(key): where is the value associated with a given "key" stored?

- And, do the above while providing
  - Fault Tolerance
  - Scalability
  - Consistency

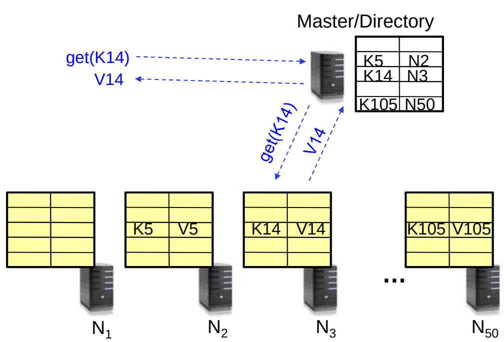
Directory-Based Architecture

- Have a node maintain the mapping between **keys** and the **machines (nodes)** that store the **values** associated with the **keys**



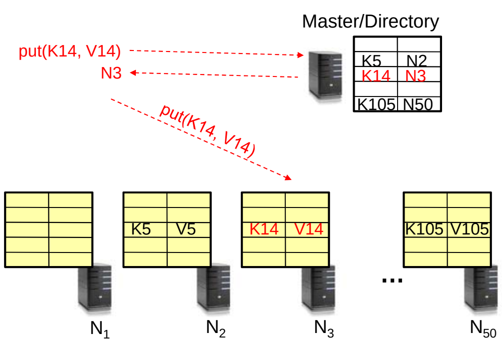
Directory-Based Architecture

- Have a node maintain the mapping between **keys** and the **machines (nodes)** that store the **values** associated with the **keys**



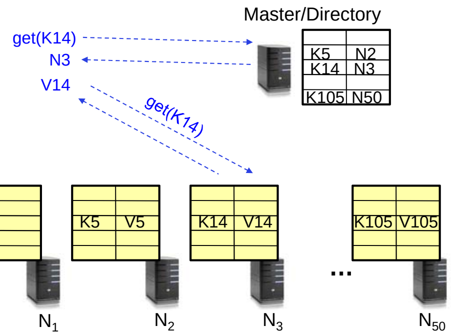
Directory-Based Architecture

- Having the master relay the requests → **recursive query**
- Another method: **iterative query** (this slide)
  - Return node to requester and let requester contact node

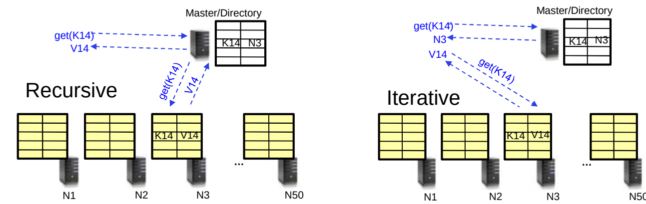


Directory-Based Architecture

- Having the master relay the requests → **recursive query**
- Another method: **iterative query**
  - Return node to requester and let requester contact node



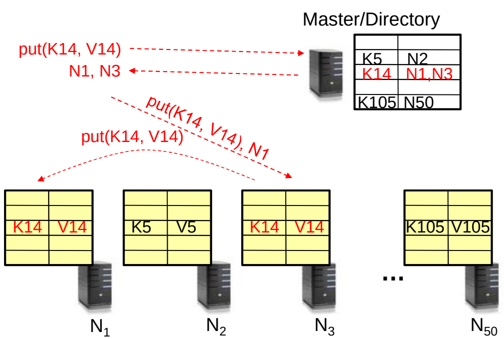
Discussion: Iterative vs. Recursive Query



- Recursive Query:
  - Advantages:
    - » Faster (latency), as typically master/directory closer to nodes
    - » Easier to maintain consistency, as master/directory can serialize puts()/gets()
  - Disadvantages: scalability bottleneck, as all "Values" go through master/directory
- Iterative Query
  - Advantages: more scalable
  - Disadvantages: slower (latency), harder to enforce data consistency

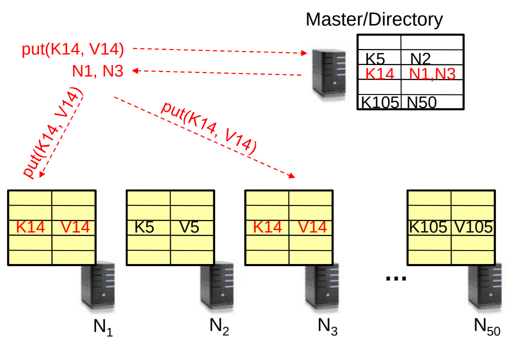
Fault Tolerance

- Replicate value on several nodes
- Usually, place replicas on different racks in a datacenter to guard against rack failures (recursive version)



