Note that this is an "extended answer"

P8q3
SMH

Digital Communications II (Part II): Tripos Questions ~~1999~~/2000

Paper 8 Question 3 (20 Marks)

What information does a host require to transmit an IP datagram to a given destination IP address? [3 marks]

Describe how the host determines this information. How does this change in the presence of subnets? [3 marks]

Routing protocols can be classified as either *link-state* or *vector distance.*

- Briefly describe the operation of each of these schemes. [4 marks]
- In which case does the overall state converge more rapidly after a change. Justify your answer. [2 marks]
- Which scheme is more scalable? Justify your answer. [2 marks]

Sketch a design for a transport protocol providing reliable data transmission over IP multicast. What are the main issues here? Which changes within the network could lead to a more efficient implementation? [6 marks]

Answers to Paper 8 Question 3

Required Information

1 mark for each the below (or any viable alternatives)

To transmit an IP datagram the host needs to know:

a) the outgoing interface — that is, the physical device and associated IP address. The former is required so that the packet can actually be sent, the latter so that the source IP address can be stamped into the header.

b) the MTU on this interface: this is so that it can fragment the datagram if required.

c) the next-hop (or final hop) mac-address to actually allow the transmission of the packet.

Routing in Host

Two marks for the basic IP routing algorithm; one mark for explaining version with subnets.

The host determines this information by *routing* the datagram. This takes place with the assistance of a *routing* table which contains a set of entries of the form { prefix, next-hop, outgoing interface, mtu, metric }. To route a datagram, we simply (a) determine the prefix of the destination IP address, and (b) look this up in the routing table. If we do not have subnets, obtaining the prefix of the destination address is easy; we simply work it out depending on whether the address is class A, B or C.

Typically we allow some special cases such as *host routes*, i.e. routes based on the entire destination address rather than simply its prefix. A host route overrides a route for the network within which that host resides. We also allow the use of a *default route* to be used in the case that no match is found.

In the presence of subnets we need to keep information about the length of the prefix on a per routing table entry (usually done by using the "netmask" technique), and we also need to do longest prefix match. Once we've done this it turns out we can fold the host route and default cases into the same scheme (with netmasks of 255.255.255.255 and 0.0.0.0 respectively).

Routing Protocol Descriptions

Two marks for each description

Link-state schemes require that each participating router maintain complete topology information, including information about the *cost* of each link in the network (i.e. each node holds a weighted graph). Given this information, each router can (and does) independently run Dijkstra, and hence can route datagrams to the requisite network via the shortest path.

With link-state, each node monitors the state of its [direct] links by 'pinging'[1] each of its neighbours. Whenever a node decides the link state has changed (viz. from "up" to "down" or vice versa), it *broadcasts* this information to all other routers. In a nutshell, link state schemes communicate *local* information to a *global* set of recipients.

In vector distance (aka distance vector aka Bellman-Ford) routing schemes, each node only holds partial information about the topology. This information is actually its routing table, i.e. an entry for each network it can reach giving the next hop and the *cost*. The cost here is the number of hops to that network.

Periodically each router sends a copy of its routing table to each of its neighbours. This is called *route advertisement*. When a router hears an advertised route from a neighbour, it knows that is could reach the given network at a cost of (advertised-cost + 1). If this is better than its existing metric, or if it has no entry for that network, it updates its own

[1] Doesn't actually use ping! e.g. OSPF uses "Hello" messages.

routing table. In a nutshell, link state schemes communicate *global* information to a *local* set of recipients.

Routing Protocol Convergence

In general link-state schemes converge far more rapidly. This is because updated information is immediately propagated to all participating routers; after this, all that is required is that each router recompute the shortest paths.

An unmodified distance vector scheme converges after on average $d/2$ "update periods" where d is the diameter of the network (in hops). This is particularly poor in the case of "bad news" (e.g. when a link fails) — get the counting to infinity problem. Hence triggered updates, split horizon, poison reverse, etc.

Routing Protocol Scalability

Either of the below answers is acceptable.

Distance-vector schemes are less scalable since:

a) the size of the messages depends on the size of the routing tables and hence of the number of routes in the entire system. This is $O(n^2)$ where n is the number of links in the network. In link-state schemes the size of messages is $O(n)$.

b) as the network gets larger, the diameter gets larger, and hence the update time (which is $kd/2$) gets worse.

Alternatively one can argue that link-state schemes are less scalable since they require broadcasting to every participating router. As the number of routers gets large and/or widely distributed, this becomes very inefficient. [hence e.g. OSPF has special support for multi-access networks].

Reliable Multicast

This key thing here is how one handles (a) ACKs or whatever and (b) windows. Marks will be given for a suitable exposition of the issues, and a plausible solution.

The main issues with reliability are:

- Who is responsible for detecting the errors?
  A reliable multicast protocols must detect the error in order to request its retransmission. One approach is to have the sender detect that an error has occurred (e.g. making sure that each receiver acknowledges the having received the packet). An

alternative is to make each receiver independently responsible for detecting its errors (e.g. a gap in the sequence numbers may signal an error).

- How are errors signaled?
  When an error is detected, it must be signaled to whomever is going to retransmit the data (usually the sender). Common practices are to send a special control packet such as an ACK(positive acknowledgement) or an NAK (negative acknowledgement). Since this may result in an implosion of control packets being sent to the sender, a number of researchers are looking into techniques to reduce the number of control packets returned (statistical backoffs, control packet aggregation).

- How are errors retransmitted?
  The final problem for reliable multicast is how the error is retransmitted. Some approaches include unicasting the packet to the appropriate receiver; multicasting it to all receivers and letting the receivers filter out the duplicate packets; having the routers perform the retransmission; or requesting the missing data from a neighboring receiver.

Given the above, a plausible design might be to use (a) receiver-based error detection since this scales better and (b) a custom form of selective acknowledgement to minimize the 'back traffic' — for example a single selective acknowledgement packet might be sent per receiver per n successfully received packets. Retransmission by the server could dynamically move between k unicasts and multicast depending on k and the size of the multicast group [if known!]. This also brings up the issue of fragmentation : perhaps the simplest solution would be only every to use 536 byte datagrams, although this would probably result in reduced performance.

The second main issue is dealing with flow control. Controlling the packet rate in multicasting is complicated by the fact that the protocol must accommodate multiple receivers simultaneously. How this is performed can have significant impact on the overall performance. For example, if the sender always operates at the rate of the slowest receiver, then faster receivers sit idle while they wait for additional information.

One solution here is to use multiple multicast groups which receivers can join or leave depending on the rate at which they are prepared to receive. Unfortunately this complicates the sender significantly. A similar scheme can be used to handle different MTUs.

Other issues are fragmentation (see above) and ordering (i.e. do we require strict ordering semantics on messages). The latter might be a requirement for some security or transaction-based protocols, it can probably be avoided in e.g. multicast file-transfer protocols.

Support in multicast routers could make this whole problem a lot easier; for example, the routers could buffer received packets and retransmit missing packets [on e.g. one interface only]. Taking this even further, multicast routers could also actively get involved in flow control. Basically, having the hierarchy get involved means that most things can be done more efficiently (in terms of network traffic). This is roughly the approach taken by LGMP.