

CS162 – Section # 12, 04/22/2003

Rodrigo Fonseca

Announcements

- Proj. 3 Code due Thursday
 - Next Tuesday: Midterm 2 !!
 - Study from Lecture Notes, Book (see relevant sections in Schedule), Section notes (4 TAs' to choose from), Additional reading, Project Concepts
 - 1 Pimentel, 5pm – 7pm
 - No Section! I will hold Office Hours from 11-12
 - Midterm review this Wednesday, during lecture
 - Nachos Phase 4 Initial Design: due Monday, 5/5: Networking (TCP like protocol)
 - Today: Networking. Reliable Transfer. Good for studying, good for Nachos ph. 4, may be not-so-new if you took EE122!
-

Quick Question:

Suppose you had a directory that is accessible to the entire class. Now you want this to be accessible only to yourself. Is this easy with ACLs? Is this easy with Capabilities? Why? How do you do it?

What you want to do in this case is basically revoke privileges that are currently granted. You want to change all (but one) values of one column of the Access Control Matrix, namely, that for the directory. Since Access Control Lists are representations of columns of this matrix, it is natural that it is going to be easier to perform this task if the system employs ACLs. So, you just have to go through the list, and erase the lines that refer to your classmates.

With Capability Lists it is much harder: they are not stored with the object, but rather with the users that have the access granted. So, you have to go through each user's capability, and verify if the user has that permission, and erase it. Since this approach may not be possible if, for example, the system is distributed, other solutions may have to be employed: you can have old capabilities expire (and renew the ones to keep), or create revocation lists.

Network Protocols

Problem: reliable communication over unreliable channel

IP

routes packets from one machine to another.
connects multiple networks
Best Effort: unreliable, unordered

Packets can get:

Lost
Reordered
Damaged

Loss is assumed to be due to congestion: buffers get full somewhere between the two end points. Also, if a packet is damaged, use checksum, and drop

How to obtain reliable communication?

Flow Control: sliding window

Reconcile the speeds of sender and receiver

The receiver must have a way to tell the sender to stop sending if it is overwhelmed.

Add acknowledgement for packets.

The solution is to tell how many packets there can be which are not acknowledged. This is called the sliding window.

How to size the window? What if size is one? What if too large? The window controls the effective transmission rate.

Two competing goals:

large window → high throughput. More buffer space required.

small window → low throughput. Simpler protocol.

Loss and reordering

What if we add the facts that messages can be lost and reordered?

How can a sender identify the loss of a message?

Need 3 things:

1. Unique identifiers
2. Acknowledgements
3. Timeouts

Why do you need unique identifiers?

Why do you need acknowledgements?

Why do you need timeouts?

A packet that is lost must be resent.

Acknowledgements can also be lost!

A protocol may deal with acknowledgements in different ways. You can send acks for each packet you receive, or in blocks, stating the last one you received. You can also send negative acks, for specific packets you are missing.

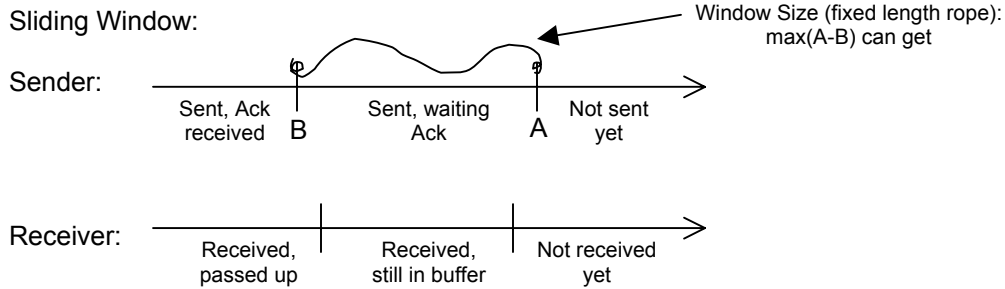
Message Corruption

What if messages can get corrupted along the way?

How do you cope with message corruption? You can add Checksums, and drop packets that are fail the check. You could also employ more sophisticated error correcting, but that is too much overhead sometimes.

TCP

Has flow control/congestion control by use of a sliding window. The window is sized so that the buffer in the client is not overrun. Also, it is dynamically adjusted to adapt to network conditions (congestion along the way).



Dynamically varies timeout: based on estimation on current Round Trip Time.

What if timer is too short?

What if timer is too long?

Window size: dynamically changes

Slow start: start with small window. Increase with each ack, if no timeouts. If a timeout occurs, cut window in half → TCP “searches” for the maximum window size that will not cause timeouts

Initializing sequence numbers:

TCP-A: “I want to connect, will start sending from number S” (SYN packet)

TCP-B: “Acknowledged S+1. Will start from R” (SYNACK packet)

TCP-A: “Acknowledged R+1” (ACK packet)

They are now synchronized, and can start communicating in any direction.

Each one keeps send and receive state.

TCP Throughput

Window: larger window, higher throughput

Latency: larger latency, smaller throughput (given a fixed window)

In Nachos:

Life is simpler:

Network will not reorder packets → easier to detect losses

Network will not corrupt packets → no need for checksums

Delay is not going to be unpredictable → no need for dynamic estimation of timers

No need for dynamic sizing of the window (no congestion in the network, bound by receiver buffer)

Another Solution: Reed-Solomon Codes or Erasure Coding

(This is just a sketch!)

Break message M in k pieces, create a polynomial f (degree is k-1) in which the coefficients are the message pieces. If you know the polynomial, you know the message.

Evaluate the polynomial in n points, and send the pairs $\langle x_i, f(x_i) \rangle$. How many points do you need to determine a polynomial of degree (k-1)?

Now, when the receiver gets **any** k pieces, she can reconstruct the message. This process can be made efficient in time, and in number of bits transmitted (you can guarantee that each $f(x_i)$ piece is m/k bits long).

Very cool!

This is all performed in modular arithmetic (modulo a large enough prime p), and there are other details. You can also add a few (**very** clever) tricks, to decode the message even in the presence of at most $((n-k-1)/2)$ messages with errors (this is known as the Berlekamp-Welsh code).

A transmission protocol using these codes may not depend on the latency, and the source can just keep sending new messages, without the need to retransmit old ones, until the receiver sends a message of completion.

See www.digitalfountain.com.