Fault Tolerance

- Again, we can have
  - **Recursive** replication (previous slide)
  - **Iterative** replication (this slide)



Scalability

- Storage: use more nodes
- Request Throughput:
  - Can serve requests from all nodes on which a value is stored in parallel
  - Large "values" can be broken into blocks (HDFS files are broken up this way)
  - Master can replicate a popular value on more nodes
- Master/directory scalability:
  - Replicate it
  - Partition it, so different keys are served by different masters/directories
    - » How do you partition? (p2p DHDT, end of semester)

Scalability: Load Balancing

- Directory keeps track of the storage availability at each node
  - Preferentially insert new values on nodes with more storage available
- What happens when a new node is added?
  - Cannot insert only new values on new node. Why?
  - Move values from the heavy loaded nodes to the new node
- What happens when a node fails?
  - Need to replicate values from failed node to other nodes

Replication Challenges

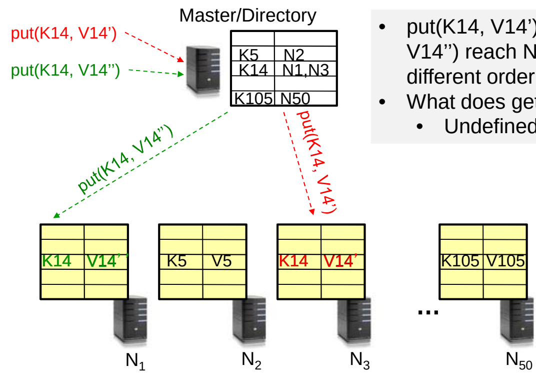
- Need to make sure that a value is replicated correctly
- How do you know a value has been replicated on every node?
  - Wait for acknowledgements from every node
- What happens if a node fails during replication?
  - Pick another node and try again
- What happens if a node is slow?
  - Slow down the entire put()? Pick another node?
- In general, with multiple replicas
  - Slow puts and fast gets

Consistency

- How close does a distributed system emulate a single machine in terms of read and write semantics?
- **Q:** Assume **put(K14, V14')** and **put(K14, V14'')** are concurrent, what value ends up being stored?
- **A:** assuming **put()** is atomic, then either **V14'** or **V14''**, right?
- **Q:** Assume a client calls **put(K14, V14)** and then **get(K14)**, what is the result returned by **get()**?
- **A:** It should be V14, right?
- Above semantics, not trivial to achieve in distributed systems

Concurrent Writes (Updates)

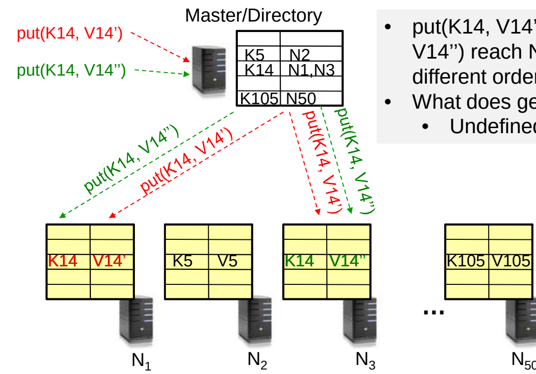
- If concurrent updates (i.e., puts to same key) may need to make sure that updates happen in the same order



- put(K14, V14') and put(K14, V14'') reach N1 and N3 in different order
- What does get(K14) return?
  - Undefined!

Concurrent Writes (Updates)

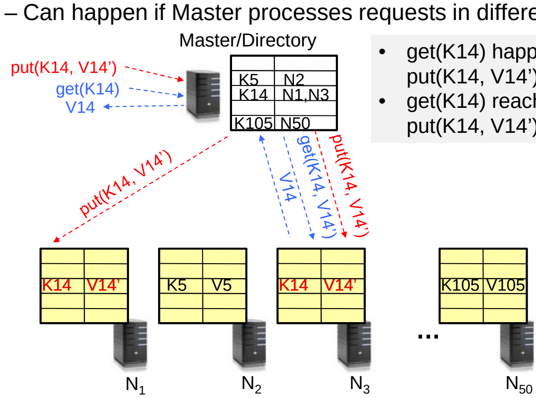
- If concurrent updates (i.e., puts to same key) may need to make sure that updates happen in the same order



- put(K14, V14') and put(K14, V14'') reach N1 and N3 in different order
- What does get(K14) return?
  - Undefined!

