SOLUTION NOTES

Digital Communication II 2001 Paper 8 Question 3 (SMH)

Definition of congestion

Congestion occurs in a network when the arrival rate at a given router (or 'node') exceeds the service rate, causing the queue length to build up. This will increase the average experienced packet delay, and can eventually lead to packet loss (real queues are finite).

Evolution of TCP congestion control

Roughly four marks per variant. The precise definition of a "variant" is somewhat tricky, so marks will be given for any plausible, well-argued schemes.

In the beginning TCP had no congestion control *per se*; rather it had a *flow control* scheme which dealt with matching sender behaviour to receiver capabilities. Unfortunately as usage increased people observed *congestion collapse* — a situation in which a positive feedback loop could cause an entire network to cease useful operation.

The first congestion control scheme for TCP was proposed by VJ back in 1988, and is essentially an additive-increase multiplicative-decrease (AIMD) scheme. A *congestion window* is introduced to act as an upper bound on the transmitter's send window. Then, whenever an ACK arrives for a outstanding segment the congestion window is increased by a fraction of a packet, and whenever a packet is lost (i.e. a timeout occurs), the congestion window is halved. The "fraction of the packet" mentioned above is computed such that the congestion window will increase by an entire packet each round-trip time (RTT).

This variant also included the ill-named *slow start*, used when beginning a connection, or when recovering from loss. When slow start is in operation, the congestion window is incremented by an entire packet for every valid ACK received.

The variant just described only used timeouts to detect loss. However observe that TCP receivers issue an ACK whenever they receive a segment (this is the "self clocking" behaviour). So if a segment is lost (or re-ordered), the receiver will issue a *duplicate ACK*. Hence if the sender receives a duplicate ACK, there is a strong suggestion that a segment has been lost. If k duplicate ACKs are received in succession, this suggestion becomes an almost certainty.

This observation is used by the *fast retransmit* scheme: if three successive duplicate ACKs are received, loss is assumed and congestion recovery is initiated. Since fast retransmit was first added to the original TCP congestion scheme in the Tahoe release of BSD 4.3, this variant (slow start, congestion avoidance, congestion recovery and fast retransmit) is often called "TCP Tahoe".

The next variant, often called 'TCP Reno", added the *fast recovery* technique in which congestion recovery is *not* used when loss has been detected by the receipt of duplicate ACKs. Instead the congestion window is effectively set to half its previous value, and additive increase resumes. This is less drastic than using slow start, and is motivated by observing that the loss in question was apparently an isolated occurrence (since we've been receiving ACKs triggered by later segments).

Beyond Reno, we have a few "enhanced network" schemes such as RED, which drops packets probabilistically before queue overrun in order to avoid having to drop a large number of packets once the queue length is exceeded, and ECN, which allows routers to explicitly mark packets rather than dropping them, with a similar goal of hoping to get back off from end systems before a critical situation arises.

Returning to schemes implemented strictly at the end-systems we have "TCP Vegas". This has the sender explicitly estimate how much data it expects to transmit to the receiver in a given time period — i.e. the sender has an *expected rate* of throughput. The sender then measures the actual achieved rate and linearly adjusts its congestion window depending on the difference: if the difference is "too small", then the congestion window is linearly increased; if the difference is "too large", the congestion window is linearly decreased; if the difference is "just right", the congestion window remains the same.

Most recently, we have the idea of *congestion pricing*. This is a scheme in which congestion is explicitly signalled to end-systems (as with ECN), and where hosts adapt to these signals appropriately. The key difference is that it is up to each sender to decide what "appropriately" means. By assuming that (a) congestion causes packets to be marked, and (b) that users gain negative utility from each mark (viz. "marks cost something"), then the situation will reach pareto equilibrium.

The congestion pricing scheme has the advantage that it does not require any form of compliance from end-system implementations (something which is required for vanilla TCP congestion control schemes). Further, it allows proportional sharing of bandwidth, and is protocol agnostic.

Comparison with congestion control in ATM

Most of the congestion control schemes used with TCP expect little or no support from the network itself. Instead, they expect all of the hard work to be done by the end-systems. In ATM, both major ways of handling congestion suppose a more sophisticated set of intermediate nodes.

For CBR and VBR traffic flows, "congestion control" happens during the admission control phase of connection set-up. Per-flow parameters are signalled to each switch, and a decision is made based on expected impact of the new flow on existing ones. In general, each switch will admit flows only if it believes that the chance of congestion is very small.

For ABR traffic flows, a more dynamic scheme is needed since the flow is attempting to

adjust to the "spare" capacity within the network. This is rather more like what TCP attempts to do, and uses not dissimilar techniques.

An ATM VBR stream comprises 31 data cells followed by a *resource management* (RM) cell. As an RM cell passes though the network, switches en route can set the *congestion indication* (CI) or *no increase* (NI) bits.

These are reflected back by the peer end-system (much as per ECN in the internet), and hence return to the sender. The sender multiplicatively decreases its rate if the CI bit is set, does nothing if the NI bit is set, and otherwise linearly increases its sending rate. This is not unlike the behaviour of TCP with ECN, save that there are three states rather than two (i.e. a "do nothing" state as per TCP Vegas).

In addition, each switch can act in unison with another to perform "sub-path" congestion control; that is, the switches themselves can generate RMs and send them backwards towards the transmitter. This can result in faster reaction to critical network conditions. No variant of TCP includes anything like this.