

cmisc 417 programming assignment #3

March 9, 2006

Goal: Implement unreliable rate-controlled message delivery to neighbours.

1 Deadline

DUE: April 3. That's a monday. The late deadline will be Friday, April 7; you have only one late deadline to play with.

2 Context

This project extends the previous project by providing rate-controlled message delivery called `sendmsg`.¹ The rate-control requirement ensures that messages from a host do not swamp its neighbours or the network. Because the service runs over UDP (over IP), it is unreliable.

The rate control is the so-called "leaky bucket" algorithm. Water (packets) pours into a leaky bucket (queue). When the bucket has water, it leaks steadily (sends at fixed rate). If the incoming water flow is so fast that it fills the bucket (queue is filled), water spills out (packets are dropped).

3 Service specification

The `sendmsg` service :

- Takes `dst-address`, the neighbour's logical address (the 32 bit source address in the packet header), and `msg`, a message to send, as parameters
- Limits the rate of sending to 10 packets (of maximum size 1000 bytes each) per second to each neighbour
- Uses a per-neighbour output queue of maximum size 10 packets (irrespective of packet size) to limit the sending rate

When `sendmsg` is invoked, it does the following in sequence:

- Look up the IP address of the neighbour in the neighbour table. (Don't send messages to the multicast address!)
- Check if last packet was sent 0.1 seconds ago or earlier:
 - If so, send the message immediately using UDP `sendto`
 - If not, enqueue the message in the output queue subject to maximum queue size
 - If the queue is full, drop packet
- Ensures that the queue, if not empty, is drained by one packet every 0.1 seconds.

The `sendmsg` service can return the following errors:

- Unknown or dead neighbour
- Queue overflow

¹Think of `sendmsg` as a wrapper over UDP `sendto`. This is an example of how services are composed over layers, i.e., a higher protocol layer uses a lower layer's service specification to provide something more.

4 Implementation requirements

In addition to the requirements of P2, P3 should:

- Implement a command from STDIN with signature `sendmsg dst-address message` providing the semantics of the `sendmsg` service.
- Handle queue overflow by printing “ERROR:NOBUFF”. Do not block the `sendmsg` command waiting for the queue to drain.
- Handle unknown or dead neighbour by printing “ERROR:NOROUTE”
- Not leak memory

5 Hints

- Figure out what events will be required to drain the buffer (queue) periodically and use your event queue appropriately.
- Be careful when `freeing` packets; you want to free them, but only when you actually can.
- You should be able to test by sending to yourself.

6 Objectives

At the end of this assignment, you should understand:

- maintaining queues of packets without coredumps :)
- scheduling events to manipulate queues of packets
- simple rate control