Read after Write

- Read not guaranteed to return value of latest write
  - Can happen if Master processes requests in different threads



- get(K14) happens right after put(K14, V14')
- get(K14) reaches N3 before put(K14, V14')!

Consistency (cont'd)

- Large variety of consistency models:
  - Atomic consistency (linearizability): reads/writes (gets/puts) to replicas appear as if there was a single underlying replica (single system image)
    - » Think "one updated at a time"
    - » Transactions (later in the class)
  - Eventual consistency: given enough time all updates will propagate through the system
    - » One of the weakest forms of consistency; used by many systems in practice
  - And many others: causal consistency, sequential consistency, strong consistency, …

Strong Consistency

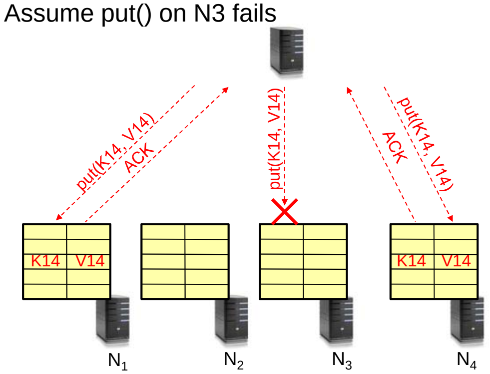
- Assume Master serializes all operations
- Challenge: master becomes a bottleneck
  – Not addressed here
- Still want to improve performance of reads/writes → quorum consensus

Quorum Consensus

- Improve **put()** and **get()** operation performance
- Define a replica set of size N
- **put()** waits for acks from at least W replicas
- **get()** waits for responses from at least R replicas
- W+R > N
- Why does it work?
  – There is at least one node that contains the update

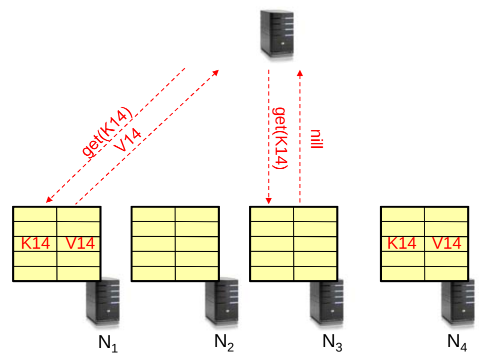
Quorum Consensus Example

- N=3, W=2, R=2
- Replica set for K14: {N1, N3, N4}
- Assume put() on N3 fails



Quorum Consensus Example

- Now, issuing get() to any two nodes out of three will return the answer



Summary: Key-Value Store

- Very large scale storage systems
- Two operations
  – put(key, value)
  – value = get(key)
- Challenges
  – Fault Tolerance → replication
  – Scalability → serve get()'s in parallel; replicate/cache hot tuples
  – Consistency → quorum consensus to improve put/get performance

Administrivia

- Project 2 code due 11:59pm on Thursday 10/31.
- Project 2 group evals due 11:59pm on Friday 11/1.

Watch slip days! Remember there are only 4 of these, after that there is an automatic (non-negotiable) 10% deduction for each day late. Projects 3 and 4 are challenging!

5min Break

Quiz 15.1: Key-Value Store

- Q1: True _ False _ Distributed Key-Value stores should always be Consistent, Available and Partition-Tolerant (CAP)
- Q2: True _ False _ On a single node, a key-value store can be implemented by a hash-table
- Q3: True _ False _ A Master can be a bottleneck point for a key-value store
- Q4: True _ False _ Iterative PUTs achieve lower throughput than recursive PUTs on a loaded system
- Q5: True _ False _ With quorum consensus, we can improve read performance at expense of write performance

Quiz 15.1: Key-Value Store

- Q1: True _ False _x Distributed Key-Value stores should always be Consistent, Available and Partition-Tolerant (CAP)
- Q2: True x False _ On a single node, a key-value store can be implemented by a hash-table
- Q3: True x False _ A Master can be a bottleneck point for a key-value store
- Q4: True _ False _x Iterative PUTs achieve lower throughput than recursive PUTs on a loaded system
- Q5: True _x False _ With quorum consensus, we can improve read performance at expense of write performance

What Is A Protocol?

