

Announcements

- Hopefully, you're done with project 3?
- If you need to know your slip day status, drop me an email.

Today's Menu:

- Quick Review: Protocols and Performance
- TCP
- Distributed Systems: Two-Phase Commit

Protocol Stacks and Performance Measures

- An example of a protocol stack: the ISO/OSI Seven Layers of Networking
 - o Physical Layer
 - Ex: this layer defines the size of Ethernet coaxial cable, the type of BNC connector used, and the termination method.
 - o Data Link Layer
 - Ex: this layer defines the framing, addressing and checksumming of Ethernet packets.
 - o Network Layer
 - Ex: this layer defines the addressing and routing structure of the Internet.
 - o Transport Layer
 - Ex: this layer defines if and how retransmissions will be used to ensure data delivery.
 - o Session Layer
 - Ex: this layer describes how request and reply packets are paired in a remote procedure call.
 - o Presentation Layer
 - Ex: this layer describes how floating point numbers can be exchanged between hosts with different math formats.
 - o Application Layer
 - Ex: this layer would interact with users, i.e. a web browser.
- A few common examples for each layer.
 - o Application: Telnet, FTP, HTTP, etc.
 - o Transport: TCP, UDP, etc.
 - o Network: IP, ICMP, IGMP, etc.
 - o Link: Ethernet, PPP, etc.
- What are the components of Networking Performance?
 - o Overhead - CPU time to put packet on wire.
 - o Latency - How long to send one byte packet

- Throughput - maximum bytes per second
- **QUESTION:** How long would it take to transfer 1 MB across country from Berkeley to Boston where the latency is 15ms and the bandwidth is 10Mbit/s?
- **ANSWER:**
 - $10\text{Mbit/s} * 1 \text{ Byte}/8 \text{ bit} = 1.25\text{MB/s}$
 - $0.015\text{s} + 1\text{MB}/1.25\text{MB/s} = 0.815\text{s}$
- Bandwidth*Delay Product
 - Bandwidth * Delay = number of bytes in flight to fill path
 - TCP needs a receive window (rwin) equal to or greater than the BW * Delay product to achieve maximum throughput

Transmission Control Protocol - TCP

- IP by itself not too useful because it is UNRELIABLE.
 - Applications need some kind of guarantee their data is going to get to the destination.
 - TCP provides reliable byte-streams over IP
- TCP provides a CONNECTION-ORIENTED service
 - Two nodes must first perform a HANDSHAKE before they can communicate
 - The protocol maintains state for the connection.
- TCP provides RELIABILITY
 - Messages are broken into segments which are then encapsulated in IP.
 - Each segment has a checksum associated with it to detect errors.
 - When receiving a segment, a node sends an acknowledgement (ACK) to indicate that it was received OK.
 - Nodes perform RETRANSMISSION
 - If an ACK is not received after a certain amount of time, the node retransmits segment.
 - Since IP can duplicate and reorder packets, the TCP layer must deal with this.
- TCP also provides FLOW CONTROL
 - Each end of the connection has a finite amount of buffer space
 - A node cannot place more packets on the network if the receiver's buffer space is full.
- A TCP segment looks like this:

16-bit source port
 16-bit destination port
 32-bit sequence number
 32-bit ack number
 [various flags]
 16-bit window size -- used for flow control

[other flags]
data

- Each end of the connection is identified by a 16-bit port number.
 - o Multiple TCP connections can exist at once between two hosts.
 - Note that the TCP header does not include any host addresses!
 - IP handles this.
- Each TCP packet contains a sequence number to order packets (remember that IP can drop and reorder packets).
 - o The sequence number indicates the BYTE NUMBER in the stream.
 - o Sequence number typically starts at random
- Each packet contains a 32-byte ACK field.
 - o Tells other end of the connection how many bytes have been successfully received.
 - Does this via indicating the next sequence number that the node expects to receive.
 - ACKs can be "piggybacked" onto data flowing between two nodes, saving bandwidth.
- Establishing a TCP connection: the three-way handshake
 - o Special flag set in initial packet called 'SYN'
 - o The response is a packet that has the SYN and ACK flags set.
 - o The initial requester then sends an ACK

EXAMPLE: Assume we have two nodes, A and B, communicating. After a connection has been established, a packet trace of their traffic might look like the following:

```
A -> B sequence 48, 100 bytes of data
B -> A sequence 1006, ack 148, 3 bytes of data
A -> B sequence 148, ack 1009, 100 bytes of data
B -> A sequence 1009, ack 248, 3 bytes of data
```

Of course, A might send multiple packets to B at a time:

```
A -> B sequence 48, 25 bytes of data
A -> B sequence 73, 25 bytes of data
A -> B sequence 98, 25 bytes of data
A -> B sequence 123, 25 bytes of data
```

Say that the second of these 4 packets is dropped. Then the only ACK that B can send is

```
B -> A sequence 1006, ack 73, 3 bytes of data
```

Both sides maintain a retransmission timer. A will time out waiting for the ACKs from B for all 4 packets (it wants to see an ACK containing the value 148), and will retransmit the last 3 packets again.