- A protocol is an agreement on how to communicate
- Includes
  - Syntax: how a communication is specified & structured
    - » Format, order messages are sent and received
  - Semantics: what a communication means
    - » Actions taken when transmitting, receiving, or when a timer expires

Examples of Protocols in Human Interactions

- Telephone
  1. (Pick up / open up the phone)
  2. Listen for a dial tone / see that you have service
  3. Dial
  4. Should hear ringing …
  5. Callee: "Hello?"
  6. Caller: "Hi, it's John…."
     Or: "Hi, it's me" (← what's *that* about?)
  7. Caller: "Hey, do you think … blah blah blah …" **pause**
  8. Callee: "Yeah, blah blah blah …" **pause**
  9. Caller: Bye
  10. Callee: Bye
  11. Hang up

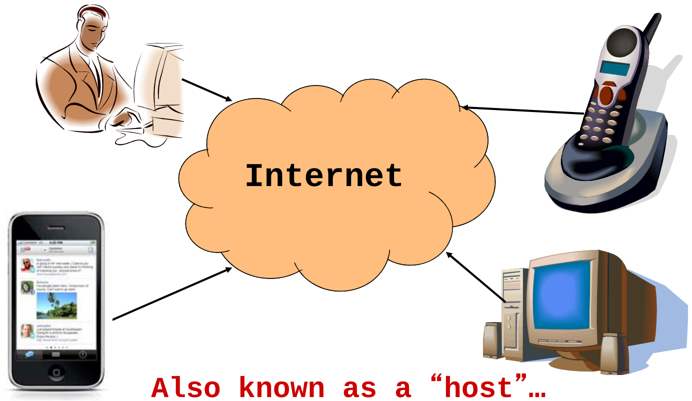
Examples of Protocols in Human Interactions

Asking a question

1. Raise your hand
2. Wait to be called on

3. Or: wait for speaker to **pause** and vocalize

End System: Computer on the 'Net

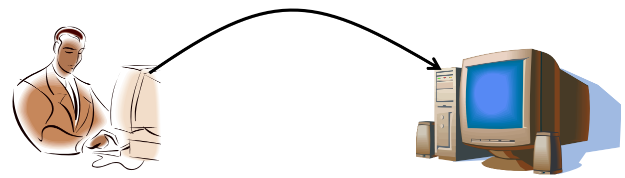


Also known as a "host"…

Clients and Servers

- Client program
  - Running on end host
  - Requests service
  - E.g., Web browser

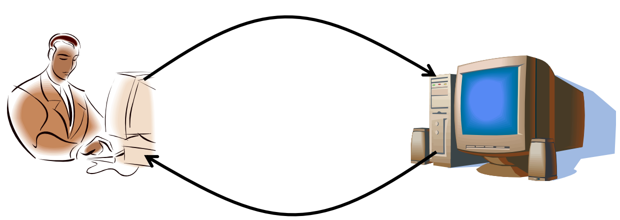
GET /index.html



Clients and Servers

- Client program
  - Running on end host
  - Requests service
  - E.g., Web browser
- Server program
  - Running on end host
  - Provides service
  - E.g., Web server

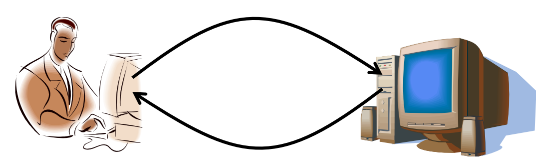
GET /index.html



"Site under construction"

Client-Server Communication

- Client "sometimes on"
  - Initiates a request to the server when interested
  - E.g., Web browser on your laptop or cell phone
  - Doesn't communicate directly with other clients
  - Needs to know the server's address
- Server is "always on"
  - Services requests from many client hosts
  - E.g., Web server for the *www.cnn.com* Web site
  - Doesn't initiate contact with the clients
  - Needs a fixed, well-known address



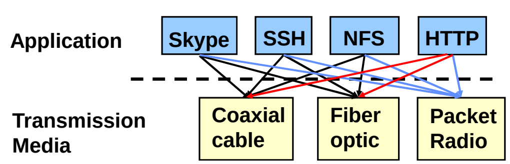
Peer-to-Peer Communication

- No always-on server at the center of it all
  - Hosts can come and go, and change addresses
  - Hosts may have a different address each time
- Example: peer-to-peer file sharing (e.g., BitTorrent)
  - Any host can request files, send files, query to find where a file is located, respond to queries, and forward queries
  - Scalability by harnessing millions of peers
  - Each peer acting as both a client and server

The Problem

- Many different applications
  - email, web, P2P, etc.
- Many different network styles and technologies
  - Wireless vs. wired vs. optical, etc.
- How do we organize this mess?

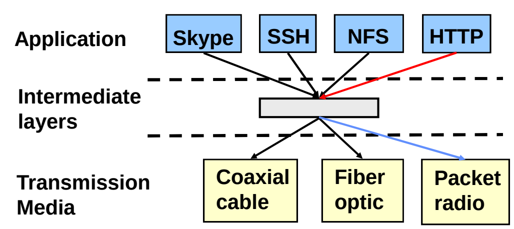
The Problem (cont'd)



- Re-implement every application for every technology?
- No! But how does the Internet design avoid this?

Solution: Intermediate Layers

- Introduce intermediate layers that provide set of abstractions for various network functionality & technologies
  - A new app/media implemented only once
  - Variation on "add another level of indirection"



Software System Modularity

Partition system into modules & abstractions:

- Well-defined interfaces give flexibility
  - ***Hides*** implementation - thus, it can be freely changed
  - Extend functionality of system by adding new modules
- E.g., libraries encapsulating set of functionality
- E.g., programming language + compiler abstracts away not only how the particular CPU works …
  - … but also the basic computational model
- Well-defined interfaces hide information
  - Present high-level abstractions
  - **But can impair performance**

Network System Modularity

Like software modularity, but:

- Implementation distributed across many machines (routers and hosts)
- Must decide:
  - How to break system into modules:
    - » **Layering**
  - What functionality does each module implement:
    - » **End-to-End Principle:** don't put it in the network if you can do it in the endpoints.
- We will address these choices next lecture

Layering: A Modular Approach

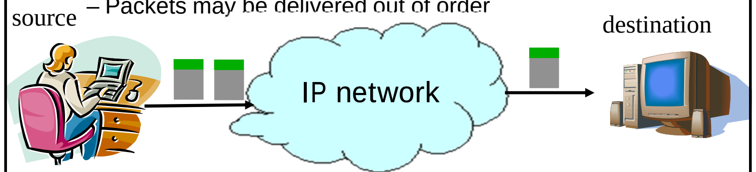
- Partition the system
  - Each layer solely relies on services from layer below
  - Each layer solely exports services to layer above
- Interface between layers defines interaction
  - Hides implementation details
  - Layers can change without disturbing other layers

Protocol Standardization

- Ensure communicating hosts speak the same protocol
  - Standardization to enable multiple implementations
  - Or, the same folks have to write all the software
- Standardization: Internet Engineering Task Force
  - Based on working groups that focus on specific issues
  - Produces "Request For Comments" (RFCs)
    - » Promoted to standards via rough consensus and running code
  - IETF Web site is ***http://www.ietf.org/***
  - RFCs archived at ***http://www.rfc-editor.org/***
- De facto standards: same folks writing the code
  - P2P file sharing, Skype, <your protocol here>…

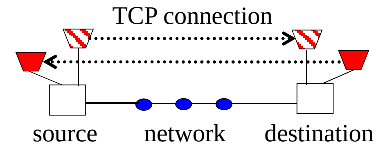
Example: The Internet Protocol (IP): "Best-Effort" Packet Delivery

- Datagram packet switching
  - Send data in packets
  - Header with source & destination address
- Service it provides:
  - Packet arrives quickly (if it does)
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order



Example: Transmission Control Protocol (TCP)

- Communication service
  - Ordered, reliable byte stream
  - Simultaneous transmission in both directions
- Key mechanisms at end hosts
  - Retransmit lost and corrupted packets
  - Discard duplicate packets and put packets in order
  - Flow control to avoid overloading the receiver buffer
  - Congestion control to adapt sending rate to network load



Quiz 15.2: Protocols

- Q1: True _ False _ Protocols specify the syntax and semantics of communication
- Q2: True _ False _ Protocols specify the implementation
- Q3: True _ False _ Layering helps to improve application performance
- Q4: True _ False _ "Best Effort" packet delivery ensures that packets are delivered in order
- Q5: True _ False _ In p2p systems a node is both a client and a server
- Q6: True _ False _ TCP ensures that each packet is delivered within a predefined amount of time

Quiz 15.2: Protocols

- Q1: True x False _ Protocols specify the syntax and semantics of communication
- Q2: True _ False x Protocols specify the implementation
- Q3: True _ False x Layering helps to improve application performance
- Q4: True _ False x "Best Effort" packet delivery ensures that packets are delivered in order
- Q5: True x False _ In p2p systems a node is both a client and a server
- Q6: True _ False x TCP ensures that each packet is delivered within a predefined amount of time

Summary

- Roles of
  - Standardization
  - Clients, servers, peer-to-peer
- Layered architecture as a powerful means for organizing complex networks
  - Though layering has its drawbacks too
- Next lecture
  - Layering
  - End-to-end arguments