- Tearing down a TCP connection
 - o Again, special flag is set, FIN
 - o From there, timeouts set waiting for ack
 - Either FIN + ACK will be received, or we will time out.
 - Either way, connection will be closed.

Distributed Systems and Two-Phase Locking

- Event ordering is a hard problem!

QUESTION: Given events A and B in a distributed system, can you tell whether A happened before B?

ANSWER: In general: NO! Why not? In general networks can drop/reorder/delay messages indefinitely, making absolute notion of time infeasible. (“General’s Paradox”)

- In the distributed-systems nomenclature we talk about "processes" rather than "machines".
 - o Each "process" is a distributed entity.
 - o Processes can only interact by sending messages to one another.
 - o The network can do all kinds of nasty things to messages, like reorder and delay them.
- In complex distributed systems, network links and machines can fail independently of one another.
 - o Need to be able deal with failures in the system that can cause state to become inconsistent.
- The basic building-block of solutions to this sort of distributed problem is a TRANSACTION
 - o Transaction is an action that may affect state in multiple distributed resources, but is guaranteed to happen just ONCE and leave the system in a consistent state.
 - This definition should ring a bell. . .
 - o Getting transactions right is hard: Larry Ellison isn't a multibillionaire for nothing!
- In the database world it is often stated that transactions should have "ACID semantics". ACID stands for:
 - o Atomicity:
 - The transaction should appear to be a single, indivisible action
 - o Consistency:
 - The transaction leaves the distributed system in a consistent state, with all processes agreeing on what the state is.
 - o Isolation:
 - Execution of the transaction is isolated from that of others ; that is, transactions do not interfere with one another.
 - o Durability:

- If the transaction completes successfully, the new state is durable (its effects persist - system crashes etc. do not cause the system to revert to the old state)
- Getting ACID semantics isn't always easy.
 - Multiple levels of locking and logs.
 - Some systems have explored alternatives which are easier to implement and more lax on the restrictions.
- Two Phase Commit (2PC)\
 - Every process participating in a transaction maintains a persistent log (e.g., on disk)
 - This will retain state even if the process crashes.
 - The processes use this log to store information on the state of the transaction
 - if they crash they can recover the state.
- Every 2PC has a "coordinator" which is responsible for driving the protocol. The protocol is as follows:
 - 1) The coordinator C writes a log entry "prepare T" and commits the log to disk.
 - 2) C sends a "prepare T" message to every participant in the transaction.
 - 3) Each process P decides whether it can commit the transaction T. A node may not be able to commit a transaction if the state of the transaction is out of whack with its view of reality, or if the request is disallowed (i.e. requesting to withdraw more money than is in the bank account).
 - 4) If P can commit, it writes "ready T" to its log, and sends an "OK T" message to C.

If not, it writes "no T" to its log and sends an "abort T" message to C.
 - 5) If C receives an "OK T" message from every process P, it writes "commit T" to its log, and sends a "commit T" message to every process P.

If C receives an "abort T" from *any* process P, or times out, it writes "abort T" to its log and sends an "abort T" message to every process P.
 - 6) When P receives "commit T" or "abort T" it performs the commit/abort action and writes that message to its log.
- In 2PC, processes can crash at any time and the entire transaction will either be committed or aborted at every process eventually.
 - If a process crashes it can
 - read its log

- find out what it was doing before it crashed
 - find out from the coordinator what the result of the transaction was.
- If P recovers and sees the message "commit T" in its log. . .
 - P knows that the transaction was completed and performs a local "redo" of whatever the transaction was.
- If P recovers and sees "abort T"
 - P knows that the transaction was aborted and does whatever is needed to locally "undo" the transaction.
- If P recovers and sees "ready T"
 - P made a promise to commit the transaction locally - but, some other process may have caused an abort.
 - P asks C (or some other node that participated in the transaction) the state of the transaction and either commits or aborts based on C's answer.
- If P recovers and sees no log message for T
 - P must have crashed before responding to the prepare message from C.
 - P aborts the transaction locally (assuming that C will time out the transaction causing an abort).
- What happens if the coordinator crashes?
 - C can check with participating processes to tell if the transaction committed or aborted.
 - In some cases however, processes have to wait for C to recover before the state of the transaction can be decided -- this is called the **BLOCKING PROBLEM**.
- What is network links fail?
 - This makes it appear as though those site cut off from the rest of the world have failed, so the 2PC recovery mechanisms kick in and (eventually) cause the transaction to reach a consistent state.